# **Homology--based generation of a genome--scale metabolic models for *Hansenula polymorpha* using the RAVEN toolbox**

# **1 Introduction**

Cellular metabolism of organisms are complex modular networks involving several hundreds of reactions and metabolites. To meaningfully describe the behaviour of such complicated systems requires sophisticated computational approaches. It is precisely for this reason that genome-scale metabolic models were developed. A genome-scale metabolic model (GEM) represents a mathematical description of the cellular metabolism of a given organism in a quantitative manner. The mathematical formulation of GEMs is based on the S-matrix, an elegant summary of the stoichiometry an organism’s specific biochemical pathways and metabolic capabilities. The S-matrix is sparse and of dimensions *m* by *n*, where *m* represents the number of metabolites and *n* the number of reactions present in the metabolism of a given organism. Computational models such as GEMs are generally based on a bottom-up approach, leveraging the plethora of biochemical and omics information generated by the scientific community to reconstruct the complex network of metabolism. Well developed and curated GEMs can be powerful computational tools suitable for a large range of analyses. Perhaps most notably, constraining the model with measurable reaction rates, such as the carbon uptake, allows for an estimation of all the metabolic fluxes using flux balance analysis (FBA). Therefore, GEMs can be used for the prediction of phenotypic effects caused by genetic modifications. This has a vast range of applications including antibiotic discovery, production strain engineering, and microbial community interaction analysis, which have been reviewed in detail. In short, well curated GEMs are useful frameworks that allow researchers to supplement *in vitro/in vivo* experiments with *in silico* simulations.

The first published GEM, which debuted in 1999, aimed to model the metabolism of the modest *Haemophilus influenzae Rd*. Since then, GEMs for larger and more complex organisms have been developed and extensively curated. In 2010, Thiele and Palsson reported the existence of over 30 published GEMs. In the decade since then, that number has increased significantly as the GEM reconstruction process continues to mature. In fact, a number of toolboxes have been developed to facilitate the tedious reconstruction and curation process. Some notable examples include COBRA, RAVEN, and more recently CarveMe. While the reconstruction of GEMs has come a long way since the turn of the millennium, there are a number of challenges that persist in the reconstruction process. For example, manual curation and model refinement represents a glaring bottleneck in the reconstruction process, as this step can take anywhere from weeks to years.

The following protocol demonstrates a practical implementation of the various steps necessary for the homology--based reconstruction of GEMs using the RAVEN toolbox. Additionally, complementary tools and methods will be discussed when relevant. As an illustrative example, a draft model for the industrially relevant filamentous yeast *Hansenula polymorpha* will be reconstructed. This organism was chosen due to its high scientific interest and lack of published high quality GEM. Indeed, this organism was recently demonstrated to be conveniently editable with a CRISPR/Cas9 system, rendering it an even more attractive target for GEM reconstruction.

## **2 Materials**

GEM reconstruction generally requires a number of different software packages. While the following protocol focuses on a MATLAB-based approach, it also makes use of Python when necessary. In addition to this, various files related to the organism of interest and the template models are required.

### 2.1 Software

The following table summarizes information about the software used in this reconstruction approach. Note that with the exception of MATLAB, all software used is open source and available through GitHub.

Table 1: Summary of software dependencies and sources

|  |  |  |
| --- | --- | --- |
| **Software** | **Dependencies** | **Source** |
| MATLAB | License | https://mathworks.com/products/matlab.html |
| RAVEN | libSBML MATLAB API, Solver | https://github.com/SysBioChalmers/RAVEN |
| COBRA | MATLAB | https://github.com/opencobra/cobratoolbox |
| BOFdat | Python | https://github.com/jclachance/BOFdat |
| meneco | Python | https://github.com/bioasp/meneco |
| Gurobi | License | https://www.gurobi.com/products/gurobi-optimizer/ |
| libSBML | MATLAB | http://sbml.org/Software/libSBML/Downloading\_libSBML |

#### 2.1.1 MATLAB

Although not open source, MATLAB is a widely used programming language in science and engineering. A MATLAB license can be purchased privately or obtained through an academic institution. This protocol requires version 2013b or higher.

#### 2.1.2 RAVEN toolbox

Clone or download RAVEN from the Chalmers Sysbio GitHub. RAVEN, short for Reconstruction, Analysis and Visualization of Metabolic Networks, provides a useful framework for GEM development. RAVEN and works on Windows, Mac and Unix. Note that dependencies include the [libSBML MATLAB API](https://sourceforge.net/projects/sbml/files/libsbml/5.16.0/stable/MATLAB%20interface/) and a linear programming solver (e.g gurobi, mosek, or COBRA solver). For easiest usage, it is recommended that users also install the COBRA toolbox, as this satisfies the aforementioned dependencies in one step.

#### 2.1.3 BOFdat

Short for Biomass Objective Function data, BOFdat is an open source python package used for the determination of organism-specific biomass equation stoichiometric coefficients from experimental data. Note that BOFdat, unlike RAVEN or COBRA, runs on Python instead of MATLAB. This tool can be easily installed, locally or on a server, using the pip install command.

#### 2.1.4 meneco

Short for MEtabolic NEtwork COmpletion, this python based tool is used to gapfill draft reconstructions in order to generate fully functional models. Again, note that this tool is used outside of MATLAB, and can be installed locally or on a server using the pip install command.

### 2.2 Files

GEM reconstruction also requires a number of different types of files, including template GEMs, DNA FASTA, protein FASTA, and other omics information. The following table summarizes the files used for the approach presented in this protocol. These files can be obtained individually from their respective original sources, or collectively downloaded from the *Hansenula polymorpha* GEM GitHub repository.

Table 2: Summary of required organism specific file types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **File Type** | **Usage** | **Organism** | **Source** |
| Template Organism | GEM | Template network | *S. cerevisiae* | https://github.com/SysBioChalmers/yeast-GEM/releases/tag/v8.3.0 |
| GEM | Template network | *R. toruloides* | https://github.com/SysBioChalmers/rhto-GEM/releases/tag/v1.1.1 |
| Protein FASTA | BLAST | *S. cerevisiae* | https://github.com/SysBioChalmers/rhto-GEM/blob/master/ComplementaryData/genome/sce\_s288c.faa |
| Protein FASTA | BLAST | *R. toruloides* | https://github.com/SysBioChalmers/rhto-GEM/blob/master/ComplementaryData/genome/rhto\_np11.faa |
| Target Organism | Protein FASTA | BLAST | *H. polymorpha* | https://genome.jgi.doe.gov/portal/Hanpo2/download/Hanpo2\_GeneCatalog\_proteins\_20100927.aa.fasta.gz |
| Genbank | Biomass curation | *H. polymorpha* | ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/001/664/045/GCF\_001664045.1\_Hanpo2/GCF\_001664045.1\_Hanpo2\_genomic.gbff.gz |
| DNA FASTA | Biomass curation | *H. polymorpha* | ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/001/664/045/GCF\_001664045.1\_Hanpo2/GCF\_001664045.1\_Hanpo2\_genomic.fna.gz |

### 2.3 Literature data

Any available knowledge on the metabolic behaviour and capabilities of the organism of interest can be used to curate the model. More specifically, BOFdat requires estimates for the dry cell weight composition of the target organism, as well as relevant omic dataset (i.e. transcriptomic, lipidomic, and proteomic). Through a short literature search, we identified two relevant papers which characterize the lipid profile and carbohydrate profile of *H. polymorpha*, respectively [1,2]. These papers also contain information about the dry cell weight fractions of *H. polymorpha* biomass. Any required information that could not be found specifically for *H. polymorpha* was estimated from *S. cerevisiae* data [3].

[1] Petersen, Gene R. “Determining a Carbohydrate Profile for Hansenula Polymorpha.” *Enzyme and Microbial Technology*, vol. 7, no. 7, 1985, pp. 339–345., doi:10.1016/0141-0229(85)90113-9.

[2] Wijeyaratne, Sushila Chandrani, et al. “Lipid Composition of a Thermotolerant Yeast,Hansenula Polymorpha.” *Agricultural and Biological Chemistry*, vol. 50, no. 4, 1986, pp. 827–832., doi:10.1080/00021369.1986.10867502.

[3] Gombert, A K, et al. “Network Identification and Flux Quantification in the Central Metabolism of Saccharomyces Cerevisiae under Different Conditions of Glucose Repression.” *Journal of Bacteriology*, American Society for Microbiology, Feb. 2001, www.ncbi.nlm.nih.gov/pmc/articles/PMC95019/.

## **3 Methods**

As detailed in subsection 3.7, development of the GEM for *Hansenula polymorpha* is tracked in a git repository. All required files and scripts can be found on GitHub . Clone or download this repository and set folder as the MATLAB working directory. Setting the repository folder as the working directory tells MATLAB where to look for the necessary scripts and files.

### **3.1 Install RAVEN**

Once the required software and files are obtained (see Methods), the relevant software needs to be installed. Refer to the RAVEN wiki for an installation guide, and detailed information regarding requirements and dependencies. The solver Gurobi 8.0 is used in this protocol. To verify that RAVEN and its dependencies been properly installed, simply run the checkInstallation function in the MATLAB command window. Note that this function does not take any input arguments. The output of this function is useful for troubleshooting installation problems.

### **3.2 Import template models**

RAVEN can create draft GEMs for an organisms of interest by using a models from phylogenetically related organisms as templates. This functionality allows users to leverage well curated models to develop GEMs for new organisms of relevance. To reconstruct a GEM for *Hansenula polymorpha*, the highly curated model for the ascomycete *Saccharomyces cerevisiae* will be used as primary template, while a model of the basidiomycete *Rhodosporidium toruloides* is used as complementary template model. While there is no limit to the number of template models one can use, it is essential that the model components (i.e. metabolites, reactions, genes) use the same identifiers.

First let us import the *S. cerevisiae* GEM and ensure that the model is functional.

modelSce=importModel('yeastGEMv8.3.0.xml');

modelSce.id = 'sce';

TheimportModelfunction imports the .xml format *S. cerevisiae* GEM into a RAVEN format MATLAB structure.The model id is set to ‘sce’, as this will be used below to identify each model. The model can then be exported in Excel format for a more user-friendly inspection.

exportToExcelFormat(modelSce,'modelSce.xlsx');

This function takes two input arguments: the model structure and a character string of the desired name for the excel file. The generated excel file has five sheets: RXNS, METS, COMPS, GENES, and MODEL. These contain information regarding each reaction, metabolite, compartment, gene, and model, respectively. Repeat this step for any additional template models. In our example case, we also import the *R. toruloides* model.

### **3.3 Generate models from homology**

In this section we will make use of the obtained protein FASTA files to construct a draft model for *H. polymorpha* using a homology-based approach. Please ensure that gene identifiers in the template model and FASTA file match, as discussed below in section 3.3.1. Two key RAVEN functions necessary for constructing a draft model from homology are getBlast and getModelFromHomology.

#### 3.3.1 Match protein FASTA IDs

It is crucial that the identifiers in the protein FASTA files of the homologous organisms match the gene identifiers in their respective template models. This can be checked by opening the FASTA files using a text editor, and comparing the protein ID format found here with the gene identifiers found in the respective template models. In this example, the downloaded protein FASTA file contains some extra information within each identifier, which will cause problems with the getModelFromHomology function in the subsequent protocol section. To avoid problems, these unnecessary annotations need to be removed. This can be done using a text editor or with unix, for example using the sed command. Please refer to the additional notes in section 4.2 to see a detailed example of how this could be done.

#### 3.3.2 Generate GEM from homology using *S. cerevisiae* template

First, we perform a bi-directional protein BLAST search using the getBlast function as shown below:

hpoSceBlast = getBlast('hpo','hpo.faa',{'sce'},{'sce.faa'});

The getBlast function takes four input arguments: a character string with the target organism model ID (‘hpo’), a character string of the target organism protein FASTA filename (‘hpo.faa’), a cell containing the character string of the template organism ({‘sce’}), and a cell containing the character string of the template organism’s protein FASTA filename ({‘sce.faa’}). It is worth noting that this function can use more than one template organism FASTA file to create a BLAST structure, in which case the last two input arguments would be lists rather than single element cell arrays. Also, bear in mind that if the FASTA files are not in MATLAB’s current working directory, then the file paths need to be specified along with the filenames.

We can now obtain a homology based draft model for *H. polymorpha* using the getModelFromHomology function, as shown below:

modelHpoHomology1=getModelFromHomology({modelSce},hpoBlastStructure,'hpo',{},1,false,10^-20,100,35,true)

Refer to the additional notes in section 4.3 to get an in depth explanation for each of the input parameters of this important function.

Next, it is necessary to also import exchange reactions from the template model, since these reactions are not gene annotated and therefore not imported by the getModelFromHomology function. Note that exchange reactions are necessary to simulate the flow of metabolites into and out of our system, the cell. We might not require all exchange reactions present in the template model, but we can easily remove any unconnected reactions later.

exRxns=getExchangeRxns(modelSce);

modelHpoHomology1=addRxnsGenesMets(modelHpoHomology1,modelSce,exRxns,false,'Exchange reaction from S.ce template model',1);

Same as with the exchange reactions, we wish to add all non-gene annotated transport and diffusion reactions. Additionally, we wish to import the biomass (growth) reaction from this template model organism, as it is more closely related to our target *H. polymorpha*. Depending on the nature of the template model, the biomass equation may be present as one long reaction, or broken up into smaller pseudo-reactions. In the case of our *S. cerevisiae* template, the latter is the case, and therefore we wish to include all biomass-associated pseudo-reactions in our target draft model. A quick way of identifying and adding these aforementioned reactions is shown below.

First, find all indexes for reactions with no associated gene:

noGeneIdx=find(cellfun(@isempty,modelSce.grRules));

Next, find all reactions containing the keywords “transport”, “diffusion”, or “pseudoreaction” in their names, and obtain their corresponding reaction indexes.

rxnIdx=regexp(modelSce.rxnNames,'(transport)|(diffusion)|(pseudoreaction)');

rxnIdx=find(~cellfun('isempty',rxnIdx));

Determine which reaction indexes are both non-gene-annotated and contain the aforementioned keywords. Using these reaction indexes, obtain reaction IDs from the template model.

rxnIdx=intersect(rxnIdx,noGeneIdx);

exRxns=modelSce.rxns(rxnIdx);

Finally, use the addRxnsGenesMets function to import required non-gene-annotated reactions.

modelHpoHomology1=addRxnsGenesMets(modelHpoHomology,modelSce,exRxns,false,'Non-gene-annotated reaction from S.ce template model',1);

#### 3.3.3 Generate GEM from homology using *R. toruloides* template

In the section we mimic the steps taken in 3.3.2 in order to generate a draft model for our target organism, this time using *R. toruloides* as a homologous template model.

First obtain blast structure:

hpoRhtoBlast = getBlast('hpo','hpo.faa',{'rhto'},{'rhto\_np11.faa'});

Next use the getModelFromHomology function:

modelHpoHomology2=getModelFromHomology({modelRhto},hpoRhtoBlast,'hpo',{},1,false,10^-20,100,35,true);

### **3.4 Merge models**

Note that you can also generate models using KEGG and Metacyc, which could be merged into the homology models. Please refer to the additional notes in section 4.4 to read about these other methods.

Use mergeModels function to merge the two homology-derived drafts:

modelHpoMerged = mergeModels({modelHpoHomology1,modelHpoHomology2})

Contract obtained model, as iso-enzymes have been expanded to separate reactions:

modelHpoMerged = contractModel(modelHpoMerged)

Change ID from MERGED to hpo and add model description:

modelHpoMerged.id = 'hpo';

modelHpoMerged.description = 'H. polymorpha GEM genearated by merging 2 homology based models from S.ce and Rh.to';

### **3.5 Curate biomass stoichiometry**

We now have an almost functional draft model for our target organism, *H. polymorpha*. One of the final tasks is to generate an organism specific biomass equation. This generally requires some experimental data such as dry cell weight fractions of macromolecular components and omics datasets. The generation of of organism specific biomass objective function falls outside the scope of the RAVEN toolbox. However, it is possible to manually curate the template model’s biomass stoichiometric coefficients to use as a starting point. There are tools to help to define species specific biomass composition, while it also often involves manual curation using literature data. Here we demonstrate the use of BOFdat to obtain some species specific biomass objective function stoichiometric coefficients for *H. polymorpha*.

#### 3.5.1 BOFdat

As was previously mentioned, this tool estimates organism specific stoichiometric coefficients for the biomass equation of a GEM based on omics data. To make full use of BOFdat’s functionality it is necessary to obtain growth and uptake rate, transcriptomic, lipomic, and proteomic datasets. These datasets can be generated experimentally or found in the scientific literature. As this tool fall outside the scope of the RAVEN toolbox, we will restrict our demonstration to obtaining DNA stoichiometric coefficients only.

The BOFdat tool makes use of BIGG IDs to identify metabolites. As our template models did not have BIGG IDs, we will therefore need to convert some metabolite IDs. Since we will only be using BOFdat to obtain coefficients for DNA precursors, we only need to convert the IDs for these metabolites (i.e. dATP, dGTP, dTTP and dCTP). For simplicity, we will make a backup copy of our draft model (.xml format) and only make the metabolite ID modifications to one of the copies, which will be discarded after serving its purpose. Please refer to the additional notes section 4.5 to see an example of how this metabolite renaming could be accomplished using unix.

Next, launch Python and import the following modules:

from BOFdat import step1

from BOFdat.util import update

import pandas as pd

import cobra

This tool requires the user to provide estimates of the dry cell weight composition for the target organism. This includes DNA, RNA, protein, lipid, and metabolite fractions, which should be provided in the following manner:

dna\_weight\_fraction = 0.05

rna\_weight\_fraction = 0.1

protein\_weight\_fraction = 0.13

lipid\_weight\_fraction = 0.01

metabolite\_weight\_fraction = 0.71

In addition to providing the template model, BOFdat requires a DNA fasta file and a genbank file for our target organism. These should be provided in the following way:

genome = '/path/to/file/BOFdat/hpo.fna'

genbank = '/path/to/file/BOFdat/hpo.gbff'

model = '/path/to/file/BOFdat/modelHpoMerged.xml'

Next, use the step1.generate\_dna\_coefficients function to generate estimates for the DNA stoichiometric coefficients of the biomass equation.

dna\_coefficients=step1.generate\_dna\_coefficients(genome,model,DNA\_WEIGHT\_FRACTION=dna\_weight\_fraction)

By calling the newly defined dna\_coefficients variable, we can see the newly calculated DNA stoichiometric coefficients for the biomass equation of *H. polymorpha*.

>>> dna\_coefficients

{<Metabolite datp\_c at 0x7f2b6904fd90>: -0.00855716169972182, <Metabolite dctp\_c at 0x7f2b6904fe90>: -0.008628417496747516, <Metabolite ppi\_c at 0x7f2b690aa250>: 0.03354733319457753, <Metabolite dgtp\_c at 0x7f2b69064250>: -0.007579394372551919, <Metabolite dttp\_c at 0x7f2b69064910>: -0.00878235962555628}

Although we restrict ourselves to demonstrating the acquisition of DNA stoichiometric coefficients for the biomass equation, we recommend users to make full use of the BOFdat workflow if possible.

#### 3.5.2 Manual curation

Since we did not carry out the complete BOFdat workflow, which yields a model with updated biomass stoichiometric coefficients, we need to manually change these coefficients in MATLAB. Checking the excel file version of our model, we can identify r\_4050 as corresponding to the DNA pseudo-reaction necessary for growth. We can easily find the index for this reaction using the following one-liner:

rxnIndex= getIndexes(modelHpoMerged,{'r\_4050'},'rxns')

Next, double check that the stoichiometry indeed matches the equation found in the excel version of our model with the following line:

modelHpoMerged.S(:,index)

Finally, we can manually edit the stoichiometric matrix to implement BOFdat output.

modelHpoMerged.S(380,index)= -0.00855716169972182;

modelHpoMerged.S(385,index)= -0.008628417496747516;

modelHpoMerged.S(397,index)= -0.007579394372551919;

modelHpoMerged.S(419,index)= -0.00878235962555628;

Note about not using phophate sc, dxTP vs dxMP.

### **3.6 Gap-filling**

Homology based draft reconstructions will generally tend to have many gaps within their metabolic network. These gaps can be thought of as missing reactions that are necessary for the production of biomass precursors. To convert a draft model into a functional model, we perform automated gap-filling. While the RAVEN toolbox provides a fillGaps function, it is generally unable to gap-fill draft models with extensive gaps present within their metabolic network. Therefore it is often necessary to combine multiple gap-filling approaches. In this protocol, we make use of the gap-filling tool meneco. After most of the network gaps have been filled with meneco, we finally use the RAVEN fillGaps function to generate a functional model. Refer to the additional notes section 4.6 to read about other gap-filling-related RAVEN functions not explicitly used in this protocol.

#### 3.6.1 meneco

This tool takes four input arguments: the draft model to be gap-filled, a functional template model, an SBML format seed file, and an SMBL format target file. The seed file simply contains information about which metabolites are available in the environment or media for the organism to uptake. The target file contains information about which metabolites the organism or model should be able to produce, in this case we selected all the biomass precursors. Refer to the GitHub repository to inspect the structure of these files.

First we generate the contents of the targets file in MATLAB using the following code.

rxnIdx=find(contains(modelHpoMerged.rxnNames,'pseudoreaction'));

targets=find(any(modelHpoMerged.S(:,rxnIdx)<0,2));

[modelHpoMerged.mets(targets), modelHpoMerged.metNames(targets)]

targetSBML=strcat('<species id="M\_',modelHpoMerged.mets(targets),...

'" name="',modelHpoMerged.metNames(targets),'"/>');

The information stored in targetSBML can then be used to generate a properly formatted SBML targets file. The seed file was directly taken from \*elaborate on how it was made\*.

Next, run meneco on the command line. Ensure that all files are present in the directory you are working in, otherwise specify the full file path for each file.

meneco.py -d modelHpoMerged.xml -r modelSce.xml -s seeds.sbml -t targets.sbml > meneco.out

The meneco.out file may contain some warning about exchange reactions, these can be safely ignored. The information we require for this protocol can be found at the end of the meneco.out file. More specifically, we need the list of reactions which follow “Computing union of reactions from all completion with size X ... done.”. In this case, it is a list of 77 reactions with the format "R\_r\_XXXX". Copy this list into a new text file named rxns.txt, and remove all quotation marks, as well as the unnecessary “R\_” prefix from all entries.

Next, import the newly created rxns.txt file into MATLAB using the following:

fid = fopen ('C:\path\to\file\rxns.txt');

menecoRxns = textscan(fid,'%s'); fclose(fid);

menecoRxns = menecoRxns{1};

Finally, obtain the corresponding required reactions from the template model using getAllRxnsFromGenes, and add them to the draft model using addRxnsGenesMets.

menecoRxns=getAllRxnsFromGenes(modelSce,menecoRxns);

modelHpoGF=addRxnsGenesMets(modelHpoMerged,modelSce,menecoRxns,true,...

'Identified by MENECO to produce biomass components',1); % Add reactions and metabolites

Inspecting the meneco.out file, we can see that the gap-filled network is still unable to produce two metabolites necessary for growth. In our case, this information appears on lines 997-999 of meneco.out. Note that we are using the addRxnsGenesMets to define a new model called “modelHpoGF” instead of “modelHpoMerged”.

#### 3.6.2 RAVEN

Since the draft model remains unable to produce biomass, we turn to the RAVEN fillGaps function to generate a functional model. First, force model to push flux through the biomass reaction by setting the lower bound of this reaction to an arbitrary positive number:

modelHpoGF = setParam(modelHpoGF,'lb','r\_4041',0.01);

Next, run the fillGaps function on the newly constrained model, which is now forced to produce biomass.

[newConnected, cannotConnect, addedRxns, modelHpoGF, exitFlag]=fillGaps(modelHpoGF,modelSce,false,true);

Now we reset the lower bound of the biomass reaction to zero in the new gap filled model to test if growth is achievable.

modelHpoGF = setParam(modelHpoGF,'lb','r\_4041',0);

Ensure that the objective function set to growth.

modelHpoGF=setParam(modelHpoGF,'obj','r\_4041',1);

Perform FBA to solve for and print fluxes:

sol=solveLP(modelHpoGF,1);

printFluxes(modelHpoGF,sol.x);

Note that the exchange flux value for growth is non-zero and positive, signifying that our model has been successfully gap-filled to produce biomass.

### **3.7 Save to GitHub**

In a boxed text, give overview of GitHub standardized commit terminology. Track changes using txt-and yaml-versions of the model. Make first commit with homology draft model. Alternatively, this could also be addressed in the section *Curation and maintenance*

### **3.8 Simulation**

Perform some basic simulations to demonstrate that the model works. No need for very advanced analysis, that's not the focus of this paper.

Before running a test flux balance analysis (FBA) simulation to check that the model is functional, we need to constrain the exchange reactions as shown below:

idx=getExchangeRxns(modelPch,'in');

modelPch = setParam(modelPch,'eq',idx,0);

modelPch = setParam(modelPch,'ub',{'piIN','nh3IN','hno3IN','hno2IN',...

'o2IN','slfIN','h2sIN','sulfurIN','h2so3IN','thmIN','pimIN'},1000);

modelPch = setParam(modelPch,'ub',{'dglcIN'},1);

First, we use the getExchangeReactions function to define a vector, *idx*, containing the reaction identifiers for all exchange reactions going into the GEM. Next, we use the setParam function, this time to set the upper and lower bounds of all exchange reactions going into the GEM to zero. The following line of code uses setParam yet again, this time to set the upper bound of some metabolites, which are needed for growth, to a value of 1000. Essentially, this means that these fluxes are unbounded, and will be constrained by the carbon source uptake. Note that the units of flux, and therefore the units of the lower and upper bounds, are in terms of mmol\*gDCW-1\*hr-1. The only exception is the growth flux, which is in units of gDCW\*gDCW-1\*hr-1, i.e. hr-1. Next, we set the upper bound of our carbon source, in this case glucose, to a value of 1. Note that we can use either the reaction identifier or reaction name as an argument for the setParam function.

Now that the model has been properly constrained, we move on to check whether or not the linear algebra problem defined by the model stoichiometry and constraints is solvable.

Next, the objective function of the model is set to growth using the setParam function. This function takes four input arguments: the model to be adjusted, the parameter type to be adjusted, the reaction or reaction list to adjust, and the magnitude of the value to be used.

Now that the model has an objective function to optimize, we move on to check whether or not the linear algebra problem defined by the model’s stoichiometry and constraints is solvable.

sol=solveLP(modelSce,1)

printFluxes(modelSce,sol.x)

To solve the linear programming problem defined by the objective function, we use the solveLP function, and store the solution in a structure called *sol*. This structure is composed of the fields *sol.f*, containing the negative value of the calculated objective function solution, and *sol.x*, containing the calculated flux values for all other reactions. Next we use the printFluxes function to see the calculated exchange reaction values in the command window. This function takes the appropriate model structure and *sol.x* field as input arguments.

## **4 Notes**

### 4.1 RAVEN documentation

With the extensive functionality of RAVEN Toolbox, users might find it useful to consult the documentation that can be found in the docs folder. The RAVEN repository on GitHub has a Wiki with useful information, while the Issues section can be used to report bugs. As example, the Text Analytics Toolbox of MATLAB 2017b and more recent versions has been identified as conflicting with RAVEN Toolbox and a workaround is provided here <https://github.com/SysBioChalmers/RAVEN/issues/55>. Additionally, help can be requested a chat forum at <https://gitter.im/SysBioChalmers/RAVEN>.

### 4.2 Example: Matching protein FASTA IDs using sed

As discussed in section 3.3.1, it is essential to ensure that protein FASTA IDs match within every model-protein FASTA file pair. One of many possible ways of accomplishing this is to edit the protein FASTA files using the sed command in unix.

Let us first address the S. cerevisiae model-FASTA file pair. Considering that we simply wish to remove everything after the first whitespace in every protein ID, we can simply use:

sed -i 's/ .\*$//' sce.faa

This replaces instances such as “>YAL001C TFC3 SGDID:S000000001, Chr I from 151006-147594,151166-151097, Genome Release 64-1-1, reverse complement, Verified ORF, "Largest of six subunits of the RNA polymerase III transcription initiation factor complex (TFIIIC); part of the TauB domain of TFIIIC that binds DNA at the BoxB promoter sites of tRNA and similar genes; cooperates with Tfc6p in DNA binding"” , with the corresponding protein identifier, in this case simply: “>YAL001C”.

This procedure should be carried out for each model-FASTA pair. Since the *R. toruloides* FASTA file has already been cleaned up to match the model IDs, we proceed to the last model-FASTA file pair corresponding to H. polymorpha. In this case, ID cleanup can be accomplished using sed as shown below.

sed -i 's/jgi|//' hpo.faa

sed -i 's/|[^|]\*//2g' hpo.faa

sed -i 's/|/\_/' hpo.faa

This should replace instances such as “>jgi|Hanpo2|5743|gm1.1\_g” with the corresponding identifier, in this case: “>Hanpo2\_5743”.

### 4.3 RAVEN function: getModelFromHomology

As can be seen from the example usage in section 3.3.2, this function takes a total of ten input arguments. The first three correspond to: a cell structure containing the previously discussed template model, the aforementioned bidirectional BLAST structure, and a character string containing the organism ID for the target organism. Note that this target organism ID *must* match with the target organism ID provided in the input arguments of the getBlast function described above. The fourth input argument, left intentionally blank for the default value, corresponds to the prefered order of template models from which to add reactions. Since we only use one template model in this case, we need not concern ourselves with this option. The next two input arguments correspond to options that tweak the criteria for adding reactions from the template model to the target model. The default values are used in our case. Please refer to the help documentation for more detailed descriptions of these options. The seventh argument, which corresponds to the max E-value threshold, is set to 10-20, instead of the default 10-30. This means that the function will only look at genes with an E-value less than or equal to 10-20, resulting in a less strict E-value cutoff. The next input argument, which corresponds to the minimum alignment length cutoff, is set to 100 instead of the default 200. Similarly to the last parameter, a less strict cutoff is used to capture more information from the template model. The ninth input argument, which refers to the minimum gene identity cutoff, is set to 35% instead of the default 40%. Again, this tweak is implemented to reduce the strictness of gene filtering. The final input argument is used to determine how to match genes if there are no one-to-one orthologs. The default value is used, which establishes that new genes should be mapped to old genes.

### 4.4 Generating and merging additional models

Besides making model from homology, RAVEN can also generate de novo reconstructions using KEGG or MetaCyc as reaction databases. For this, one uses the functions getModelFromKEGG or getModelFromMetaCyc, only requiring the fasta file of the organism of interest. This is of particular interest to find reactions that are not part of any template model. Database models can be easily generated, but can be trickier to merge with homology based models. When attempting this, it is strongly advised to pay close attention to ensure matching of metabolite, reaction, and gene IDs. When combining reactions from different models, it’s crucial that the metabolites are matched. Otherwise there could be two copies of the same metabolite, with different names or metabolite IDs. Depending on the function used for merging models, metabolites are matched by name and/or metabolite ID. In the example presented for this protocol, we did not require any metabolite ID adjustment since both of the template models use the same naming convention. In the case of merging models with different metabolite IDs, e.g. merging a KEGG-based model with a MetaCyc-based model, one could potentially adjust metabolite IDs by mapping to the MetaNetX database.

### 4.5 Automatically translating metabolite IDs using unix

In order to translate the metabolite IDs, one could for example create a conversion table with two columns. Each row should correspond to a particular metabolite, and one column should have BIGG IDs while the other should have the metabolite IDs present in your given model. This table can then be fed to a while loop that replaces our metabolite IDs with the required BIGG IDs. An example of how this could be done using sed in command line is shown below:

while read line ;do

BIGG=$(echo "$line"|cut -f 1);

MODEL=$(echo "$line"|cut -f 2);

sed -i "s/$MODEL/$BIGG/g" modelHpoMerged.xml;

done < conversionTableDNA.txt

### 4.5 Gap-filling with RAVEN

gapReport, canProduce, canConsume,

Additional notes:

* Localization of the reactions is also crucial. Here, we assumed that localization can be assumed to be identical as in the template model. However, predictLocalization can be used to use localization predictions to assign reactiosn to different compartments.
* Merging Compartments:

Next, we will merge the compartments of our template model in order to allow for easier model merging in section seven. The rationale behind this is that merging compartments does not change the biosynthetic capabilities of the model, and it is easier to combine models if all the reactions are in one compartment. For this purpose, we use the mergeCompartments function. Localization of reactions will be reestablished in section eight of this protocol, after the models derived from different approaches are combined.

[modelSce,deletedRxns,duplicateRxns]=mergeCompartments(modelSce,true)

As can be seen above, the mergeCompartments function takes two input arguments and has three output arguments. The first input argument corresponds to the model structure whose compartments are to be merged, while the second input corresponds to an option to keep unconstrained metabolite exchange reactions. There are in fact several more options for this function, but since the default values are used for these options, we can omit their inclusion in the input arguments of the function. Refer to the help documentation to learn about the other options for this function. Regarding the output arguments, these correspond to: the resulting model structure with merged compartments, a list of reactions which were deleted due to having only one metabolite, and a list of reactions which were deleted for having duplicates after the merging of compartments.

## **5 References**

**Springer Guidelines Summary**

**0. Cover Page**

* Names
* Affiliations
* Email of corresponding author

**i. Running head**

* Short title at the top of every page of chapter

**ii. Abstract**

* 1-2 Paragraphs, will appear on sites such as pubmed

**Iii. Key Words**

* 5-10 buzzwords/key terms for search engines

**Structure**

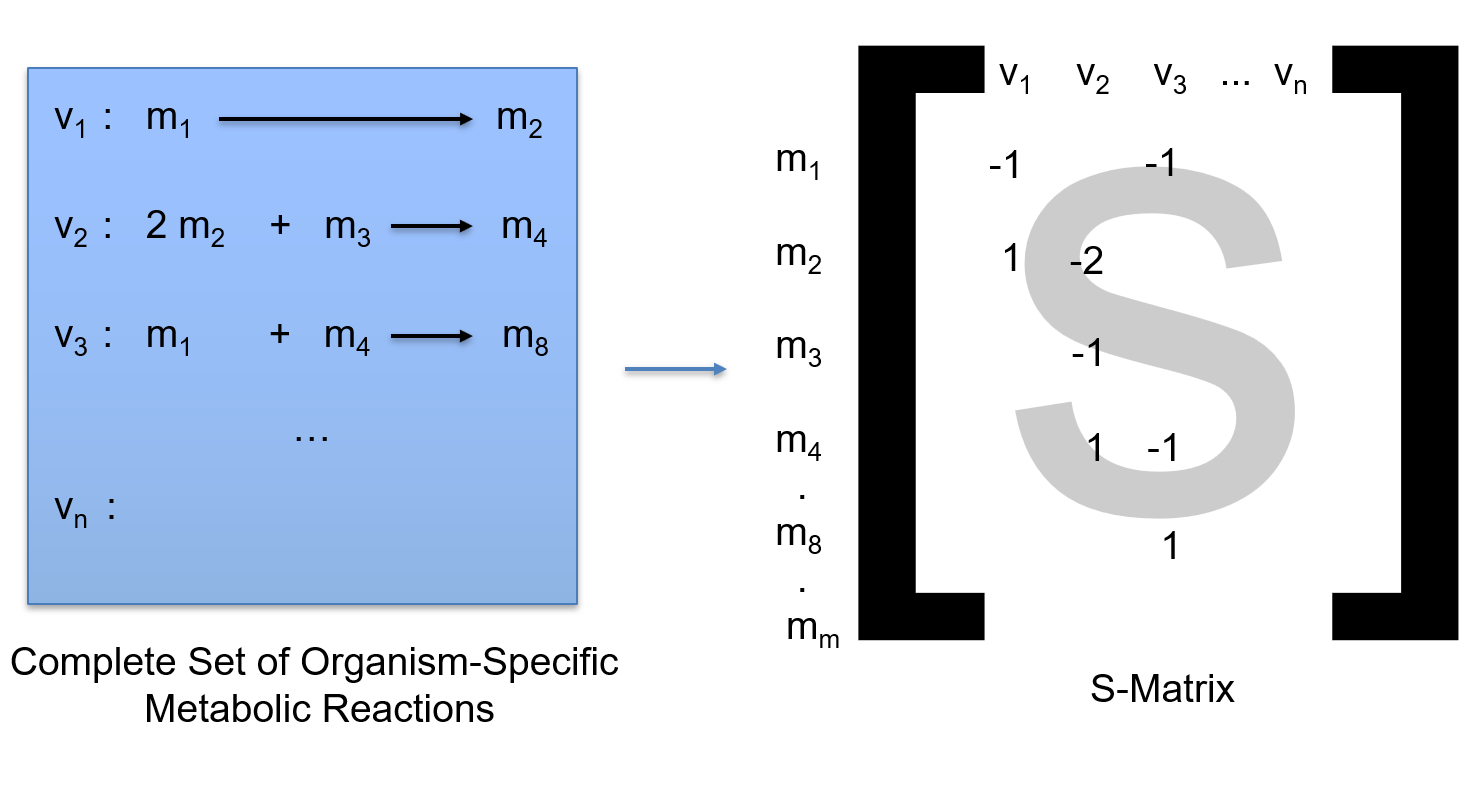
1. Introduction
2. Materials
3. Method
   * 3.1 INSTALL RAVEN
   * 3.2 IMPORT TEMPLATE MODELS
     + 3.2.1 SCE
     + 3.2.2 RHTO
   * 3.3 GENERATE TARGET MODELS FROM HOMOLOGY
     + 3.3.1 SCE
     + 3.3.2 RHTO
   * 3.4 MERGE MODELS
   * 3.5 GENERATE BOF
   * 3.6 GAP-FILLING
   * 3.7 SAVE TO GITHUB
   * 3.8 SIMULATIONS
4. Notes
   * Indicate common errors, discuss possible solutions, suggest alternative approaches, give tips & tricks, etc
5. References
   * Ordered as they appear in text. Use bold italic number in parenthesis in text.

**Things to bear in mind**

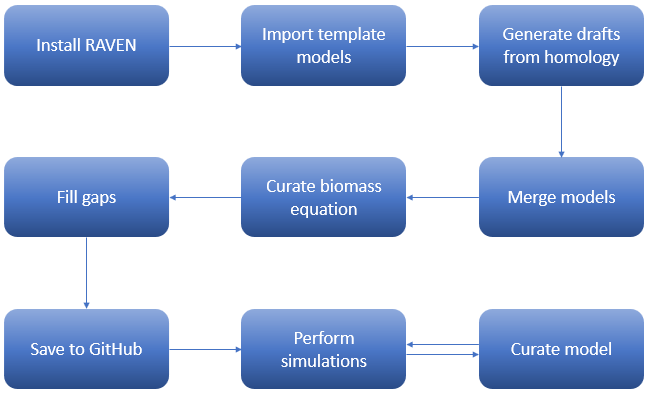
* Submit protocol as MS WORD document, double spaced, dont justify or indent text
* Submit images as separate documents, ideally TIFF format @ 600 dpi, dont make too small, make sure they are referenced in text

**Figure Ideas**

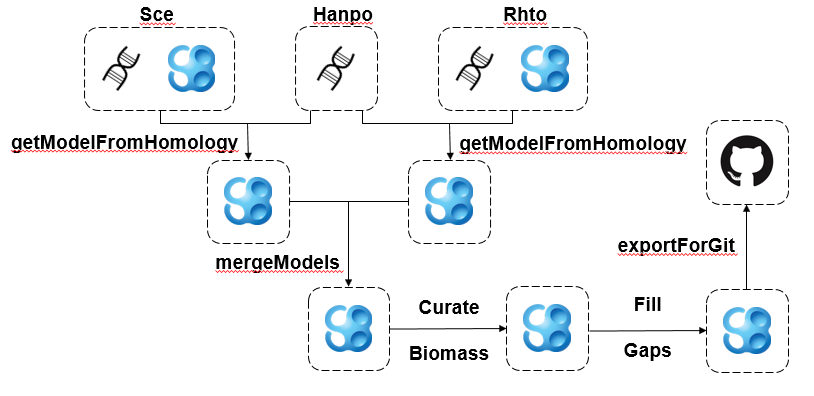
1. Considering the audience will be mostly wet lab people, In the introduction we may want to have some visual showing the connection between reactions/metabolites in cells and how they are ordered in S matrix form. Maybe also showing some other aspects of GEMs like gene annotation or constraints like reversibility. Very rough sketch of idea below. A small example, like a glycolysis reaction or two might be useful for explaining this concretely.



1. Overview of protocol steps as a flow diagram, very rough sketch below. Show all steps and suggest uses for GEM at the end. Add more detail? Represent in different way? Include step number inside each box?



1. Detailed flow diagram of how different files are used by different functions (indicated in consolas font).



Key:

* HM1 = Homologous Model 1 (Sce)
* HM1F = Homologous Model 1 protein Fasta file (Sce)
* TF = Target protein Fasta file (Hpo)
* TMH1= Target Model from Homology 1 (Hpo)
* HM2 = Homologous Model 2 (Rto)
* HM2F = Homologous Model 2 protein Fasta file (Rto)
* TMH2 = Target Model from Homology 2 (Hpo)
* ITM = Intermediate Target Model (Hpo)
* TM = Target Model

# 