nature protocols

PROTOCOL UPDATE
https://doi.org/10.1038/s41596-018-0098-2

Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0

Laurent Heirendt[1,24], Sylvain Arreckx[1,24], Thomas Pfau[2], Sebastián N. Mendoza[3,4], Anne Richelle[5], Almut Heinken[1], Hulda S. Haraldsdóttir[1], Jacek Wachowiak[1], Sarah M. Keating[6], Vanja Vlasov[1], Stefania Magnusdóttir[1], Chiam Yu Ng[7], German Preciat[1], Alise Žagare[1], Siu H. J. Chan[7], Maike K. Aurich[1], Catherine M. Clancy[1], Jennifer Modamio[1], John T. Sauls[8], Alberto Noronha[1], Aarash Bordbar[9], Benjamin Cousins[10], Diana C. El Assal[1], Luis V. Valcarcel[11], Iñigo Apaolaza[11], Susan Ghaderi[1], Masoud Ahookhosh[1], Marouen Ben Guebila[1], Andrejs Kostromins[12], Nicolas Sompairac[13], Hoai M. Le[1], Ding Ma[14], Yuekai Sun[15], Lin Wang[7], James T. Yurkovich[16], Miguel A. P. Oliveira[1], Phan T. Vuong[1], Lemmer P. El Assal[1], Inna Kuperstein[13], Andrei Zinovyev[13], H. Scott Hinton[17], William A. Bryant[18], Francisco J. Aragón Artacho[19], Francisco J. Planes[11], Egils Stalidzans[12], Alejandro Maass[3,4], Santosh Vempala[10], Michael Hucka[20], Michael A. Saunders[14], Costas D. Maranas[7], Nathan E. Lewis[5,21], Thomas Sauter[2], Bernhard Ø. Palsson[16,22], Ines Thiele[1] and Ronan M. T. Fleming[1,23]*

Constraint-based reconstruction and analysis (COBRA) provides a molecular mechanistic framework for integrative analysis of experimental molecular systems biology data and quantitative prediction of physicochemically and biochemically feasible phenotypic states. The COBRA Toolbox is a comprehensive desktop software suite of interoperable COBRA methods. It has found widespread application in biology, biomedicine, and biotechnology because its functions can be flexibly combined to implement tailored COBRA protocols for any biochemical network. This protocol is an update to the COBRA Toolbox v.1.0 and v.2.0. Version 3.0 includes new methods for quality-controlled reconstruction, modeling, topological analysis, strain and experimental design, and network visualization, as well as network integration of chemoinformatic, metabolomic, transcriptomic, proteomic, and thermochemical data. New multi-lingual code integration also enables an expansion in COBRA application scope via high-precision, high-performance, and nonlinear numerical optimization solvers for multi-scale, multi-cellular, and reaction kinetic modeling, respectively. This protocol provides an overview of all these new features and can be adapted to generate and analyze constraint-based models in a wide variety of scenarios. The COBRA Toolbox v.3.0 provides an unparalleled depth of COBRA methods.

This protocol is an update to *Nat. Protoc.* 2, 727–738 (2007): https://doi.org/10.1038/nprot.2007.99 and *Nat. Protoc.* 6, 1290–1307 (2011): https://doi.org/10.1038/protex.2011.234

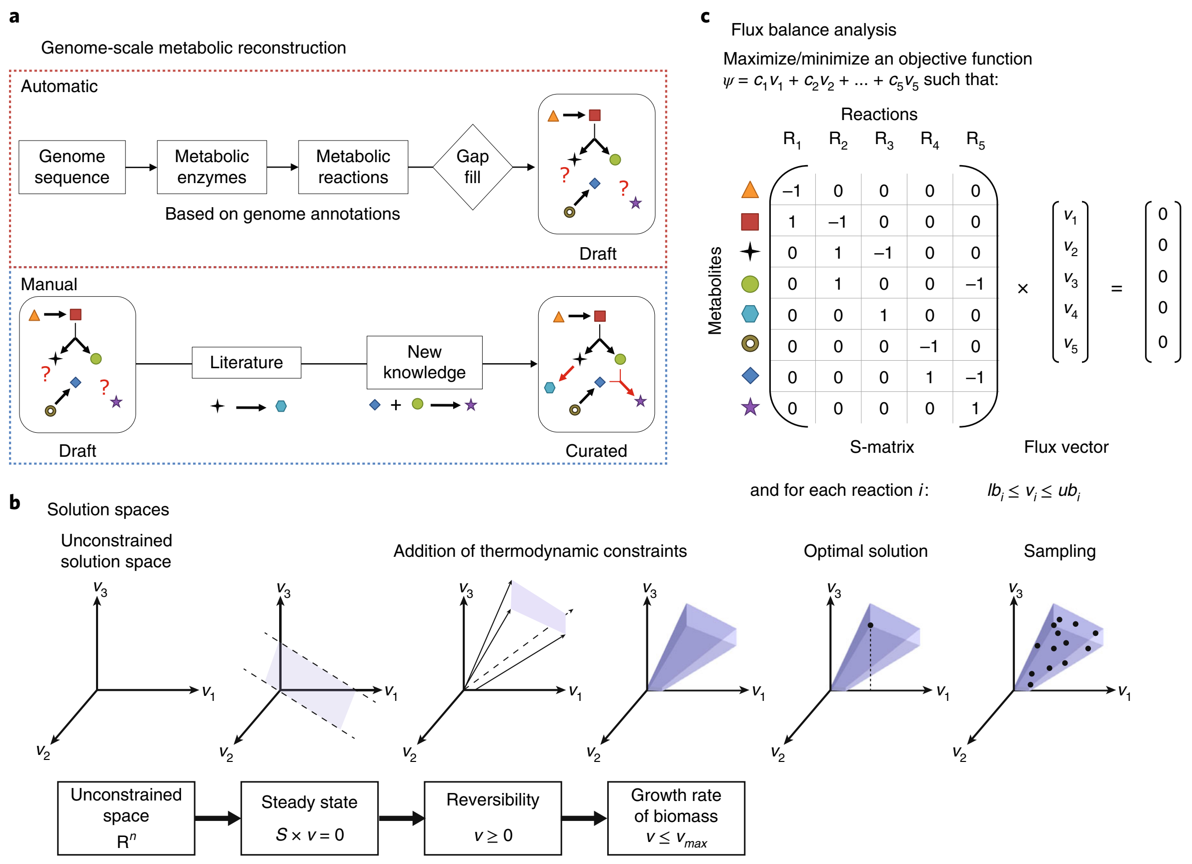
[1]Luxembourg Centre for Systems Biomedicine, University of Luxembourg, Belvaux, Luxembourg. [2]Life Sciences Research Unit, University of Luxembourg, Belvaux, Luxembourg. [3]Center for Genome Regulation (Fondap 15090007), University of Chile, Santiago, Chile. [4]Mathomics, Center for Mathematical Modeling, University of Chile, Santiago, Chile. [5]Department of Pediatrics, University of California, San Diego, School of Medicine, La Jolla, CA, USA. [6]European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Hinxton, Cambridge, UK. [7]Department of Chemical Engineering, The Pennsylvania State University, State College, PA, USA. [8]Department of Physics, and Bioinformatics and Systems Biology Program, University of California, San Diego, La Jolla, CA, USA. [9]Sinopia Biosciences, San Diego, CA, USA. [10]Algorithms and Randomness Center, School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA. [11]Biomedical Engineering and Sciences Department, TECNUN, University of Navarra, San Sebastián, Spain. [12]Institute of Microbiology and Biotechnology, University of Latvia, Riga, Latvia. [13]Institut Curie, PSL Research University, Mines Paris Tech, Inserm, U900, Paris, France. [14]Department of Management Science and Engineering, Stanford University, Stanford, CA, USA. [15]Department of Statistics, University of Michigan, Ann Arbor, MI, USA. [16]Department of Bioengineering, University of California, San Diego, La Jolla, CA, USA. [17]Utah State University Research Foundation, North Logan, UT, USA. [18]Centre for Integrative Systems Biology and Bioinformatics, Department of Life Sciences, Imperial College London, London, UK. [19]Department of Mathematics, University of Alicante, Alicante, Spain. [20]Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, USA. [21]Novo Nordisk Foundation Center for Biosustainability, University of California, San Diego, La Jolla, CA, USA. [22]Novo Nordisk Foundation Center for Biosustainability, Technical University of Denmark, Kemitorvet, Lyngby, Denmark. [23]Division of Systems Biomedicine and Pharmacology, Leiden Academic Centre for Drug Research, Faculty of Science, Leiden University, Leiden, The Netherlands. [24]These authors contributed equally: Laurent Heirendt, Sylvain Arreckx. *e-mail: ronan.mt.fleming@gmail.com

Introduction

Development of the protocol

COBRA[1] is a mechanistic integrative analysis framework that is applicable to any biochemical system with prior mechanistic information, including those for which mechanistic information is incomplete. The overall approach is to mechanistically represent the relationship between genotype and phenotype by mathematically and computationally modeling the constraints that are imposed on the phenotype of a biochemical system by physicochemical laws, genetics, and the environment[2] (Fig. 1). This protocol updates and extends previous protocols on the COBRA Toolbox v.1.0 (ref. [3]) and v.2.0 (ref. [4]). It provides an introduction to the practical application of many of the novel COBRA methods developed in recent years.

Early in the development of the COBRA framework, the need for ease of reproducibility and demand for reuse of COBRA methods were recognized. This necessity led to the development of COBRA Toolbox v.1.0 (ref. [3])—an open source software package running in the MATLAB environment—which facilitated quantitative prediction of metabolic phenotypes using a selection of the COBRA methods available at the time. With the expansion of the COBRA community and the growing phylogeny of COBRA methods, the need for the amalgamation and transparent dissemination of COBRA methods was recognized. This demand led to the COBRA Toolbox v.2.0


a
Genome-scale metabolic reconstruction
Automatic
Genome sequence
Metabolic enzymes
Metabolic reactions
Gap fill
Based on genome annotations
Draft
Manual
Literature
New knowledge
Draft
Curated
b
Solution spaces
Unconstrained solution space
Addition of thermodynamic constraints
Optimal solution
Sampling
Unconstrained space R^n
Steady state $S \times v = 0$
Reversibility $v \geq 0$
Growth rate of biomass $v \leq v_{max}$
c
Flux balance analysis
Maximize/minimize an objective function $\psi = c_1v_1 + c_2v_2 + ... + c_5v_5$ such that:
Reactions
Metabolites
S-matrix
Flux vector
and for each reaction i: $lb_i \leq v_i \leq ub_i$

$$\begin{array}{c|ccccc} & R_1 & R_2 & R_3 & R_4 & R_5 \\ \hline ▲ & -1 & 0 & 0 & 0 & 0 \\ ■ & 1 & -1 & 0 & 0 & 0 \\ ✦ & 0 & 1 & -1 & 0 & 0 \\ ● & 0 & 1 & 0 & 0 & -1 \\ ⬢ & 0 & 0 & 1 & 0 & 0 \\ ◎ & 0 & 0 & 0 & -1 & 0 \\ ◆ & 0 & 0 & 0 & 1 & -1 \\ ★ & 0 & 0 & 0 & 0 & 1 \end{array} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Fig. 1 | Overview of key constraint-based reconstruction and analysis concepts. a, A genome-scale metabolic reconstruction is a structured knowledge base that abstracts pertinent information on the biochemical transformations taking place within a chosen biochemical system, e.g., the human gut microbiome[38]. Genome-scale metabolic reconstructions are built in two steps. First, a draft metabolic reconstruction based on genome annotations is generated using one of several platforms. Second, the draft reconstruction is refined on the basis of known experimental and biochemical data from the literature[6]. Novel experiments can be performed on the organism and the reconstruction can be refined accordingly. **b**, A phenotypically feasible solution space is defined by specifying certain assumptions, e.g., a steady-state assumption, and then converting the reconstruction into a computational model that eliminates physicochemically or biochemically infeasible network states. Various methods are used to interrogate the solution space. For example, optimization for a biologically motivated objective function (e.g., biomass production) identifies a single optimal flux vector (v), whereas uniform sampling provides an unbiased characterization via flux vectors uniformly distributed in the solution space. **c**, Flux balance analysis is an optimization method that maximizes a linear objective function, $\psi(v) = c^Tv$, formed by multiplying each reaction flux v_j by a predetermined coefficient, c_j, subject to a steady-state assumption, $Sv = 0$, as well as lower and upper bounds on each reaction flux (lb_j and ub_j, respectively).

Table 1 | Main COBRA methods currently available in the COBRA Toolbox

src/	Narrative	Novelty in the COBRA Toolbox v.3.0 compared to v.2.0
B	Initialize and verify the installation	Software dependency audit, e.g., solvers, binaries, and git
R	Input and output of reconstructions and models	Support for latest standards, e.g., SBML flux balance constraints[64]
R	Reconstruction: rBioNet	New software for quality-controlled reconstruction[48]
R	Reconstruction: create a functional generic subnetwork	New methods for selecting different types of subnetworks
R	Reconstruction exploration	New methods, e.g., find adjacent reactions
R	Reconstruction refinement	Maintenance of internal model consistency, e.g., upon subnetwork generation[29]
R	Numerical reconstruction properties	Flag a reconstruction requiring a multi-scale solver[54]
R	Convert a reconstruction into a flux balance analysis model	Identification of a maximal flux and stoichiometrically consistent subset[69]
I	Atomically resolve a metabolic reconstruction	New algorithms and methods for working with molecular structures, atom-mapping, identification of conserved moieties[110,122]
I	Integration of metabolomic data	New methods for analysis of metabolomic data in a network context[65,142]
I	Integration of transcriptomic and proteomic data	New algorithms for generation of context-specific models[89]
A	Flux balance analysis and its variants	New flux balance methods, multi-scale model rescaling and multi-scale solvers, additional solver interfaces, thermodynamically feasible methods[42,60,128,132,143,144]
A	Variation on reaction rate bounds in flux balance analysis	Increased computational efficiency
A	Parsimonious flux balance analysis	New method for parsimonious flux balance analysis[145]
A	Sparse flux balance analysis	New method for sparse flux balance analysis
A	Gap filling	Increased computational efficiency[82]
A	Adding biological constraints to a flux balance model	New methods for coupling reaction rates[38,146]
A	Testing biochemical fidelity	Human metabolic function test suite[17]
A	Testing basic properties of a metabolic model (sanity checks)	New methods to minimize occurrence of modeling artifacts[66]
A	Minimal spanning pathway vectors	New method for determining minimal spanning pathway vectors[100]
A	Elementary modes and pathway vectors	Extended functionality by integration with CellNetAnalyzer[55]
A	Minimal cut sets	Extended functionality by integration with CellNetAnalyzer[147,148], and new algorithms for genetic MCS[57]
A	Flux variability analysis	Increased computational efficiency[101]
A	Uniform sampling of steady-state fluxes	New algorithm, guaranteed convergence to uniform distribution[102]
I	Thermodynamically constrain reaction directionality	New algorithms and methods for estimation of thermochemical parameter estimation in multi-compartmental, genome-scale metabolic models[126,127]
A	Variational kinetic modeling	New algorithms and methods for genome-scale kinetic modeling[68,135–137]
D	Metabolic engineering and strain design	New methods, e.g., OptForce, interpretation of new strain designs. New modeling language interface to GAMS[59]
V	Human metabolic network visualization: ReconMap	New method for genome-scale metabolic network visualization[50,51,149]
V	Variable scope visualization with automatic layout generation	New method for automatic visualization of network parts[141]
B	Contributing to the COBRA Toolbox with MATLAB.devTools	New software application enabling contributions by those unfamiliar with version-control software
B	Engaging with the COBRA Toolbox Forum	>800 posted questions with supportive replies connecting problems and solutions

Each method available in the COBRA Toolbox v.3.0 is made accessible with a narrative tutorial that illustrates how the corresponding function(s) are combined to implement each COBRA method in the respective src/ directories (https://github.com/opencobra/cobratoolbox/tree/master/src): base (B), reconstruction (R), dataIntegration (I), analysis (A), design (D), and visualization (V).

(ref. [4]), which offered an enhanced range of methods to simulate, analyze, and predict a variety of phenotypes using genome-scale metabolic reconstructions. Since then, the increasing functional scope and size of biochemical network reconstructions, as well as the increasing breadth of physicochemical and biological constraints that are represented within constraint-based models, naturally resulted in the development of a broad arbor of new COBRA methods[5].

The present protocol provides an overview of the main novel developments within v.3.0 of the COBRA Toolbox (Table 1), especially the expansion of functionality to cover new biochemical network reconstruction and modeling methods. In particular, this protocol includes the input and output of new standards for sharing reconstructions and models, an extended suite of supported general-purpose optimization solvers, new optimization solvers developed especially for constraint-based modeling problems, enhanced functionality in the areas of computational efficiency and high-precision computing, numerical characterization of reconstructions, conversion of reconstructions into various forms of constraint-based models, comprehensive support for flux balance analysis (FBA) and its variants, integration with omics data, uniform sampling of high-dimensional models, atomic resolution of metabolic reconstructions via molecular structures, estimation, and application of thermodynamic constraints, visualization of metabolic networks, and genome-scale kinetic modeling.

This protocol consists of a set of methods that are introduced in sequence but can be combined in a multitude of ways. The overall purpose is to enable the user to generate a biologically relevant, high-quality model that enables novel predictions and hypothesis generation. Therefore, we implement and enforce standards in reconstruction and simulation that have been developed by the COBRA community over the past two decades. All explanations of a method are also accompanied by explicit computational commands.

First, we explain how to initialize and verify the installation of the COBRA Toolbox in MATLAB (MathWorks). The main options for importing and exploring the content of a biochemical network reconstruction are introduced. For completeness, a brief summary of methods for manual and algorithmic reconstruction refinement is provided, with reference to the established reconstruction protocol[6]. We also explain how to characterize the numerical properties of a reconstruction, especially with respect to detection of a reconstruction requiring a multi-scale numerical optimization solver. We explain how to semi-automatically convert a reconstruction into a constraint-based model suitable for FBA. This is followed by an extensive explanation of how to carry out FBA and its variants. The procedure to fill gaps in a reconstruction, due to missing reactions, is also explained.

We provide an overview of the main methods for integration of metabolomic, transcriptomic, proteomic, and thermochemical data to generate context-specific, constraint-based models. Various methods are explained for the addition of biological constraints to a constraint-based model. We then explain how to test the chemical and biochemical fidelity of the model. After a high-quality model has been generated, we explain how to interrogate the discrete geometry of its stoichiometric subspaces, how to efficiently measure the variability associated with the prediction of steady-state reaction rate using flux variability analysis, and how to uniformly sample steady-state fluxes. We introduce various approaches for prospective use of a constraint-based model, including strain design and experimental design.

We explain how to atomically resolve a metabolic reconstruction by connecting it with molecular species structures and how to use cheminformatic algorithms for atom mapping and identification of conserved moieties. Using molecular structures for each metabolite, and established thermochemical data, we estimate the transformed Gibbs energy of each subcellular compartment-specific reaction in a model of human metabolism in order to thermodynamically constrain reaction directionality and constrain the set of feasible kinetic parameters. Sampled kinetic parameters are then used for variational kinetic modeling in an illustration of the utility of recently published algorithms for genome-scale kinetic modeling. We also explain how to visualize predicted phenotypic states using a recently developed approach for metabolic network visualization. We conclude with an explanation of how to engage with the community of COBRA developers, as well as contribute code to the COBRA Toolbox with MATLAB.devTools, a newly developed piece of software for community contribution of COBRA methods to the COBRA Toolbox.

All documentation and code is released as part of the openCOBRA project (https://github.com/opencobra/cobratoolbox). Where reading the extensive documentation associated with the COBRA Toolbox does not suffice, we describe the procedure for effectively engaging with the community via a dedicated online forum (https://groups.google.com/forum/#!forum/cobra-toolbox). Taken together, the COBRA Toolbox v.3.0 provides an unparalleled depth of interoperable COBRA methods and a proof of concept that knowledge integration and collaboration by large numbers of scientists can lead to cooperative advances impossible to achieve by a single scientist or research group alone[7].

Applications of COBRA methods

Constraint-based modeling of biochemical networks is broadly applicable to a range of biological, biomedical, and biotechnological research questions[8]. Fundamentally, this broad applicability arises from the common phylogenetic tree, shared by all living organisms, that manifests in a set of shared mathematical properties that are common to biochemical networks in normal, diseased, wild-type, or mutant biochemical networks. Therefore, a COBRA method developed primarily for use in one scenario can usually be quickly adapted for use in a variety of related scenarios. Often, this adaptation retains the mathematical properties of the optimization problem underlying the original

Table 2 | A selection of actively developed software applications with constraint-based modeling (COBRA) capabilities

Name	Implementation	Interface	Development	Distribution	OS
COBRA Toolbox	MATLAB (and others)	Script/narrative	Open source[a]	git	All
RAVEN[150]	MATLAB	Script	Open source[a]	git	All
CellNetAnalyzer[55]	MATLAB (and others)	Script/GUI	Closed source[a]	zip	All
FBA-SimVis[151]	Java + MATLAB	GUI	Closed source[b]	zip	Windows
OptFlux[152]	Java	Script	Open source[a]	svn	All
COBRA.jl[42]	Julia	Script/narrative	Open source[a]	git	All
Sybil[53]	R package	Script	Open source[a]	zip	All
COBRApy[40]	Python	Script/narrative	Open source[a]	git	All
CBMPy[52]	Python	Script	Open source[a]	zip	All
Scrumpy[153]	Python	Script	Open source[a]	tar	All
SurreyFBA[47]	C++	Script/GUI	Open source[a]	zip	All
FASIMU[154]	C	Script	Open source[b]	zip	Linux
FAME[155]	Web-based	GUI	Open source[b]	zip	All
PathwayTools[43]	Web-based	GUI/script	Closed source[a]	zip	All
KBase[41]	Web-based	Script/narrative	Open source[a]	git	All

GUI, graphical user interface; NA, not applicable. The COBRA Toolbox: https://opencobra.github.io/cobratoolbox; RAVEN: https://github.com/SysBioChalmers/RAVEN; CellNetAnalyzer: https://www2.mpi-magdeburg.mpg.de/projects/cna/cna.html; FBA-SimVis: https://immersive-analytics.infotech.monash.edu/fbasimvis; OptFlux: http://www.optflux.org; COBRA.jl: https://opencobra.github.io/COBRA.jl; Sybil: https://rdrr.io/cran/sybil; COBRApy: http://opencobra.github.io/cobrapy; CBMPy: http://cbmpy.sourceforge.net; SurreyFBA: http://sysbio.sbs.surrey.ac.uk/sfba; FASIMU: http://www.bioinformatics.org/fasimu; FAME: http://f-a-m-e.org; Pathway Tools: http://bioinformatics.ai.sri.com/ptools; KBase: https://kbase.us. Software may be distributed by version-controlled repositories (git, svn) or as compressed files (zip, tar). The 'All' label in the OS column means that the application is compatible with Windows, Linux and Mac operating systems. [a]Active project. [b]Inactive project.

constraint-based modeling method. By adapting the input data and interpreting the output results in a different way, the same method can be used to address a different research question.

Biotechnological applications of constraint-based modeling include the development of sustainable approaches for chemical[9] and biopharmaceutical production[10,11]. Among these applications is the computational design of new microbial strains for production of bioenergy feedstocks from non-food plants, such as microbes capable of deconstructing biomass into their sugar subunits and synthesizing biofuels, either from cellulosic biomass or through direct photosynthetic capture of carbon dioxide.

Another prominent biotechnological application is the analysis of interactions between organisms that form biological communities and their surrounding environments, with a view toward utilization of such communities for bioremediation[12] or nutritional support of non-food plants for bioenergy feedstocks. Biomedical applications of constraint-based modeling include the prediction of the phenotypic consequences of single-nucleotide polymorphisms[13], drug targets[14], and enzyme deficiencies[15–18], as well as side and off-target effects of drugs[19–21]. COBRA has also been applied to generate and analyze normal and diseased models of human metabolism[17,22–25], including organ-specific models[26–28], multi-organ models[29,30], and personalized models[31–33]. Constraint-based modeling has also been applied to understanding of the biochemical pathways that interlink diet, gut microbial composition, and human health[34–38].

Key features and comparisons

Aside from the COBRA Toolbox, COBRA can be carried out with a variety of software tools. In 2012, Lakshmanan et al.[39] made a comprehensive, comparative evaluation of the usability, functionality, graphical representation, and interoperability of the tools available for FBA. Each of these evaluation criteria is still valid when comparing the current version of the COBRA Toolbox with other software with constraint-based modeling capabilities. The rapid development of novel constraint-based modeling algorithms requires continuity of software development. Short-term investment in new COBRA modeling software applications has led to a plethora of COBRA modeling applications[39]. Each usually provides some unique capability initially, but many have become antiquated because of lack of maintenance, failure to upgrade, or failure to support new standards in model exchange formats (http://sbml.org/Documents/Specifications). Therefore, we also restrict our comparison to software in active development (Table 2).

Each software tool for constraint-based modeling has varying degrees of dependency on other software. Web-based applications exist for the implementation of a limited number of standard constraint-based modeling methods. Their only local dependency is on a web browser. The COBRA Toolbox depends on MATLAB (MathWorks), a commercially distributed, general-purpose computational tool. MATLAB is a multi-paradigm programming language and numerical computing environment that allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran, and Python. All software tools for constraint-based modeling also depend on at least one numerical optimization solver. The most robust and efficient numerical optimization solvers for standard problems are distributed commercially, but often offer free licenses for academic use, e.g., Gurobi Optimizer (http://www.gurobi.com). Stand-alone constraint-based modeling software tools also exist, and their dependency on a numerical optimization solver is typically satisfied by GLPK (https://gnu.org/software/glpk), an open-source linear optimization solver.

Some perceive a commercial advantage to depending only on open-source software. However, there are also commercial costs associated with dependency on open-source software. That is, in the form of increased computation times as well as increased time required to install, maintain, and upgrade open-source software dependencies. This is an important consideration for any research group whose primary focus is on biological, biomedical, or biotechnological applications, rather than on software development. The COBRA Toolbox v.3.0 strikes a balance by depending on closed-source, general purpose, commercial computational tools, yet all COBRA code is distributed and developed in an open-source environment (https://github.com/opencobra/cobratoolbox).

The availability of comprehensive documentation is an important feature in the usability of any modeling software. Therefore, a dedicated effort has been made to ensure that all functions in the COBRA Toolbox v.3.0 are comprehensively and consistently documented. Moreover, we also provide a new suite of >35 tutorials (https://opencobra.github.io/cobratoolbox/latest/tutorials) to enable beginners, as well as intermediate and advanced users to practice a wide variety of COBRA methods. Each tutorial is presented in a variety of formats, including as a MATLAB live script, which is an interactive document, or narrative (https://mathworks.com/help/matlab/matlab_prog/what-is-a-live-script.html), that combines MATLAB code with embedded output, formatted text, equations, and images in a single environment viewable with the MATLAB Live Editor (v.R2016a or later). MATLAB live scripts are similar in functionality to Mathematica Notebooks (Wolfram) and Jupyter Notebooks (https://jupyter.org). The latter support interactive data science and scientific computing for >40 programming languages. To date, only the COBRA Toolbox v.3.0, COBRApy[40], KBase[41], and COBRA.jl[42] offer access to constraint-based modeling algorithms via narratives.

KBase is a collaborative, open environment for systems biology of plants, microbes, and their communities[41]. It also has a suite of analysis tools and data that support the reconstruction, prediction, and design of metabolic models in microbes and plants. These tools are tailored toward the optimization of microbial biofuel production and the identification of minimal media conditions under which that fuel is generated, and predict soil amendments that improve the productivity of plant bioenergy feedstocks. In our view, KBase is currently the tool of choice for the automatic generation of draft microbial metabolic networks, which can then be imported into the COBRA Toolbox for further semi-automated refinement, which has recently successfully been completed for a suite of gut microbial organisms[38]. However, KBase[41] currently offers a modest depth of constraint-based modeling algorithms.

MetaFlux[43] is a web-based tool for the generation of network reconstructions directly from pathway and genome databases, proposing network refinements to generate functional flux balance models from reconstructions, predict steady-state reaction rates with FBA, and interpret predictions in a graphical network visualization. MetaFlux is tightly integrated within the PathwayTools[44] environment, which provides a broad selection of genome, metabolic, and regulatory informatics tools. As such, PathwayTools provides breadth in bioinformatics and computational biology, whereas the COBRA Toolbox v.3.0 provides depth in constraint-based modeling, without providing, for example, any genome informatics tools. Although an expert can locally install a PathwayTools environment, the functionality is closed source and only accessible via an application programming interface. This approach does not permit the level of repurposing possible with open-source software. As recognized in the computational biology community[45], open-source development and distribution is scientifically important for tractable reproducibility of results, as well as reuse and repurposing of code[46].

Lakshmanan et al.[39] consider the availability of a graphical user interface to be an important feature in the usability of modeling software. For example, SurreyFBA[47] provides a command-line tool and graphical user interface for constraint-based modeling of genome-scale metabolic reaction networks. The time lag between the development of a new modeling method and its availability via a graphical user interface necessarily means that graphically driven COBRA tools permit a limited depth of novel constraint-based modeling methods. Whereas MATLAB provides a generic graphical user interface, the COBRA Toolbox is controlled either by scripts or narratives, rather than graphically. Exceptions include the input of manually curated data during network reconstruction[48], the assimilation of genome-scale metabolic reconstructions[49], and the visualization of simulation results in biochemical network maps[50] via specialized network visualization software[51].

Owing to the relative simplicity of the MATLAB programming language, new COBRA Toolbox users, including those without software development experience, can rapidly become familiar with the basics of constraint-based modeling. This initial learning effort is worthwhile for the flexibility it opens up, especially considering the broad array of constraint-based modeling methods now available within the COBRA Toolbox v.3.0. Although it should be technically possible to generate a computational specification of the point-and-click analysis steps that are required to generate results using a graphical user interface, to our knowledge, none of the graphically driven modeling tools in Table 2 offers this facility. Such a specification would be required for another scientist to reproduce the same results using the same tool. This weakness limits the ability to reproduce analytical results, as verbal specification is not sufficient for reproducibility[46].

Each language-specific COBRA implementation has its benefits and drawbacks, which are mainly associated with the programming language itself. PySCeS-CBM[52] and COBRApy[40] both provide support for a set of COBRA methods implemented in the Python programming language. Python is a multi-paradigm, interpreted programming language for general-purpose programming. It has a broad development community and a wide range of open-source libraries, especially in bioinformatics. As such, it is well suited to the amalgamation and management of heterogeneous experimental data. At present, the COBRA software tools in Python provide access to standard COBRA methods. In COBRApy[40], this functionality can be extended by using Python to invoke MATLAB and use the COBRA Toolbox. Achieving such interoperability between COBRA software implemented in different programming languages and developed together by a united open-source community is the primary objective of the openCOBRA project (https://opencobra.github.io).

Sybil[53] is an open-source, object-oriented software library that implements a limited set of standard constraint-based modeling algorithms in the programming language R, which is a free, platform-independent environment for statistical computing and graphics. Sybil is available for download from the comprehensive R archive network (CRAN) but does not follow an open-source development model. The COBRA Toolbox is primarily implemented in MATLAB, a proprietary, multi-paradigm, programming language that is interpreted for execution rather than compiled before execution. As such, MATLAB code typically runs more slowly than compiled code, but its main advantage is the ability to rapidly and flexibly implement sophisticated numerical computations by leveraging the extensive libraries for general-purpose numerical computing, both those supplied commercially within MATLAB and those distributed freely by the community (https://mathworks.com/matlabcentral).

For the application of computationally demanding constraint-based modeling methods to high-dimensional or high-precision constraint-based models, the COBRA Toolbox v.3.0 comes with an array of integrated, pre-compiled extensions and interfaces that employ complementary programming languages and tools. These include a quadruple precision Fortran 77 optimization solver implementation for constraint-based modeling of multi-scale biochemical networks[54] and a high-level, high-performance, open-source implementation of FBA in Julia[42]. The latter is tailored to solve multiple flux balance analyses on a subset or all the reactions of large- and huge-scale networks, on any number of threads or nodes. To enumerate elementary modes or minimal cut-sets, we provide an interface to CellNetAnalyzer[55,56] (https://www2.mpi-magdeburg.mpg.de/projects/cna/cna.html), which excels at computationally demanding enumerative, discrete geometry calculations of relevance to biochemical networks. In addition, we included an updated implementation of the genetic minimal cut-sets approach[57], which extends the concept of minimal cut-sets to gene knockout interventions.

In summary, the COBRA Toolbox v.3.0 provides an unparalleled depth of COBRA methods, has a highly active and supportive open-source development community, is accompanied by extensive documentation and narrative tutorials, leverages the most comprehensive library for numerical computing, and is distributed with extensive interoperability with a range of complementary programming languages that exploit their particular strengths to realize specialized constraint-based modeling methods. A list of the main COBRA methods now available in the COBRA Toolbox is given in Table 1. Moreover, all of this functionality is provided within one accessible software environment.

Experimental design

The COBRA Toolbox v.3.0 is designed for flexible adaptation into customized pipelines for COBRA in a wide range of biological, biochemical, or biotechnological scenarios, from single organisms to communities of organisms. To become proficient in adapting the COBRA Toolbox to generate a protocol specific to one's situation, it is wise to first familiarize oneself with the principles of constraint-based modeling. This can best be achieved by studying the educational material already available. The textbook *Systems Biology: Constraint-based Reconstruction and Analysis*[1] is an ideal place to start. It is accompanied by a set of lecture videos that accompany the various chapters (http://systemsbiology.ucsd.edu/Publications/Books/SB1-2LectureSlides). The textbook *Optimization Methods in Metabolic Networks*[58] provides the fundamentals of mathematical optimization and its application in the context of metabolic network analysis. A study of this educational material will accelerate one's ability to utilize any software application dedicated to COBRA.

Once one is cognizant of the conceptual basis of COBRA, one can then proceed with this protocol, which summarizes a subset of the key methods that are available within the COBRA Toolbox. To adapt this protocol to one's situation, users can combine the COBRA methods implemented within the COBRA Toolbox in numerous ways. The adaption of this protocol to one's situation may require the development of new customized MATLAB scripts that combine existing methods in a new way. Owing to the aforementioned benefits of narratives, the first choice should be to implement these customized scripts in the form of MATLAB live scripts. To get started, the existing tutorial narratives, described in Table 1, can be repurposed as templates for new analysis pipelines. Narrative figures and tables can then be generated from raw data and used within the main text of scientific articles and converted into supplementary material to enable full reproducibility of computational results. The narratives specific to individual scientific articles can be shared with peers within a dedicated repository (https://github.com/opencobra/COBRA.papers).

New tutorials can be shared with the COBRA community (https://github.com/opencobra/COBRA.tutorials). Depending on one's level of experience, or the novelty of an analysis, the adaptation of this protocol to a particular situation may require the adaption of existing COBRA methods, development of new COBRA methods, or both.

Software architecture of the COBRA Toolbox

The source code of the COBRA Toolbox (https://github.com/opencobra/cobratoolbox/tree/master/src) is divided into several top-level folders, which either mimic the main classes of COBRA methods (*reconstruction*, *dataIntegration*, *analysis*, *visualization*, *design*) or contain the basic functions (*base*) available for use within many COBRA methods. For example, the input or output of reconstructions and models in various formats, as well as all the interfaces to optimization solvers, is contained within the *base* folder. The *reconstruction* folder contains all of the methods associated with the reconstruction and refinement of a biochemical network to match experimental data, as well as the conversion of a reconstruction into various forms of constraint-based models (see Table 3 for a description of the main fields of a COBRA model). The *dataIntegration* folder contains the methods for integration of metabolomic, transcriptomic, proteomic, and thermodynamic data with a reconstruction or model. The *analysis* folder contains all of the methods for interrogation of the properties of a reconstruction or model, and combinations thereof, as well as the prediction of biochemical network states using constraint-based models. The *visualization* folder contains all of the methods for the visualization of predictions within a biochemical network context, using various biochemical cartography tools that interoperate with the COBRA Toolbox. The *design* folder contains new strain design methods and a new modeling language interface to GAMS (general algebraic modeling system), a high-level modeling system for mathematical optimization[59].

Open-source software development with the COBRA Toolbox

Understanding how the COBRA Toolbox is developed is most important for developers, so beginners may skip this section at first. With an increasing number of contributions from developers around the world, the code base is evolving at a fast pace. The COBRA Toolbox has evolved from monolingual MATLAB software to a multilingual software suite via integration with C, FORTRAN, Julia, Perl, and Python code, as well as pre-compiled binaries, for specific purposes. For example, the integration with quadruple precision numerical optimization solvers, implemented in FORTRAN, for robust and efficient modeling of multi-scale biochemical networks, such as those obtained with integration[60] of metabolic[61] and macromolecular synthesis[62] reconstructions, represents a new peak in terms of

Table 3 | A description of the main fields of a standard model structure

Field name	Size	Data type	Field description
.b	$m \times 1$	Double	The coefficients of the constraints of the metabolites ($Sv = b$)
.csense	$m \times 1$	Char	The sense of the constraints represented by b; each row is either 'E' (equality), 'L' (less than), or 'G' (greater than)
.metCharges	$m \times 1$	Numeric	The charge of the respective metabolite (NaN if unknown)
.metFormulas	$m \times 1$	Cell of char	Elemental formula for each metabolite
.metInChIString	$m \times 1$	Cell of char	Formula for each metabolite in the InCHI strings format
.metNames	$m \times 1$	Cell of char	Full name of each corresponding metabolite
.mets	$m \times 1$	Cell of char	Identifiers of the metabolites
.metSmiles	$m \times 1$	Cell of char	Formula for each metabolite in SMILES format
.c	$n \times 1$	Double	The objective coefficient of the reactions
.grRules	$n \times 1$	Cell of char	A string representation of the gene–protein–reaction rules defined in a readable format
.lb	$n \times 1$	Double	Lower bounds for fluxes through the reactions
.rxnConfidenceScores	$n \times 1$	Numeric	Confidence scores for reaction presence (0–5, with 5 being the highest confidence)
.rxnECNumbers	$n \times 1$	Cell of char	Enzyme Commission (EC) number for each reaction
.rxnNames	$n \times 1$	Cell of char	Full name of each corresponding reaction
.rxnNotes	$n \times 1$	Cell of char	Description of each corresponding reaction
.rxnReferences	$n \times 1$	Cell of char	Description of references for each corresponding reaction
.rxns	$n \times 1$	Cell	Identifiers of the reactions
.subSystems	$n \times 1$	Cell of cell of char	Subsystem assignments for each reaction (one reaction may correspond to multiple subsystems)
.ub	$n \times 1$	Double	Upper bounds for fluxes through the reactions
.S	$m \times n$	Numeric	The stoichiometric matrix containing the model structure (for large models, a sparse format is suggested)
.geneNames	$g \times 1$	Cell of char	Full name of each corresponding gene
.genes	$g \times 1$	Cell of char	Identifiers of the genes in the model
.proteinNames	$g \times 1$	Cell of char	Full name for each protein
.proteins	$g \times 1$	Cell of char	Proteins associated with each gene (one protein per gene)
.rxnGeneMat	$n \times g$	Numeric or logical	Matrix with rows corresponding to reactions and columns corresponding to genes
.compNames	$c \times 1$	Cell of char	Descriptions of the compartments (compNames(m) is associated with comps(m))
.comps	$c \times 1$	Cell of char	Symbols for compartments
.osenseStr	1×3	Char	The objective sense: either 'max' (maximization) or 'min' (minimization)

Cell, a data type with indexed data containers called cells, where each cell can contain any type of data; Cell of cell of char, a data type where each indexed cell corresponds to another indexed series of cells, each of which contains a character array; Cell of char, a series of character arrays; Char, a character array; Double, a double-precision array; logical, an array of logical values, 1 (true) or 0 (false); Numeric, numeric variable.

biochemical comprehensiveness and predictive capacity[63]. These developments warranted an industrial approach to software development of the COBRA Toolbox. Therefore, we implemented a continuous-integration approach with the aim of guaranteeing a consistent, stable, and high-quality software solution for a broad user community.

The COBRA Toolbox is version-controlled using Git (https://git-scm.com), a free and open-source distributed, version-control system that tracks changes in computer files and is used for coordinating work on those files by multiple people. The continuous-integration environment facilitates contributions from the fork of COBRA developers to a development branch while ensuring that robust, high-quality, well-tested code is released to end users on the *master* branch. To lower the technological barrier to the use of the aforementioned software development tools, we have developed MATLAB.devTools (https://github.com/opencobra/MATLAB.devTools), a new user-friendly software extension that enables submission of new COBRA software and tutorials. A server-side, semi-automated continuous-integration environment ensures that the code in each new submission is first verified automatically, via a comprehensive test suite that detects bugs and integration errors, and,

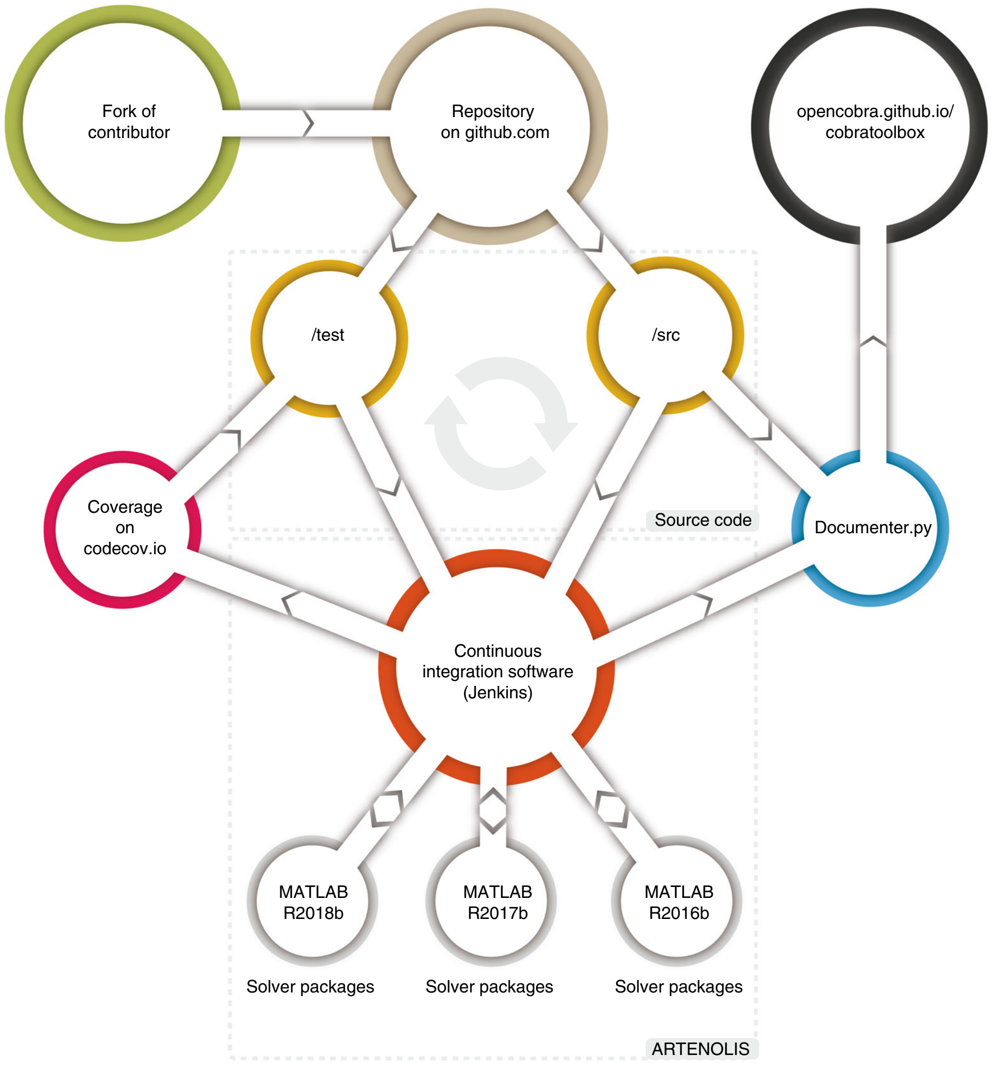


Fig. 2 | Continuous integration of newly developed code is performed on a dedicated server running Jenkins. The main code is located in the *src* folder and test functions are located in the *test* folder. A test not only runs a function (first-degree testing), but also tests the output of that function (second-degree testing). The continuous-integration setup relies on end-of-year releases of MATLAB only. Soon after the latest stable version of MATLAB is released, full support will be provided for the COBRA Toolbox. After a successful run of tests on the three latest end-of-year releases of MATLAB using various solver packages, the documentation based on the headers of the functions (docstrings) is extracted, generated, and automatically deployed. Immediate feedback through code coverage reports (https://codecov.io/gh/opencobra/cobratoolbox) and build statuses is reported on GitHub. With this setup, the impact of local changes in the code base is promptly revealed. This newly developed software architecture is termed ARTENOLIS (Automated Reproducibility and Testing Environment for Licensed Software).

second, reviewed manually by at least one domain expert, before integration with the development branch. Third, each new contribution to the *development* branch is evaluated in practice by active COBRA researchers before it becomes part of the *master* branch.

Until recently, the code-quality checks of the COBRA Toolbox were primarily static: the code was reviewed by experienced users and developers, and occasional code inspections led to discoveries of bugs. The continuous-integration setup defined in Fig. 2 aims at dynamic testing with automated builds, code evaluation, and documentation deployment. Often a function runs properly independently and yields the desired output(s), but when called within a different part of the code, logical errors are thrown. The unique advantage of continuous integration is that logical errors are mostly avoided.

In addition to automatic testing, manual usability testing is performed regularly by users and is key to providing a tested and usable code base to the end user. These users provide feedback on the usability of the code base, as well as the documentation, and report eventual issues online (https://github.com/opencobra/cobratoolbox/issues). The documentation is automatically deployed to the COBRA Toolbox website (https://opencobra.github.io/cobratoolbox) based on function headers. Moreover, each of the narrative tutorials is presented in a format suitable for web browsers in a dedicated part of the COBRA Toolbox website (https://opencobra.github.io/cobratoolbox/stable/tutorials).

Controls

COBRA is part of an iterative systems biology cycle[1]. As such, it can be used as a framework for integrative analysis of experimental data in the context of prior information on the biochemical network underlying one or many complementary experimental datasets. Moreover, it can be used to predict the outcome of new experiments, or it can be used in both of these scenarios at once. Assuming all of the computational steps are errorless, the appropriate control for any prediction derived from a computational model is the comparison with independent experimental data, that is, experimental data that were not used for the model-generated predictions. It is also important to introduce quality controls to check that the computational steps are free from certain errors that may arise during adaptation of existing COBRA protocols or development of new ones.

There are various strategies for the implementation of computational quality controls. Within the COBRA Toolbox v.3.0, substantial effort has been devoted to automatically testing the functionality of existing COBRA methods. We have also embedded a large number of sanity checks, which evaluate whether the input data could possibly be appropriate for use with a function. These sanity checks have been accumulated over more than a decade of continuous development of the COBRA Toolbox. Their objective is to rule out certain known classes of obviously false predictions that might result from an inappropriate use of a COBRA method, but they do not (and are not intended) to catch every such error, as it is impossible to imagine all of the eventual erroneous inputs that may be presented to a COBRA Toolbox function. It is advisable for users to create their own narratives with additional sanity checks, which will depend heavily on the modeling scenario. Examples of such narratives can be found within the COBRA Toolbox website (https://opencobra.github.io/cobratoolbox/stable/tutorials).

Required expertise

Most of this protocol can be implemented by anyone with a basic familiarity with the principles of constraint-based modeling. Some methods are only for advanced users. If one is a beginner with respect to MATLAB, Supplementary Manual 1 provides pointers to get started. MATLAB is a relatively simple programming language to learn, but it is also a powerful language for an expert because of the large number of software libraries for numerical and symbolic computing that it provides access to. Certain specialized methods within this protocol, such as that for thermodynamically constraining reaction directionality, depend on the installation of other programming languages and software, which may be too challenging for a beginner with a non-standard operating system.

If the documentation and tutorials provided within the COBRA Toolbox are not sufficient, then Step 103 guides the user toward sources of COBRA community support. The computational demands associated with the implementation of this protocol for one's reconstruction or model of choice are dependent on the size of the network concerned. For a genome-scale model of metabolism, a desktop computer is usually sufficient. However, for certain models, such as a microbial community of genome-scale metabolic networks, a multi-scale model of metabolism and macromolecular synthesis, or a multi-tissue model, more powerful processors and extensive memory capacity are required, ranging from a workstation to a dedicated computational cluster. Embarrassingly parallel, high-performance computing is feasible for most model analysis methods implemented in the COBRA Toolbox, which will run in isolation with invocation from a distributed computing engine. It is currently an ongoing topic of research, beyond the scope of this protocol, to fully exploit high-performance computing environments with software developed within the wider open COBRA environment, although some examples[42] are already available for interested researchers to consult.

Limitations

A protocol for the generation of a high-quality, genome-scale reconstruction, using various software applications, including the COBRA Toolbox, has previously been disseminated[6]; therefore, this protocol focuses more on modeling than reconstruction. The COBRA Toolbox is not meant to be a general-purpose computational biology tool, as it is focused on COBRA. For example, although various forms of generic data analysis methods are available within MATLAB, the input data for integration with reconstructions and models within the COBRA Toolbox are envisaged to have already been preprocessed by other tools. Within its scope, the COBRA Toolbox aims for complete coverage of

COBRA methods. The first comprehensive overview of the COBRA methods available for microbial metabolic networks[5] requires an update to encompass many additional methods that have been reported to date, in addition to the COBRA methods targeted toward other biochemical networks. The COBRA Toolbox v.3.0 provides the most extensive coverage of published COBRA methods. However, there are certainly some methods that have yet to be incorporated directly as MATLAB implementations, or indirectly via a MATLAB interface to a software dependency. Although, in principle, any COBRA method could be implemented entirely within MATLAB, it may be more efficient to leverage the core strength of another programming language that could provide intermediate results that can be incorporated into the COBRA Toolbox via various forms of MATLAB interfaces. Such a setup would enable one to overcome any current limitation in coverage of existing methods.

Materials

Equipment

Input data

The COBRA Toolbox offers support for several commonly used data formats for describing models, including models in Systems Biology Markup Language (SBML) and Excel sheets (.xls). The COBRA Toolbox fully supports the standard format documented in the SBML Level 3 v.1 with the Flux Balance Constraints (fbc) package v.2 specifications (http://www.sbml.org/specifications/sbml-level-3/version-1/fbc/sbml-fbc-version-2-release-1.pdf).

Required hardware

- A computer with any 64-bit Intel or AMD processor and at least 8 GB of RAM ▲**CRITICAL** Depending on the size of the reconstruction or model, more processing power and more memory may be needed, especially if it is also desired to store the results of analysis procedures within the MATLAB workspace.
- A hard drive with free storage of at least 10 GB
- A working and stable Internet connection is required during installation and while contributing to the COBRA Toolbox.

Required software

- A Linux, macOS, or Windows operating system that is MATLAB qualified (https://mathworks.com/support/sysreq.html) ▲**CRITICAL** Make sure that the operating system is compatible with the MATLAB version by checking the requirements at https://mathworks.com/support/sysreq/previous_releases.html. Follow the upgrade and installation procedures on the supplier's website or ask your system administrator for help if required.
- MATLAB (MathWorks, https://mathworks.com/products/matlab.html), v.R2014b or above is required. Version R2016a or above is required for running MATLAB live scripts (tutorials .mlx files). Note that the tutorials can be run on R2014b using the provided .m files. Install MATLAB and its license by following the official installation instructions (https://mathworks.com/help/install/ug/install-mathworks-software.html) or ask your system administrator. ▲**CRITICAL** No support is provided for versions older than R2014b. MATLAB is released on a twice-yearly schedule. After the latest release (version b), it may be a couple of months before certain methods with dependencies on other software become compatible. For example, the latest releases of MATLAB may not be compatible with the existing solver interfaces, necessitating an update of the MATLAB interface provided by the solver developers, an update of the COBRA Toolbox, or both.
- The COBRA Toolbox (https://github.com/opencobra/cobratoolbox), v.3.0 or above. Install the COBRA Toolbox by following the procedures given at https://opencobra.github.io/cobratoolbox/stable/installation.html. ▲**CRITICAL** Make sure that all system requirements outlined at https://opencobra.github.io/cobratoolbox/docs/requirements.html are met. If an installation of the COBRA Toolbox is already present, there is no need to re-clone the full repository. Instead, you can update the repository from MATLAB or from the terminal (Equipment setup).
- A working *bash* terminal (or shell) with UNIX tools. *curl* v.7.0 or above must be installed to ensure connectivity between the COBRA Toolbox and the remote GitHub server. The version control software *git* v.1.8 or above is required to be installed and accessible through system commands. On Linux and macOS, a *bash* terminal with *git* and *curl* is readily available. Supplementary Manual 2 provides a brief guide to the basics of using a terminal. ▲**CRITICAL** On Windows, the shell integration included with *git Bash* (https://git-for-windows.github.io) utilities must be installed. The command-line tools such as *git* or

curl will be installed together with *git Bash*. Make sure that you select <*Use git Bash and optional Unix tools from the Windows Command prompt during the installation process*> of git Bash. After installing *git Bash*, restart MATLAB. On macOS, a working installation of Xcode (https://developer.apple.com/xcode) v.8.0 or above and command-line tools is mandatory. The *Xcode* command-line tools can be installed by following the instructions at https://railsapps.github.io/xcode-command-line-tools.html.

Optional software

- The ability to read and write models in SBML format requires the MATLAB interface from the libSBML application programming interface, v.5.15.0 or above. The COBRA Toolbox v.3.0 supports the latest SBML Level 3 Flux Balance Constraints version 2 package (http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/fbc). The libSBML package, v.5.15.0 or above, is already packaged with the COBRA Toolbox via the COBRA.binary submodule for all common operating systems. Alternatively, binaries can be downloaded separately and installed by following the procedure at http://sbml.org/Software/libSBML. The COBRA Toolbox developers work closely with the SBML team to ensure that the COBRA Toolbox supports the latest standards, and moreover that standard development is also focused on meeting the evolving requirements of the constraint-based modeling community. After the latest release of MATLAB, there may be a short time lag before input and output become fully compatible. For example, the input and output of .xml files in the SBML standard formats rely on platform-dependent binaries that we maintain (https://github.com/opencobra/COBRA.binary) for each major platform, but the responsibility for maintenance of the source code[64] lies with the SBML team (http://sbml.org), which has a specific forum for raising interoperability issues (https://groups.google.com/forum/#!forum/sbml-interoperability).
- The MATLAB Image Processing Toolbox, the Parallel Computing Toolbox, the Statistics and Machine Learning Toolbox, and the Optimization Toolbox and Bioinformatics Toolbox (https://mathworks.com/products) must be licensed and installed to ensure certain model analysis functionality, such as topology-based algorithms, flux variability analysis, or sampling algorithms. The individual MATLAB toolboxes can be installed during the MATLAB installation process. If MATLAB is already installed, the toolboxes can be managed using the built-in MATLAB add-on manager as described at https://mathworks.com/help/matlab/matlab_env/manage-your-add-ons.html.
- The Chemaxon Calculator Plugins (ChemAxon, https://chemaxon.com/products/calculator-plugins), v.16.9.5.0 or above, constitute a suite offering a range of cheminformatics tools. Standardizer is ChemAxon's solution for transforming chemical structures into customized, canonical representations to achieve best reliability with chemical databases. The Chemaxon Calculator Plugins, v.16.9.5.0 or above, can be installed by following the installation procedures outlined in the user guide at https://chemaxon.com/products/calculator-plugins. A license is freely available for academics.
- Java (https://java.com/en/download/help/download_options.xml), v.8 or above, is a programming language that enables platform-independent applications. Java, v.8 or above, can be installed by following the procedures given at https://java.com/en/download/help/index_installing.xml.
- Python (https://python.org/downloads), v.2.7, is a high-level programming language for general-purpose programming and is required to run NumPy or to generate the documentation locally (relevant when contributing). Python v.2.7 is already installed on Linux and macOS. On Windows, the instructions at https://wiki.python.org/moin/BeginnersGuide/Download will guide you to install Python.
- NumPy (http://numpy.org), version 1.11.1 or above, is a fundamental package for scientific computing with Python. NumPy can be installed by following the procedures accessible via https://scipy.org.
- OpenBabel (https://openbabel.org), v.2.3 or above, is a chemical toolbox designed to speak the many languages of chemical data. OpenBabel can be installed by following the installation instructions at http://openbabel.org/wiki/Category:Installation.
- Reaction Decoder Tool (RDT; https://github.com/asad/ReactionDecoder/releases), v.1.5.0 or above, is a Java-based, open-source atom-mapping software tool. The latest version of the RDT can be installed by following the procedures at https://github.com/asad/ReactionDecoder#installation.

Solvers

- Table 4 provides an overview of supported optimization solvers. At least one linear programming (LP) solver is required for basic constraint-based modeling methods. Therefore, by default, the COBRA Toolbox installs certain open-source solvers, including the LP and mixed-integer LP (MILP) solver GLPK (https://gnu.org/software/glpk). However, for more efficient and robust linear optimization, we recommend that an industrial numerical optimization solver be installed. On Windows, the OPTI solver suite (https://inverseproblem.co.nz/OPTI) must be installed separately in order to use the OPTI

Table 4 | An overview of the types of optimization problems solved by each optimization solver

	Name	Version	Interface	LP	MILP	QP	MIQP	NLP
Active support	DQQ	—	dqqMinos	*				
	GLPK	2.7+	glpk	*	*			
	GUROBI	7.0+	gurobi	*	*	*	*	
	ILOG CPLEX	12.7.1+	ibm_cplex	*	*	*		
	MATLAB	R2014b+	matlab	*				*
	MINOS	—	quadMinos	*				*
	MOSEK	8.0+	mosek	*	*	*		
	PDCO	—	pdco	*		*		*
	Tomlab CPLEX	8.0+	cplex_direct	*	*	*	*	
			tomlab_cplex	*	*	*	*	
Passive	OPTI	2.27+	opti	*	*	*	*	*
	QPNG	—	qpng			*		
	Tomlab SNOPT	8.0+	tomlab_snopt					*
Legacy	GUROBI	7.0+	gurobi_mex	*	*	*	*	
	LINDO	2.0+	lindo_old	*				
			lindo_legacy	*				
	MATLAB	R2014b+	lp_solve	*				

Asterisks (*) indicate that the solver can solve the respective types of problems. The interface to certain standard optimization solvers is actively supported, whereas the interface to other non-standard solvers requires testing by the end user to ensure compatibility. A legacy solver interface might require refinement before it will be compatible with newer solver or MATLAB releases.

interface. ▲**CRITICAL** Depending on the type of optimization problem underlying a COBRA method, an additional numerical optimization solver may be required.

- Most steps of the solver installation require superuser or administrator rights (`sudo`) and eventually the setting of environment variables. Detailed instructions and links to the official installation guidelines for installing Gurobi, Mosek, Tomlab, and IBM Cplex can be found at https://opencobra.github.io/cobratoolbox/docs/solvers.html. ▲**CRITICAL** Make sure that environment variables are properly set in order for the solvers to be correctly recognized by the COBRA Toolbox.

Application-specific software

▲**CRITICAL** Certain solvers have additional software requirements, and some binaries provided in the COBRA.binary (https://github.com/opencobra/COBRA.binary) repository might not be compatible with your system.

- The *dqqMinos* and *Minos* solvers. These can be used only on Unix. The C-shell *csh* (http://bxr.su/NetBSD/bin/csh) is required. On Linux or macOS, the C-shell *csh* can be installed by following the instructions at https://en.wikibooks.org/wiki/C_Shell_Scripting/Setup.
- The GNU C compiler *gcc* v.7.0 or above (https://gcc.gnu.org). The library of the gcc compiler is required for generating new binaries of *fastFVA* with a different version of the CPLEX solver from that officially supplied. The *gcc* compiler can be installed by following the link given at https://opencobra.github.io/cobratoolbox/docs/compilers.html.
- The GNU Fortran compiler *gfortran* v.4.1 or above (https://gcc.gnu.org/fortran). The library of the *gfortran* compiler is required for running *dqqMinos*. Most Linux distributions come with this compiler preinstalled. Alternatively, the *gfortran* compiler can be installed by following the link given at https://opencobra.github.io/cobratoolbox/docs/compilers.html.

Contributing software

- MATLAB.devTools (https://github.com/opencobra/MATLAB.devTools) is highly recommended for contributing code to the COBRA Toolbox in a user-friendly and convenient way, even for those without basic knowledge of *git*. The MATLAB.devTools can be installed by following the instructions given at https://github.com/opencobra/MATLAB.devTools#installation. Alternatively, if the COBRA Toolbox is already installed, then the MATLAB.devTools can be installed directly from within MATLAB by typing:

```
>> installDevTools()
```

Equipment setup

COBRA Toolbox

Update the COBRA Toolbox from within MATLAB by running the following command:

```
>> updateCobraToolbox
```

Update from the terminal (or shell) by running the following from within the *cobratoolbox* directory.

```
$ cd cobratoolbox # change to the cobratoolbox directory
$ git checkout master # switch to the master branch
$ git pull origin master # retrieve changes
```

In the case that the update of the COBRA Toolbox fails or cannot be completed, clone the repository again. ▲**CRITICAL** The COBRA Toolbox can be updated as described above only if no changes have been made to the code in one's local clone of the official online repository. If one intends to edit the code in one's local clone of the COBRA Toolbox, then the official repository should be cloned as explained in Steps 97–102. These steps also explain how to contribute any local edits to the COBRA Toolbox code into the official online repository.

Procedure

Initialization of the COBRA Toolbox ● Timing 5–30 s

1 At the start of each MATLAB session, the COBRA Toolbox must be initialized. The initialization can be done either automatically (option A) or manually (option B). For a regular user who primarily uses the official openCOBRA repository, automatic initialization of the COBRA Toolbox is recommended. It is highly recommended to manually initialize when contributing (Steps 97–102), especially when the official version and a clone of the fork are present locally.

(A) **Automatically initializing the COBRA Toolbox**

(i) Edit the MATLAB *startup.m* file and add a line with `initCobraToolbox` so that the COBRA Toolbox is initialized each time that MATLAB is started.

```
>> edit startup.m
```

▲**CRITICAL STEP** During initialization, a check for software dependencies is performed and reported to the command window. It is not necessary that all possible dependencies be satisfied before beginning to use the toolbox; e.g., satisfaction of a dependency on a multi-scale linear optimization solver is not necessary for modeling with a mono-scale metabolic model. However, it is essential to satisfy other software dependencies; e.g., dependency on a linear optimization solver must be satisfied for any method that uses FBA.

? TROUBLESHOOTING

(B) **Manually initializing the COBRA Toolbox**

(i) Navigate to the directory where you installed the COBRA Toolbox and initialize by running:

```
>> initCobraToolbox;
```

▲**CRITICAL STEP** During initialization, a check for software dependencies is performed and reported to the command window. It is not necessary that all possible dependencies be satisfied before beginning to use the toolbox; e.g., satisfaction of a dependency on a multi-scale linear optimization solver is not necessary for modeling with a mono-scale metabolic model. However, it is essential to satisfy other software dependencies; e.g., dependency on a linear optimization solver must be satisfied for any method that uses FBA.

? TROUBLESHOOTING

2 At initialization, one from a set of available optimization solvers will be selected as the default solver. If Gurobi is installed, it is used as the default solver for LP problems, quadratic problems (QPs), and MILP problems. Otherwise, the GLPK solver is selected for LPs and MILP problems. It is important to check whether the solvers installed are satisfactory. A table stating the solver

compatibility and availability is printed for the user during initialization. Check the currently selected solvers with the following command:

```
>> changeCobraSolver;
```

▲**CRITICAL STEP** A dependency on at least one linear optimization solver must be satisfied for FBA.

Verification and testing of the COBRA Toolbox ● Timing ~17 min

3 (Optional) Test the functionality of the COBRA Toolbox locally. This is recommended if one encounters an error running a function. The test suite runs tailored tests that verify the output and proper execution of core functions on the locally configured system. The full test suite can be invoked by typing the following command:

```
>> testAll
```

? TROUBLESHOOTING

Importation of a reconstruction or a model ● Timing 10 s–2 min

4 The COBRA Toolbox offers support for several commonly used data formats for describing models, including models in SBML, Excel sheets (.xls) and different SimPheny formats. The COBRA Toolbox fully supports the standard format documented in the SBML level 3 v.1 with the Flux Balance Constraints (fbc) package v.2 specifications (http://www.sbml.org/specifications/sbml-level-3/version-1/fbc/sbml-fbc-version-2-release-1.pdf). In order to load a model with `fileName` into the MATLAB workspace as a COBRAv3 model structure, run the following command:

```
>> model = readCbModel(fileName);
```

When `filename` is left blank, a file-selection dialogue window is opened. If no file extension is provided, the code will automatically determine the appropriate format from the given filename. The `readCbModel` function also supports reading of normal MATLAB files for convenience, and checks whether those files contain valid COBRA models. Legacy model structures saved in a .mat file are loaded and converted. The fields are also checked for consistency with the current definitions.

▲**CRITICAL STEP** We advise that `readCbModel()` be used to load new models. This is also valid for models provided in .mat files, as `readCbModel` checks the model for consistency with the COBRA Toolbox v.3.0 field definitions and automatically performs necessary conversions for models with legacy field definitions or field names. To develop future-proof code, it is good practice to use `readCbModel()` instead of the built-in function `load`.

? TROUBLESHOOTING

Exportation of a reconstruction or a model ● Timing 10 s–2 min

5 The COBRA Toolbox offers a set of different output methods. The most commonly used formats are SBML .xml and MATLAB .mat files. SBML is the preferred output format, as it can be read by most applications in the field of computational systems biology. However, some information cannot be encoded in standard SBML, so a .mat file might contain information not present in the corresponding SBML output.

To output a COBRA model structure in either format, use the following command:

```
>> writeCbModel(model, fileName);
```

The extension of the `fileName` provided is used to identify the type of output requested. The model will consequently be converted and saved in the respective format. When exporting a reconstruction or model, it is necessary that the model adhere to the `model` structure in Table 3, and that fields contain valid data. For example, all cells of the `rxnNames` field should contain only data of type `char` and not data of type `double`.

? TROUBLESHOOTING

Use of *rBioNet* to add reactions to a reconstruction ● Timing 1 s–17 min

▲CRITICAL We highly recommend using *rBioNet*[48] (a graphical user interface–based reconstruction tool) for the addition or removal of reactions and of gene–reaction associations.

▲CRITICAL A stoichiometric representation of a reconstructed biochemical network is contained within the `model.S` matrix. This is a stoichiometric matrix with `m` rows and `n` columns. The entry `model.S(i,j)` corresponds to the stoichiometric coefficient of the i^{th} molecular species in the j^{th} reaction. The coefficient is negative when the molecular species is consumed in the reaction and positive if it is produced in the reaction. If `model.S(i,j)== 0`, then the molecular species does not participate in the reaction. To manipulate an existing reconstruction in the COBRA Toolbox, one can use *rBioNet*, use a spreadsheet, or generate scripts with reconstruction functions. Each approach has its advantages and disadvantages. When adding a new reaction or gene–protein–reaction association, *rBioNet* ensures that reconstruction standards are satisfied, but it may make the changes less tractable when many reactions are added. A spreadsheet-based approach is tractable, but allows only for the addition, and not the removal, of reactions. By contrast, using reconstruction functions provides an exact specification for all the refinements made to a reconstruction. One can also combine these approaches by first formulating the reactions and gene–protein–reaction associations with *rBioNet* and then adding sets of reactions using reconstruction functions.

▲CRITICAL If you do not have existing *rBioNet* metabolite, reaction, and compartment databases, the first step is to create these files. Refer to the *rBioNet* tutorial provided in the COBRA Toolbox for instructions on how to add new metabolites and reactions to an *rBioNet* database. Make sure that all the relevant metabolites and reactions that you wish to add to your reconstruction are present in your *rBioNet* databases.

6 There are two options for using *rBioNet* functionality to add reactions to a reconstruction: using the *rBioNet* graphical interface (option A) or not using the interface (option B). If you wish to add the reactions only to the *rBioNet* database, hence benefiting from the included quality control and assurance measures, but then afterward decide to use the COBRA Toolbox commands to add reactions to the reconstruction, use option B.

(A) **Adding reactions from an *rBioNet* database to a reconstruction using the *rBioNet* graphical user interface**

(i) Verify your *rBioNet* settings. First, make sure the paths to your *rBioNet* reaction, metabolite, and compartment databases are set correctly.

```
>> rBioNetSettings;
```

(ii) Load the .mat files that hold your reaction, metabolite, and compartment databases.

(iii) To add reactions from an *rBioNet* database to a reconstruction, invoke the *rBioNet* graphical user interface with the following command:

```
>> ReconstructionTool;
```

Select *File* > *Open Model Creator*.

(iv) Load your reconstruction by selecting *File* > *Open Model* > *Complete Reconstruction*.

(v) Add reactions from the *rBioNet* database by selecting *Add Reaction* and selecting a reaction. Repeat for all reactions that should be added to the reconstruction.

(vi) Save your updated reconstruction by selecting *File* > *Save* > *As Reconstruction Model*. As *rBioNet* was created using the old COBRA model structure, use the following command to convert your model to the new model structure:

```
>> model = convertOldStyleModel(model);
```

(B) **Adding reactions from the *rBioNet* database without using the *rBioNet* interface**

(i) Load (or create) a list of reaction abbreviations `ReactionList` to be added from the *rBioNet* reaction database:

```
>> load('Reactions.mat');
```

(ii) Load the *rBioNet* reaction database `'rxnDB'`:

```
>> load('rxnDB.mat');
```

(iii) Then, add new reactions:

```
>> for i = 1:length(ReactionList)
model = addReaction(model, ReactionList{i}, 'reactionFormula',
... rxnDB(find(ismember(rxn(:, 1), ReactionList{i})), 3));
 end
```

Use of a spreadsheet to add reactions to a reconstruction ● Timing 1 s–17 min

7 Load reactions from a spreadsheet with a pre-specified format[17] into a new model structure `modelNewR`:

```
>> modelNewR = xls2model('NewReactions.xlsx');
```

8 Merge the existing reconstruction `model` with the new model structure `modelNewR` to obtain a reconstruction with expanded content, `modelNew`:

```
>> modelNew = mergeTwoModels(model, modelNewR, 1);
```

Use of scripts with reconstruction functions ● Timing 1 s–2 min

9 To ensure traceability of all manipulations to a reconstruction, generate, execute, and save a script that calls reconstruction functions rather than using the command line. Use the function addReaction to add a reaction to a reconstruction:

```
>> model = addReaction(model, 'GAPDH', 'metaboliteList', {'g3p[c]',
'nad[c]', 'pi[c]',...
'13bpg[c]', 'nadh[c]', 'h[c]'}, 'stoichCoeffList', [-1; -1; -2; 1; 1; 1]);
```

The use of `metaboliteList` provides a cell array of compartment-specific molecular species abbreviations, whereas `stoichCoeffList` is used to provide a numeric array of stoichiometric coefficients. If particular metabolites do not exist in `model.mets`, then this function will add them to the list of metabolites. In the function `addReaction()`, duplicate reactions are recognized, even when the order of metabolites or the abbreviations of the reaction are different. Certain types of reactions, such as exchange, sink, and demand reactions[65], can also be added using the functions `addExchangeRxn`, `addSinkReactions`, or `addDemandReaction`, respectively. After adding one or multiple reactions to a reconstruction, it is important to verify that these reactions can carry flux, that is, that they are functionally connected to the remainder of the network.

10 Check whether the added reaction(s) have a nonzero flux value (in other words, can carry flux). To do this for each newly added reaction `NewRxn`, change it to be the objective function using:

```
>> model = changeObjective(model, 'NewRxn');
```

then maximize (`'max'`) and minimize (`'min'`) the flux through this reaction.

```
>> FBA = optimizeCbModel(model, 'max');
>> FBA = optimizeCbModel(model, 'min');
```

If the reaction should have a negative flux value (e.g., a reversible metabolic reaction or an uptake exchange reaction), then the minimization should result in a negative objective value `FBA.f < 0`. If both maximization and minimization return an optimal flux value of zero (i.e., `FBA.f == 0`), then this newly added reaction cannot carry a non-zero flux value under the given simulation condition and the cause for this must be identified. If the reaction(s) can carry non-zero fluxes, make sure to carry out Steps 17 and 47 to ensure stoichiometric consistency as a check for chemical and biochemical fidelity.

11 Remove reactions. To remove reactions from a reconstruction, use:

```
>> modelOut = removeRxns(model, rxnRemoveList);
```

For example, if manual curation of the literature reveals that a reaction in the generic human metabolic reconstruction, Recon3[66], is not active in a specific cell type being modeled, then one should remove the corresponding reaction from the reconstruction.

12 Remove metabolites. To remove metabolites only, run the following command:

```
>> model = removeMetabolites(model, metaboliteList, removeRxnFlag);
```

Note that the removal of one or more metabolites makes sense only if they do not appear in any reactions or if one wishes to remove all reactions associated with one or more metabolites. For example, if a network contains reactions $A + B \leftrightarrow C$ and $A \leftrightarrow C$, removing metabolite C will remove the former reaction also.

13 Remove trivial stoichiometry. If metabolites with zero rows, or reactions with zero columns are present in a stoichiometric matrix, they can be removed with:

```
>> modelOut = removeTrivialStoichiometry(model);
```

After removing one or more reactions (or metabolites) from the reconstruction, please repeat Steps 9–13 in order to check that these modifications did not alter existing metabolic functions of the reconstruction-derived models.

Checking the scaling of a reconstruction ● Timing 1s–2 min

14 Most optimization solvers are designed to work with data (e.g., stoichiometric coefficients, bounds, and objective coefficients in linear optimization problems) that are well scaled. Standard solvers are based on 16-digit double-precision floating-point arithmetic, so the input data should not require a solution with >8 significant digits, in order to ensure that solutions are accurate to the remaining 8 digits of precision. Such a solution approach is sufficient for most metabolic models, except, for instance, if micro and macronutrients are simultaneously being considered. Multi-scale models of metabolism and macromolecular synthesis require higher-precision solvers, but they need only be used when necessary, so it is useful to check the scaling of a new reconstruction or model.

Check the scaling of a stoichiometric matrix with the following command:

```
>> [precisionEstimate, solverRecommendation, scalingProperties] =
checkScaling(model);
```

Selection of a double- or quadruple-precision optimization solver ● Timing 1–5 s

15 The COBRA Toolbox is integrated with a wide variety of different optimization solvers (cf. Table 4). Quad MINOS[54,67] is a quadruple-precision version of the general-purpose, industrial-strength linear and nonlinear optimization solver MINOS. This solver operates with 34 digits of precision, and was developed with multi-scale constraint-based modeling problems in mind. Higher-precision solvers are more precise but less computationally efficient than standard solvers. They must be used when necessary, i.e., with multi-scale reconstructions and models. To solve multi-scale linear optimization problems, the COBRA Toolbox offers a Double-Quad-Quad MINOS method (DQQ) that combines the use of double and quadruple solvers in order to improve efficiency while maintaining high accuracy in the solution. One can set the optimization solver used by the COBRA Toolbox as `solverStatus = changeCobraSolver(solverName, solverType)`, where `solverName` specifies the solver to be used, and `solverType` specifies the type of problems to solve with the solver specified by `solverName` (`'LP'` for linear optimization problem; `'MILP'` for MILPs; `'QP'` for quadratic problems; `'MIQP'` for mixed-integer quadratic problems; `'NLP'` for nonlinear problems; or `'ALL'` to change the solver for all the previously mentioned problem types). Depending on the `precisionEstimate`, there are two options: choose a double-precision solver (option A) or a quadruple-precision solver (option B).

(A) **Double-precision solver**

(i) If the recommendation shows that a double-precision solver is probably sufficient, then, for example, set the Gurobi solver to solve LP problems with the following command:

```
>> solverStatus = changeCobraSolver('gurobi', 'LP');
```

A positive `solverStatus` also indicates that the COBRA Toolbox will use Gurobi as the default linear optimization solver.

▲ **CRITICAL STEP** A dependency on at least one linear optimization solver must be satisfied for FBA. If any numerical issues arise while using a double-precision solver, then a higher-precision solver should be tested. For instance, a double-precision solver may incorrectly report that a poorly scaled optimization problem is infeasible although it actually might be feasible for a higher-precision solver. The `checkScaling` function can be used on all operating systems, but the `dqqMinos` or `quadMinos` interfaces are available only on UNIX operating systems.

? TROUBLESHOOTING

(B) **Quadruple-precision solver**

(i) If the recommendation shows that a higher-precision solver is required, then, for example, select the quadruple-precision optimization solver `dqqMinos` for solving linear optimization problems with the following command:

```
>> solverStatus = changeCobraSolver('dqqMinos', 'LP');
```

▲ **CRITICAL STEP** A dependency on at least one linear optimization solver must be satisfied for FBA. If any numerical issues arise while using a double-precision solver, then a higher-precision solver should be tested. For instance, a double-precision solver may incorrectly report that a poorly scaled optimization problem is infeasible although it actually might be feasible for a higher-precision solver. The `checkScaling` function can be used on all operating systems, but the `dqqMinos` or `quadMinos` interfaces are available only on UNIX operating systems.

? TROUBLESHOOTING

Identification of stoichiometrically consistent and inconsistent reactions ● Timing 1 s–28 h

16 All biochemical reactions conserve mass; therefore, it is essential that each biochemical reaction in a model does actually conserve mass. Reactions that do not conserve mass[68] are, however, often added to a reconstruction in order to represent the flow of mass into and out of a system, e.g., during FBA. Each reaction that does not conserve mass, but is added to a model in order to represent the exchange of mass across the boundary of a biochemical system, is henceforth referred to as an *external reaction*, e.g., $D \rightleftharpoons \emptyset$, where $\varnothing$ represents null. Each reaction that is supposed to conserve mass is referred to as an *internal reaction*. In addition to exchange reactions, a reconstruction may contain mass-imbalanced internal reactions due to incorrect or incompletely specified stoichiometry. This situation results in one or more sets of *stoichiometrically inconsistent* reactions[69]. For instance, the reactions $A + B \rightleftharpoons C$ and $C \rightleftharpoons A$ are stoichiometrically inconsistent because it is impossible to assign a positive molecular mass to all species while ensuring that each reaction conserves mass. By combining flux through both of the former reactions in the forward direction, the net effect is $B \longrightarrow \emptyset$, that is, inadvertent exchange of B across the boundary of the model.

In order to distinguish between the reactions in a model that are stoichiometrically consistent and stoichiometrically inconsistent, there are three options: identifying reactions with only one stoichiometric coefficient based on the stoichiometric matrix (option A), checking the reactions that are elementally imbalanced on the basis of the chemical formulae of molecular species (option B), or identifying the largest set of reactions in a reconstruction that are stoichiometrically consistent (option C).

(A) **Use of stoichiometric matrix or reaction names**

(i) Pinpoint external reactions by identifying reactions with only one stoichiometric coefficient, or reactions with the `model.rxns` abbreviation prefixes `EX_`, `DM_` and `sink_`, for exchange, demand and sink reactions, respectively:

```
>> model = findSExRxnInd(model);
```

In the result, `model.SIntRxnBool` gives a Boolean vector of reactions that are heuristically thought to be internal.

▲ **CRITICAL STEP** Any supposedly internal reaction that is actually stoichiometrically inconsistent with the remainder of a reconstruction should be omitted from a model that is intended to be subjected to FBA; otherwise erroneous predictions may result due to inadvertent violation of the steady-state mass conservation constraint.

? TROUBLESHOOTING

(B) **Checking elementally imbalanced reactions on the basis of chemical formulae**

(i) When `model.metFormulas` is populated with the chemical formulae of molecular species, it is possible to check which reactions are elementally imbalanced with the following command:

```
>> [massImbalance] = checkMassChargeBalance(model);
```

The output `massImbalance` is an $n \times t$ matrix with a non-zero entry for any elemental imbalance in a reaction. The other outputs from this function can also be used to analyze imbalanced reactions to suggest modifications to the stoichiometric specification that can resolve the imbalance. A resolution of mass imbalance should ensure that the reaction stoichiometry is consistent with the known biochemical mechanism of the reaction.

▲CRITICAL STEP Any supposedly internal reaction that is actually stoichiometrically inconsistent with the remainder of a reconstruction should be omitted from a model that is intended to be subjected to FBA; otherwise erroneous predictions may result due to inadvertent violation of the steady-state mass conservation constraint.

? TROUBLESHOOTING

(C) **Identifying the largest set of reactions that are stoichiometrically consistent**

(i) Given stoichiometry alone, a non-convex optimization problem can be used to approximately identify the largest set of reactions in a reconstruction that are stoichiometrically consistent.

```
>>[~, SConsistentRxnBool, ~, SInConsistentRxnBool, ~, unknownS
ConsistencyRxnBool,...  model]  =  findStoichConsistentSubset
(model, massBalanceCheck);
```

When checking for stoichiometric inconsistency, external reactions identified via option B can be used to warm-start the algorithm for option C if `massBalanceCheck == 1`. The non-zero entries of `unknownSConsistencyRxnBool` and `unknownSConsistencyMetBool` denote reactions and uniquely involved molecular species for which consistency could not be established.

▲CRITICAL STEP Any supposedly internal reaction that is actually stoichiometrically inconsistent with the remainder of a reconstruction should be omitted from a model that is intended to be subjected to FBA; otherwise erroneous predictions may result due to inadvertent violation of the steady-state mass conservation constraint.

? TROUBLESHOOTING

Identification of stoichiometrically consistent and inconsistent molecular species ● Timing 1 s–17 min

17 Identify the molecular species that participate only in reactions that are stoichiometrically inconsistent using the following command:

```
>>[SConsistentMetBool, ~, SInConsistentMetBool, ~, unknownSConsistency
MetBool,  ~,  model]  =...  findStoichConsistentSubset(model,  mass
BalanceCheck);
```

Setting of simulation constraints ● Timing 1 s–17 min

18 To set the constraints on a model, type

```
>> model = changeRxnBounds(model, rxnNameList, value, boundType);
```

The list of reactions for which the bounds should be changed is given by `rxnNameList`, whereas the vector `value` contains the new boundary reaction rate values. This type of bound can be set to a lower (`'l'`) or upper bound (`'u'`). Alternatively, both bounds can be changed simultaneously (`'b'`).

▲CRITICAL STEP The more biochemically realistic the applied constraints are with respect to a particular context, the more likely network states that are specific to that context are to be predicted, as opposed to those predicted from a generic model. All else being equal, a model derived from a comprehensive, yet generic, reconstruction will be less constrained than a model derived from a less comprehensive, yet generic, reconstruction. That is, in general, the more comprehensive a reconstruction is, the greater attention must be paid to setting simulation constraints.

Identification of molecular species that leak, or siphon, across the boundary of the model ● Timing 1 s–17 min

19 Identification of internal and external reactions using `findSExRxnInd` in Step 16A is the fastest option, but it may not always be accurate. It is therefore wise to check whether there exist molecular species that can be produced from nothing (leak) or consumed, giving nothing (siphon) in a reconstruction, with all external reactions blocked. If `modelBoundsFlag == 1`, then the leak testing uses the model bounds on internal reactions, and if `modelBoundsFlag == 0`, then all internal reactions are assumed to be reversible. To do this, type the following commands:

```
>> modelBoundsFlag = 1;
>> [leakMetBool, leakRxnBool, siphonMetBool, siphonRxnBool] =...
findMassLeaksAndSiphons(model, model.SIntMetBool, model.SIntRxn-
Bool, modelBoundsFlag);
```

▲ **CRITICAL STEP** Non-zero entries in `leakMetBool` or `siphonMetBool` indicate that the corresponding molecular species can be produced from nothing or consumed, giving nothing, and may invalidate any FBA prediction.

Identification of flux-inconsistent reactions ● Timing 1 s–17 min

20 In FBA, the objective is to predict reaction fluxes subject to a steady-state assumption on internal molecular species and a mass balance assumption for molecular species exchanged across the boundary of the model. It is therefore useful to know, before making any FBA prediction, which reactions do not admit a non-zero steady-state flux, i.e., the reactions that are flux–inconsistent, also known as blocked reactions.

To identify reactions that do not admit a non-zero flux, use the following command:

```
>>[fluxConsistentMetBool, fluxConsistentRxnBool, fluxInConsistentMet-
Bool,... fluxInConsistentRxnBool] = findFluxConsistentSubset(model);
```

Flux balance analysis ● Timing 1 s–2 min

21 In standard notation, an FBA[70] solution is obtained by solving the following linear optimization problem:

$$\begin{aligned}\max_{v\in\mathbb{R}^n} \rho(v) &:= c^T v\\ \text{s.t.} Sv &= 0,\\ l \le v &\le u,\end{aligned} \tag{1}$$

where $c \in \mathbb{R}^n$ is a parameter vector that linearly combines one or more reaction fluxes to form the objective function, denoted as $\rho(v)$. In the COBRA Toolbox, `model.c` contains the objective coefficients. $S \in \mathbb{R}^{m\times n}$ is the stoichiometric matrix stored in `model.S`, and the lower and upper bounds on reaction rates, $l, u \in \mathbb{R}^n$ are stored in `model.lb` and `model.ub`, respectively. The equality constraint represents a steady-state constraint (production = consumption) on internal metabolites and a mass balance constraint on external metabolites (production + input = consumption + output). The solution to optimization problem (1) can be obtained using a variety of LP solvers that have been interfaced with the COBRA Toolbox. Table 4 gives the various options. A typical application of FBA is to predict an optimal steady-state flux vector that optimizes a microbial biomass production rate[71], subject to literature-derived bounds on certain reaction rates. Deciphering the most appropriate objective function for a particular context is an important open research question. The objective function in problem (1) can be modified by changing `model.c` directly or by using the convenient function as follows:

```
>> model = changeObjective(model, rxnNameList, objectiveCoeff);
```

A cell array `rxnNameList` and numeric array `objectiveCoeff` are used to give the reaction abbreviation and corresponding linear objective coefficient for one or more reactions to be optimized. By default, `objectiveCoeff(p) > 0` and `objectiveCoeff(q) < 0` correspond to maximization and minimization of the pth and qth reaction abbreviations in `rxnNameList`.

22 FBA, and many of its variants, can be computed using the versatile function `optimizeCbModel`. That is, the default method implemented by `optimizeCbModel` is FBA, as defined in problem (1), but depending on the optional arguments provided to `optimizeCbModel`, many methods that are variations on FBA are also implemented and accessible with slight changes to the input arguments. Here, there are two options: either compute an FBA solution (option A) or compute a unique flux balance solution (option B).

The solution structure `FBAsolution` from `optimizeCbModel` always has the same form, even if the meanings of the fields change depending on the optional input arguments to the function. The field `.stat` contains a standardized solver status. If `FBAsolution.stat == 1`, then an optimal solution has been found and will be returned. The field `.v` is a flux vector such that the optimal value of the objective function is attained, `.y` yields the vector of dual variables for the equality constraints, and `.w` contains the vector of optimal dual variables for the inequality constraints. The field `.stat` is translated from the solver-specific status `.origStat`. The latter is idiosyncratic to each numerical optimization solver, and this is translated to the standardized solver status in order to enable other functions within the COBRA Toolbox to operate in a manner invariant with respect to the underlying solver, to the maximum extent possible. If `FBAsolution.stat == 2`, then the lower and upper bounds are insufficient to limit the value of the objective function and the problem is unbounded, so no optimal solution is returned. If `FBAsolution.stat == 0`, then the constraints in problem (1) do not admit any feasible steady-state flux vector, and therefore no optimal solution exists. If `FBAsolution.stat == -1`, then no solution is reported, because of a time limit or numerical issues.

(A) **Computing a flux balance analysis solution**

(i) Compute a solution to the flux balance analysis problem (1) using the following command:

```
>> FBAsolution = optimizeCbModel(model);
```

▲**CRITICAL STEP** Assuming the constraints are feasible, the optimal objective value `FBAsolution.f` is unique; however, the optimal flux vector `FBAsolution.v` is most likely not unique. It is unwise to base any biological interpretation on a single optimal flux vector if it is one of many alternative optima, because the optimal vector returned can vary depending on the solver chosen to solve the problem. Therefore, when a flux vector is interpreted, it should be a unique solution to some optimization problem.

? TROUBLESHOOTING

(B) **Computing the unique flux balance analysis solution**

(i) To predict a unique optimal flux vector, it is necessary to regularize the objective by subtracting a strictly concave function from it. That is $\rho(v) = c^T v - \theta(v)$, where $\theta(v)$ is a strictly convex function. To do this, type the following command:

```
>> osenseStr = 'max';
>> minNorm = 1e-6;
>> solution = optimizeCbModel(model, osenseStr, minNorm);
```

Assuming the constraints are feasible, the optimal objective value `solution.f`and the optimal flux vector `solution.v` are unique. Setting `minNorm` to 10^{-6} is equivalent to maximizing the function $p(v) := c^T v - \frac{\sigma}{2} v^T v$ with $\sigma = 10^{-6}$ and $\theta(v) = \frac{\sigma}{2} v^T v$ is a regularization function. With high-dimensional models, it is wise to ensure that the optimal value of the regularization function is smaller than the optimal value of the original linear objective in problem (1), that is $\rho(v^*) \gg \theta(v^*)$. A pragmatic approach is to specify `minNorm = 1e-6;`, and then reduce it if necessary.

? TROUBLESHOOTING

Relaxed flux balance analysis ● Timing 1 s–17 min

23 Each solution to problem (1) must satisfy $Sv = 0$ and $l \le v \le u$, independent of any objective chosen to optimize over the set of constraints. It may occur that these constraints are not all simultaneously feasible; i.e., the system of inequalities is infeasible. This situation might be caused by an incorrectly

specified reaction bound. To resolve the infeasibility, one can use relaxed FBA, which is an optimization problem that minimizes the number of bounds to relax in order to render an FBA equation feasible. The refluxed flux balance analysis problem is

$$\begin{aligned} \min_{v,r,p,q} \quad & \lambda\|r\|_0+\alpha\|p\|_0+\alpha\|q\|_0 \\ \text{s.t.} \quad & Sv+r=b, \\ & l-p\le v\le u+q, \\ & p,q,r\ge 0, \end{aligned} \tag{2}$$

where $S\in\mathbb{R}^{m\times n}$ denotes a stoichiometric matrix, p, $q\in\mathbb{R}^n$ denote the relaxations of the lower and upper bounds (l and u) on reaction rates of the flux vector v, and $r\in\mathbb{R}^m$ denotes a relaxation of the mass balance constraint. The latter is useful when there are non-zero boundary constraints forcing secretion or uptake of biochemical species from the environment; that is, $b\in\mathbb{R}^m\neq 0$. Non-negative parameters $\lambda\in\mathbb{R}^n_{\ge 0}$ and $\alpha\in\mathbb{R}^n_{\ge 0}$ can be used to trade off between relaxation of mass balance or bound constraints, e.g., relaxation of bounds on exchange reactions rather than internal reactions or mass balance constraints. The optimal choice of parameters depends heavily on the biochemical context. A relaxation of the minimum number of constraints is desirable because, ideally, one should be able to justify the relaxation of each bound with reference to the literature. The scale of this task is proportional to the number of bounds proposed to be relaxed, motivating the sparse optimization problem to minimize the number of relaxed bounds. Implement relaxed FBA with the following command:

```
>> solution = relaxFBA(model, relaxOption);
```

The structure `relaxOption` can be used to prioritize the relaxation of one type of bound over another. For example, in order to disallow relaxation of bounds on all internal reactions, set the field `.internalRelax` to `0`, and to allow the relaxation of bounds on all exchange reactions, set the field `.exchangeRelax` to `2`. If there are certain reaction bounds that should not be relaxed, then this can be specified using the Boolean vector field `.excludedReactions`. The first application of `relaxFBA` to a model may predict bounds to relax that are not supported by the literature or other experimental evidence. In this case, the field `.excludedReactions` can be used to disallow the relaxation of bounds on certain reactions.

Sparse flux balance analysis ● Timing 1 s –17 min

24 The prediction of the minimal number of active reactions required to carry out a particular set of biochemical transformations[72], consistent with an optimal objective derived from FBA, is based on a cardinality minimization problem termed sparse FBA:

$$\begin{aligned} \min_{v} \quad & \|v\|_0 \\ \text{s.t.} \quad & Sv=b, \\ & l\le v\le u, \\ & c^Tv=p^*, \end{aligned} \tag{3}$$

where the last constraint is optional and represents the requirement to satisfy an optimal objective value ρ^* derived from any solution to problem (1). The optimal flux vector can be considered as a steady-state biochemical pathway with minimal support, subject to the bounds on reaction rates and satisfaction of the optimal objective of problem (1). There are many possible applications of such an approach; here, we consider one example.

Sparse FBA is used to find the smallest active stoichiometrically balanced cycle that can produce ATP at a maximal rate using the ATP synthase reaction (https://www.vmh.life/#reaction/ATPS4m). We use the `Recon3Dmodel.mat`[66] (naming subject to change), which does not have such a cycle active due to bound constraints, but does contain such an active cycle with all internal reactions set to be irreversible. First the model is loaded, then the internal reactions are identified and blocked, and finally the objective is set to maximize the ATP synthase reaction rate. Thereafter, the sparse FBA solution is computed. To do this, use the following command:

```
>> model = readCbModel('Recon3Dmodel.mat');
>> model = findSExRxnInd(model);
>> modelClosed = model;
>> modelClosed.lb(model.SIntRxnBool) = 0;
>> modelClosed.ub(model.SIntRxnBool) = 0;
>> modelClosed_ATPS4mi = changeObjective(modelClosed, ‘ATPS4mi’, 1);
>> osenseStr = ‘max’;
>> minNorm = ‘zero’;
>> sparseFBAsolution = optimizeCbModel(modelClosed_ATPS4mi, osenseStr,
minNorm);
```

Identification of dead-end metabolites and blocked reactions ● Timing ~2 min

25 Manually curated, as well as automatically created, genome-scale metabolic reconstructions contain dead-end metabolites, which can be either only produced or only consumed in the metabolic network (including transport to/from the system boundary). Given a `model`, the function `detectDeadEnds`identifies all dead-end metabolites in a model. To do this, use the following command:

```
>> deadEndMetabolites = detectDeadEnds(model);
```

26 The `deadEndMetabolites` can be split into `downstreamGaps` and `rootGaps`. Metabolites that cannot be produced or consumed by any of the reactions in the network are referred to as `rootGaps`. To list these, use the following command:

```
>> [deadEndMetabolites, rootGaps, downstreamGaps] = gapFind(model,
‘true’);
```

Dead-end metabolites listed in `deadEndMetabolites` are metabolites that are both produced and consumed on the basis of network topology alone but are still dead-end metabolites because there are not any two reactions that can actively produce and consume the metabolite in any steady state.

27 Both the root and the downstream metabolites are part of reactions that cannot carry any flux (i.e., blocked reactions), given the network topology subject to the current bounds on reaction rates. To identify blocked reactions, use the following command:

```
>> blockedReactions = findBlockedReaction(model);
```

Gap-filling a metabolic network ● Timing 2 min–28 h

28 Dead-end metabolites show that there are missing reactions in the network that must enable their consumption/production. Thus, they define the boundaries of network gaps that must be filled with one or more reactions to complete our representation of the full metabolic network. These gaps are due to incompleteness of our current knowledge, even in well-studied model organisms[73]. This is partially due to orphan enzymes, whose biochemical functions have been described but for which no corresponding gene sequences have yet been found in any organism[74]. Such biochemical functions cannot be added to reconstructions by automatic (sequence-based) inference, and instead must be added manually or by some non-sequence-related computational approach. Moreover, gene annotations have been experimentally validated in only a limited number of organisms, which may lead to annotation errors when annotations are propagated across a large number of genes using sequence-based methods only[75]. Genome-scale metabolic reconstructions can assist in identifying missing knowledge by detecting and filling network gaps, as has been demonstrated for various organisms, including *Escherichia coli*[76,77], *Chlamydomonas reinhardtii*[78], and *Homo sapiens*[79,80].

The COBRA Toolbox facilitates the identification and filling of gaps using `gapFind`[81] and `fastGapFill`[82]. `fastGapFill` uses a reference database (*U*, e.g. KEGG REACTION) and a transport and exchange reaction database *X* that consists of transport and exchange reactions for each metabolite in both the reference database and the reconstruction. Reactions and pathways are proposed for addition to the metabolic reconstruction during gap filling from the combined *UX* database. `fastGapFill` works for both compartmentalized and decompartmentalized reconstructions. It relies on *fastcc.m*, which was developed within `fastCORE` in order to approximate the most compact (i.e., least) number of reactions to be added to fill the highest possible number of gaps.

Prioritize reaction types in the reference database to use for filling gaps using a `weights` parameter structure. The parameters `weights.MetabolicRxns`, `weights.ExchangeRxns`, and `weights.TransportRxns` allow different priorities to be set for internal metabolic reactions, exchange reactions, and transport reactions, respectively. Transport reactions include intracellular and extracellular transport reactions. The lower the weight for a reaction type, the higher is its priority. Generally, a metabolic reaction should be prioritized in a solution over transport and exchange reactions with, for example, the following commands:

```
>> weights.MetabolicRxns = 0.1;
>> weights.ExchangeRxns = 0.5;
>> weights.TransportRxns = 10;
```

29 Use the function `prepareFastGapFill` to prepare a gap-filling problem. A reconstruction is given as a `model` structure along with the optional inputs: list of compartments (`listCompartments`); a parameter `epsilon` that is needed for the fastCORE algorithm; the `fileName` for the universal database (e.g., KEGG; default: `'reaction.lst'`); `dictionaryFile`, which lists the universal database IDs and their counterpart in the reconstruction as defined in `model.mets` (default: `'KEGG_dictionary.xls'`); and `blackList`, which permits the exclusion of certain reactions from the universal database (default: no blacklist).

```
>> [consistModel, consistMatricesSUX, blockedRxns] =...
prepareFastGapFill(model, listCompartments, epsilon, fileName,
dictionaryFile, blackList);
```

The first output variable is `consistModel`, which contains a flux-consistent subnetwork of the input model.

`consistMatricesSUX` represents the flux-consistent *SUX* matrix, which contains the flux-consistent S matrix (model), the universal database placed in all cellular compartments, along with transport reactions for each metabolite from cytosol to compartment and exchange reactions for all extracellular metabolites. Finally, `blockedRxns` lists again the blocked reactions in the input model.

The main aim of the `fastGapFill` function is to find a compact set of reactions from the *UX* matrix to be added to the input model to close the gaps in the model. The main result is a list of candidate reactions to be added to the metabolic reconstructions. These reactions need to be evaluated for their biological and physiological plausibility in the organism, or cell type, under consideration.

30 Perform gap filling using option A or B, depending on the amount of metadata required to aid the interpretation of proposed reactions to be added to the model to fill gaps.

▲**CRITICAL** Algorithmic approaches help to identify new candidate reactions, but these candidates must be manually curated before being added to a reconstruction. This step is critical for obtaining a high-quality metabolic reconstruction. Adding the least number of reactions to fill gaps may not be the most appropriate assumption from a biological viewpoint. Consequently, the reactions proposed to be added to reconstruction require further manual assessment. Proposed gap-filling solutions must be rejected if they are biologically incorrect.

▲**CRITICAL** The mapping between the metabolite abbreviations in the universal database (e.g., KEGG) and the reconstruction metabolite abbreviations in `model.mets`, will ultimately limit how many blocked reactions might be resolved with `fastGapFill`. The larger the number of metabolites that map between these different namespaces, the larger the pool of metabolic reactions from the universal database that can be proposed to fill gaps. The mapping between the reconstruction and universal metabolite database can be customized using the `dictionaryFile`, which lists the universal database identifiers and their counterparts in the reconstruction.

(A) **Gap filling without return of additional metadata**

(i) To fill gaps without returning additional metadata, run the following command:

```
>> epsilon = 1e-4;
>> addedRxns = fastGapFill(consistMatricesSUX, epsilon, weights);
```

The parameter `epsilon` defines the minimum non-zero flux requested in a blocked reaction when filling gaps. In a multi-scale model, the value of epsilon may need to be decreased when using a quadruple precision solver (Step 15). The output `addedRxns` contains the reactions from the *UX* matrix added to fill the gap(s).

(B) **Gap filling with return of additional metadata**

(i) To return additional metadata for assistance with the manual evaluation of proposed reactions, use the following command:

```
>> addedRxnsExtended = postProcessGapFillSolutions(addedRxns,
model, blockedRxns);
```

The output structure `addedRxnsExtended`contains the information present in `addedRxns`, as well as the statistics and whether desired pathways contain the flux vectors.

Integration of extracellular metabolomic data ● Timing 17 min–28 h

31 The COBRA Toolbox can integrate metabolomics data with metabolic models. Metabolomics is an indispensable analytical method in many biological disciplines, including microbiology, plant sciences, biotechnology, and biomedicine. In particular, extracellular metabolomic data are often generated from cell lines in order to characterize and phenotype them under different experimental conditions (e.g., drug treatment or hypoxia). However, the analysis and interpretation of metabolomic data are still in their infancy, limiting the interpretation to potential metabolic pathways rather than providing a comprehensive understanding of the underlying mechanistic basis of the observed data.

MetaboTools is a COBRA Toolbox extension[65] that integrates semi-quantitative and quantitative extra-cellular metabolomic data with metabolic models. The resulting models allow for the interpretation and generation of experimentally testable mechanistic hypotheses. With *Metabo-Tools*, extracellular metabolomic data are integrated with a COBRA model structure, e.g., the generic human metabolic model[66], in a way that ensures the integration of a maximal number of measured metabolites, while adding a minimal number of additional uptake and secretion metabolites such that the specified constraints on the metabolic network can be sustained.

It is assumed that the extracellular metabolomic experiments are carried out with a defined fresh medium and that the corresponding model can take up only the components of the medium (plus dissolved gases). To apply constraints that are representative of the chemical composition of the fresh medium used in an experiment, use the `setMediumConstraints`function:

```
>> modelMedium = setMediumConstraints(starting_model, set_inf,
current_inf,... medium_composition, met_Conc_mM, cellConc, t, cell-
Weight, mediumCompounds,... mediumCompounds_lb);
```

The `starting_model` is the model before addition of fresh medium constraints. The `current_inf` input argument allows one to specify a value for the large-magnitude finite number that is currently used to represent an effectively infinite reaction rate bound, then harmonize it to a new value specified by `set_inf`. When no information on the bounds of a reaction is known, the ideal way to set reaction bounds is by use of the commands `model.lb(j) = -inf;` and `model.ub(j)= inf;`. However, depending on the optimization solver, an infinite lower or upper bound may or may not be accepted. Therefore, when no information on the bounds of a reaction is known, except perhaps the directionality of the reaction, then the upper or lower bound may be a large-magnitude finite number, e.g., `model.ub(j)= 1000;`. The fresh medium composition must be specified with a vector of exchange reaction abbreviations for metabolites in the cell medium (`medium_composition`) and the corresponding millimolar concentration of each medium component (`met_Conc_mM`). The density of the culture (`cellConc`, cells per mL), the time between the beginning and the end of the experiment (*t*, hours), and the measured cellular dry weight (`cellWeight`, gDW) must also be specified. Basic medium components (`mediumCompound`), such as protons, water, and bicarbonate, and the corresponding lower bounds on exchange reactions (`mediumCompounds_lb`), must also be specified. Even though they are present, they are not usually listed in the specification of a commercially defined medium, but they are needed for cells and the generic human metabolic model in order to support the synthesis of biomass. The `modelMedium` is a new model with external reaction bounds set according to the defined fresh medium.

32 Next, prepare the quantitative exometabolomic data using the `prepIntegrationQuant` function:

```
>> prepIntegrationQuant(modelMedium, metData, exchanges, sample-
Names, test_max, test_min,... outputPath);
```

The fluxes for each metabolite are given as uptake (negative) and secretion (positive) flux values in a metabolomic data matrix `metData`, in which each column represents a sample in `sampleNames` and each row in `exchanges` represents an exchanged metabolite. The units used for fluxes must be consistent within a model. For the input model in `modelMedium`, the `prepIntegrationQuant` function tests whether the qualitative uptake (`test_max`, e.g., ±500) and secretion (`test_min`, e.g., 10^{-5}) values of the metabolites are possible for each sample defined in the metabolomic data matrix `metData`. If a metabolite cannot be secreted or taken up, it will be removed from the data matrix for that particular sample. Possible reasons for this could be missing production or degradation pathways or blocked reactions. For each sample, the uptake and secretion profile compatible with the input model in `modelMedium` is saved to the location specified in `outputPath` using the unique sample name.

33 Use the model constrained by the defined fresh medium composition `modelMedium` and the output of the `prepIntegrationQuant` function to generate a set of functional, contextualized, condition-specific models using the following command:

```
>> [ResultsAllCellLines, OverViewResults] = setQuantConstraints(mod-
elMedium, samples, tol,... minGrowth, obj, no_secretion, no_uptake,
{}, {}, 0, outputPath);
```

A subset of samples can be specified with `samples`. All fluxes smaller than `tol` will be treated as zero. A lower bound (`minGrowth`, e.g., 0.008 per h) on a specified objective function (e.g., `obj = biomass_reaction2;`) needs to be defined, along with metabolites that should not be secreted (e.g., `no_secretion = 'EX_o2[e]'`) or taken up (`no_uptake = 'EX_o2s'`). The function returns a `ResultsAllCellLines` structure containing the context-specific models, as well as an overview of model statistics in `OverViewResults`. For each sample, a condition-specific model is created, in which the constraints have been set in accordance with the medium specification and the measured extracellular metabolomic data. This set of condition-specific models can then be phenotypically analyzed using the various additional functions present in the COBRA Toolbox as detailed in the *MetaboTools* protocol[65].

Integration of intracellular metabolomic data ● Timing 2 min–2.8 h

34 COBRA methods have also been developed for integration with intracellular metabolomic measurements[83–85], further improving the ability of the COBRA Toolbox to be used for the integration and interpretation of metabolomic data. In particular, unsteady-state FBA (uFBA[85]) enables the integration of absolutely quantified time-course metabolomic data into a metabolic model, generating constraints on intracellular fluxes even when intracellular metabolite levels are not at steady state. The main steps in the `uFBA` method are illustrated in Fig. 3. As a first step, experimentally quantify the absolute concentrations of a set of extracellular and intracellular metabolites at regular time intervals[85].

35 Plot the time-course metabolomic data. If the data are nonlinear, use principal-component analysis to define a sequence of temporal stages during which the time-course metabolomic data can be considered piecewise linear.

36 Use linear regression to estimate the rate of change of concentration with respect to time for each measured metabolite and for each temporal stage.

37 Load a standard COBRA `model` structure containing the fields `.S`, `.b`, `.lb`, `.ub`, `.mets`, and `.rxns`.

38 Integrate the rate of change in concentration for each measured metabolite with a COBRA model with the following command:

```
>> uFBAoutput = buildUFBAmodel(model, uFBAvariables);
```

The `uFBAvariables` structure must contain the following fields: `.metNames` is a list of measured metabolites, `.changeSlopes` provides the rate of change of concentration with respect

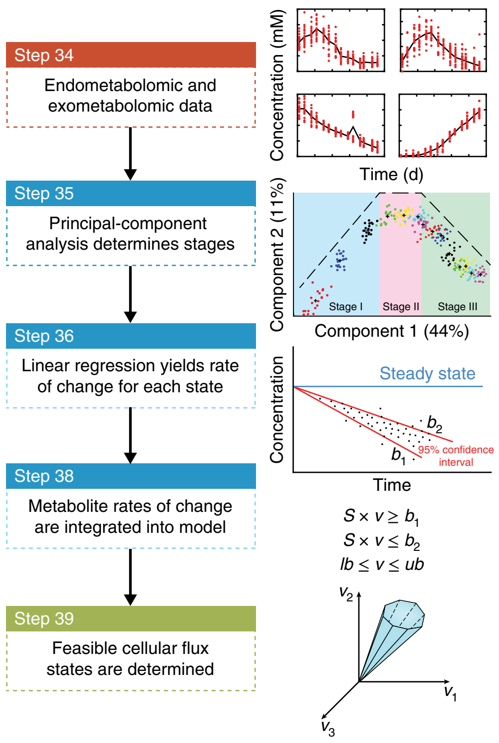


Fig. 3 | Unsteady-state flux balance analysis. Conceptual overview of the main steps involved in the unsteady-state flux balance analysis (uFBA) method.

to time for each measured metabolite, `.changeIntervals` yields the difference between the mean rate of change of concentration with respect to time and the lower bound of the 95% confidence interval. The list `ignoreSlopes` contains metabolites whose measurements should be ignored because of an unsubstantial rate of change.

The output is a `uFBAoutput` structure that contains the following fields: `.model`, a COBRA model structure with constraints on the rate of change of metabolite concentrations; `.metsToUse`, containing a list of metabolites with metabolomic data integrated into the model; and `.relaxedNodes`, containing a list of metabolites that deviate from steady state along with the direction (i.e., accumulation or depletion) and magnitude (i.e., reaction bound) of deviation. The uFBA algorithm automatically determines sink or demand reactions needed to return a model with at least one feasible flux balance solution, by automatically reconciling potentially incomplete or inaccurate metabolomic data with the model structure. The added sink or demand reactions allow the corresponding metabolites, defined by `.relaxedNodes`, to deviate from a steady state to ensure model feasibility. The default approach is to minimize the number of metabolites that deviate from steady state.

The `buildUFBAmodel` function integrates quantitative time course metabolomic data with a model by setting rates of change with respect to time for a set of measured intracellular and extracellular metabolites. A set of sink reactions, demand reactions, or both, may have been added to certain nodes in the network to ensure that the model admits at least one feasible mass balanced flux.

39 Use `optimizeCbModel` to minimize the obtained model:

```
>> model_ufba = optimizeCbModel(uFBAoutput.model);
```

Integration of transcriptomic and proteomic data ● Timing 2 min–2.8 h

40 Given a generic reconstruction of a biochemical network for a particular organism, some reactions may be active only in a specific tissue or cell type, or under specific environmental conditions. It is necessary to extract a context-specific model from a generic model in order to create a model that is representative of the part of the biochemical network that is active within a particular context. Each context-specific model is therefore a subset of a generic model. A variety of experimental data can

be used to determine the set of reactions that must be part of a context-specific model, including transcriptomic, proteomic, and metabolomic data, as well as complementary experimental data from the literature. Several model extraction methods have been developed, with different underlying assumptions, and each has been the subject of multiple comparative evaluations[86–88]. The selection of a model extraction method and its parameterization, as well as the methods chosen to preprocess and integrate the aforementioned omics data, substantially influences the size, functionality, and accuracy of the resulting context-specific model. Currently, there is insufficient evidence to assert that one model extraction method universally provides the most physiologically accurate models. Therefore, a pragmatic approach is to test the biochemical fidelity of context-specific models generated using a variety of model extraction methods. The COBRA Toolbox offers six different model extraction methods; to access these, use this common interface:

```
>> tissueModel = createTissueSpecificModel(model, options);
```

The different methods and associated parameters are selected via the `options` structure. The `.solver` field indicates which method will be used. The other fields of the `options` structure vary depending on the method and often depend on bioinformatic preprocessing of input omics data. There are additional optional parameters for all algorithms, with the default being the values indicated in the respective papers. Please refer to the original papers reporting each algorithm for details on the requirements for preprocessing of input data.

- *The FASTCORE[89] algorithm.* One set of core reactions that is guaranteed to be active in the extracted model is identified by `FASTCORE`. Then, the algorithm finds the minimum number of reactions possible to support the core; the `.core` field provides the core reactions, which have to be able to carry flux in the resulting model.
- *The GIMMME[90] algorithm.* With this algorithm, the usage of low-expression reactions is minimized while keeping the objective (e.g., biomass) above a certain value; the `.expressionRxns`field provides the reaction expression, with `-1` for unknown reactions or reactions not linked to genes; the `.threshold` field sets the threshold above which a reaction is assumed to be active.
- *The iMAT[91] algorithm.* `iMAT` finds the optimal trade-off between including high-expression reactions and removing low-expression reactions; the `.expressionRxns` field is defined as above; the `.threshold_lb` field is the threshold below which reactions are assumed to be inactive; the `.threshold_ub` field is the threshold above which reactions are assumed to be active.
- *The INIT[92] algorithm.* The optimal trade-off between including and removing reactions on the basis of their given weights is determined by this algorithm; .the `weights` field provides the weights w for each reaction in the objective of `INIT`$\left(\max\left(\sum_{i\in R} w_i y_i + \sum_{j\in M} x_j\right)\right)$. Commonly, a high expression leads to higher positive values and low or no detection leads to negative values.
- *The MBA[93] algorithm.* `MBA` defines high-confidence reactions to ensure activity in the extracted model. Medium-confidence reactions are kept only when a certain parsimony trade-off is met. The `.medium_set` field provides the set of reactions that have a medium incidence, and the `.high_set` field provides the set of reactions that have to be in the final model. Any reaction not in the medium or high set is assumed to be inactive and preferably not present in the final model.
- *The mCADRE[94] algorithm.* A set of core reactions is first found, and all other reactions are then pruned on the basis of their expression, connectivity to core, and confidence score. Reactions that are not necessary to support the core or defined functionalities are thus removed. Core reactions are removed if they are supported by a certain number of zero-expression reactions. The `.confidenceScores` field provides reliability for each reaction, generally based on the literature, and the `.ubiquityScore` field provides the ubiquity score of each reaction in multiple replicates, i.e., the number of times the reaction was detected as active in experimental data under the investigated condition.

▲ CRITICAL STEP When integrating omics data, parameter selection is critical, especially selection of the threshold for binary classification, e.g., the threshold for genes to be placed into active or inactive sets. Algorithmic performance often strongly depends on parameter choices and on the choice of data preprocessing method[87]. However, `createTissueSpecificModel` does not offer data preprocessing tools, because the selection of the discretization method and parameters depend on the origin of the data. However, the COBRA Toolbox offers functionality to map preprocessed expression data to reactions via the function `mapExpressionToReactions (model, expression)`.

Adding biological constraints to a flux balance model ● Timing ~2 min

41 A cell-type or organ-specific model can be converted into a condition-specific model, on the basis of the imposition of experimentally derived constraints. There are several types of constraints that can be imposed on a metabolic network, such as biomass maintenance requirements, environmental constraints, or maximal enzyme activities. In general, biomass constraints[71] are added as part of a biomass reaction. In some instances, however, a cell type (e.g., neuron) does not divide, but is required only to turn over its biomass components. Turnover rates are commonly expressed as half-lives and represent the time required for half of the biomass precursor to be replaced[95]. A model can be constrained with inequality constraints so as to require a minimal rate of turnover for a metabolite. If that metabolite possesses only one degradation pathway, then it is sufficient to adjust the bounds on a reaction in that pathway. However, if there are multiple possible degradation pathways, then it is necessary to impose a lower bound on the total rates of a set of irreversible degradation reactions, one for each possible degradation pathway of the metabolite in question. The implementation of such a constraint is illustrated in the following example. In the brain, phosphatidylcholine (PC) can be degraded by three different metabolic pathways[96]:

- Phospholipase D acts on the choline/phosphate bond of PC to form choline and phosphatidic acid (PCHOLP_hs, https://www.vmh.life/#reaction/PCHOLP_hs).
- Phospholipase A2 acts on the bond between the fatty acid and the hydroxyl group of PC to form a fatty acid (e.g., arachidonic acid or docosahexaenoic acid) and lysophosphatidylcholine (PLA2_2, https://vmh.life/#reaction/PLA2_2).
- Ceramide and PC can also be converted to sphingomyelin by sphingomyelin synthetase (SMS, https://vmh.life/#reaction/SMS).

Load a COBRA model and define the set of reactions that will represent degradation of the metabolite in question; for this example type:

```
>> multipleRxnList = {'PCHOLP_hs', 'PLA2_2', 'SMS'};
```

▲**CRITICAL STEP** Correctly converting the literature data into bound constraints with the same units used for the model fluxes may be a challenge. Indeed, the curation of biochemical literature to abstract the information required to quantitatively bound turnover rates can take between 4 and 8 weeks when the target is to retrieve the biomass composition and the turnover rates of each of the different biomass precursors. Once all the constraints are available, imposing the corresponding reaction bounds takes <5 min.

42 Verify that all the reactions are irreversible (the lower and upper bounds should be ≥0).

```
>> rxnInd = findRxnIDs(model, multipleRxnList);
>> model.lb(rxnInd);
>> model.ub(rxnInd);
```

43 Generate and add the constraint:

```
>> c = [1, 1, 1];
>> d = 2.674;
>> ineqSense = 'G';
>> modelConstrained = constrainRxnListAboveBound(model, multiple
RxnList, c, d, ineqSense);
```

where `c` is a vector forming the inequality constraint $c^T v \geq d$, and `d` is a scalar. `ineqSense` encodes the sense of these inequalities (`'L'`for a lower inequality, or `'G'`for an upper inequality). In this example, all entries of `c` are positive as we seek for the sum of the rates of the three reactions (irreversible in the forward direction) to be greater than `d`. This extra constraint is encoded in the `model.C` field.

44 Check that the constraints are correctly added to the model:

```
>> [nMet, nRxn] = size(modelConstrained.S);
```

45 Solve the FBA problem with the extra constraint $c^T v \geq d$:

```
>> solution = optimizeCbModel(modelConstrained, 'max', 1e-6);
```

46 Check the values of the added fluxes. The sum of fluxes should be greater than or equal to the value of d:

```
>> solution.v(rxnInd);
>> sum(c*FBAsolution.v(rxnInd));
```

Qualitative chemical and biochemical fidelity testing ● Timing 2–17 min

47 Once a context-specific model is generated, it is highly advisable to frequently compare preliminary model predictions with published experimental data[6]. Such predictions must be compared directly with an unbiased selection of appropriate independent biological literature in order to identify possible sources of misconception or computational misspecification. It is challenging to compare genome-scale predictions with experimental data that may be available for only a subset of a biochemical network. In this context, it is important to first turn to literature relevant to the aspect of the biological network being represented by a model and then check if the literature result is correctly predicted by the model. Inevitably, this is an iterative approach, with multiple rounds of iterative refinement of the reconstruction and the model derived from it before finalization of a model version and comparison of final predictions with independent experimental data.

A draft model should be subjected to a range of quantitative and qualitative chemical and biochemical fidelity tests. As described in Step 16, chemical fidelity testing includes testing for stoichiometric consistency. This should not be necessary if one starts with a stoichiometrically consistent generic model and extracts a context-specific model from it. However, it is possible that mis-specified reactions might have been inadvertently added during refinement of a reconstruction; therefore, retest for stoichiometric consistency. Beyond chemical fidelity, it is advised to test again for biochemical fidelity. Such tests are very specific to the particular biological domain that is being modeled. Here, we focus on human metabolism and use `modelClosed`, the `Recon3Dmodel`[66] with all external reactions closed, from Steps 21 and 22. Encode and conduct qualitative fidelity tests for anticipated true negatives.

As an example, the following test is for the production of ATP from water alone in a closed model:

```
>> modelClosedATP = changeObjective(modelClosed, 'DM_atp[c]');
>> modelClosedATP = changeRxnBounds(modelClosedATP, 'DM[atp_c]', 0, 'l');
>> modelClosedATP = changeRxnBounds(modelClosedATP, 'EX_h2o[e]', -1, 'l');
>> FBAsol = optimizeCbModel(modelClosedATP);
```

If `FBAsol.stat == 0`, then the model is incapable of producing ATP from water, as expected. If `FBAsol.stat == 1`, then the supposedly closed model can produce ATP from water. This indicates that there are stoichiometrically inconsistent reactions in the network, which need to be identified. See Step 16 for instructions how to approach this analysis.

48 Encode and conduct qualitative fidelity tests for anticipated true positives. The following metabolic function test is for the production of mitochondrial succinate from 4-aminobutanoate in a model that is closed to exchange of mass across the boundary of the system, except for the metabolites `'gly[c]'`, `'co2[c]'`, and `'nh4[c]'`.

```
>> modelClosed = addSinkReactions(modelClosed, {'gly[c]', 'co2[c]',
'nh4[c]'},...
[-100, -1; 0.1, 100; 0.1, 100]);
>> modelClosed = changeObjective(modelClosed, 'sink_nh4[c]');
>> sol = optimizeCbModel(modelClosed, 'max', 'zero');
```

If `FBAsol.stat == 1`, then it is feasible for the model to produce mitochondrial succinate from 4-aminobutanoate. If `FBAsol.stat == 0`, then this metabolic function is infeasible. This is not anticipated and indicates that further gap filling is required (Steps 28–30).

Quantitative biochemical fidelity testing ● Timing 2–17 min

49 Check if a model can reproduce or closely approximate known quantitative features of the biochemical network being represented. Here, we illustrate how to predict the ATP yield from different carbon sources under aerobic or anaerobic conditions for Recon3D[66]. These are compared with the values for the corresponding ATP yields obtained from the biochemical literature. This approach can be adapted for condition- and cell-type-specific models derived from Recon3D in order to test whether these models are still able to produce physiologically relevant ATP yields. Add and define the ATP hydrolysis reaction `DM_atp[c]` to be the objective reaction in the model with the following commands:

```
>> modelClosed = addReaction(modelClosed,...
'DM_atp[c]', 'h2o[c] + atp[c] -> adp[c] + h[c] + pi[c]');
>> modelClosed = changeObjective(modelClosed, 'DM_atp[c]');
```

50 Allow the model to uptake oxygen and water, and then provide 1 mol/gdw/h of a carbon source, e.g., glucose (Virtual Metabolic Human database (VMH; http://vmh.life) ID: glc_D[e]):

```
>> modelClosed.lb(find(ismember(modelClosed.rxns, 'EX_o2[e]'))) =
-1000;
>> modelClosed.lb(find(ismember(modelClosed.rxns, 'EX_h2o[e]'))) =
-1000;
>> modelClosed.ub(find(ismember(modelClosed.rxns, 'EX_h2o[e]'))) =
1000;
>> modelClosed.ub(find(ismember(modelClosed.rxns, 'EX_co2[e]'))) =
1000;
>> modelClosed.lb(find(ismember(modelClosed.rxns, 'EX_glc_D[e]')))
= -1;
>> modelClosed.ub(find(ismember(modelClosed.rxns, 'EX_glc_D[e]')))
= -1;
```

51 Compute an FBA solution with maximum flux through the `DM_atp[c]` reaction:

```
>> FBAsolution = optimizeCbModel(modelClosed, 'max', 'zero');
```

MinSpan pathways: a sparse basis of the nullspace of a stoichiometric matrix ● Timing 2 min–2.8 h

52 COBRA models are often mathematically deconstructed into feasible steady-state flux vectors in biochemical networks that can be biologically conceptualized as pathways. Much development and analysis has been done for such pathway vectors in terms of elementary flux modes[97], extreme pathways[98], and elementary flux vectors[99]. As the number of elementary or extreme vectors scales exponentially with the size of a typical metabolic network, increasingly efficient algorithms become essential for enumerating elementary or extreme vectors at genome scale. An alternative approach[100] is to approximately compute a set of $n-$ rank(S) sparse linearly independent flux vectors that together form a basis of the right nullspace of a stoichiometric matrix and also satisfy specified constraints on reaction directionality. This approach requires the solution of a greedy sequence of mixed-integer linear optimization problems, each of which computes a sparse flux mode that is linearly independent from the rest of the vectors within a nullspace basis. The end result is a sparse set of linearly independent flux modes denoted *MinSpan* pathways. If you have a model with a stoichiometric matrix, reaction bounds, and reaction identifiers, invoke the *MinSpan* algorithm with:

```
>> Z = detMinSpan(model, params);
```

If you want to change key parameters, do this using the `params` structure. Among others, the main parameters include the amount of time `.timeLimit` for each iterative solve in seconds and the number of threads for the MILP solver to use. The output $Z \in \mathbb{R}^{n\times(n-rank(S))}$ is a sparse set of $n-$ rank(S) linearly independent flux modes, each corresponding to a *MinSpan* pathway.

Low-dimensional flux variability analysis ● Timing 1 s–17 min

53 FBA does not, in general, return a unique optimal flux vector. That is, problem (1) returns an optimal flux vector, $v^* \in \mathbb{R}^n$ with one flux value for each reaction, but typically an infinite set of steady-state flux vectors exist that can satisfy the same requirement for an optimal objective, $c^T v^* = c^T v$, as well as the other equalities and inequalities in problem (1). Flux variability analysis is a widely used method for evaluating the minimum and maximum range of each reaction flux that can still satisfy the aforementioned constraints using two optimization problems for each reaction of interest

$$\begin{aligned} \max_v \backslash \min v_j & \\ \text{s.t.} \mathcal{S}_v &= 0 \\ l \leq v &\leq u, \\ c^T v &= c^T v^*. \end{aligned} \tag{4}$$

Just as there are many possible variations on FBA, there are many possible variations on flux variability analysis. The COBRA Toolbox offers a straightforward interface to implement standard flux variability analysis and a wide variety of options to implement variations on FBA.

(A) **Standard flux variability analysis**

(i) Use the following command to compute standard flux variability analysis:

```
>> [minFlux, maxFlux] = fluxVariability(model);
```

The result is a pair of n-dimensional column vectors, `minFlux` and `maxFlux`, with the minimum and maximum flux values satisfying problem (4).

(B) **Advanced flux variability analysis**

(i) Access the full spectrum of flux variability analysis using the command:

```
>> [minFlux, maxFlux, Vmin, Vmax] = fluxVariability(model, optPer-
centage, osenseStr,... rxnNameList, verbFlag, allowLoops, method);
```

The `optPercentage` parameter allows one to choose whether to consider solutions that give at least a certain percentage of the optimal solution. For instance `optPercentage = 0` would just find the flux range of each reaction, without of any requirement to satisfy any optimality with respect to FBA. Setting the parameters `osenseStr = 'min'`or `osenseStr = 'max'`determines whether the FBA problem is first solved as a minimization or maximization.

The `rxnNameList` accepts a cell array list of reactions upon which to selectively perform flux variability. This is useful for high-dimensional models, for which the computation of a flux variability for all reactions is more time consuming. The additional $n \times k$ output matrices `Vmin` and `Vmax` return the flux vector for each of the $k \leq n$ fluxes selected for flux variability.

The `verbFlag` input determines how much output will be printed. The parameter `allowLoops == 0` invokes an MILP implementation of thermodynamically constrained flux variability analysis for each minimization or maximization of a reaction rate. The `method` input argument determines whether the output flux vectors also minimize the `0-norm`, `1-norm`, or `2-norm` while maximizing or minimizing the flux through one reaction.

The default result is a pair of maximum and minimum flux values for each reaction. Optional parameters can be set. For instance, parameters can be set to control which subset of $k \leq n$ reactions of interest will be obtained, or to determine the characteristics of each of the $2 \times k$ flux vectors.

High-dimensional flux variability analysis ● Timing 1 s–28 h

54 Besides FBA, flux variability analysis is the most widely used constraint-based modeling method for high-dimensional models. However, its use in this setting requires a more sophisticated computational approach, with a multi-core processor[101] or computational cluster[42], and a commercial-grade linear optimization solver. In this setting, advanced users have two options:

(A) **Use fastFVA with MATLAB**

(i) Solve the $2 \times k$ linear optimization problems using multiple threads running on parallel processors with `fastFVA`, which depends on the CPLEX solver (IBM), using the following command:

```
>> [minFlux, maxFlux, optsol] = fastFVA(model, optPercentage,
osenseStr);
```

The output argument `optsol` returns the optimal solution of the initial FBA.

(B) **Use *distributedFBA.jl* with Julia**

(i) Alternatively, solve the $2 \times k$ linear optimization problem using multiple threads running on parallel processors or a cluster using *distributedFBA.jl*, an openCOBRA extension that permits the solution of FBA, a distributed set of flux balance problems, or a flux variability analysis using a common of solver (GLPK, CPLEX, Clp, Gurobi, and Mosek). Assuming that *distributedFBA.jl* has been correctly installed and configured, the commands to go back and forth between a model or results in MATLAB and the computations in Julia are as follows:

```
>> save('high_dimensional_model.mat', unimodel);
# -- switch to Julia --
julia  >  model  =  loadModel("high_dimensional_model.mat");
julia> workersPool, nWorkers = createPool(128);
julia > minFlux, maxFlux, optSol, fbaSol, fvamin, fvamax =
distributedFBA(model,...
solver, nWorkers=nWorkers, optPercentage=optPercentage, pre-
FBA=true);   julia>   saveDistributedFBA("high_dimensional_
FVA_results.mat");
# -- switch to MATLAB --
>> load('high_dimensional_FVA_results.mat');
```

Here, `nWorkers = 128` will distribute the flux variability analysis problem among 128 Julia processes on one or more computing nodes in a computational cluster.

Uniform sampling of steady-state fluxes ● Timing 1 s–17 min

55 An unbiased characterization of the set of flux vectors consistent with steady-state, mass balance, and reaction-bound constraints can be obtained by uniformly sampling the feasible set $\Omega := \{v \mid Sv = 0; l \leq v \leq u\}$. The feasible set for sampling should be defined based on biochemically justifiable constraints. These are the same conditions that apply when formulating the FBA problem (1), except that there is no need to formulate a linear objective. To ensure that the sample is statistically representative of the entire feasible set, a sufficiently large number of flux vectors must be collected, and the flux vectors must be collected randomly within the feasible set. Recently, we distributed new software to uniformly sample feasible sets of steady-state fluxes[102] on the basis of a coordinate hit-and-run with rounding (CHRR) algorithm[103,104] that is guaranteed to return a statistically uniform distribution when appropriately utilized. The CHRR sampling algorithm is therefore used by default. Figure 4 illustrates the basics of this algorithm. Sample the model by either using the default setting or tailoring the parameters with more arguments to the interface. A pragmatic approach is to first try option (A) with the default parameters, and then check the quality of the marginal flux distribution for a subset of reactions (Fig. 4). Especially for higher dimensional models, it may be necessary to tune the parameters with option (B).

(A) **Sampling of a mono-scale model with <~2,000 variables**

(i) Use the default settings to sample a model that contains <2,000 variables:

```
>> [modelSampling, samples] = sampleCbModel(model);
```

The `samples` output is an $n \times p$ matrix of sampled flux vectors, where p is the number of samples. To accelerate any future rounds of sampling, use the `modelSampling` output. This is a model storing extra variables acquired from preprocessing the model for sampling (Fig. 4).

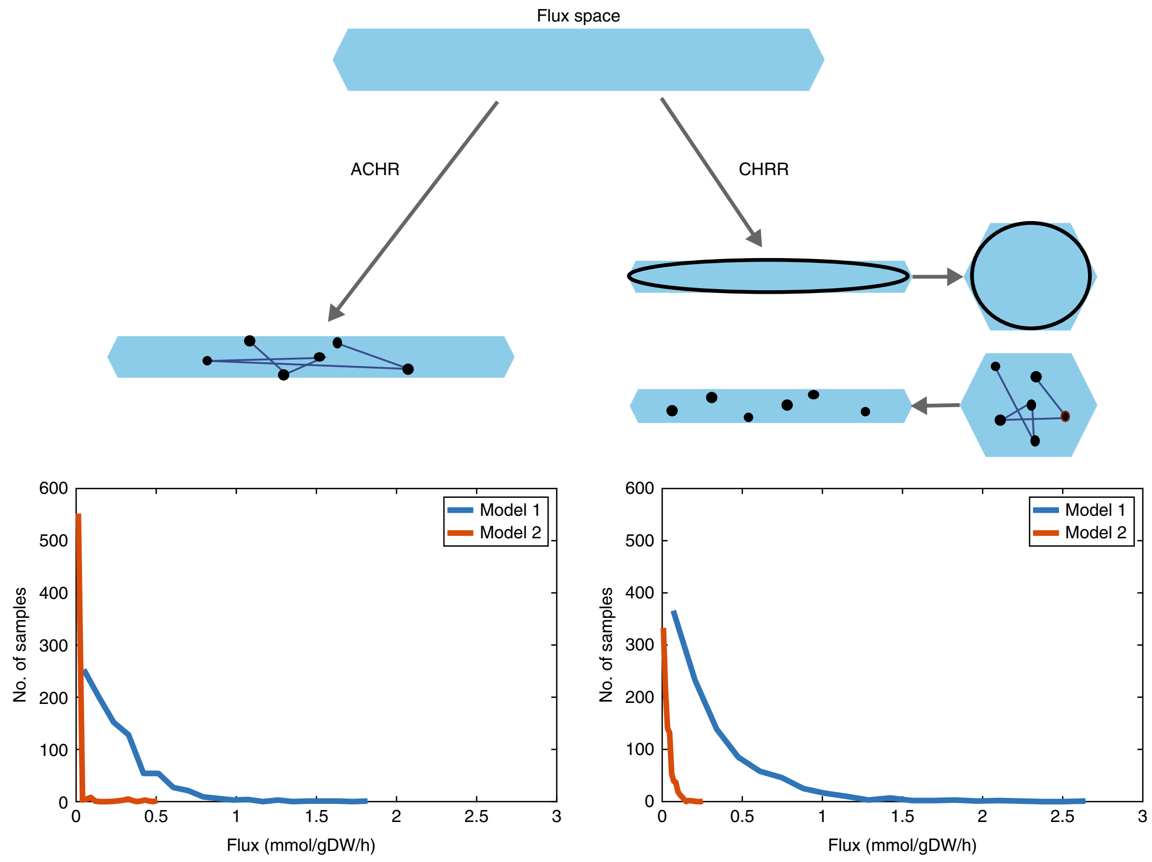


Fig. 4 | Solution spaces from steady-state fluxes are anisotropic, that is, long in some directions and short in others. This impedes the ability of any sampling algorithm taking a random direction to evenly explore the full feasible set (*artificial centering hit-and-run* (ACHR) algorithm). The CHRR (*coordinate hit-and-run with rounding*) algorithm first rounds the solution space based on the maximum-volume ellipsoid. Then, the rounded solution space is uniformly sampled using a provably efficient coordinate hit-and-run random walk. Finally, the samples are projected back onto the anisotropic feasible set. This leads to a more distributed uniform sampling, so that the converged sampling distributions for the selected reactions become smoother.

(B) **Sampling of a model with <~10,000 variables**

(i) Sample larger models containing <10,000 variables by tuning the optional input parameters:

```
>> [modelSampling, samples] = sampleCbModel(model, sample-
File, samplerName, options,...
modelSampling);
```

The variable `sampleFile` contains the name of a .mat file used to save the sample vectors to disk. A string passed to `samplerName` can be used to sample with non-default solvers. The `options` structure contains fields that control the number of sampling steps taken per sample point saved (`.nStepsPerPoint`) and the number of sample points saved (`.nPointsReturned`). Reasonable parameter values are nPointsSaved = $8 \times (n - \text{rank}(S))^2$ and nStepsPerPoint = 200. The output `modelSampling` is a model that can be used in subsequent rounds of sampling. Although rounding of large models is computationally demanding, the results can be reused when sampling the same model more than once. The CHRR algorithm provably converges to a uniform stationary sampling distribution, if enough samples are obtained, and has been tested with mono-scale metabolic models with up to 10,000 reactions. The default parameters are set using heuristic rules to estimate a sufficiently large number of samples, which balances this requirement against the desire to complete the sampling procedure in a practically useful period of time.

? TROUBLESHOOTING

Identification of all genetic manipulations leading to targeted overproductions

● **Timing 10 s–28 h**

56 A variety of strain design algorithms[58] are implemented within the COBRA Toolbox, including OptKnock[105], OptGene[106], Genetic Design Local Search (GDLS[107]), and OptForce[108]. Whereas OptKnock, OptGene, and GDLS can identify gene-deletion strategies, the OptForce method can identify not only gene deletion but also up- and downregulation strategies. As the OptForce method is new to this version of the COBRA Toolbox, we provide an illustrative example of strain design using this method. Consider the problem of finding a set of interventions of size *K* such that when these interventions are applied to a wild-type strain, the mutant created will produce a particular target of interest at a higher yield than the wild-type strain. The interventions could be knockouts (zero out the flux for a particular reaction), upregulations (increase the flux for a particular reaction), or downregulations (decrease the flux for a particular reaction). As an example, we will use the OptForce method to identify all genetic manipulations leading to the overproduction of succinate in *E. coli*[108]. The OptForce method consists of the following set of steps: define the constraints for both wild-type and mutant strains, perform flux variability analyses for both wild-type and mutant strains, find the sets of reactions that must alter their flux in order to achieve the desired phenotype in the mutant strain, and, finally, find the interventions needed to ensure an increased production of the target of interest (Steps 69–73). First, select a commercial-grade solver and select the local directory in which to save the generated results with the following command:

```
>> changeCobraSolver('gurobi', 'ALL');
```

57 Load an illustrative model that comprises only 90 reactions, describing the central metabolism in *E. coli*[109].

```
>> readCbModel('AntCore.mat');
```

58 Set the objective function to maximize the biomass reaction (`R75`). Change the lower bounds such that the *E. coli* model will be able to consume glucose, oxygen, sulfate, ammonium, citrate, and glycerol.

```
>> model = changeObjective(model, 'R75', 1);
>> for rxn = {'EX_gluc', 'EX_o2', 'EX_so4', 'EX_nh3', 'EX_cit',
'EX_glyc'} model = changeRxnBounds(model, rxn, -100, 'l');
end
```

59 Define the constraints for both wild-type and mutant strains:

```
>> constrWT = struct('rxnList', {{'R75'}}, 'rxnValues', 14,
'rxnBoundType', 'b');
>> constrMT = struct('rxnList', {{'R75', 'EX_suc'}}, 'rxnValues',
[0, 155.55],... 'rxnBoundType', 'bb');
```

▲ **CRITICAL STEP** In this example, we provide the constraints for both wild-type and mutant strains, but in a typical scenario, the definition of differential constraints on wild-type and mutant strains requires additional research. This step could take a few days or weeks, depending on the information available for the species of interest. Flux bounds (i.e., uptake rate and minimum biomass yield target) are required inputs. New experiments might be required to be performed in addition to the literature curation task in order to obtain such data. Assumptions may also be made when describing the phenotypes of both strains, which will reduce the dependency on literature curation. It is important that the two strains be sufficiently different in order to be able to anticipate differences in reaction ranges.

60 Perform flux variability analysis for both wild-type and mutant strains with the following commands:

```
>> [minFluxesW, maxFluxesW, minFluxesM, maxFluxesM] = FVAoptForce
(model, constrWT, constrMT);
>> disp([minFluxesW, maxFluxesW, minFluxesM, maxFluxesM]);
```

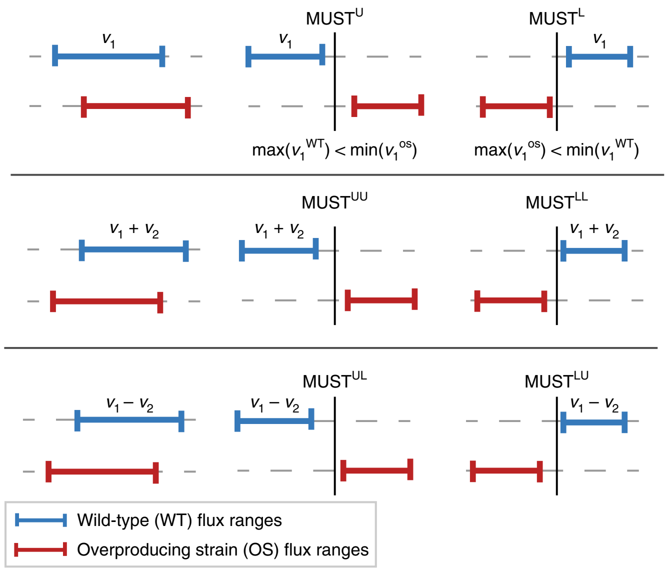

MUST^U
MUST^L
v_1
$\max(v_1^{WT}) < \min(v_1^{os})$
$\max(v_1^{os}) < \min(v_1^{WT})$
MUST^UU
MUST^LL
$v_1 + v_2$
MUST^UL
MUST^LU
$v_1 - v_2$
Wild-type (WT) flux ranges
Overproducing strain (OS) flux ranges

Fig. 5 | In the OptForce procedure, the MUST sets are determined by contrasting the flux ranges obtained using flux variability analysis (FVA) of a wild-type (blue bars) and an overproducing strain (red bars). The first order MUST sets (top panel) are denoted MUST^L and MUST^U. For instance, a reaction belongs to the MUST^U set if the upper bound of the flux range in the wild-type is less than the lower bound of the flux range of the overproducing strain. The center and bottom panels show all possible second-order MUST sets.

61 The MUST sets are the sets of reactions that must increase or decrease their flux in order to achieve the desired phenotype in the mutant strain. As shown in Fig. 5, the first-order MUST sets are `MustU` and `MustL`, and second-order MUST sets are denoted as `MustUU`, `MustLL`, and `MustUL`. After parameters and constraints are defined, the functions `findMustL` and `findMustU` are run to determine the `mustU` and `mustL` sets, respectively. Define an ID for the run with the following command:

```
>> runID = 'TestoptForceM';
```

Each time the MUST sets are determined, folders are generated to read inputs and store outputs, i.e., reports. These folders are located in the directory defined by the uniquely defined `runID`.

62 To find the first-order MUST sets, define the constraints:

```
>> constrOpt = struct('rxnList', {{'EX_gluc', 'R75', 'EX_suc'}},
'values', [-100; 0; 155.5]);
```

63 Determine the first-order MUST set MustL by running the following command:

```
>> [mustLSet, pos_mustL] = findMustL(model, minFluxesW, maxFluxesW,...
'constrOpt', constrOpt, 'runID', runID);
```

If `runID` is set to '`TestoptForceL'`, a folder *TestoptForceL* is created, in which two additional folders *InputsMustL* and *OutputsMustL* are created. The *InputsMustL* folder contains all the inputs required to run the function `findMustL`, and the *OutputsMustL* folder contains the `mustL` set found and a report that summarizes all the inputs and outputs. To maintain a chronological order of computational experiments, the report is timestamped.

64 Display the reactions that belong to the `mustL` set using the following command:

```
>> disp(mustLSet)
```

65 Determine the first-order MUST set `MustU` by running the following command:

```
>> [mustUSet, pos_mustU] = findMustU(model, minFluxesW, maxFluxesW,...
'constrOpt', constrOpt, 'runID', runID);
```

The results are stored and available in a format analogous to the `mustL` set. The reactions that belong to the `mustU` can be displayed in the same way as `mustL`.

66 Define the reactions that will be excluded from the analysis. The reactions found in the previous step, as well as exchange reactions, will be included.

```
>> constrOpt = struct('rxnList', {{'EX_gluc', 'R75', 'EX_suc'}},
'values', [-100, 0, 155.5]');
>> exchangeRxns = model.rxns(cellfun(@isempty, strfind(model.rxns,
'EX_')) == 0);
>> excludedRxns = unique([mustUSet; mustLSet; exchangeRxns]);
```

67 Determine the second-order MUST set MustUU by running the following command:

```
>> [mustUU, pos_mustUU, mustUU_linear, pos_mustUU_linear] = findMus-
tUU(model, minFluxesW,... maxFluxesW, 'constrOpt', constrOpt,...
'excludedRxns', excludedRxns,'runID', runID);
```

The results are stored and available in a format analogous to that of the `mustL` set. The reactions of the `mustUU` can be displayed using the `disp` function.

68 Repeat the above steps to determine the second-order MUST sets `MustLL` and `MustUL` by using the functions `findMustLL` and `findMustUL`, respectively. The results are stored and available in a format analogous to that of the `mustL` set. In the present example, `mustLL` and `mustUL` are empty sets.

? TROUBLESHOOTING

69 To find the interventions needed to ensure an increased production of the target of interest, define the `mustU` set as the union of the reactions that must be upregulated in the first- and second-order MUST sets. Similarly, `mustL` can be defined with the following commands:

```
>> mustU = unique(union(mustUSet, mustUU));
>> mustL = unique(union(mustLSet, mustLL));
```

70 Define the number of interventions `k` allowed, the maximum number of sets to find (`nSets`), the reaction producing the metabolite of interest (`targetRxn` (in this case, succinate)), and the constraints on the mutant strain (`constrOpt`) with the following commands:

```
>> k = 1; nSets = 1; targetRxn = 'EX_suc';
>> constrOpt = struct('rxnList', {{'EX_gluc','R75'}}, 'values',
[-100, 0]);
```

71 Run the OptForce algorithm and display the reactions identified by `optForce` with the following command:

```
>> [optForceSets, posoptForceSets, typeRegoptForceSets, flux_optFor-
ceSets] =... optForce(model, targetRxn, mustU, mustL, minFluxesW,
maxFluxesW, minFluxesM,... maxFluxesM, 'k', k, 'nSets', nSets, 'con-
strOpt', constrOpt, 'runID', runID);
>> disp(optForceSets)
```

72 To find non-intuitive solutions, increase the number of interventions `k` and exclude the `SUCt` reaction from upregulations. Increase `nSets` to find the 20 best sets. Change the `runID` to save this second result in a separate folder from that of the previous result, and then run `optForce` again as in Step 71.

```
>> k = 2; nSets = 20; runID = 'TestoptForceM2';
>> excludedRxns = struct('rxnList', {{'SUCt'}}, 'typeReg','U');
```

73 Display the reactions determined by `optForce` using `disp(optForceSets)`. The complete set of predicted interventions can be found in the folders created inside the `runID` folder in which inputs and outputs of `optForce` and associated `findMust*` functions are stored. The input folders *InputsFindMust** contain .mat files for running the functions to solve each one of the bilevel optimization problems. The output folders *OutputsFindMust** contain results of the algorithms (saved as .xls and .txt files), as well as a report (a .txt file) that summarizes the outcome of the steps performed during the execution of each function. The `optForce` algorithm will find sets of reactions that should increase the production of a specified target. The first sets found should be the best ones because the production rate will be the highest. The last ones will be the worst, as the production rate is the lowest.
▲**CRITICAL STEP** Be aware that some sets may not guarantee a minimum production rate for a target, so check the minimum production rate, e.g., by using the function `testoptForceSol`.

Atomically resolving a metabolic reconstruction ● Timing 10 s–28 h

74 Obtain chemical structures for each metabolite. In most genome-scale metabolic models, it is not explicit that the stoichiometric matrix represents a network of biochemical reactions. It is implicit that each row of the stoichiometric matrix corresponds to some molecular species, but when computing properties of the model, the atomic structure of each molecular species is not represented. It is also implicit that each column of the stoichiometric matrix corresponds to some biochemical reaction. However, when computing properties of the model, the mechanisms of the underlying biochemical reaction, in terms of the structures of the metabolites and the atomically resolved chemical transformations that take place, are not represented. Recent developments in genome-scale metabolic modeling have generated genome-scale metabolic reconstructions in which the molecular structures are specified[110] and the reaction mechanisms are represented by atom mappings between substrate and product atoms[66,111].

An atom mapping is a one-to-one association between a substrate atom and a product atom. An instance of a chemical reaction can be represented by a set of atom mappings, with one atom mapping between each substrate and product atom. A single chemical reaction can admit multiple chemically equivalent atom mappings when chemically equivalent atoms are present in a substrate, a product, or both. Therefore, each chemical reaction can be represented by one set, or multiple chemically equivalent sets, of atom mappings. Together, a set of atom mappings for a chemical reaction specifies key aspects of the reaction mechanism, e.g., chemical bond change, breakage, and formation. The VMH (http://vmh.life) provides atom mappings for 9,610 reactions in Recon3D[66] and chemical structures for 4,831 metabolites from Recon3D[66] and the human gut microbiota[38] combined. Metabolite structures are provided in canonically ordered MOL and InChI formats. Atom mapping data are provided in RXN format. This explicit representation of metabolite and reaction structure offers the possibility of a broader range of biological, biomedical, and biotechnological applications than with stoichiometry alone.

Use chemoinformatic software tools, manually interrogate metabolic databases, or manually draw structures of metabolites to obtain chemical structures for each metabolite.

Suitable cheminformatics software tools[110] are useful if you want to automatically obtain metabolite identifiers in metabolic network reconstructions and download the corresponding structure from a database.

Databases such as VMH (http://vmh.life), PubChem[112], KEGG[113], ChEBI[114], LMSD[115], BioPath database[116], ChemSpider database[117], and HMDB[118] can be interrogated manually to provide chemical structures for metabolites in a network.

Based on chemical knowledge, one can also manually draw structures of metabolites using tools such as ChemDraw (PerkinElmer, http://www.perkinelmer.com/category/chemdraw).

75 Obtain an atom mapping for a metabolic reaction. To do this, the reaction stoichiometry and the chemical structures of the corresponding metabolites must be available. With this information, use software tools, manually interrogate metabolic databases, or manually draw the mappings.
- *Software tools*. A comparative study has been performed using Recon3D as a test case[111]. Due to its accuracy and availability, Reaction Decoder Tool (RDT[119]) is considered to be the most suitable algorithm to atom-map the reactions from a genome-scale metabolic network. Nevertheless, note that the Canonical Labelling for Clique Approximation (CLCA[120]) algorithm can map reactions

with explicit hydrogen atoms for fully protonated reactions while RDT can only atom-map reactions with implicit hydrogen atoms.

- *Manual database interrogation*. Databases such as BioPath[116] and KEGG RPAIR[121] disseminate manually curated atom mappings. VMH (http://vmh.life) also contains manually curated atom mappings for a subset of human metabolic reactions.
- *Manually draw atom mappings*. Based on chemical knowledge, one can draw atom mappings using tools such as ChemDraw (PerkinElmer, http://www.perkinelmer.com/category/chemdraw).

76 Given a model structure and the directory containing the chemical structure files (`molFileDir`) in MDL MOL file format, invoke RDT to atom-map a metabolic model using the following command:

```
>> balancedRxns = obtainAtomMappingsRDT(model, molFileDir, output-
Dir,... maxTime, standardiseRxn);
```

This function computes atom-mapping data for the balanced and unbalanced reactions in the metabolic network and saves them in the `outputDir` directory. The optional `maxTime` parameter sets a runtime limit for atom mapping of a reaction. If `standardiseRxn == 1`, then atom mappings are also canonicalized, which is necessary in order to obtain a consistent interoperable set of atom mappings for certain applications, e.g., computation of conserved moieties in Step 77. The output `balancedRxns` contains the balanced atom-mapped metabolic reactions.
? TROUBLESHOOTING

77 With a set of canonicalized atom mappings for a metabolic network, the set of linearly independent conserved moieties for a metabolic network can be identified[122]. Each of these conserved moieties corresponds to a molecular substructure (set of atoms in a subset of a molecule) whose structure remains invariant despite all the chemical transformations in a given network. A conserved moiety is a group of atoms that follow identical paths through metabolites in a metabolic network. Similarly to a vector in the (right) nullspace of a stoichiometric matrix that corresponds to a pathway (Step 52), a conserved moiety corresponds to a vector in the left nullspace of a stoichiometric matrix. Metabolic networks are hypergraphs[123], whereas most moiety subnetwork are graphs. Therefore, conserved moieties have both biochemical and mathematical significance and, once computed, can be used for a wide variety of applications. Given a metabolic network of exclusively mass-balanced reactions, one can identify conserved moieties by a graph theory analysis of its atom transition network[122].

First compute an atom transition network for a metabolic network using the following command:

```
>> ATN = buildAtomTransitionNetwork(model, rxnfileDir);
```

where `rxnfileDir` is a directory containing only atom-mapped files from balanced reactions, which can be obtained as explained in Step 76. The output `ATN` is a structure with several fields: `.A` is a $p \times q$ sparse incidence matrix for the atom transition network, where p is the number of atoms and q is the number of atom transitions, `.mets` is a $p \times 1$ cell array of metabolite identifiers to link each atom to its corresponding metabolites, `.rxns` is a $q \times 1$ cell array of reaction identifiers to link atom transitions to their corresponding reactions, and `.elements` is a $p \times 1$ cell array of element symbols for atoms in `.A`.

▲ CRITICAL STEP All the RXN files needed to compute the atom transition network must be in a canonical format. This can be achieved by setting `standardiseRxn = 1`.

78 To identify the conserved moieties in the metabolic network, invoke:

```
>> [L, ~, moietyFormulas] = identifyConservedMoieties(model, ATN);
```

where `L` represents the conserved moieties in the metabolic network. That is, `L` is an $m \times r$ matrix of r moiety vectors in the left null space of the stoichiometric matrix and `moietyFormulas` is an $mr \times 1$ cell array with a chemical formula corresponding to each of the computed conserved moieties.

Thermodynamically constraining a metabolic model ● Timing 1 s–17 min

79 In FBA of genome-scale stoichiometric models of metabolism, the principal constraints are uptake or secretion rates, the steady-state mass conservation assumption, and reaction directionality. The COBRA Toolbox extension *vonBertalanffy*[124] is a set of methods for integration of thermochemical data with constraint-based models[125–127] as well as application of thermodynamic laws to increase the physicochemical fidelity of constraint-based modeling predictions[128]. A full exposition of the

method for thermodynamically constraining a genome-scale metabolic model is beyond the scope of this protocol. Therefore, only several key steps are highlighted.

Given a set of experimentally derived `training_data` on standard transformed Gibbs energies of formation, a state-of-the art quantitative estimation of standard Gibbs energy of formation for metabolites with similar chemical substructures can be obtained using an implementation of the component contribution method[127]. We assume that the input `model` has been anatomically resolved as described in Steps 74–78. Access to a compendium of stoichiometrically consistent metabolite structures[110,122] is a prerequisite. When these are in place, invoke the component contribution method as follows:

```
>> model = componentContribution(model, training_data);
```

The `model.DfG0` field gives the estimated standard Gibbs energy of formation for each metabolite in the model with the `model.DfG0_Uncertainty` field expressing the uncertainty in these estimates, which is smaller for metabolites structurally related to metabolites in the training set. All thermodynamic estimates are given in units of kJ/mol.

80 Transform the standard Gibbs energy of formation for each metabolite according to the environment of each compartment of the model[126], i.e., the temperature, pH, ionic strength, and electrical potential specific to each compartment. Estimate the thermodynamic properties of reactions, given `model.concMin` and `model.concMax`, where one can supply lower and upper bounds on compartment-specific metabolite concentrations (mol/L), using the following command:

```
>> model = setupThermoModel(model, confidenceLevel);
```

In the output, field `.DfGt0` of `model` gives the estimated standard transformed Gibbs energy of formation for each metabolite and `.DrGt0` gives the estimated standard transformed Gibbs energy for each reaction. Subject to a `confidenceLevel` specified as an input, the upper and lower bounds on standard transformed Gibbs energy for each reaction are provided in `.DrGtMin` and `.DrGtMax`, respectively.

▲ **CRITICAL STEP** In a multi-compartmental model, this step must be done for an entire network at once in order to ensure that thermodynamic potential differences, arising from differences in the environment between compartments, are properly taken into account. See ref. [126] for a theoretical justification for this assertion.

81 Quantitatively assign reaction directionality based on the aforementioned thermodynamic estimates using the following command:

```
>> [modelThermo, directions]= thermoConstrainFluxBounds(model,
confidenceLevel,...
DrGt0_Uncertainty_Cutoff);
```

If `model.DrGtMax(j)< 0`, then the *j*th reaction is assigned to be forward, and if `model.DrGtMin(j)> 0` then the *j*th reaction is assigned to be reverse, unless the uncertainty in estimation of standard transformed reaction Gibbs energy exceeds a specified cutoff (`DrGt0_Uncertainty_Cutoff`). In this case, the qualitatively assigned reaction directionality, specified together by `model.lb(j)` and `model.ub(j)`, takes precedence. The `directions` output provides a set of Boolean vector fields that can be used to analyze the effect of qualitatively versus quantitatively assigning reaction directionality using thermochemical parameters.

82 Invoke thermodynamically constrained FBA by disallowing flux around stoichiometrically balanced cycles, also known as loops, using the `allowLoops` parameter to `optimizeCbModel` with the following command:

```
>> allowLoops = 0;
>> solution = optimizeCbModel(model, [], [], allowLoops);
```

The `solution` structure is the same as for FBA (see problem (1)), except that this `solution` satisfies additional constraints that ensure the predicted steady-state flux vector is thermodynamically feasible[129]. The solution satisfies energy conservation and the second law of thermodynamics[130].

Conversion of a flux balance model into a kinetic model ● Timing 1 s–17 min

83 Consider the conditions that need to be met in order to generate a kinetic model that is internally consistent. To analyze biochemical networks at genome scale, systems biologists often use a linear optimization technique called FBA. Linear approximation to known nonlinear biochemical reaction network function is sufficient to obtain biologically meaningful predictions in some situations. However, there are many biochemical processes for which a linear approximation is insufficient, which motivates the quest for developing variational kinetic modeling[131–133]. Certain conditions are required to be met in order to generate a kinetic model that is internally consistent. First, we describe those conditions, and then we demonstrate how to ensure that they are met. Consider a biochemical network with m molecular species and n reversible reactions. We define forward and reverse stoichiometric matrices, F, $R \in \mathbb{Z}_{+}^{m \times n}$, respectively, where F_{ij} denotes the stoichiometry of the i^{th} molecular species in the j^{th} forward reaction and R_{ij} denotes the stoichiometry of the i^{th} molecular species in the j^{th} reverse reaction. We assume that the network of reactions is stoichiometrically consistent[69], that is, that there exists at least one positive vector $l \in \mathbb{R}_{\geq 0}^{n}$ satisfying $(R - F)^{T} l = 0$. Equivalently, we require that each reaction conserves mass. The matrix $N := R - F$ represents net reaction stoichiometry and can be viewed as the incidence matrix of a directed hypergraph[123]. We assume that there are fewer molecular species than there are net reactions, that is $m < n$. We assume the cardinality of each row of F and R is at least 1, and the cardinality of each column of $R - F$ is at least 2. The matrices F and R are sparse, and the particular sparsity pattern depends on the particular biochemical network being modeled. Moreover, we assume that rank($[F, R]$) $= m$, which is a requirement for kinetic consistency[134].

Computation of a non-equilibrium kinetic steady state ● Timing 1 s–17 min

84 Let $c \in \mathbb{R}_{\geq 0}^{m}$ denote a variable vector of molecular species concentrations. Assuming constant strictly positive elementary kinetic parameters $k_f, k_r \in \mathbb{R}_{>}^{m}$, we assume elementary reaction kinetics for forward and reverse elementary reaction rates as $s(k_f, c) := \exp(\log(k_f) + F^{T}\log(c))$ and $r(k_r, c) := \exp(\log(k_r) + R^{T}\log(c))$, respectively, where $\exp(\cdot)$ and $\log(\cdot)$ denote the respective component-wise functions[134,135]. Then the deterministic dynamical equation for time evolution of molecular species concentration is given by

$$\begin{aligned} \frac{dc}{dt} &\equiv N\big(s(k_f, c) - r(k_r, c)\big) \\ &= N\big(\exp(\log(k_f) + F^{T}\log(c)) - \exp(\log(k_r) + R^{T}\log(c))\big) =: -f(c). \end{aligned} \tag{5}$$

A vector c^{*} is a steady state if and only if it satisfies $f(c^{*}) = 0$, leading to the nonlinear system of equations

$$f(x) = 0.$$

There are many algorithms that can handle this nonlinear system by minimizing a nonlinear least-squares problem; however, particular features of this mapping, such as sparsity of stoichiometric matrices F and R and non-unique local zeros of mapping f, motivate the quest to develop several algorithms for efficiently dealing with this nonlinear system. A particular class of such mappings, called duplomonotone mappings, was studied for biochemical networks[136], and three derivative-free algorithms for finding zeros of strongly duplomonotone mappings were introduced. Further, it was shown that the function $\|f(x)\|^2$ can be rewritten as a difference of two convex functions that is suitable for minimization with DC programming methods[135]. Therefore, a DC algorithm and its acceleration by adding a line-search technique were proposed for finding a stationary point of $\|f(x)\|^2$. Because the mapping f has locally non-unique solutions, it does not satisfy classic assumptions (e.g., nonsingularity of the Jacobian) for convergence theory. As a result, it was proven that the mapping satisfies the so-called Hölder metric subregularity assumption[137], and an adaptive Levenberg–Marquardt method was proposed to find a solution for this nonlinear system if the starting point is close enough to a solution. To guarantee the convergence of the Levenberg–Marquardt method with an arbitrary starting point, it is combined with globalization techniques such as line search or trust region, which leads to computationally efficient algorithms. Note that a stationary point of $\|f(x)\|^2$ may not correspond to a solution x, such that $f(x) = 0$, when $\nabla f(x) f(x) = 0$ does not imply $f(x) = 0$.

Compute a non-equilibrium kinetic steady state by running the function `optimizeVKmodel`. The mandatory inputs for computing steady states are a model `vKModel` containing F and R, the name of a solver to solve the nonlinear system, an initial point `x0`, and parameters for the considered solvers. For example, for specifying a solver, we write `solver = 'LMR';`. Optional parameters for the selected algorithm can be given to `optimizeVKmodel` by the `params`struct as follows:

```
>> params.MaxNumIter = 1000; params.adaptive = 1; params.kin = kin;
```

Otherwise, the selected algorithm will be run with the default parameters assigned for each algorithm. Run the function `optimizeVKmodel` by typing the following command:

```
>> output = optimizeVKmodel(vKModel, solver, x0, params);
```

The `output` struct contains information related to the execution of the solver.

Computation of a moiety-conserved non-equilibrium kinetic steady state

● Timing 1 s–17 min

85 Let us note that a vector c^* is a steady state of the biochemical system if and only if

$$s(k_f, c^*) - r(k_r, c^*) \in \mathcal{N}(N),$$

where $\mathcal{N}$ (N) denotes the null space of N. Therefore, the set of steady states $\Omega = \{c \in \mathbb{R}^m_{\geq 0} | f(c) = 0\}$ is unchanged if the matrix N is replaced by a matrix $\overline{N}$ with the same kernel. Suppose that $\overline{N} \in \mathbb{Z}^{r \times n}$ is the submatrix of N whose rows are linearly independent; then rank $(\overline{N})$ = rank(N) =:r. If one replaces N with $\overline{N}$ and transforms (5) to logarithmic scale, then, by letting $x := \log(c) \in \mathbb{R}^m$, k:= $[\log(k_f)^T, \log(k_r)^T]^T \in \mathbb{R}^{2n}$, the right-hand side of (5) is equal to the function

$$\overline{f}(x) := [\overline{N}, -\overline{N}] \exp\left(k + [F, R]^T x\right), \tag{6}$$

where [,] stands for the horizontal concatenation operator. Let $L \in \mathbb{R}^{m-r,m}$ denote a basis for the left nullspace of N, which implies $LN = 0$. We have rank$(L) = m - r$. We say that the system satisfies moiety conservation if for any initial concentration $c_0 \in \mathbb{R}^m_{\geq 0}$,

$$Lc = L\exp(x) = l_0,$$

where $l_0 \in \mathbb{R}^m_{\geq 0}$. It is possible to compute L such that each corresponds to a structurally identifiable conserved moiety in a biochemical network[122]. The problem of finding the moiety-conserved steady state of a biochemical reaction network is equivalent to solving the nonlinear system of equations

$$h(x) := \begin{pmatrix} \overline{f}(x) \\ L\ \exp(x) - l_0 \end{pmatrix} = 0. \tag{7}$$

Among algorithms mentioned in the previous section, the local and global Levenberg–Marquardt methods[137] are designed to compute either a solution for the nonlinear system (7) or a stationary point of the merit function $\frac{1}{2}\|h(x)\|^2$. To compute the conserved non-equilibrium kinetic steady state for a moiety, run the `optimizeVKmodel` function in the same way as in the previous section. Then pass a model `vKModel` containing F and R, L and l_0 to `optimizeVKmodel` together with the name of one of the Levenberg–Marquardt solvers.

Human metabolic network visualization with ReconMap ● Timing 1 s–2 min

86 The visualization of biochemical pathways is an important tool for biological interpretation of predictions generated by constraint-based models. It can be an invaluable aid for developing an understanding of the biological meaning implied by a prediction. Biochemical network maps permit the visual integration of model predictions with the underlying biochemical context. Patterns that are very difficult to discern in a vector can often be much more easily discerned by studying a generic map contextualized with model predictions. Genome-scale biochemical network visualization is particularly demanding. No currently available software satisfies all the requirements that might be desired for visualization of predictions from genome-scale models. Automatic layout of genome-scale biochemical networks is insufficiently developed to generate an aesthetically pleasing map, yet manual layout of such maps is very labor intensive and there is no global reference coordinate system for such maps, so each person might lay out a global map differently. Software applications for graph visualization are often not suited to displaying metabolic

hypergraphs[138]. Client-server software models have to trade off between highly interactive display of subsystem maps[139] and less interactive display of genome-scale maps[51]. An additional challenge with genome-scale models is that there is too much detail to visually process if an entire genome-scale map is visualized at once, necessitating the application of techniques to dimensionally reduce the presentation, e.g., semantic zooming[140]. With these caveats in mind, we present a method for genome-scale visualization of human metabolic network predictions using ReconMap 2.01 (ref. [50]), a manual layout of the reactions in the human metabolic reconstruction Recon 2.04 (ref. [17]), visualized with the Molecular Interaction Network Visualization (MINERVA[51]), a stand-alone web service built on the Google Maps (Google) application programming interface that enables low-latency web display and navigation of genome-scale molecular interaction networks.

Visualization of context-specific predictions in ReconMap via a web browser depends on access to a server running MINERVA, which requests user credentials for remote access. Public access to this server is provided free of charge.

To request user credentials, navigate with a web browser to http://vmh.life/#reconmap, select ADMIN (bottom left), and click on *Request an account* to send an email to the MINERVA team and subsequently receive your user credentials.

87 To prepare for remote access from within MATLAB, load the details of the MINERVA instance on the remote server, which are provided within the COBRA Toolbox during installation, and then add your user credentials to it:

```
>> load('minerva.mat');
>> minerva.login = 'username';
>> minerva.password = 'password'; minerva.map = 'ReconMap-2.01';
```

88 Load a human metabolic model into MATLAB with the following command:

```
>> model = readCbModel('Recon2.v04.mat');
```

89 Change the objective function to maximize ATP production through complex V (ATP synthase, `'ATPS4m'`) in the electron transport chain with the following command:

```
>> modelATP = changeObjective(model, 'ATPS4m');
```

90 Although the optimal objective value of FBA problem (1) is unique, the optimal flux vector itself is most likely not. When visualizing a flux vector, it is important that a unique solution to some optimization problem be displayed. For example, we can predict a unique network flux by regularizing the FBA problem (1) by redefining $\rho(v) := c^T v - \sigma/2 \times v^T v$ and $\sigma = 10^{-6}$ (Step 21). To obtain a unique optimal flux vector, run the following command:

```
>> solution = optimizeCbModel(modelATP, 'max', 1e-6);
```

91 Build the context-specific overlay of a flux vector on ReconMap by instructing the COBRA Toolbox to communicate with the remote MINERVA server using the following command:

```
>> identifier = 'your_overlay_title';
>> response = buildFluxDistLayout(minerva, model, solution, identifier);
```

The only new input variable is the text string in the `identifier` that enables you to name each overlay according to a unique title. The `response` status will be set to 1 if the overlay was successfully received by the MINERVA server.

? TROUBLESHOOTING

92 Visualize context-specific ReconMaps using a web browser. Navigate to http://vmh.life/#reconmap, log in with your credentials above, and then select 'OVERLAYS'. The list of USER-PROVIDED OVERLAYS will appear. To see the map from Step 91, check the box adjacent to the unique text string provided by `identifier`.

93 To export context-specific ReconMaps as publishable graphics, at least two options are available: portable document format (.pdf) or portable network format (.png). The former is useful for external editing, whereas the latter essentially produces a snapshot of the visual part of the map.

(A) **PDF export**

(i) Zoom out until the entire map is visible. Right-click on the map, select *Export as image* > *PDF*. A file named *model.pdf* will be downloaded to the default directory of the browser. This PDF is a scalable network graphic, so one can optionally use a PDF editor to zoom in or crop the PDF as desired.

(B) **PNG export**

(i) Navigate and zoom until the desired region of the map is visible. Right-click on the map, select *Export as image* > *PNG*, and a file named *model.png* will be downloaded to the default directory of the browser.

Variable scope visualization of a network with Paint4Net ● Timing 1 s–17 min

94 During model validation or optimization, visualization of a small-scale fragment of the network area of interest is often sufficient and is especially convenient during network reconstruction when a manual layout may not yet be available. Automatic generation of a hypergraph layout for a chosen subset of the network can be achieved with the COBRA Toolbox extension Paint4Net[141]. A subset of a network can be visualized, and the directionality and the fluxes for selected reactions can be shown. Details on each reaction (ID, name and synonyms, and formula) and metabolite (ID, name and synonyms, and charged formula) pop up when a cursor is placed over the corresponding item.

First compute a flux vector, e.g., with FBA, using the following command:

```
>> FBAsolution = optimizeCbModel(model);
```

95 Visualize a selected network fragment around a list of reactions in a `model`, contextualized using a flux vector `flux`, by running the following commands:

```
>> flux = FBAsolution.v;
>> involvedMets = draw_by_rxn(model, rxns, drawMap, direction,
initialMet, excludeMets, flux);
```

The `rxns` input provides a selection of reactions of interest. The remaining inputs are optional and control the appearance of the automatic layout. For example, `excludeMets` provides a list of metabolites that can be excluded from the network visualization, e.g., cofactors such as NAD and NADP.

96 To visualize a model fragment with a specified radius around a specified metabolite of interest, such as `'etoh[c]'`, run the following command:

```
>> metAbbr = {'etoh[c]'};
>> [involvedRxns, involvedMets] = draw_by_met(model, metAbbr,
'true', 1, 'struc', {''}, flux);
```

Contributing to the COBRA Toolbox with MATLAB.devTools ● Timing 1–30 s

97 A comprehensive code base such as the COBRA Toolbox evolves constantly. The open-source community is very active, and collaborators submit their contributions frequently. The more a new feature or bug fix is interlinked with existing functions, the higher the risk of a new addition instantly breaking code that is heavily used on a daily basis. In order to decrease this risk, a continuous-integration setup interlinked with the version-control system *git* has been set up. A *git*-tracked repository is essentially a folder with code or other documents of which all incremental changes are tracked by date and user. Any incremental changes to the code are called commits. The main advantage of *git* over other version-control systems is the availability of branches. In simple terms, a branch contains a sequence of incremental changes to the code. A branch is also commonly referred to as a feature. Consequently, a contribution generally consists of several commits on a branch.

To make changes to the openCOBRA repository, or, in other words, contribute, you must obtain your own personal copy first. You must register on the GitHub website (https://github.com) in order to obtain a *username*. First, click on the *FORK* button at the top right corner of the official

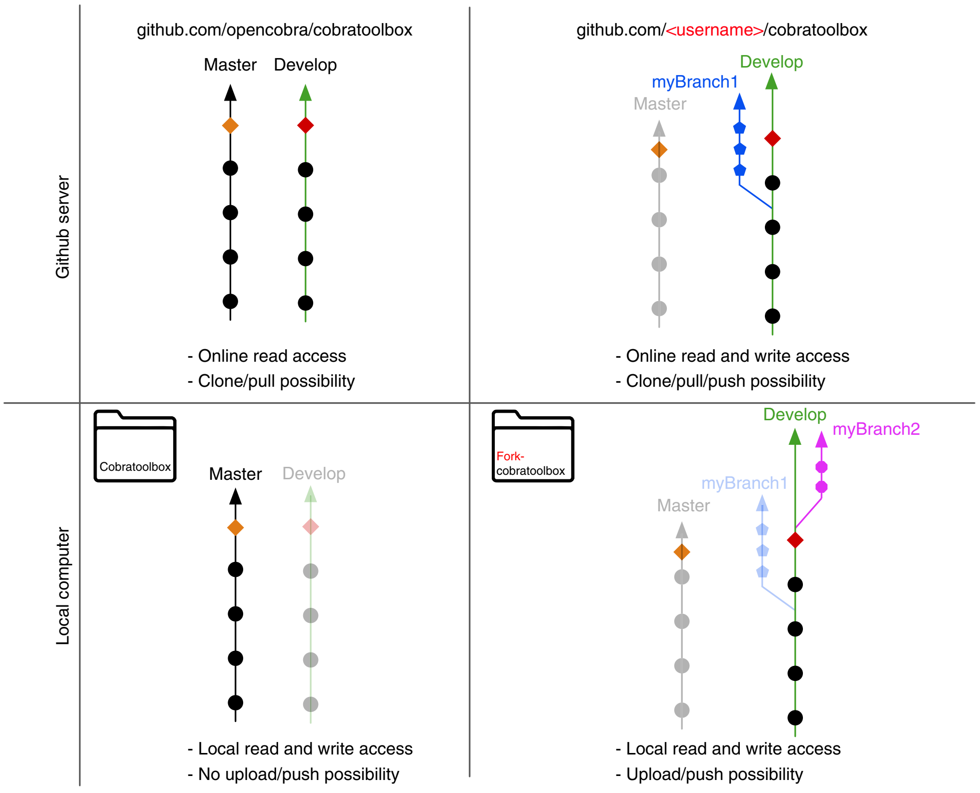

Github server
Local computer
github.com/opencobra/cobratoolbox
Master
Develop
- Online read access
- Clone/pull possibility
github.com/<username>/cobratoolbox
Develop
myBranch1
Master
- Online read and write access
- Clone/pull/push possibility
Cobratoolbox
Master
Develop
- Local read and write access
- No upload/push possibility
Fork-cobratoolbox
Develop
myBranch2
myBranch1
Master
- Local read and write access
- Upload/push possibility

Fig. 6 | Development branching model of the COBRA Toolbox. The openCOBRA repository and the fork of a contributor located on the GitHub server can be cloned to the local computer as *cobratoolbox* and *fork-cobratoolbox* folders, respectively. Each repository might contain different branches, but each repository contains the *master* and *develop* branches. Note that contributors only have read access to the openCOBRA repository. The stable branch is the *master* branch (black branch), and the development of code is done on the *develop* branch (green branch). The *master* branch will be checked out when using the *cobratoolbox* repository, whereas contributors can create new branches originating from the *develop* branch (local *fork-cobratoolbox* directory and online *<username>/cobratoolbox* repository). In the present example, *myBranch1* (blue branch) has already been pushed to the forked repository on the GitHub server, whereas *myBranch2* (pink branch) is present only locally. The branch *myBranch1* can be merged into the *develop* branch of the openCOBRA repository by opening a pull request. To submit the contributions (commits) on *myBranch2*, the contributor must first push the commits to the forked repository (https://github.com/<username>/cobratoolbox) before opening a pull request. Any commit made on the *develop* branch (red square) will be merged with the *master* branch if the *develop* branch is stable overall (orange square).

openCOBRA repository website (https://github.com/opencobra/cobratoolbox) in order to create a personal copy (or *fork*) with write and read access of the openCOBRA repository. This copy is accessible under https://github.com/<username>/cobratoolbox. These branches can be accessed by following procedure [2] in Step 99.

Contributing to the COBRA Toolbox is straightforward. As a contributor to the COBRA Toolbox is probably more familiar with MATLAB than with the internal mechanics of *git*, the MATLAB.devTools (https://github.com/opencobra/MATLAB.devTools) have been developed specifically to assist users in contributing to a *git*-tracked repository located on the GitHub server. In Fig. 6, an overview of the two online repositories, as well as their local copies, is given.

After the official openCOBRA version of the COBRA Toolbox has been installed, install the MATLAB.devTools from within MATLAB with the following command.

```
>> installDevTools
```

With this command, the directory *MATLAB.devTools* is created next to the *cobratoolbox* installation directory. The MATLAB.devTools can also be installed from the terminal (or shell) with the following command:

```
$ git clone git@github.com:opencobra/MATLAB.devTools
```

After initialization of the MATLAB.devTools, the user and developer may have two folders: a *cobratoolbox* folder with the stable *master* branch checked out, and a *fork-cobratoolbox* folder with the *develop* branch checked out. Detailed instructions for troubleshooting and/or contributing to the COBRA Toolbox using the terminal (or shell) are provided in Supplementary Manual 3.

▲CRITICAL STEP A working Internet connection is required, and *git* and *curl* must be installed. Installation instructions are provided on the main repository page of the MATLAB.devTools. A valid passphrase-less SSH key must be set in the GitHub account settings in order to contribute without entering a password while securely communicating with the GitHub server.

? TROUBLESHOOTING

98 The MATLAB.devTools are configured on the fly or whenever the configuration details are not present. The first time a user runs `contribute`, the personal repository (fork) is downloaded (cloned) into a new folder named *fork-cobratoolbox* at the location specified by the user. In this local folder, both *master* and *develop* branches exist, but it is the *develop* branch that is automatically selected (checked out). Any new contributions are derived from the *develop* branch.

Initializing a contribution using the MATLAB.devTools is straightforward. In MATLAB, type the following command:

```
>> contribute % then select procedure [1]
```

If the MATLAB.devTools are already configured, procedure [1] updates the fork (if necessary) and initializes a new branch with a name requested during the process. Once the contribution is initialized, files can be added, modified or deleted in the *fork-cobratoolbox* folder. A contribution is successfully initialized when the user is presented with a brief summary of configuration details. Instructions on how to proceed are also provided.

▲CRITICAL STEP The location of the fork must be specified as the root directory. There will be a warning issued if the path already contains another git-tracked repository.

? TROUBLESHOOTING

99 An existing contribution can be continued after a while. This step is particularly important in order to retrieve all changes that have been made to the openCOBRA repository in the meantime. Retrieve the changes using:

```
>> contribute % then select procedure [2]
```

Procedure [2] pulls all changes from the openCOBRA repository and rebases the existing contribution. In other words, existing commits are shifted forward and placed after all commits made on the *develop* branch of the openCOBRA repository.

▲CRITICAL STEP Before attempting to continue working on an existing feature, make sure that you published your commits as explained in Step 100.

? TROUBLESHOOTING

100 Publishing a contribution means uploading the incremental code changes to your fork; that is, your changes are then available to the public via your online fork of the COBRA Toolbox but are not yet available in the openCOBRA version of the COBRA Toolbox. A contribution will be accepted in the official repository only after a pull request has been submitted. It is not necessary to open a pull request if you want to simply upload your contribution to your fork. Submit a pull request using the following command:

```
>> contribute % then select procedure [3]
```

When running procedure [3], choose between 'simple contribution' and 'publishing and opening a pull request'.

- *Simple contribution without opening a pull request*. All changes to the code are individually listed and the user is asked explicitly which changes should be added to the commit. Once all changes have been added, a commit message must be entered. Upon confirmation, the changes are pushed to the online fork automatically.
- *Publishing and opening a pull request*. The procedure for submitting a pull request is the same as for the simple contribution, with the difference that when selecting to open a pull request, a link is

provided that leads to a pre-configured website according to the contributing guidelines. The pull request is then one click away.

▲ **CRITICAL STEP** After following procedures [1] and [2], all changes should be published using procedure [3] before stopping work on that contribution. When following procedure [3], the incremental changes are uploaded to the remote server. We advise publishing often and making small, incremental changes to the code. There is no need for opening a pull request immediately if there are more changes to be made. A pull request can be opened at any time, even manually and directly from the GitHub website. Unless the pull request is accepted and merged, the changes submitted are not available on the *develop* or *master* branches of the openCOBRA version of the COBRA Toolbox.

? TROUBLESHOOTING

101 If a contribution has been merged into the *develop* branch of the openCOBRA repository (accepted pull request), the contribution (feature or branch) can be safely deleted both locally and remotely on the fork by running `contribute` and selecting procedure [4]. Note that deleting a contribution deletes all the changes that have been made on that feature (branch). It is not possible to selectively delete a commit using the MATLAB.devTools. Instead, create a new branch by following procedure [1] (Step 98), and follow the instructions to 'cherry-pick' in the Supplementary Manual 3.

▲ **CRITICAL STEP** Make sure that your changes are either merged or saved locally if you need them. Once procedure [4] is concluded, all changes on the deleted branch are removed, both locally and remotely. No commits can be recovered.

? TROUBLESHOOTING

102 It is sometimes useful to simply update the fork without starting a new contribution. The local fork can be updated using procedure [5] of the *contribute* menu.

```
>> contribute % then select procedure [5]
```

▲ **CRITICAL STEP** Before updating your fork, make sure that no changes are present in the local directory *fork-cobratoolbox*. You can do so by typing the following command:

```
>> checkStatus
```

If there are changes listed, publish them by selecting procedure [3] of the *contribute* menu as explained in Step 100.

? TROUBLESHOOTING

Engaging with the COBRA Toolbox forum ● Timing 1 s–2 min

103 Visit the Frequently Asked Questions (FAQ) and the public forum associated with the COBRA Toolbox. The FAQ section of the documentation (https://opencobra.github.io/cobratoolbox/docs/FAQ.html) is a good starting point to find answers to commonly posed questions one may face. The public forum associated with the COBRA Toolbox, available at https://groups.google.com/forum/#!forum/cobra-toolbox, is a great way to search for solutions to previously recognized problems that may be similar to problems novel to the user. This is especially so with respect to recent installation and configuration issues that have arisen because of asynchronous development of the many software packages integrated with the COBRA Toolbox.

Questions posted in the forum are welcome, provided that some simple guidelines are followed:

- Before posting a question, apply for an application for membership at https://groups.google.com/forum/#!forum/cobra-toolbox; this is required to eliminate spam.
- Make the question as detailed as possible to increase the probability of a rapid and helpful reply.
- Append your message with the result of running `generateSystemConfigReport` so that repository maintainers are aware of the system configuration. That is often the first question that comes to mind when considering a response.

Users are also encouraged to reply to questions in the online COBRA Toolbox forum. Community contributions are welcomed to help users overcome any issues they face and are noticed by existing COBRA community members. Generally, responses to questions can be expected within 1–2 d of posting, provided that the posting guidelines are followed.

Troubleshooting

Troubleshooting advice can be found in Table 5.

Table 5 | Troubleshooting table

Step	Problem	Possible reason	Solution
1	The `initCobraToolbox` function displays warnings or error messages	Incompatible third-party software or an improperly configured system	First, read the output of the initialization script in the command window. Any warning or error messages, although often brief, may point toward the source of the problem during initialization if read literally. Second, verify that all software versions are supported and have been correctly installed as described in the 'Materials' section. Third, ensure that you are using the latest version of the COBRA Toolbox, cf. Steps 97–102. Fourth, verify and test the COBRA Toolbox as described in Step 3. Finally, if nothing else works, consult the COBRA Toolbox forum, as described in Step 103
3	Some tests are listed as failed when running `testAll`	Some third-party dependencies are not properly installed or the system is improperly configured	Verify that all required software has been correctly installed as described in the 'Materials' section. The specific test can then be run individually to determine the exact cause of the error. If the error can be fixed, try to use the MATLAB. devTools and contribute a fix. Further details on how to approach submitting a contribution are given in Steps 97–102. If the error cannot be determined, reach out to the community as explained in Step 103
4	The `readCbModelfunction` fails to import a model	The input file is not correctly formatted or the SBML file format is not supported	Specifications for Excel sheets accepted by the COBRA Toolbox can be found on GitHub (http://opencobra.github.io/cobratoolbox/docs/COBRAModelFields.html). An Excel template is available at https://github.com/opencobra/cobratoolbox/blob/master/docs/source/notes/COBRA_structure_fields.xlsx. Files with legacy SBML formats can be imported, but some information from the SBML file might be lost. In addition to constraint-based information encoded by fields of the fbc package, the COBRA-style annotations introduced in the COBRA Toolbox v.2.0 (ref. [4]) are supported for backward compatibility. Some information is still stored in this type of annotation. The data specified with the latest version of the fbc package is used in preference to other fields, e.g., legacy COBRA-style notes, which may contain similar data
	The `readCbModel` function fails to import a model saved as a .mat file	The model may contain deprecated fields or fields that have invalid values	Old MATLAB models saved as .mat files sometimes contain deprecated fields or fields that have invalid values. Some of these instances are checked and corrected when `readCbModel` is run, but there might be instances when `readCbModel` fails. If this happens, it is advisable, to load the .mat file, run the `verifyModel` function on the loaded model, and manually adjust all indicated inconsistent fields. After this procedure, we suggest saving the model again and using `readCbModel` to load the model
	The `readCbModel` function fails to import an SBML file	The model might be invalid	If an SBML file produces an error during input-output, check that the file is valid SBML by using the SBMLValidator (http://sbml.org/Facilities/Validator)
5	The `writeCbModel` function fails to export a model	Some of the required fields of the `model` structure are missing or the model contains invalid data	Before a reconstruction or model is exported, a summary of invalid data in the model can be obtained by running `verifyModel(model)`. A list of required fields for the model structure is presented in Table 3
15	The `dqqMinos` or `quadMinos` interfaces are not working as intended	The binaries might not be compatible with your operating system	Make sure that all relevant system requirements described in the 'Materials' section are satisfied. If you are still unable to use the respective interfaces, reach out to the community as explained in Step 103
16(A)	The `findSExRxnInd` function fails to identify some exchange, demand and sink reactions	Some exchange, demand and sink reactions do not start with any of anticipated prefixes	Try an alternative approach, e.g., `findStoichConsistentSubset`, which finds the subset of a stoichiometric matrix that is stoichiometrically consistent
16(B)	The function `checkMassChargeBalance` returns incorrect results	Some formulae are missing or a formula is incorrectly specified, leading one or more reactions to be incorrectly identified as being elementally balanced	Try an alternative approach, e.g., `findStoichConsistentSubset`, which finds the subset of a stoichiometric matrix that is stoichiometrically consistent
16(C)	Erroneous predictions	Inadvertent violation of the steady-state mass conservation constraint	Manually inspect the reaction formulae for each reaction to identify any obviously mass-imbalanced reactions; omit them from the reconstruction and run `findStoichConsistentSubset` again
22			

Table continued

Table 5 (continued)

Step	Problem	Possible reason	Solution
	The solution status given by `FBAsolution.stat` is −1	A too-short runtime limit has been set or numerical issues occurred during the optimization procedure	Check the value of `FBAsolution.origStat` and compare it with the documentation provided by the solver in use for further information. If one is using a double-precision solver to solve a model that could be multi-scale but is not yet recognized as such, then `FBAsolution.stat == -1` can be symptomatic of this situation. In that case, refer to Steps 14 and 15 to learn how to numerically characterize a reconstruction model
55	The sampling distribution is not uniform (revealed by a non-uniform marginal flux distribution)	The values of the sampling parameters `options.nSkip` and `options.nSamples` are set too low	Increase the values of the `options.nSkip` and `options.nSamples` parameters until smooth and unimodal marginal flux distributions are obtained
68	No reaction is found in the MUST sets	The wild-type or mutant strain may not be sufficiently constrained	A solution is to add more constraints to the strains until differences in the reaction ranges are shown. If no differences are found, another algorithm might be better suited. If there is an error when running the `findMust*` functions, a possible reason is that the inputs are not well defined or a solver may not be set. Verify the inputs and use `changeCobraSolver` to change to a commercial-grade optimization solver (see Table 4 for a list of supported solvers)
76	Some reactions could not be mapped	Too-short runtime limit or a reaction that the algorithm could not atom-map	Increase the runtime limit of the algorithm
91	The remote MINERVA server refuses to build a new overlay	The text string in the identifier input variable is not uniquely defined in your account	Change the identifier text string of your overlay
97	An error message claims permission is denied	The SSH key of the computer is not configured properly	The installation of the MATLAB.devTools is dependent on a correctly configured GitHub account. The SSH key of the computer must be set in the GitHub account settings or else errors will be thrown. If the *git clone* command works, the SSH key is properly set. In that case, delete the SSH key locally (generally located in the *ssh* folder in the home directory) and remotely on GitHub, and generate a new SSH key
98	When running `contribute` an error message is generated claiming that the fork cannot be reached or that the local fork cannot be found	The local forked folder cannot be found or has been moved, or the remote fork cannot be reached	It may occur that the configuration of the MATLAB.devTools is faulty or has been mistyped. In that case, try to reset the configuration by typing: >> `resetDevTools`
	Procedure [1] fails when running `contribute`	The local *fork-cobratoolbox* folder is too old or has not been updated for a while	In that case and if no local changes are present, back up and remove the local *fork-cobratoolbox* folder and run the `contribute` command again. Alternatively, try to delete the forked repository online and re-fork the openCOBRA repository. When one is sure that everything is fine, the backup can be safely deleted, but it is wise to store it for some time, in case later one realizes that some updates to the code have gone missing
		There are changes in the local *fork-cobratoolbox* folder	Contribute the changes manually as described in Supplementary Manual 3
		The forked repository cannot be reached online or the SSH key is not configured properly	Set the SSH key in your GitHub account and make sure that the forked repository can be reached. This can easily be checked by re-cloning the MATLAB.devTools in the terminal as explained in Step 97 and by browsing to the forked repository online
99	Procedure [2] fails when running `contribute`	Your contribution has been deleted online or is no longer available locally	When the rebase process fails, the user is asked to reset the contribution, which will reset the contribution to the online version of the branch in the fork. In general, when the rebase fails, there have been changes made on the openCOBRA repository that are in conflict with the local changes. You can check the status of the local repository by typing: >> `checkStatus`. If there are conflicts that you do not know how to resolve, check the official repository or ping the developers at https://groups.google.com/forum/#!forum/cobra-toolbox as explained in Step 103. If you already have published changes, try to submit a pull request as explained in Step 100 to help the developers to understand the situation. Alternatively, you can try to resolve the conflicts manually. More information on how to solve conflicts is given as Supplementary Manual 3
100	Procedure [3] fails when running `contribute`	The forked repository cannot be reached online or the SSH key is not configured properly	Check to make sure the SSH key in your GitHub account is set properly and make sure that the forked repository can be reached
	When opening a pull request, GitHub cannot automatically merge	There have been changes made on the openCOBRA repository and on your local fork	Submit the pull request anyway; another developer will help you rebase your contribution manually

Table continued

Table 5 (continued)

Step	Problem	Possible reason	Solution
101	Procedure [4] fails when running `contribute`	Your local changes are not yet published (committed)	Follow procedure [3] of the *contribute* menu in order to publish your changes first as explained in Step 100
102	Procedure [5] fails when running `contribute`	There are some local changes that have not yet been published (committed)	Back up eventual modifications, remove the *fork-cobratoolbox folder*, and run the `contribute` command again
		Too many changes have been made in the openCOBRA repository	Back up your modified files to a separate location, and reset your branch manually by typing the following into the terminal (be careful; this will delete all your changes locally, but not remotely): `$ git reset --hard origin/<yourBranch>` Then copy your file back into the *fork-cobratoolbox* folder and `contribute` normally

Timing

Steps 1 and 2, initialization of the COBRA Toolbox: 5–30 s
Step 3, verification and testing of the COBRA Toolbox: ~17 min
Step 4, importation of a reconstruction or model: 10 s–2 min
Step 5, exportation of a reconstruction or model: 10 s–2 min
Step 6, use of *rBioNet* to add reactions to a reconstruction: 1 s–17 min
Steps 7 and 8, use of a spreadsheet to add reactions to a reconstruction: 1 s–17 min
Steps 9–13, use of scripts with reconstruction functions: 1 s–2 min
Step 14, checking the scaling of a reconstruction: 1 s–2 min
Step 15, selection of a double- or quadruple-precision optimization solver: 1–5 s
Step 16, identification of stoichiometrically consistent and inconsistent reactions: 1 s–28 h
Step 17, identification of stoichiometrically consistent and inconsistent molecular species: 1 s–17 min
Step 18, setting of simulation constraints: 1 s–17 min
Step 19, identification of molecular species that leak, or siphon, across the boundary of the model: 1–17 min
Step 20, identification of flux-inconsistent reactions: 1 s–17 min
Steps 21 and 22, flux balance analysis: 1 s–2 min
Step 23, relaxed flux balance analysis: 1 s–17 min
Step 24, sparse flux balance analysis: 1 s–17 min
Steps 25–27, identification of dead-end metabolites and blocked reactions: ~2 min
Steps 28–30, gap-filling a metabolic network: 2 min–28 h
Steps 31–33, integration of extracellular metabolomic data: 17 min–28 h
Steps 34–39, integration of intracellular metabolomic data: 2 min–2.8 h
Step 40, integration of transcriptomic and proteomic data: 2 min–2.8 h
Steps 41–46, adding biological constraints to a flux balance model: ~2 min
Steps 47 and 48, qualitative chemical and biochemical fidelity testing:2–17 min
Steps 49–51, quantitative biochemical fidelity testing: 2–7 min
Step 52, MinSpan pathways: a sparse basis of the nullspace of a stoichiometric matrix: 2 min–2.8 h
Step 53, low-dimensional flux variability analysis: 1 s–17 min
Step 54, high-dimensional flux variability analysis: 1 s–28 h
Step 55, uniform sampling of steady-state fluxes: 1 s–17 min
Steps 56–73, identification of all genetic manipulations leading to targeted overproductions: 10 s–28 h
Steps 74–78, atomically resolving a metabolic reconstruction: 10 s–28 h
Steps 79–82, thermodynamically constraining a metabolic model: 1 s–17 min
Step 83, conversion of a flux balance model into a kinetic model: 1 s–17 min
Step 84, computation of a non-equilibrium kinetic steady state: 1 s–17 min
Step 85, computation of a moiety-conserved non-equilibrium kinetic steady state: 1 s–17 min
Steps 86–93, human metabolic network visualization with ReconMap: 1 s–2 min
Steps 94–96, variable scope visualization of a network with Paint4Net: 1 s–17 min
Steps 97–102, contributing to the COBRA Toolbox with MATLAB.devTools: 1–30 s
Step 103, engaging with the COBRA Toolbox forum, 1 s–2 min

Anticipated results

Initialization of the COBRA Toolbox

Step 1

The initialization step automatically checks the configuration of all of the required and some of the optional software dependencies. During initialization, all *git* submodules are updated. The solver paths are set when available and compatible. A system-dependent table with the solver status is returned, together with solver suggestions as shown in Fig. 7. The user is also presented with options for updating the COBRA Toolbox.

```
 _____   _____   _____   _____     _____     |
/  ___| /  _  \ |  _  \ |  _  \   / ___ \    |   COnstraint-Based Reconstruction and Analysis
| |     | | | | | |_| | | |_| |  | |___| |   |   The COBRA Toolbox - 2017
| |     | | | | |  _  { |  _  /  |  ___  |   |
| |___  | |_| | | |_| | | | \ \  | |   | |   |   Documentation:
\_____| \_____/ |_____/ |_|  \_\ |_|   |_|   |   http://opencobra.github.io/cobratoolbox
                                             |

> Checking if git is installed ...  Done.
> Checking if the repository is tracked using git ...  Done.
> Checking if curl is installed ...  Done.
> Checking if remote can be reached ...  Done.
> Initializing and updating submodules ... Done.
> Adding all the files of The COBRA Toolbox ...  Done.
> Define CB map output... set to svg.
> Retrieving models ...   Done.
> TranslateSBML is installed and working properly.
> Configuring solver environment variables ...
- [*---] ILOG_CPLEX_PATH: /Users/syarra/Applications/IBM/ILOG/CPLEX_Studio1271/cplex/matlab/x86-64_osx
- [-*--] GUROBI_PATH: /Library/gurobi702/mac64/matlab
- [----] TOMLAB_PATH :  --> set this path manually after installing the solver ( see instructions )
- [----] MOSEK_PATH :  --> set this path manually after installing the solver ( see instructions )
Done.
> Checking available solvers and solver interfaces ... Done.
> Setting default solvers ... Done.
> Saving the MATLAB path ... Done.
- The MATLAB path was saved in the default location.

> Summary of available solvers and solver interface s

                 Support           LP     MILP      QP     MIQP     NLP
-------------------------------------------------------------------------
cplex_direct     full              0       0        0       0        -
dqqMinos         full              1       -        -       -        -
glpk             full              1       1        -       -        -
gurobi           full              1       1        1       1        -
ibm_cplex        full              1       1        1       -        -
matlab           full              1       -        -       -        1
mosek            full              0       0        0       -        -
pdco             full              1       -        1       -        -
quadMinos        full              1       -        -       -        1
tomlab_cplex     full              0       0        0       0        -
qpng             experimental      -       -        1       -        -
tomlab_snopt     experimental      -       -        -       -        0
gurobi_mex       legacy            0       0        0       0        -
lindo_old        legacy            0       -        -       -        -
lindo_legacy     legacy            0       -        -       -        -
lp_solve         legacy            1       -        -       -        -
opti             legacy            0       0        0       0        0
-------------------------------------------------------------------------
Total            -                 8       3        4       1        2

+ Legend: - = not applicable, 0 = solver not compatible or not installed, 1 = solver installed.


> You can solve LP problems using: 'dqqMinos' - 'glpk' - 'gurobi' - 'ibm_cplex' - 'matlab' - 'pdco' - 'quadMinos' - 'lp_solve'
> You can solve MILP problems using: 'glpk' - 'gurobi' - 'ibm_cplex'
> You can solve QP problems using: 'gurobi' - 'ibm_cplex' - 'pdco' - 'qpng'
> You can solve MIQP problems using: 'gurobi'
> You can solve NLP problems using: 'matlab' - 'quadMinos'

> Checking for available updates ...
> The COBRA Toolbox is up-to-date.
```

Fig. 7 | Output of initialization of the COBRA Toolbox with initCobraToolbox.

Step 2
A list of solvers assigned to solve each class of optimization solvers is returned:

```
Defined solvers are:
CBT_LP_SOLVER: gurobi
CBT_MILP_SOLVER: gurobi
CBT_QP_SOLVER: qpng
CBT_MIQP_SOLVER: gurobi
CBT_NLP_SOLVER: matlab
```

Step 3
The test suite starts by initializing the COBRA Toolbox and thereafter all the tests are run. At the end of the test run, a comprehensive summary table is presented in which the respective tests and their outcomes are shown. On a fully configured system that is compatible with the most recent version of the COBRA Toolbox, all tests should pass. It may not be necessary to have a fully configured system to use one's particular subset of methods.

Importation of a reconstruction or model

Step 4
The reconstruction or model is loaded into the MATLAB workspace within a structure named `model`, regardless of whether the `fileName` specified a reconstruction or model. The model structure should contain all the information in different fields. Table 3 provides an overview of the individual model fields and their content. Very large SBML models may take some time to load.

Step 5
A file is exported that contains the information from the model to the location and in the format specified by the `fileName` variable.

Checking the scaling of a reconstruction

Step 14
The `checkScaling` function returns a `precisionEstimate` string that is either `'double'`or `'quad'`. The scaling estimate is based on the order of magnitude of the ratio of the maximum and minimum row and column scaling coefficients, which are determined such that the scaled stoichiometric matrix has entries close to unity. In addition, a summary of scaling properties included in `scalingProperties` may be returned.

Selection of a double- or quadruple-precision optimization solver

Step 15
If the selected solver is installed, `solverStatus == 1` will be returned, meaning the solver interface to MATLAB is configured correctly and the solver is compatible with the system environment. If the `dqqMinos` solver has been selected and `solverStatus == 1`, then LP equation solutions are computed somewhat more slowly than with a double-precision solver, but with the advantage that solutions are computed with a feasibility and optimality tolerance of 10^{-15}, which becomes an advantage for a multi-scale model, for which the typical tolerance of 10^{-6} for a double-precision solver may be insufficient.

Identification of stoichiometrically consistent and inconsistent molecular species

Step 17
Any molecular species corresponding to a non-zero entry within `SConsistentMetBool` is always involved in mass-imbalanced reactions, indicating that the stoichiometry is mis-specified. Double-check the chemical formulae involved in the corresponding reactions to ensure that, e.g., the stoichiometry for protons and cofactors leads to balanced reactions.

Step 20
Any non-zero entry in `fluxInConsistentRxnBool` indicates a flux-inconsistent reaction, i.e., a reaction that does not admit a non-zero flux. Blocked reactions can be resolved by manual reconstruction[6], algorithmic reconstruction[82], or a combination of the two.

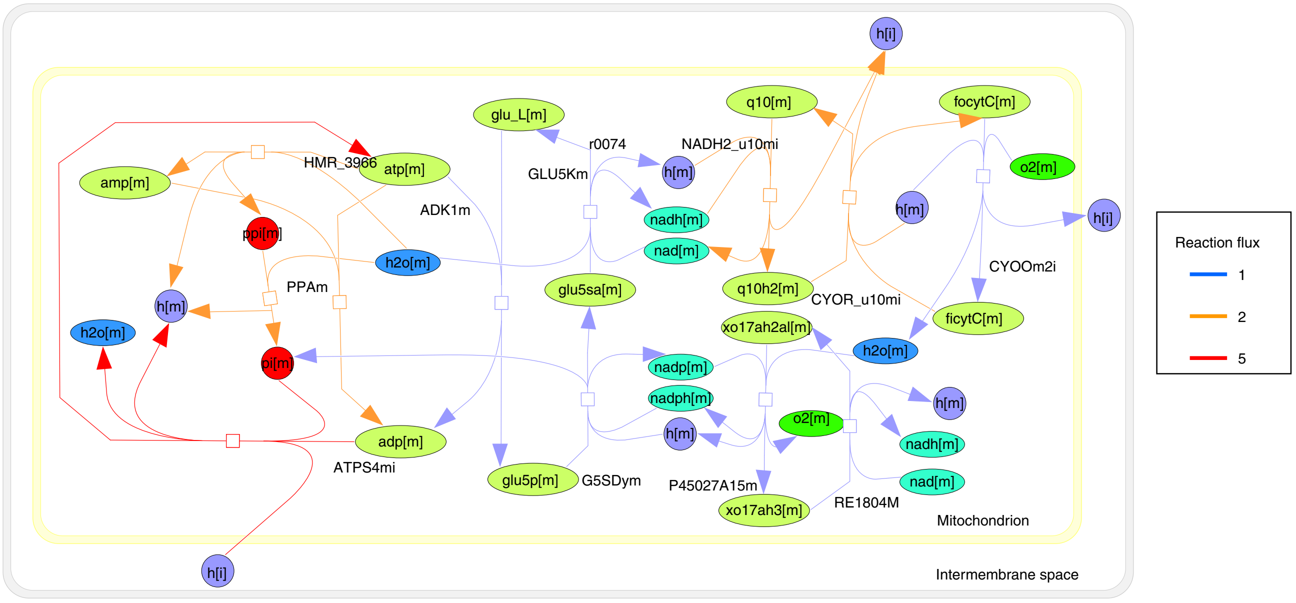


Fig. 8 | An energy-generating stoichiometrically balanced cycle. The smallest stoichiometrically balanced cycle that produces ATP at a maximal rate using the ATP synthase reaction, in Recon3D, with all internal reactions. Relative reaction flux magnitudes are indicated by the color of each directed hyperedge shown in the legend. All metabolite and reaction abbreviations are primary keys in the Virtual Metabolic Human database (https://vmh.life): reaction abbreviation, reaction name: ADK1m, adenylate kinase, mitochondrial; G5SDym, glutamate-5-semialdehyde dehydrogenase, mitochondrial; GLU5Km, glutamate 5-kinase, mitochondrial; P45027A15m, 5-beta-cytochrome P450, family 27, subfamily A, polypeptide 1; PPAm, inorganic diphosphatase; r0074, L-glutamate 5-semialdehyde:NAD+ oxidoreductase; HMR_3966, nucleoside-triphosphate giphosphatase; ATPS4mi, ATP synthase (four protons for one ATP); CYOR_u10mi, ubiquinol-6 cytochrome *c* reductase, complex III; NADH2_u10mi, NADH dehydrogenase, mitochondrial; CYOOm2i, cytochrome *c* oxidase, mitochondrial complex IV.

Sparse flux balance analysis

Step 24

There should be no such active stoichiometrically balanced cycle in a network if one is using bounds that are sufficiently constrained. Figure 8 illustrates the cycle obtained from Recon3D with all internal reaction bounds set to zero.

Integration of transcriptomic and proteomic data

Step 41

`createTissueSpecificModel` returns a COBRA model that is constrained by the context of the data provided to it. Usually this means enrichment of reactions with high expression and omission of reactions with low expression profiles. Each method returns a flux-consistent model; hence, it is likely that certain reactions, without experimental evidence, are added to the context-specific model in order to enable non-zero net flux through reactions for which supporting experimental evidence for activity exists.

Quantitative biochemical fidelity testing

Step 51

For the `Recon3Dmodel`, the anticipated yield is 32 ATP per unit of glucose, which compares favorably with the ATP yield of 31 ATP obtained from the biochemical literature.

Uniform sampling of steady-state fluxes

Step 55

The marginal flux distributions for each reaction should be smooth and uni-modal for any biochemical network with feasible set $\Omega := \{v \mid Sv = 0; l \leq v \leq u\}$.

Identification of all genetic manipulations leading to targeted overproductions

Step 71

The identified reaction is `SUCt`, i.e., a transporter for succinate (an intuitive solution). However, changing the parameters will enable `optForce` to find non-intuitive solutions.

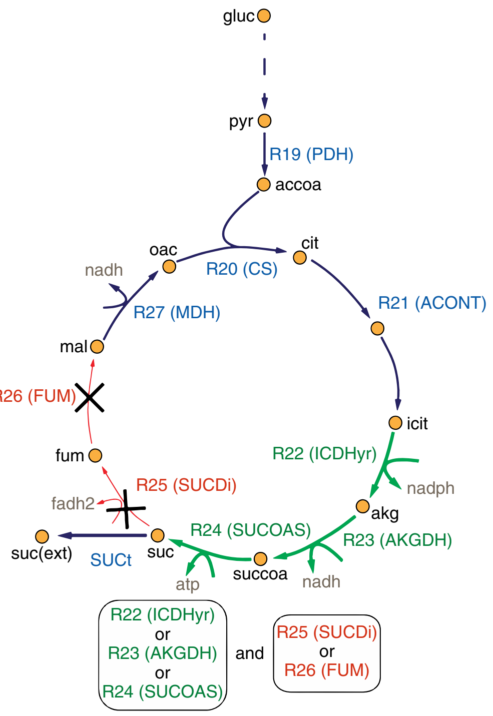


Fig. 9 | The interventions predicted by the OptForce method for succinate overproduction in *E. coli* (AntCore model) under aerobic conditions. Reactions that need to be upregulated (green arrows and labels) and knocked out (red arrows and labels) are shown in this simplified metabolic map. The strategies include upregulation of reactions generating succinate such as isocitrate dehydrogenase, α-ketoglutarate dehydrogenase, or succinyl-CoA synthetase, along with knockout of reactions draining succinate, such as those for succinate dehydrogenase or fumarate hydratase. Note that each of these reactions may associate with one or more genes in *E. coli*. R before a number denotes the index of the corresponding reaction in the stoichiometric martrix. Each metabolite and reaction is abbreviated: accoa, acetyl-CoA; ACONT, aconitase; akg, oxoglutaric acid; AKGDH, 2-oxoglutarate dehydrogenase; atp, adenosine triphosphate; cit, citric acid; CS, citrate synthase; fadh2, reduced form of flavin adenine dinucleotide; fum, fumaric acid; FUM, fumarase; ICDHyr, isocitrate dehydrogenase; icit, isocitric acid; gluc, glucose; mal, malate; MDH, malate dehydrogenase; nadh, reduced nicotinamide–adenine dinucleotide; oac, oxaloacetate; pyr, pyruvate; suc, succinate; SUCDi, succinate dehydrogenase (irreversible); suc(ext), succinate (extracellular); succoa, succinyl-CoA; SUCOAS, succinyl-CoA synthetase (ADP-forming); SUCt, succinate transport.

Identification of all genetic manipulations leading to targeted overproductions

Step 73

Figure 9 illustrates the interventions predicted by the `OptForce` method for succinate overproduction in the AntCore *E. coli* model under aerobic conditions.

Thermodynamically constraining a metabolic model

Step 82

Thermodynamically constrained flux predictions can differ markedly from those obtained with FBA. An open challenge is acquisition of sufficient thermochemical training data, as well as sufficient quantitative metabolomic data, such that estimates of transformed reaction Gibbs energies can be made with sufficiently low uncertainty to constrain reaction directionality with high confidence. The degree of confidence typically differs markedly between reactions. Therefore, a pragmatic approach is to rank-order reaction directionality assignments by the probability that the thermodynamically assigned reaction directionality is forward, reversible, or reverse (see Fig. 10 for an application to Recon3D).

Human metabolic network visualization with ReconMap

Step 93

Figure 11 shows an overlay of an optimal regularized FBA solution within ReconMap within a web browser window. The full image can be accessed as the Supplementary Data file.

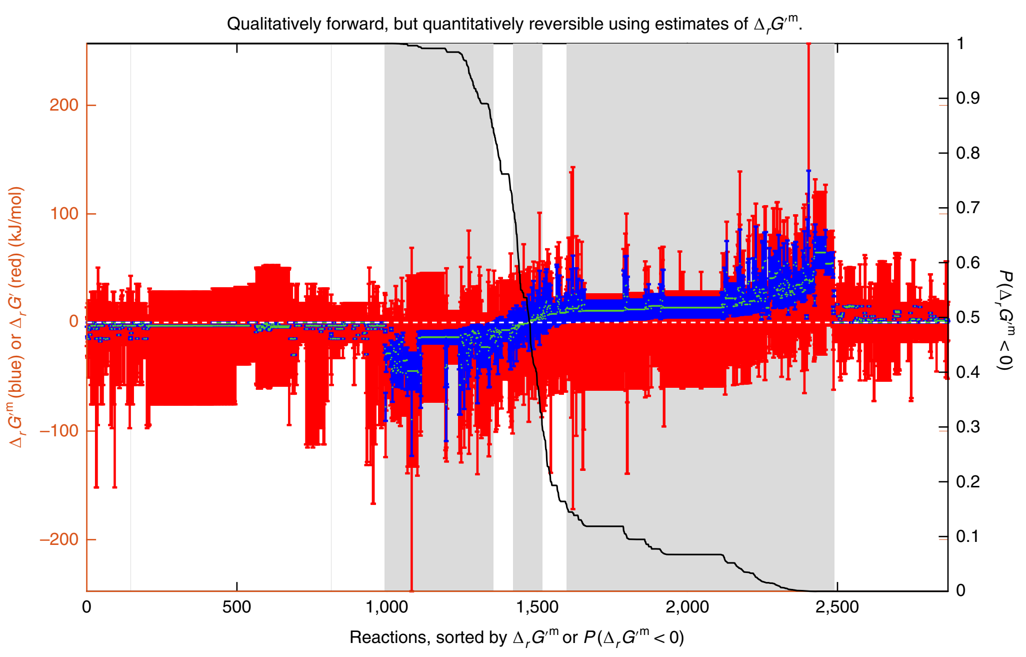


Fig. 10 | Qualitatively forward, quantitatively reverse reactions in a multi-compartmental, genome-scale model. In Recon3D, the transformed reaction Gibbs energy could be estimated for 7,215 reactions. Of these reactions, 2,868 reactions were qualitatively assigned to be forward in the reconstruction, but were quantitatively assigned to be reversible using subcellular compartment-specific thermodynamic parameters, the component contribution method, and broad bounds on metabolite concentrations (10^{-5}–0.02 mol/L), except for certain cofactors. The geometric mean (green) and feasible range (between maximum and minimum) of estimated millimolar standard transformed reaction Gibbs energy ($\Delta_r G'^m$, blue) and transformed reaction Gibbs energy ($\Delta_r G'$, red) are illustrated. The relative uncertainty in metabolite concentrations versus uncertainty in thermochemical estimates is reflected by the relative breadth of the red and blue bars for each reaction, respectively. The reactions are rank ordered by the cumulative probability that millimolar standard transformed reaction Gibbs energy is less than zero, $P(\Delta_r G'^m < 0)$, (black descending line from left to right). This assumes that all metabolites are at a millimolar concentration (1 mM) and a Gaussian error is assumed in component contribution estimates. In this ordering, forward transport reactions have $P(\Delta_r G'^m < 0) = 1$ (far left) and reverse transport reactions have $P(\Delta_r G'^m < 0) = 0$ (far right). In between, from left to right are biochemical reactions with decreasing cumulative probability of being forward in direction, subject to the stated assumptions. Alternative rankings are possible. The key point is to observe that the COBRA Toolbox is primed for quantitative integration of metabolomic data as the uncertainty in transformed reaction Gibbs energy associated with thermochemical estimates using the component contribution method is now substantially lower than the uncertainty associated with the assumption of broad concentration range.

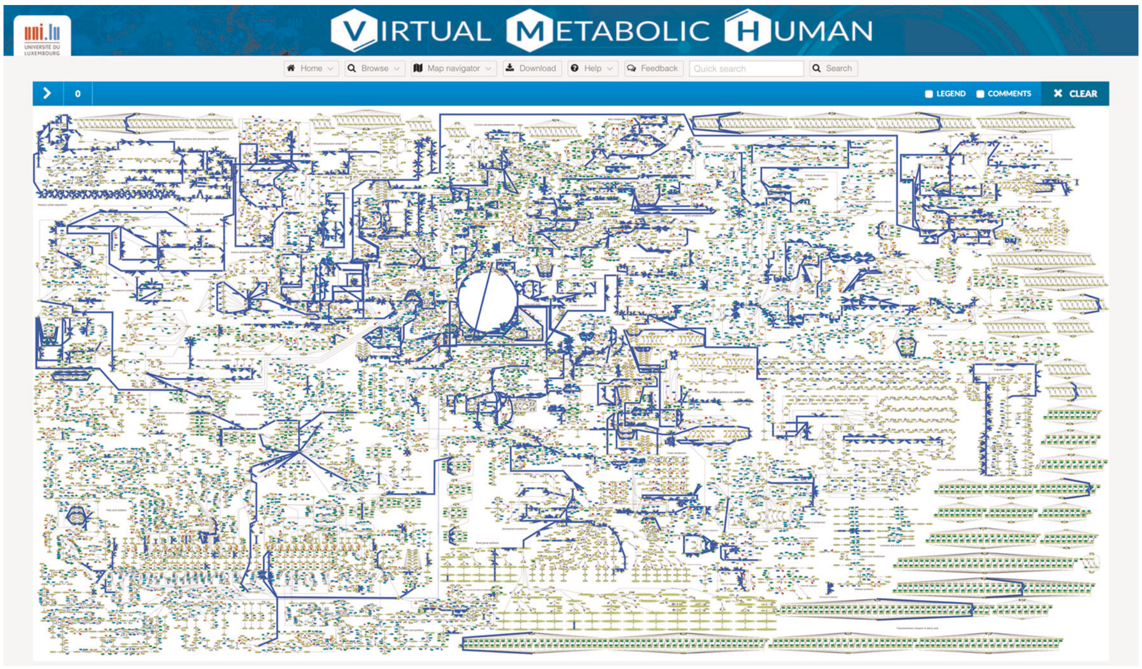


Fig. 11 | Human metabolic network visualization. Overlay of the flux vector for maximum ATP synthase flux, using flux balance analysis with regularization of the flux vector. Active fluxes are highlighted (blue). The full image can be accessed as the Supplementary Data file.

Variable scope visualization of a network with Paint4Net

Step 96

Figure 12 illustrates a fragment of a Paint4Net visualization contextualized using a flux vector to control the thickness and color of edges representing reactions. In the visualization,

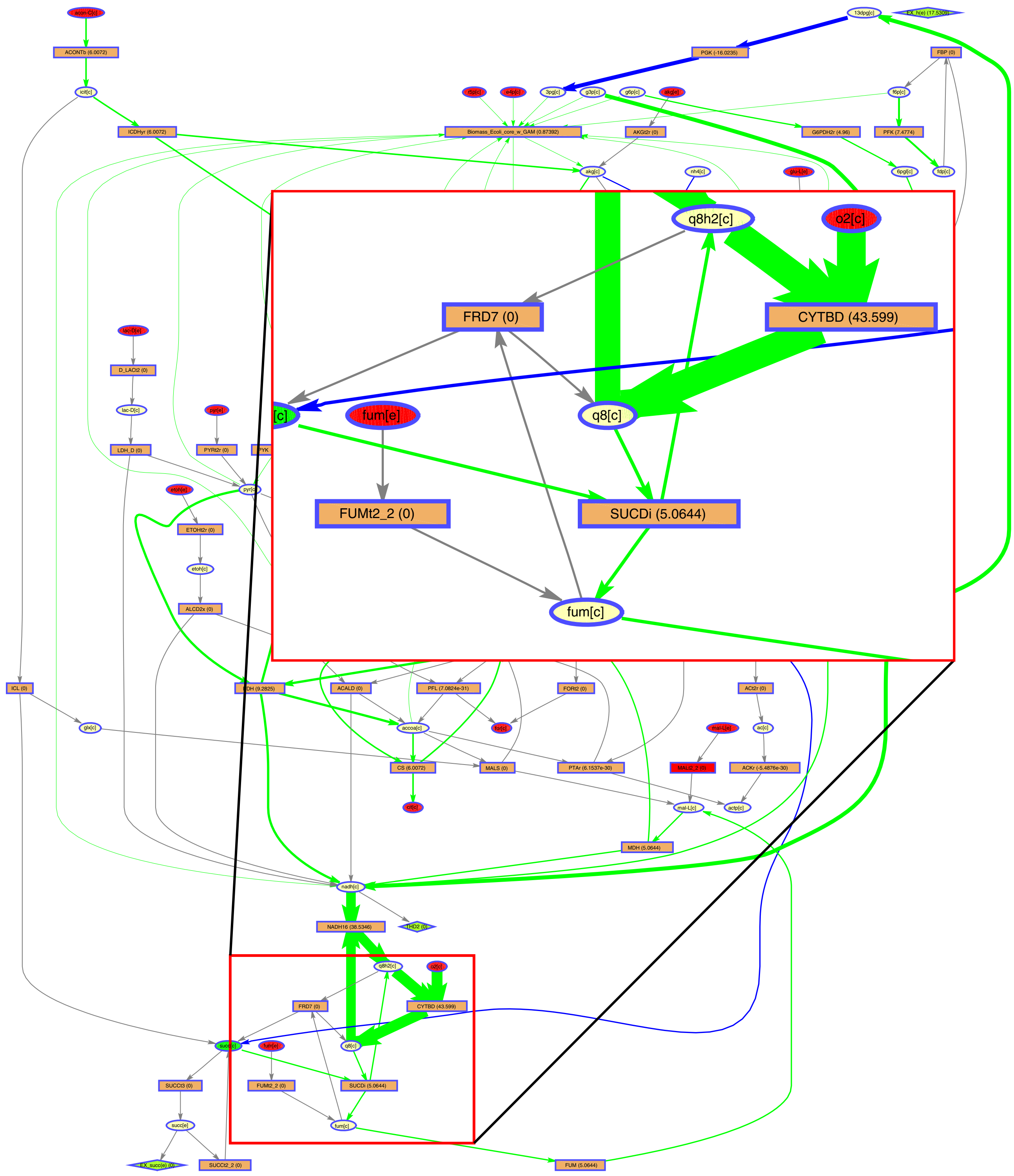


Fig. 12 | Selective scope visualization of the *E. coli* core model by Paint4Net. Rectangles represent reactions with rates of fluxes in parentheses; the red rectangles represent reactions with only one metabolite; ellipses represent metabolites; the red ellipses represent dead-end metabolites; gray arrows represent zero-rate fluxes; green arrows represent positive-rate (forward) fluxes; and blue arrows represent negative-rate (backward) fluxes. Network visualization also enables zoom to specific regions.

one can discover the isolated parts of network without any flux, as well as cycles running without any substrate.

References

1. Palsson, B. Ø. *Systems Biology: Constraint-Based Reconstruction and Analysis* (Cambridge University Press, Cambridge, 2015).
2. O'Brien, E. J., Monk, J. M. & Palsson, B. O. Using genome-scale models to predict biological capabilities. *Cell* **161**, 971–987 (2015).
3. Becker, S. A. et al. Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nat. Protoc.* **2**, 727–738 (2007).
4. Schellenberger, J. et al. Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat. Protoc.* **6**, 1290–1307 (2011).
5. Lewis, N. E., Nagarajan, H. & Palsson, B. O. Constraining the metabolic genotype–phenotype relationship using a phylogeny of in silico methods. *Nat. Rev. Microbiol.* **10**, 291–305 (2012).
6. Thiele, I. & Palsson, B. Ø. A protocol for generating a high-quality genome-scale metabolic reconstruction. *Nat. Protoc.* **5**, 93–121 (2010).
7. Kitano, H., Ghosh, S. & Matsuoka, Y. Social engineering for virtual 'big science' in systems biology. *Nat. Chem. Biol.* **7**, 323–326 (2011).
8. Bordbar, A., Monk, J. M., King, Z. A. & Palsson, B. O. Constraint-based models predict metabolic and associated cellular functions. *Nat. Rev. Genet.* **15**, 107–120 (2014).
9. Maia, P., Rocha, M. & Rocha, I. In silico constraint-based strain optimization methods: the quest for optimal cell factories. *Microbiol. Mol. Biol. Rev.* **80**, 45–67 (2016).
10. Hefzi, H. et al. A consensus genome-scale reconstruction of Chinese hamster ovary cell metabolism. *Cell Syst.* **3**, 434–443.e8 (2016).
11. Yusufi, F. N. K. et al. Mammalian systems biotechnology reveals global cellular adaptations in a recombinant CHO cell line. *Cell Syst.* **4**, 530–542.e6 (2017).
12. Zhuang, K. et al. Genome-scale dynamic modeling of the competition between *Rhodoferax* and *Geobacter* in anoxic subsurface environments. *ISME J* **5**, 305–316 (2011).
13. Jamshidi, N. & Palsson, B. Ø. Systems biology of the human red blood cell. *Blood Cells Mol. Dis.* **36**, 239–247 (2006).
14. Yizhak, K., Gabay, O., Cohen, H. & Ruppin, E. Model-based identification of drug targets that revert disrupted metabolism and its application to ageing. *Nat. Commun.* **4**, 2632 (2013).
15. Shlomi, T., Cabili, M. N. & Ruppin, E. Predicting metabolic biomarkers of human inborn errors of metabolism. *Mol. Syst. Biol.* **5**, 263 (2009).
16. Sahoo, S., Franzson, L., Jonsson, J. J. & Thiele, I. A compendium of inborn errors of metabolism mapped onto the human metabolic network. *Mol. Biosyst.* **8**, 2545–2558 (2012).
17. Thiele, I. et al. A community-driven global reconstruction of human metabolism. *Nat. Biotechnol.* **31**, 419–425 (2013).
18. Pagliarini, R. & di Bernardo, D. A genome-scale modeling approach to study inborn errors of liver metabolism: toward an in silico patient. *J. Comput. Biol.* **20**, 383–397 (2013).
19. Shaked, I., Oberhardt, M. A., Atias, N., Sharan, R. & Ruppin, E. Metabolic network prediction of drug side effects. *Cell Syst.* **2**, 209–213 (2016).
20. Chang, R. L., Xie, L., Xie, L., Bourne, P. E. & Palsson, B. Drug off-target effects predicted using structural analysis in the context of a metabolic network model. *PLoS Comput. Biol.* **6**, e1000938 (2010).
21. Kell, D. B. Systems biology, metabolic modelling and metabolomics in drug discovery and development. *Drug Discov. Today* **11**, 1085–1092 (2006).
22. Duarte, N. C. et al. Global reconstruction of the human metabolic network based on genomic and bibliomic data. *Proc. Natl. Acad. Sci. USA* **104**, 1777–1782 (2007).
23. Swainston, N. et al. Recon 2.2: from reconstruction to model of human metabolism. *Metabolomics* **12**, 109 (2016).
24. Pornputtapong, N., Nookaew, I. & Nielsen, J. Human metabolic atlas: an online resource for human metabolism. *Database* **2015**, bav068 (2015).
25. Zielinski, D. C. et al. Systems biology analysis of drivers underlying hallmarks of cancer cell metabolism. *Sci. Rep.* **7**, 41241 (2017).
26. Mardinoglu, A. et al. Genome-scale metabolic modelling of hepatocytes reveals serine deficiency in patients with non-alcoholic fatty liver disease. *Nat. Commun.* **5**, 3083 (2014).
27. Karlstädt, A. et al. CardioNet: a human metabolic network suited for the study of cardiomyocyte metabolism. *BMC Syst. Biol.* **6**, 114 (2012).
28. Gille, C. et al. HepatoNet1: a comprehensive metabolic reconstruction of the human hepatocyte for the analysis of liver physiology. *Mol. Syst. Biol.* **6**, 411 (2010).
29. Martins Conde Pdo, R., Sauter, T. & Pfau, T. Constraint based modeling going multicellular. *Front. Mol. Biosci.* **3**, 3 (2016).
30. Bordbar, A. et al. A multi-tissue type genome-scale metabolic network for analysis of whole-body systems physiology. *BMC Syst. Biol.* **5**, 180 (2011).

31. Yizhak, K. et al. Phenotype-based cell-specific metabolic modeling reveals metabolic liabilities of cancer. *Elife* **3**, e03641 (2014).
32. Mardinoglu, A. et al. Integration of clinical data with a genome-scale metabolic model of the human adipocyte. *Mol. Syst. Biol.* **9**, 649 (2013).
33. Bordbar, A. et al. Personalized whole-cell kinetic models of metabolism for discovery in genomics and pharmacodynamics. *Cell Syst.* **1**, 283–292 (2015).
34. Shoaie, S. et al. Quantifying diet-induced metabolic changes of the human gut microbiome. *Cell Metab.* **22**, 320–331 (2015).
35. Nogiec, C. D. & Kasif, S. To supplement or not to supplement: a metabolic network framework for human nutritional supplements. *PLoS ONE* **8**, e68751 (2013).
36. Heinken, A., Sahoo, S., Fleming, R. M. T. & Thiele, I. Systems-level characterization of a host-microbe metabolic symbiosis in the mammalian gut. *Gut Microbes* **4**, 28–40 (2013).
37. Heinken, A. et al. Functional metabolic map of *Faecalibacterium prausnitzii*, a beneficial human gut microbe. *J. Bacteriol.* **196**, 3289–3302 (2014).
38. Magnúsdóttir, S. et al. Generation of genome-scale metabolic reconstructions for 773 members of the human gut microbiota. *Nat. Biotechnol.* **35**, 81–89 (2017).
39. Lakshmanan, M., Koh, G., Chung, B. K. S. & Lee, D.-Y. Software applications for flux balance analysis. *Brief Bioinform.* **15**, 108–122 (2014).
40. Ebrahim, A., Lerman, J. A., Palsson, B. O. & Hyduke, D. R. COBRApy: constraints-based reconstruction and analysis for Python. *BMC Syst. Biol.* **7**, 74 (2013).
41. Arkin, A. P. et al. The United States Department of Energy Systems Biology Knowledgebase. *Nat. Biotechnol.* **36**, 566–569 (2018).
42. Heirendt, L., Thiele, I. & Fleming, R. M. T. DistributedFBA.jl: high-level, high-performance flux balance analysis in Julia. *Bioinformatics* **33**, 1421–1423 (2017).
43. Latendresse, M., Krummenacker, M., Trupp, M. & Karp, P. D. Construction and completion of flux balance models from pathway databases. *Bioinformatics* **28**, 388–396 (2012).
44. Karp, P. D. et al. Pathway Tools version 19.0 update: software for pathway/genome informatics and systems biology. *Brief Bioinform.* **17**, 877–890 (2016).
45. Sandve, G. K., Nekrutenko, A., Taylor, J. & Hovig, E. Ten simple rules for reproducible computational research. *PLoS Comput. Biol.* **9**, e1003285 (2013).
46. Ince, D. C., Hatton, L. & Graham-Cumming, J. The case for open computer programs. *Nature* **482**, 485–488 (2012).
47. Gevorgyan, A., Bushell, M. E., Avignone-Rossa, C. & Kierzek, A. M. SurreyFBA: a command line tool and graphics user interface for constraint-based modeling of genome-scale metabolic reaction networks. *Bioinformatics* **27**, 433–434 (2011).
48. Thorleifsson, S. G. & Thiele, I. rBioNet: a COBRA toolbox extension for reconstructing high-quality biochemical networks. *Bioinformatics* **27**, 2009–2010 (2011).
49. Sauls, J. T. & Buescher, J. M. Assimilating genome-scale metabolic reconstructions with modelBorgifier. *Bioinformatics* **30**, 1036–1038 (2014).
50. Noronha, A. et al. ReconMap: an interactive visualization of human metabolism. *Bioinformatics* **33**, 605–607 (2017).
51. Gawron, P. et al. MINERVA—a platform for visualization and curation of molecular interaction networks. *npj Syst. Biol. Appl.* **2**, 16020 (2016).
52. Olivier, B. G., Rohwer, J. M. & Hofmeyr, J.-H. S. Modelling cellular systems with PySCeS. *Bioinformatics* **21**, 560–561 (2005).
53. Gelius-Dietrich, G., Desouki, A. A., Fritzemeier, C. J. & Lercher, M. J. Sybil—efficient constraint-based modelling in R. *BMC Syst. Biol.* **7**, 125 (2013).
54. Ma, D. et al. Reliable and efficient solution of genome-scale models of metabolism and macromolecular expression. *Sci. Rep.* **7**, 40863 (2017).
55. Klamt, S., Saez-Rodriguez, J. & Gilles, E. D. Structural and functional analysis of cellular networks with CellNetAnalyzer. *BMC Syst. Biol.* **1**, 2 (2007).
56. Klamt, S. & von Kamp, A. An application programming interface for CellNetAnalyzer. *Biosystems* **105**, 162–168 (2011).
57. Apaolaza, I. et al. An in-silico approach to predict and exploit synthetic lethality in cancer metabolism. *Nat. Commun*. **8**, 459 (2017).
58. Maranas, C. D. & Zomorrodi, A. R. *Optimization Methods in Metabolic Networks* (Wiley, New York, 2016).
59. Chowdhury, A., Zomorrodi, A. R. & Maranas, C. D. Bilevel optimization techniques in computational strain design. *Comp. Chem. Eng.* **72**, 363–372 (2015).
60. Thiele, I. et al. Multiscale modeling of metabolism and macromolecular synthesis in *E. coli* and its application to the evolution of codon usage. *PLoS ONE* **7**, e45635 (2012).
61. Feist, A. M. et al. A genome-scale metabolic reconstruction for *Escherichia coli* K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information. *Mol. Syst. Biol.* **3**, 121 (2007).
62. Thiele, I., Jamshidi, N., Fleming, R. M. T. & Palsson, B. Ø. Genome-scale reconstruction of *Escherichia coli*'s transcriptional and translational machinery: a knowledge base, its mathematical formulation, and its functional characterization. *PLoS Comput. Biol.* **5**, e1000312 (2009).

63. Yang, L. et al. Systems biology definition of the core proteome of metabolism and expression is consistent with high-throughput data. *Proc. Natl. Acad. Sci. USA* **112**, 10810–10815 (2015).
64. Bornstein, B. J., Keating, S. M., Jouraku, A. & Hucka, M. LibSBML: an API library for SBML. *Bioinformatics* **24**, 880–881 (2008).
65. Aurich, M. K., Fleming, R. M. T. & Thiele, I. MetaboTools: a comprehensive toolbox for analysis of genome-scale metabolic models. *Front. Physiol.* **7**, 327 (2016).
66. Brunk, E. et al. Recon 3D: a resource enabling a three-dimensional view of gene variation in human metabolism. *Nat. Biotechnol.* **36**, 272–281 (2018).
67. Ma, D. & Saunders, M. A. Solving multiscale linear programs using the simplex method in quadruple precision. in *Numerical Analysis and Optimization*, Vol. 134 (eds. Al-Baali, M., Grandinetti, L. & Purnama, A.) 223–235 (Springer International Publishing, Cham, Switzerland, 2015).
68. Fleming, R. M. T. & Thiele, I. Mass conserved elementary kinetics is sufficient for the existence of a non-equilibrium steady state concentration. *J. Theor. Biol.* **314**, 173–181 (2012).
69. Gevorgyan, A., Poolman, M. G. & Fell, D. A. Detection of stoichiometric inconsistencies in biomolecular models. *Bioinformatics* **24**, 2245–2251 (2008).
70. Orth, J. D., Thiele, I. & Palsson, B. Ø. What is flux balance analysis? *Nat. Biotechnol.* **28**, 245–248 (2010).
71. Feist, A. M. & Palsson, B. O. The biomass objective function. *Curr. Opin. Microbiol.* **13**, 344–349 (2010).
72. Meléndez-Hevia, E. & Isidoro, A. The game of the pentose phosphate cycle. *J. Theor. Biol.* **117**, 251–263 (1985).
73. Orth, J. D. & Palsson, B. Ø. Systematizing the generation of missing metabolic knowledge. *Biotechnol. Bioeng.* **107**, 403–412 (2010).
74. Yamada, T. et al. Prediction and identification of sequences coding for orphan enzymes using genomic and metagenomic neighbours. *Mol. Syst. Biol.* **8**, 581 (2012).
75. Liberal, R. & Pinney, J. W. Simple topological properties predict functional misannotations in a metabolic network. *Bioinformatics* **29**, i154–i161 (2013).
76. Reed, J. L. et al. Systems approach to refining genome annotation. *Proc. Natl. Acad. Sci. USA* **103**, 17480–17484 (2006).
77. Orth, J. D. & Palsson, B. Gap-filling analysis of the iJO1366 *Escherichia coli* metabolic network reconstruction for discovery of metabolic functions. *BMC Syst. Biol.* **6**, 30 (2012).
78. Chang, R. L. et al. Metabolic network reconstruction of *Chlamydomonas* offers insight into light-driven algal metabolism. *Mol. Syst. Biol.* **7**, 518 (2011).
79. Rolfsson, O., Palsson, B. Ø. & Thiele, I. The human metabolic reconstruction Recon 1 directs hypotheses of novel human metabolic functions. *BMC Syst. Biol.* **5**, 155 (2011).
80. Rolfsson, Ó., Paglia, G., Magnusdóttir, M., Palsson, B. Ø. & Thiele, I. Inferring the metabolism of human orphan metabolites from their metabolic network context affirms human gluconokinase activity. *Biochem. J.* **449**, 427–435 (2013).
81. Satish Kumar, V., Dasika, M. S. & Maranas, C. D. Optimization based automated curation of metabolic reconstructions. *BMC Bioinformatics* **8**, 212 (2007).
82. Thiele, I., Vlassis, N. & Fleming, R. M. T. fastGapFill: efficient gap filling in metabolic networks. *Bioinformatics* **30**, 2529–2531 (2014).
83. Willemsen, A. M. et al. MetDFBA: incorporating time-resolved metabolomics measurements into dynamic flux balance analysis. *Mol. Biosyst.* **11**, 137–145 (2014).
84. Kleessen, S., Irgang, S., Klie, S., Giavalisco, P. & Nikoloski, Z. Integration of transcriptomics and metabolomics data specifies the metabolic response of *Chlamydomonas* to rapamycin treatment. *Plant J.* **81**, 822–835 (2015).
85. Bordbar, A. et al. Elucidating dynamic metabolic physiology through network integration of quantitative time-course metabolomics. *Sci. Rep.* **7**, 46249 (2017).
86. Blazier, A. S. & Papin, J. A. Integration of expression data in genome-scale metabolic network reconstructions. *Front. Physiol.* **3**, 299 (2012).
87. Opdam, S. et al. A systematic evaluation of methods for tailoring genome-scale metabolic models. *Cell Syst.* **4**, 318–329.e6 (2017).
88. Estévez, S. R. & Nikoloski, Z. Generalized framework for context-specific metabolic model extraction methods. *Front. Plant Sci.* **5**, 491 (2014).
89. Vlassis, N., Pacheco, M. P. & Sauter, T. Fast reconstruction of compact context-specific metabolic network models. *PLoS Comput. Biol.* **10**, e1003424 (2014).
90. Becker, S. A. & Palsson, B. O. Context-specific metabolic networks are consistent with experiments. *PLoS Comput. Biol.* **4**, e1000082 (2008).
91. Zur, H., Ruppin, E. & Shlomi, T. iMAT: an integrative metabolic analysis tool. *Bioinformatics* **26**, 3140–3142 (2010).
92. Agren, R. et al. Reconstruction of genome-scale active metabolic networks for 69 human cell types and 16 cancer types using INIT. *PLoS Comp. Biol.* **8**, e1002518 (2012).
93. Jerby, L., Shlomi, T. & Ruppin, E. Computational reconstruction of tissue-specific metabolic models: application to human liver metabolism. *Mol. Syst. Biol.* **6**, 401 (2010).
94. Wang, Y., Eddy, J. A. & Price, N. D. Reconstruction of genome-scale metabolic models for 126 human tissues using mCADRE. *BMC Syst. Biol.* **6**, 153 (2012).
95. Kuhar, M. J. On the use of protein turnover and half-lives. *Neuropsychopharmacology* **34**, 1172–1173 (2008).

96. Lajtha, A. & Sylvester, V. *Handbook of Neurochemistry and Molecular Neurobiology* (Springer, Boston, 2008).
97. Schuster, S. & Hilgetag, C. On elementary flux modes in biochemical reaction systems at steady state. *J. Biol. Syst.* **02**, 165–182 (1994).
98. Schilling, C. H., Letscher, D. & Palsson, B. Ø. Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. *J. Theor. Biol.* **203**, 229–248 (2000).
99. Klamt, S. et al. From elementary flux modes to elementary flux vectors: metabolic pathway analysis with arbitrary linear flux constraints. *PLoS Comput. Biol.* **13**, e1005409 (2017).
100. Bordbar, A. et al. Minimal metabolic pathway structure is consistent with associated biomolecular interactions. *Mol. Syst. Biol.* **10**, 737 (2014).
101. Gudmundsson, S. & Thiele, I. Computationally efficient flux variability analysis. *BMC Bioinformatics* **11**, 489 (2010).
102. Haraldsdóttir, H. S., Cousins, B., Thiele, I., Fleming, R. M. T. & Vempala, S. CHRR: coordinate hit-and-run with rounding for uniform sampling of constraint-based models. *Bioinformatics* **33**, 1741–1743 (2017).
103. Cousins, B. & Vempala, S. Gaussian cooling and algorithms for volume and Gaussian volume. *SIAM J. Comput.* **47**, 1237–1273 (2018).
104. Cousins, B. & Vempala, S. A practical volume algorithm. *Math. Prog. Comp.* **8**, 1–28 (2015).
105. Burgard, A. P., Pharkya, P. & Maranas, C. D. Optknock: a bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Biotechnol. Bioeng.* **84**, 647–657 (2003).
106. Patil, K. R., Rocha, I., Förster, J. & Nielsen, J. Evolutionary programming as a platform for in silico metabolic engineering. *BMC Bioinformatics* **6**, 308 (2005).
107. Lun, D. S. et al. Large-scale identification of genetic design strategies using local search. *Mol. Syst. Biol.* **5**, 296 (2009).
108. Ranganathan, S., Suthers, P. F. & Maranas, C. D. OptForce: an optimization procedure for identifying all genetic manipulations leading to targeted overproductions. *PLoS Comput. Biol.* **6**, e1000744 (2010).
109. Antoniewicz, M. R. et al. Metabolic flux analysis in a nonstationary system: fed-batch fermentation of a high yielding strain of *E. coli* producing 1,3-propanediol. *Metab. Eng.* **9**, 277–292 (2007).
110. Haraldsdóttir, H. S., Thiele, I. & Fleming, R. M. T. Comparative evaluation of open source software for mapping between metabolite identifiers in metabolic network reconstructions: application to Recon 2. *J. Cheminform.* **6**, 2 (2014).
111. Preciat Gonzalez, G. A. et al. Comparative evaluation of atom mapping algorithms for balanced metabolic reactions: application to Recon 3D. *J. Cheminform.* **9**, 39 (2017).
112. Kim, S. et al. PubChem substance and compound databases. *Nucleic Acids Res.* **44**, D1202–D1213 (2016).
113. Kanehisa, M. & Goto, S. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res.* **28**, 27–30 (2000).
114. Hastings, J. et al. The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013. *Nucleic Acids Res.* **41**, D456–D463 (2013).
115. Sud, M. et al. LMSD: LIPID MAPS structure database. *Nucleic Acids Res.* **35**, D527–D532 (2007).
116. Forster, M., Pick, A., Raitner, M., Schreiber, F. & Brandenburg, F. J. The system architecture of the BioPath system. *In Silico Biol.* **2**, 415–426 (2002).
117. Williams, A. J., Tkachenko, V., Golotvin, S., Kidd, R. & McCann, G. ChemSpider—building a foundation for the semantic web by hosting a crowd sourced databasing platform for chemistry. *J. Cheminform.* **2**, O16 (2010).
118. Wishart, D. S. et al. HMDB: the Human Metabolome Database. *Nucleic Acids Res.* **35**, D521–D526 (2007).
119. Rahman, S. A. et al. Reaction Decoder Tool (RDT): extracting features from chemical reactions. *Bioinformatics* **32**, 2065–2066 (2016).
120. Kumar, A. & Maranas, C. D. CLCA: maximum common molecular substructure queries within the MetRxn Database. *J. Chem. Inf. Model.* **54**, 3417–3438 (2014).
121. Shimizu, Y., Hattori, M., Goto, S. & Kanehisa, M. Generalized reaction patterns for prediction of unknown enzymatic reactions. *Genome Inform.* **20**, 149–158 (2008).
122. Haraldsdóttir, H. S. & Fleming, R. M. T. Identification of conserved moieties in metabolic networks by graph theoretical analysis of atom transition networks. *PLoS Comput. Biol.* **12**, e1004999 (2016).
123. Klamt, S., Haus, U.-U. & Theis, F. Hypergraphs and cellular networks. *PLoS Comput. Biol.* **5**, e1000385 (2009).
124. Fleming, R. M. T. & Thiele, I. von Bertalanffy 1.0: a COBRA toolbox extension to thermodynamically constrain metabolic models. *Bioinformatics* **27**, 142–143 (2011).
125. Fleming, R. M. T., Thiele, I. & Nasheuer, H. P. Quantitative assignment of reaction directionality in constraint-based models of metabolism: application to *Escherichia coli*. *Biophys. Chem.* **145**, 47–56 (2009).
126. Haraldsdóttir, H. S., Thiele, I. & Fleming, R. M. T. Quantitative assignment of reaction directionality in a multicompartmental human metabolic reconstruction. *Biophys. J.* **102**, 1703–1711 (2012).
127. Noor, E., Haraldsdóttir, H. S., Milo, R. & Fleming, R. M. T. Consistent estimation of Gibbs energy using component contributions. *PLoS Comput. Biol.* **9**, e1003098 (2013).
128. Fleming, R. M. T., Maes, C. M., Saunders, M. A., Ye, Y. & Palsson, B. Ø. A variational principle for computing nonequilibrium fluxes and potentials in genome-scale biochemical networks. *J. Theor. Biol.* **292**, 71–77 (2012).

129. Beard, D. A., Liang, S.-D. & Qian, H. Energy balance for analysis of complex metabolic networks. *Biophys. J.* **83**, 79–86 (2002).
130. Qian, H. & Beard, D. A. Thermodynamics of stoichiometric biochemical networks in living systems far from equilibrium. *Biophys. Chem.* **114**, 213–220 (2005).
131. Fleming, R. M. T., Thiele, I., Provan, G. & Nasheuer, H. P. Integrated stoichiometric, thermo- dynamic and kinetic modelling of steady state metabolism. *J. Theor. Biol.* **264**, 683–692 (2010).
132. Schellenberger, J., Lewis, N. E. & Palsson, B. Ø. Elimination of thermodynamically infeasible loops in steady-state metabolic models. *Biophys. J.* **100**, 544–553 (2011).
133. Soh, K. C. & Hatzimanikatis, V. Network thermodynamics in the post-genomic era. *Curr. Opin. Microbiol.* **13**, 350–357 (2010).
134. Fleming, R. M. T., Vlassis, N., Thiele, I. & Saunders, M. A. Conditions for duality between fluxes and concentrations in biochemical networks. *J. Theor. Biol.* **409**, 1–10 (2016).
135. Aragón Artacho, F.J., Fleming, R. M. T. & Vuong, P. T. Accelerating the DC algorithm for smooth functions. *Math. Program.* **169**, 95–118 (2018).
136. Artacho, F. J. A. & Fleming, R. M. T. Globally convergent algorithms for finding zeros of duplomonotone mappings. *Optim. Lett.* **9**, 1–16 (2014).
137. Ahookhosh, M., Aragón, F. J., Fleming, R. M. T. & Vuong, P. T. Local convergence of Levenberg-Marquardt methods under Hölder metric subregularity. Preprint at https://arxiv.org/abs/1703.07461 (2017).
138. Shannon, P. et al. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.* **13**, 2498–2504 (2003).
139. King, Z. A. et al. Escher: a web application for building, sharing, and embedding data-rich visualizations of biological pathways. *PLoS Comput. Biol.* **11**, e1004321 (2015).
140. Kuperstein, I. et al. NaviCell: a web-based environment for navigation, curation and maintenance of large molecular interaction maps. *BMC Syst. Biol.* **7**, 100 (2013).
141. Kostromins, A. & Stalidzans, E. Paint4net: COBRA Toolbox extension for visualization of stoichiometric models of metabolism. *Biosystems* **109**, 233–239 (2012).
142. Aurich, M. K. et al. Prediction of intracellular metabolic states from extracellular metabolomic data. *Metabolomics* **11**, 603–619 (2014).
143. Guebila, M. B. & Thiele, I. Model-based dietary optimization for late-stage, levodopa-treated, Parkinson's disease patients. *npj Syst. Biol. Appl.* **2**, 16013 (2016).
144. Sun, Y., Fleming, R. M. T., Thiele, I. & Saunders, M. A. Robust flux balance analysis of multiscale biochemical reaction networks. *BMC Bioinformatics* **14**, 240 (2013).
145. Lewis, N. E. et al. Omic data from evolved *E. coli* are consistent with computed optimal growth from genome-scale models. *Mol. Syst. Biol.* **6**, 390 (2010).
146. Thiele, I., Fleming, R. M. T., Bordbar, A., Schellenberger, J. & Palsson, B. Ø. Functional characterization of alternate optimal solutions of *Escherichia coli*'s transcriptional and translational machinery. *Biophys. J.* **98**, 2072–2081 (2010).
147. Ballerstein, K., von Kamp, A., Klamt, S. & Haus, U.-U. Minimal cut sets in a metabolic network are elementary modes in a dual network. *Bioinformatics* **28**, 381–387 (2012).
148. von Kamp, A. & Klamt, S. Enumeration of smallest intervention strategies in genome-scale metabolic networks. *PLoS Comput. Biol.* **10**, e1003378 (2014).
149. Fujita, K. A. et al. Integrating pathways of Parkinson's disease in a molecular interaction map. *Mol. Neurobiol.* **49**, 88–102 (2014).
150. Agren, R. et al. The RAVEN Toolbox and its use for generating a genome-scale metabolic model for *Penicillium chrysogenum*. *PLoS Comput. Biol.* **9**, e1002980 (2013).
151. Grafahrend-Belau, E., Klukas, C., Junker, B. H. & Schreiber, F. FBA-SimVis: interactive visualization of constraint-based metabolic models. *Bioinformatics* **25**, 2755–2757 (2009).
152. Rocha, I. et al. OptFlux: an open-source software platform for in silico metabolic engineering. *BMC Syst. Biol.* **4**, 45 (2010).
153. Poolman, M. G. ScrumPy: metabolic modelling with Python. *Syst. Biol.* **153**, 375–378 (2006).
154. Hoppe, A., Hoffmann, S., Gerasch, A., Gille, C. & Holzhütter, H.-G. FASIMU: flexible software for flux-balance computation series in large metabolic networks. *BMC Bioinformatics* **12**, 28 (2011).
155. Boele, J., Olivier, B. G. & Teusink, B. FAME, the flux analysis and modeling environment. *BMC Syst. Biol.* **6**, 8 (2012).

Acknowledgements

The Reproducible Research Results (R3) team, in particular, C. Trefois and Y. Jarosz, of the Luxembourg Centre for Systems Biomedicine, is acknowledged for their help in setting up the virtual machine and the Jenkins server. This study was funded by the National Centre of Excellence in Research (NCER) on Parkinson's disease, the U.S. Department of Energy, Offices of Advanced Scientific Computing Research and the Biological and Environmental Research as part of the Scientific Discovery Through Advanced Computing program, grant no. DE-SC0010429. This project also received funding from the European Union's HORIZON 2020 Research and Innovation Programme under grant agreement no. 668738 and the Luxembourg National Research Fund (FNR) ATTRACT program (FNR/A12/01) and OPEN (FNR/O16/11402054) grants. N.E.L. was supported by NIGMS (R35 GM119850) and the Novo Nordisk Foundation (NNF10CC1016517). M.A.P.O. was supported by the Luxembourg National Research Fund (FNR) grant AFR/6669348. A.R. was supported by the Lilly Innovation Fellows Award. F.J.P. was supported by the Minister of Economy and Competitiveness of Spain (BIO2016-77998-R) and the ELKARTEK Programme of the Basque Government (KK-2016/00026). I.A. was supported by a Basque

Government predoctoral grant (PRE_2016_2_0044). B.Ø.P. was supported by the Novo Nordisk Foundation through the Center for Biosustainability at the Technical University of Denmark (NNF10CC1016517).

Author contributions

S.A.: continuous integration, code review, opencobra.github.io/cobratoolbox, Jenkins, Documenter.py, changeCobraSolver, pull request support, tutorials, tests, coordination, manuscript, and initCobraToolbox. L.H.: continuous integration, code review, fastFVA (new version, test, and integration), MATLAB.devTools, opencobra.github.io, tutorials, tests, pull request support, coordination, manuscript, initCobraToolbox, and forum support. T.P.: input–output and transcriptomic integration, tutorials, tutorial reviews, input–output and transcriptomic integration sections of manuscript, forum support, pull request support, and code review. S.N.M.: development and update of strain design algorithms, GAMS and MATLAB integration, and tutorials. A.R.: transcriptomic data integration methods, tutorials, transcriptomic integration section of manuscript, RuMBA, pFBA, metabolic tasks, and tutorial review. A.H.: multispecies modeling code contribution, tutorial review, and testing. H.S. Haraldsdóttir: thermodynamics, conserved moiety, and sampling methods. J.W.: documentation. S.M.K.: SBML input–output support. V.V.: tutorials. S.M.: multispecies modeling, tutorial review, and testing. C.Y.N.: strain design code review, tutorial review, and manuscript (OptForce/biotech introduction). G.P.: tutorials and chemoinformatics for metabolite structures and atom mapping data. A.Ž.: metabolic cartography. S.H.J.C.: solution navigation, multispecies modeling code, and tutorial review. M.K.A.: metabolomic data integration. C.M.C.: tutorials and testing. J.M.: metabolic cartography and human metabolic network visualization tutorials. J.T.S.: modelBorgifier code and tutorial. A.N.: virtual metabolic human interoperability. A.B.: MinSpan method and tutorial, supervision on uFBA method and tutorial. B.C.: CHRR uniform sampling. D.C.E.A.: tutorials. L.V.V.: tutorials and genetic MCS implementation. I.A.: tutorials and genetic MCS implementation. S.G.: interoperability with CellNetAnalyzer. M.A.: adaptive Levenberg–Marquardt solver. M.B.G.: tutorial reviews. A.K.: Paint4Net code and tutorial. N.S.: development of metabolomic cartography tool and tutorial. H.M.L.: cardinality optimization solver. D.M.: quadruple-precision solvers. Y.S.: multiscale FBA reformulation. L.W.: strain design code review, tutorial review, and manuscript (OptForce). J.T.Y.: uFBA method and tutorial. M.A.P.O.: tutorial. P.T.V.: adaptive Levenberg–Marquardt solvers and boosted difference of convex optimization solver. L.P.E.A.: chemoinformatic data integration and documentation. I.K.: development of metabolomic cartography tool and tutorial. A.Z.: development of metabolomic cartography tool and tutorial. H.S. Hinton: *E. coli* core tutorials. W.A.B.: code refinement. F.J.A.A.: duplomonotone equation solver, boosted difference of convex optimization solver, and adaptive Levenberg–Marquardt solvers. F.J.P.: academic supervision, tutorials, and genetic MCS implementation. E.S.: academic supervision, Paint4Net, and tutorial. A.M.: academic supervision. S.V.: academic supervision and CHRR uniform sampling algorithm. M.H.: academic supervision and SBML input–output support. M.A.S.: academic supervision, quadruple-precision solvers, nullspace computation, and convex optimization. C.D.M.: academic supervision and strain design algorithms. N.E.L.: academic supervision and coding, and transcriptomic data integration, RuMBA, pFBA, metabolic tasks, and tutorial review. T.S.: academic supervision and FASTCORE algorithm. B.Ø.P.: academic supervision and openCOBRA stewardship. I.T.: academic supervision, tutorials, code contribution, and manuscript. R.M.T.F.: conceptualization, lead developer, academic supervision, software architecture, code review, sparse optimization, nullspace computation, thermodynamics, variational kinetics, fastGapFill, sampling, conserved moieties, network visualization, forum support, tutorials, and manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s41596-018-0098-2.

Reprints and permissions information is available at www.nature.com/reprints.

Correspondence and requests for materials should be addressed to R.M.T.F.

Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Published online: 20 February 2019

Related links

Key references using this protocol

Schellenberger, J. et al. *Nat. Protocols* **6**, 1290–1307 (2011): https://www.nature.com/articles/nprot.2011.308

Becker, S. A. et al. *Nat. Protocols* **2**, 727–738 (2007): https://www.nature.com/articles/nprot.2007.99