Chapter 21

Two-Scale ^{13}C Metabolic Flux Analysis for Metabolic Engineering

David Ando and Hector Garcia Martin

Abstract

Accelerating the Design–Build–Test–Learn (DBTL) cycle in synthetic biology is critical to achieving rapid and facile bioengineering of organisms for the production of, e.g., biofuels and other chemicals. The Learn phase involves using data obtained from the Test phase to inform the next Design phase. As part of the Learn phase, mathematical models of metabolic fluxes give a mechanistic level of comprehension to cellular metabolism, isolating the principle drivers of metabolic behavior from the peripheral ones, and directing future experimental designs and engineering methodologies. Furthermore, the measurement of intracellular metabolic fluxes is specifically noteworthy as providing a rapid and easy-to-understand picture of how carbon and energy flow throughout the cell. Here, we present a detailed guide to performing metabolic flux analysis in the Learn phase of the DBTL cycle, where we show how one can take the isotope labeling data from a ^{13}C labeling experiment and immediately turn it into a determination of cellular fluxes that points in the direction of genetic engineering strategies that will advance the metabolic engineering process.

For our modeling purposes we use the Joint BioEnergy Institute (JBEI) Quantitative Metabolic Modeling (jQMM) library, which provides an open-source, python-based framework for modeling internal metabolic fluxes and making actionable predictions on how to modify cellular metabolism for specific bioengineering goals. It presents a complete toolbox for performing different types of flux analysis such as Flux Balance Analysis, ^{13}C Metabolic Flux Analysis, and it introduces the capability to use ^{13}C labeling experimental data to constrain comprehensive genome-scale models through a technique called two-scale ^{13}C Metabolic Flux Analysis (2S-^{13}C MFA) [1]. In addition to several other capabilities, the jQMM is also able to predict the effects of knockouts using the MoMA and ROOM methodologies. The use of the jQMM library is illustrated through a step-by-step demonstration, which is also contained in a digital Jupyter Notebook format that enhances reproducibility and provides the capability to be adopted to the user's specific needs. As an open-source software project, users can modify and extend the code base and make improvements at will, providing a base for future modeling efforts.

Key words Flux analysis, ^{13}C Metabolic flux analysis, Omics data, Predictive biology

Electronic supplementary material: The online version of this chapter (https://doi.org/10.1007/978-1-4939-7295-1_21) contains supplementary material, which is available to authorized users.

1 Introduction

The capability to change an organism's DNA through genetic engineering has radically changed the nature of biology in the last few decades. Synthetic biology was born in the twenty-first century as a reinterpretation of genetic engineering applying systematic design [2] and traditional engineering principles. One of those engineering principles is the Design–Build–Test–Learn (DBTL) cycle: a loop used recursively to obtain a design that satisfies the desired specifications [3]. The DBTL cycle starts with the design (D) of the biological system to produce the desired outcome. That design is built (B) in the next phase from DNA parts and an appropriate microbial chassis using synthetic biology tools. The next phase involves testing (T) whether the biological system performs as desired in the original design using a variety of assays (e.g., production measurement or/and omics profiling). It is extremely unlikely that the first design behaves as desired, and further attempts will most likely be needed to meet the desired specification. It would be desirable not to do these posterior attempts randomly, but rather to use the data generated in previous rounds to converge towards engineering goals more quickly. This phase is called the learn (L) phase of the DBTL cycle and is, arguably, the hardest and most weakly supported step in current metabolic engineering practice [3]. However, given the ever increasing amounts of data provided by the postgenomics revolution and current increasingly available high-throughput workflows, there is an imperative need to efficiently use the test data to provide *actionable* items for metabolic engineers: i.e. suggestions that can be acted upon with available tools and protocols (e.g., to change a particular gene's RBS or knock out a particular gene in order to increase a specific flux or to accelerate growth).

In this chapter we will show how to use metabolomic data obtained from ^{13}C labeling experiments to generate actionable items to increase acetate production in *E. coli*. We will use the JBEI Quantitative Metabolic Modeling library (jQMM) [4] to calculate cellular fluxes and make predictions. The jQMM library is currently capable of measuring and predicting internal metabolic fluxes using three different techniques: ^{13}C Metabolic Flux Analysis (^{13}C MFA) [5], Flux Balance Analysis (FBA) [6], and two-scale ^{13}C Metabolic Flux Analysis (2S-^{13}C MFA) [1]. First we will provide a brief description of ^{13}C labeling experiments, which provide the needed experimental data, and which consist of cellular cultures in which the feed (e.g., glucose) is labeled with carbon atoms that have an extra neutron (i.e. carbon isotopes) at selected positions. We will then succinctly describe how to measure the ensuing labeling in the metabolites in the studied cells (these are steps more appropriately described as part of the test phase). Next we describe in detail how to use the labeling data from different metabolites in the cell to infer what the cell's internal metabolic fluxes are through a technique called two-scale ^{13}C Metabolic Flux Analysis (2S-^{13}C MFA). 2S-^{13}C MFA introduces the capability to use experimental ^{13}C labeling data to constrain comprehensive *genome-scale* models (rather than small models of central metabolism as done with traditional ^{13}C MFA) by taking into account the system-wide balances of metabolites [7]. Finally, we will show how to use the COBRA (Constraint-Based Reconstruction and Analysis) [8] methods of MoMA (Minimization of Metabolic Adjustment) [9] and ROOM (Regulatory On/Off Minimization) [10] to predict, based on the measured flux profiles, which gene knockouts will increase acetate production in *E. coli*.

2 Materials

The general workflow for determining and plotting of metabolic fluxes is shown in Fig. 1. First, a microbial culture is grown with ^{13}C-labeled glucose. Next, mass spectroscopy is used for the analysis of the distribution of ^{13}C in metabolites taken from cell culture

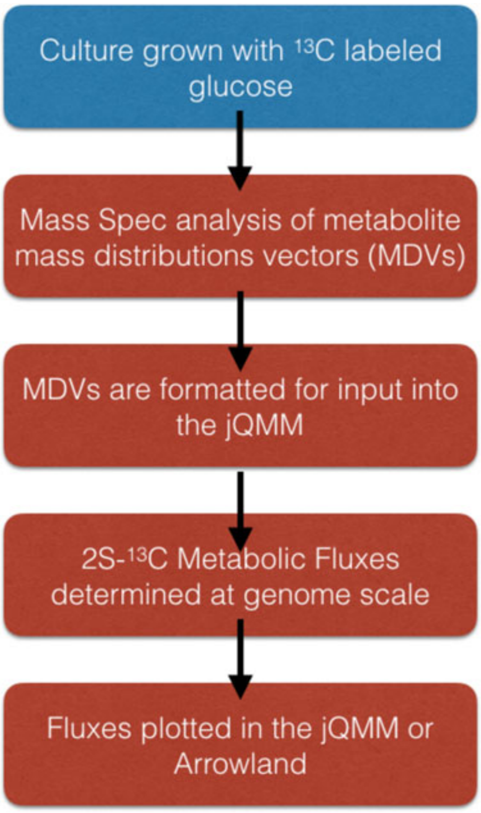
Culture grown with 13C labeled glucose
Mass Spec analysis of metabolite mass distributions vectors (MDVs)
MDVs are formatted for input into the jQMM
2S-13C Metabolic Fluxes determined at genome scale
Fluxes plotted in the jQMM or Arrowland

Fig. 1 Overview of the workflow for ^{13}C Two-Scale Metabolic Flux Analysis. The Test phase of the workflow is in *blue* while the Learn phase steps are in *red*

to create Mass Distributions Vectors (MDVs) [11] for each measured metabolite, which give the relative frequency of isotopomers. Each isotopomer of a metabolite has a different number of carbons with an extra neutron, and each isotopomer can have the heavy carbon(s) at any carbon(s) located within the metabolite. The different labeling patterns for the measured metabolites will ultimately provide enough information to determine the internal flux profile [1]. MDVs then need to be formatted for input into the jQMM, so that metabolic fluxes can be calculated at the genome scale using the 2S-^{13}C methodology. For genome-scale models, there are thousands of reactions, so fluxes can be more easily understood via plotting in Arrowland software (http://public-arrowland.jbei.org) or within the jQMM.

2.1 Computational Requirements

Procurement of a modern desktop computer or server system that is capable of running heavy computational loads for extended periods of time is necessary. The more available cores/CPUs in the system the better, as the jQMM library is parallelized and can leverage additional cores/CPUs to reduce computation time. Windows, Mac, and Linux operating systems are all compatible with running python code, which the jQMM is written in. Error-correcting code RAM (ECC RAM) is recommended to reduce the probability of faulty calculations over lengthy computations due to errors in writing and reading to RAM memory. Xeon processors from Intel for example, are specially designed for long continuous computations at high CPU load, unlike the consumer series of desktop processors sold by Intel such as the i7 and i5 series (circa 2016).

2S-^{13}C MFA computations require the following:

1. At least 32 GB of RAM and 500 GB of free disk space.
2. Ownership of both GAMS and CONOPT solver licenses.
3. Python version 2.7 installed.

2.2 Software Installation and Configuration

Installation of the required libraries for using the jQMM can be done in two different ways: the traditional way, which includes a self-installation on the hardware at hand, or via the use of a pre-configured and preinstalled Docker container (http://www.docker.com). We recommend using our pre-configured Docker container for use of the jQMM as the flux modeling environment will be immediately usable. However, for expert users or those who wish to use a custom installation, a self-installation is readily achieved.

2.2.1 Self-Installation

First install the following jQMM dependencies to the Python 2.7 environment:

1. libSBML: available at http://sbml.org/Software/libSBML
2. matplotlib: available at http://matplotlib.org/users/installing.html

3. numpy: available at http://www.scipy.org/scipylib/download.html
4. jupyter: available at http://jupyter.org/

Download the jQMM library from https://github.com/JBEI/jqmm and unpack the downloaded file. The jQMM is best used through an interactive python (iPython) notebook server called Jupyter Notebook (http://jupyter.org/), which is a web application that allows for the interactive execution, visualization, and documentation of python code. This integrated computing format greatly enhances reproducibility of results. Next, the jQMM library can be used by logging into the local Jupyter server using a web browser and navigating to the jQMM folder, and then running some of the example Jupyter notebooks contained within the jQMM. If desired, the Jupyter server can be run directly from the command line within a linux terminal via the command "jupyter notebook". GAMS and CONOPT licenses are needed and must be obtained separately.

2.2.2 Pre-configured and Preinstalled jQMM Library

Docker (https://docs.docker.com/) is a technology that is based on Linux containers that allows for building, running, testing, and deploying applications such as the jQMM library into a complete file system that contains everything it needs to run: code, runtime, system tools, system libraries, and supporting data files. This guarantees that it will always run correctly and in the same way, regardless of the system environment it is running in. The jQMM docker container can be run on virtually any cloud computing service such as AWS (Amazon Web Services), Google Cloud Platform, and Microsoft Azure. Additionally, the jQMM Docker container can be conveniently run on a personal computer running either Microsoft Windows or the Mac operating system, although we discourage this practice for anything other than training purposes given how slowly the jQMM will run on personal computers.

If choosing to run the jQMM docker container on a web-based platform one avoids the need of having to purchase an expensive high-performance server system. Pricing for cloud computing services is typically based on usage, which allows for the ability to automatically adjust the usage of computational services to as much or as little as needed, at any time. When choosing a type of instance to use on a cloud based system we recommend instances which focus on computational speed and not RAM size or disk drive access speed. On the AWS, this includes the instances of type M4 and C4, with the C4 instance currently featuring the highest performing processors and the lowest price/compute performance ratio offered by AWS (circa 2016).

The jQMM docker container (available for download at https://github.com/JBEI/jqmm) has all of the software needed to run the jQMM library preinstalled and pre-configured to work out of the box. The GAMS and CONOPT solvers (http://www.gams.com/, http://www.conopt.com/), which are preinstalled, are required to do flux analysis, but their usage requires purchase of a GAMS license and CONOPT license separately. Once these licenses are included with the GAMS and CONOPT solver installations the jQMM docker container will be fully functional.

2.3 ^{13}C Labeling Experiments and Mass Spectrometry Data Analysis

The initial step for any ^{13}C-based metabolic flux experiment is performing a ^{13}C labeling experiment with the organism of interest. Since this step is more related to the Test phase than the Learn phase of the DBTL cycle, we will only give a brief description, and refer the reader to previous protocols [12, 13]. Cultures must use minimal media, and can be grown using different types of labeling for the feed (*see* **Note 1**). A common choice is to use 20% normal glucose and 80% 1-^{13}C glucose, for example. Briefly, we recommend that culture samples are prepared by taking an aliquot of the cultured cells which were grown with ^{13}C glucose and filtering with a 0.45-μm pore-sized filter. The filter is then washed with Milli-Q water to remove the cultured cells which are then placed in methanol at 4 °C to halt metabolism. A solution at a ratio of 4 mL of chloroform to 1.6 mL of Milli-Q water is mixed with the filtered cells and then centrifuged at 2300 × *g* for 5 min at 4 °C. To remove high-molecular weight compounds the methanol layer is extracted and then passed through a Millipore 5-kDa cutoff filter via centrifugation. Finally, the filtrate is lyophilized and then dissolved in Milli-Q water before analysis on a mass spectrometry instrument.

The quantification of relative cellular metabolite isotopomer concentration, consisting of MDVs for metabolites in cellular metabolism (*see* Fig. 2 for an example), is done by determining the

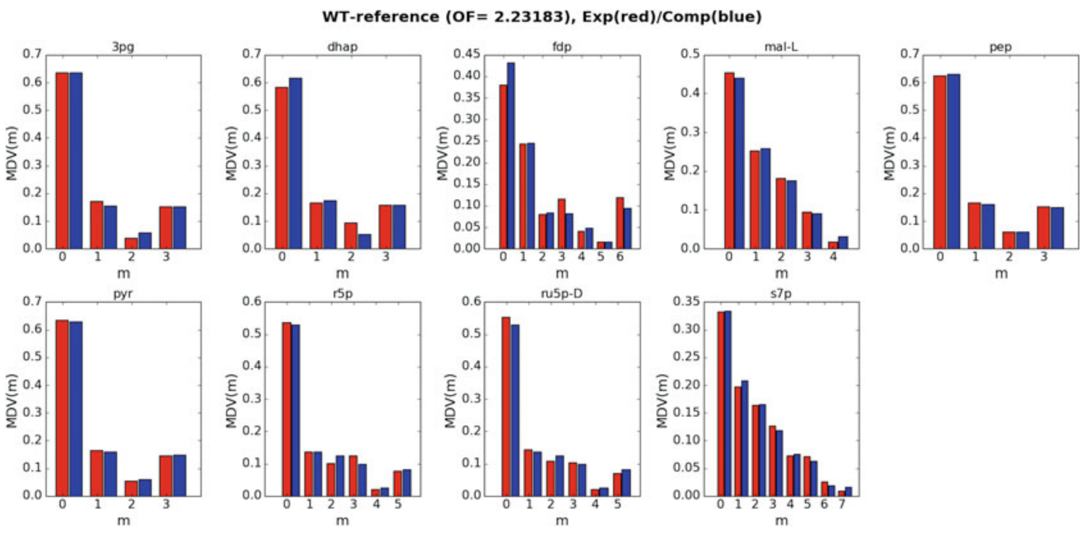

Fig. 2 Plot of wild-type *E. coli* MDVs from Toya et al. [12]: Experimentally measured MDVs are plotted in *red bars* for each metabolite, while the MDVs implied by the predicted fluxes are shown in the *blue bars*. Computational and experimental data match closely implying that the model is quantitatively correct

relative concentration of each metabolite's isotopomers via mass spectrometry. Once chromatograms from a mass spectrometry instrument have been analyzed and integrated such that the relative frequency of each metabolite's mass distribution has been determined, these data need to be prepared and formatted for input into the jQMM library so that metabolic fluxes can be determined.

2.4 Input Data Preparation

2.4.1 ^{13}C Labeling Data

Once the measured MDVs have been determined and normalized such that their sum is equal to one for each metabolite, they have to be entered into a text document with the following format:

```
Amino acid Mass distribution
       m0     m1      m2      m3      m4      m5   m6  m7   m8
3pg    M-0    0.387   0.408   0       0.204   -    -   -    -
ala-L  M-0    0.382   0.379   0.059   0.178   -    -   -    -
asp-L  M-0    0.297   0.429   0.273   0       0    -   -    -
dhap   M-0    0.348   0.464   0       0.186   -    -   -    -
```

The first two lines should remain fixed and are ignored by the jQMM library, while the following lines need to include the MDV information for every metabolite for which isotopomer data exists in the following format:

1. First, the metabolite name using the metabolite abbreviation used in the BIGG database (http://bigg.ucsd.edu) is specified and followed by a tab character.
2. Next, the tab-separated relative frequencies of each metabolite isotopomer are specified. The sum of all isotopomer frequencies needs to add to 1 for each individual metabolite.
3. Isotopomers, which do not exist for a particular metabolite, should be represented by a "-" (dash), and must not be entered in as a zero value, which would indicate that such an isotopomer does exist but is not present in the sample.

2.4.2 Choose, Edit, and Define the Metabolic Model

To define a metabolic model for the jQMM to use in the modeling of the particular organism being studied, navigate to the BIGG database (http://bigg.ucsd.edu) and download a SBML version of a metabolic model which is appropriate to the problem being studied. Smaller models run much faster in the jQMM, while more comprehensive genome-scale models are necessary for problems involving peripheral metabolism or which include the 2S-^{13}C MFA analysis methodology. Retooling of the jQMM library code may be required for the library to understand metabolite names which do not follow the naming convention used in the iJR904 metabolic model [14] format, with sample code already included for using metabolite names from the iJO1366 [15] and iAF1260 [16] models, which is located in the "sbmlio.py" python code file located in the jQMM "code/core" directory. Network reactions, which come from a heterologous pathway that has been engineered into a microbe, should be manually entered into the downloaded SBML model.

2.4.3 Defining Exchange Reaction Fluxes

Both measured fluxes through exchange reactions and the biomass flux that correspond to the time that a sample was taken for ^{13}C isotopomer analysis are detailed in a 'FLUX.txt' file. A sample exchange flux file is a follows:

```
BiomassEcoli: 0.70 [==] 0.76
GLCpts:        11.1 [==] 11.1
EX_glc(e):    -11.1 [==] -11.1
EX_ac(e):       2.6 [==] 2.6
```

The Biomass flux is in units of 1/h. and is normalized to equal the growth rate while all other exchange fluxes are in units of mMol/gdw/h (millimoles/grams of dry weight/hour). The glucose uptake rate can be measured by the HPLC determination of glucose concentration at two different times around the ^{13}C analysis sample time. If the glucose concentration is measured as g_1 and g_2 (in millimoles/volume) at times t_1 and t_2, the glucose flux can be approximated as $(g_2 - g_1)/\mathrm{gdw}_1/(t_2 - t_1)$, where gdw_1 is the grams of dry weight of cells per unit volume at time t_1. Similarly, HPLC measurements can be used to determine fluxes of acetate, lactate, and other organic acids excreted by the cell.

2.4.4 Defining Carbon Transitions in the Metabolic Model

Atom transitions can be used to represent the fate of each carbon in a reaction [5]. In the example reaction:

```
A (abc) --> B (ab) + C (c)
```

The uppercase letters represent the metabolites present in a reaction, while the lowercase letters in parentheses represent the atom transitions. (Note: It is not necessary to follow this convention of using uppercase for metabolites and lowercase for atom transitions. The parentheses delimit the start and end of each atom transition. Any alphabetic, numeric, or underscore character comprising the regular expression [a–zA–Z0–9] can be included in the metabolite names and atom transitions.) Irreversible reactions are denoted by a '−>' or '=>' arrow and reversible reactions are denoted by a '<−>' or '<=>' arrow. In the jQMM, multiple reactions are separated by carriage returns or by placing a semicolon at the end of each reaction equation.

For example, in:

```
AKGDH akg --> succoa + co2 abcde : bcde + a
```

akg gets split into succoa and co2 in reaction AKGDH, with the first four carbons going to succoa and the remaining carbon going to co2.

For 2S-^{13}C-based metabolic flux analysis in the jQMM, one needs to create a 'REACTIONS.txt' file which contains the carbon transition information for core reactions in the metabolic network and which has the following format:

```
# Metabolic Exchange fluxes
&MLIST etoh[e] 0
&MLIST ac[e] 0
&MLIST lac-L[e] 0

# Intracellular metabolite fluxes
&MLIST fdp 0
&MLIST dhap 0
&MLIST pep 0
&MLIST r5p 0
&MLIST s7p 0
&MLIST mal-L 0

# Carbon source
&SOURCE glc-D[e]

# Input Reactions
EX_glc(e) glc-D[e] <==> glcDEx abcdef : abcdef
GLCt2 glc-D[e] --> glc-D abcdef : abcdef
HEX1 glc-D --> g6p abcdef : abcdef

# Carbon Transitions
GLCpts glc-D[e] + pep --> g6p + pyr abcdef + ABC : abcdef + ABC
PGI g6p <==> f6p abcdef : abcdef
PFK f6p --> fdp abcdef : abcdef
FBA fdp <==> g3p + dhap CBAabc : abc + ABC
```

Reactions which must be specified in the 'REACTIONS.txt' file, in terms of carbon atom transition information, are those that utilize metabolites for which ^{13}C isotopomer data are input into the jQMM library and for reactions which are considered to be at the core of the metabolic network. Finally, this file also specifies a metabolite which serves as the ^{13}C carbon source (typically glucose, i.e. glc-D[e], via the &SOURCE command) and input reactions which bring this ^{13}C-labeled metabolite into the cell.

2.4.5 Defining Feed Labeling

The type of labeled glucose used in the experiment, together with its concentration relative to the amount of unlabeled glucose, is detailed in a 'FEED.txt' file. A sample feed specification file contains the following line:

```
0.4% Glucose: 30% 1-C 20% U 50% UN
```

The 0.4% at the beginning of a feed definition specifies the total glucose percentage of the initial cell culture. The percentage of 1-C glucose, which has its first carbon atom labeled, is specified next, together with the percentage of U glucose, which is uniformly labeled among all the glucose carbon atoms, and finally the percentage of normal glucose which is completely unlabeled (UN).

3 Methods

The example Jupyter notebook which is included as an attachment in the online version of this protocol, and which is also reproduced in this section, contains a description of how to use the FluxModels module in the jQMM to do 2S-^{13}C MFA and then predict the outcomes on acetate production of different reaction knockouts. The notebook provides a convenient way to reproduce results and is easily modified to fit the user's specific needs (*see* **Note 2**). It is broken into six different steps for turning experimental data into actionable predictions for increasing a targeted biochemical via genetic engineering:

1. Gathering input data
2. Creating the Reaction Network
3. Creating the two-scale metabolic model
4. Calculating internal metabolic fluxes through 2S-^{13}C MFA
5. Visualizing flux profiles
6. Predicting which genes to knock out using MoMA and ROOM

```
-- Jupyter Notebook Start --
```

Predicting KO outcomes with 2S-^{13}C MFA and COBRA methods

This Jupyter notebook presents a computable step-by-step description of how to use metabolite data from ^{13}C labeling experiments to produce actionable insights to improve acetate production in *E. coli.*

0. Setup

The first step involves specifying the correct path for the library:

```
In[1]:
%matplotlib inline
import sys, os
path = "/scratch/user"
pythonPath = path + "/quantmodel/code/core"
if pythonPath not in sys.path:
 sys.path.append(path + '/quantmodel/code/core')
os.environ["QUANTMODELPATH"] = path +'/quantmodel'
```

We then need to import the needed classes for the notebook:

```
In[2]:
from IPython.display import SVG
import FluxModels as FM
import enhancedLists, ReactionNetworks, predictions, copy,
core
```

and then move to a defined working directory where output and intermediate files will be kept:

```
In[3]:
cd /scratch/user.working_dir/tests

Out[3]:
 /scratch/user.working_dir/tests
```

1. Gathering input data

As part of the test (T) phase of the DBTL cycle, we gather all the relevant experimental data from the ^{13}C labeling experiments (*see* Subheading 2). These data involve:

1. A base *genome-scale model* that will act as the reference for all other data types [14].
2. *Exchange fluxes* containing the measured fluxes of metabolites being exchange by cells with the environment.
3. *Transition information* on the fate of each carbon in the core reaction network [17].
4. *Metabolite labeling* information in the form of Mass Distribution Vectors (MDVs).
5. *Metabolite labeling error* information.
6. *Feed labeling* information on the type of labeled glucose the cell culture was fed.

   Discussion of these data types can be seen in Subheading 2. For this demonstration, we will use the data from Toya et al. [12]:

```
In[4]:
datadir = os.environ['QUANTMODELPATH']+'/data/tests/Toya2010/
2S/wt5h/'
strain              ='wt5h'
BASEfilename        = datadir + 'EciJR904TKs.xml'
FLUXESfilename      = datadir + 'FLUX'+strain+'.txt'
TRANSITIONSfilename = datadir + 'REACTIONS'+strain+'.txt'
MSfilename          = datadir + 'GCMS'+strain+'.txt'
MSSTDfilename       = datadir + 'GCMSerr'+strain+'.txt'
FEEDfilename        = datadir + 'FEED'+strain+'.txt'
```

2. Creating the Reaction Network

Once we have gathered all the needed input files, we can condense all this information into a single sbml file. We will do this using a reaction network from the ReactionNetworks module in the jQMM library. A reaction network contains all information related to the metabolic reaction network used for the simulation:

```
In[5]:
# Load initial SBML file
reacNet = ReactionNetworks.TSReactionNetwork(BASEfilename)
# Add Measured fluxes
reacNet.loadFluxBounds(FLUXESfilename)
# Add carbon transitions
reacNet.addTransitions(TRANSITIONSfilename,translate2SBML=True)
# Add measured labeling information
reacNet.addLabeling(MSfilename,'LCMS',MSSTDfilename,min
STD=0.001)
# Add feed labeling information
reacNet.addFeed(FEEDfilename)
# Limit fluxes to 500
reacNet.capFluxBounds(500)
# Create sbml file to store the two-scale model.
# All input files are combined in a tuple of the type:
(fileName, string of contents)
SBMLfile= ('EciJR904TKs'+strain+'TS.xml',reacNet.write('to
String'))
```

3. Creating the two-scale metabolic model

The next step is to use the SBML file we just created to create a two-scale model [1] that we will use to calculate fluxes through 2S-^{13}C MFA:

```
In[6]:
TSmodel  = FM.TwoSC13Model(('EciJR904TKs'+strain+'TS.xml',
reacNet.write('toString')))
```

TSmodel now contains all the information needed to calculate fluxes along with the methods to do this calculation and other analysis [1].

4. Calculating internal metabolic fluxes through 2S-^{13}C MFA

We can now use the *findFluxesRanges* method in *TSmodel* to find the fluxes that best fit the experimentally obtained metabolite labeling data (MDVs) and find the ranges of fluxes compatible with this labeling data and the corresponding experimental error:

```
In[7]:
fluxNames = TSmodel.reactionNetwork.C13ReacNet.reactionList.
getReactionNameList(level=1)
TSresult = TSmodel.findFluxesRanges(Nrep=30,fluxNames=flux
Names,procString='proc')
```

Nrep represents the number of replicates used for the calculation. Since the problem to be solved is a nonconvex problem there is no guarantee that a single run will find the best global fit. Hence we run 30 independent processes and keep the one that best fits the data. *fluxNames* indicates the fluxes for which full flux confidence intervals will be calculated. *procString* indicates that the data (for this case) needs no derivatization correction.

We can check how accurate the model is by comparing the measured labeling distribution (MDVs, red) with the one predicted through the computational model (blue) by using the *plotExpvsCompLabelFragment* method:

```
In[8]:
%%time
TSresult.plotExpvsCompLabelFragment(titleFig='WT-reference')
```

```
Out[8]:
 CPU times: user 824 ms, sys: 12 ms, total: 836 ms
 Wall time: 831 ms
```

See Fig. 2.

or by using *plotExpvsCompLabelXvsY*, if we prefer to see these fits as an *X* vs. *Y* plot:

```
In[9]:
TSresult.plotExpvsCompLabelXvsY(titleFig = 'WT-reference')
```

```
Out[9]:
```

See Fig. 3.

The closeness of the fit data and the experimental data validate the use of this model.

Results are stored in a reaction network inside *TSresult* and can be explored through the reactionList methods.

For example, we can print the desired fluxes:

```
In[10]:
TSresult.reactionNetwork.reactionList.printFluxes(brief
="True",names="exchange")
```

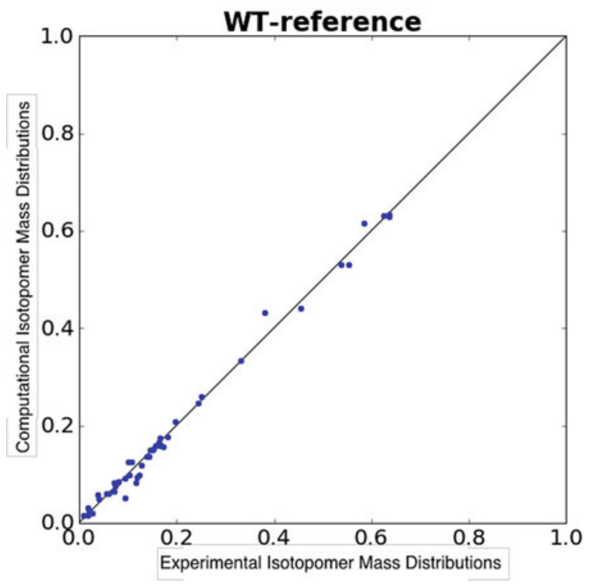

Fig. 3 Plot of experimentally measured MDVs versus the computationally predicted MDVs in an *x*–*y* plot (different way to plot data in Fig. 2), which demonstrates that experimental and predicted MDVs are comparable

```
Out[10]:
 EX_h2o_e_: 43.8140702078
 EX_co2_e_: 26.0397076138
 EX_o2_e_: -24.3853340558
 EX_h_e_: 12.3121388881
 EX_glc_e_: -11.7
 EX_nh4_e_: -8.93782429
 EX_ac_e_: 4.3
 BiomassEcoli: 0.83
 EX_pi_e_: -0.75665871
 EX_acald_e_: 0.22513003
 EX_so4_e_: -0.19343897
 EX_succ_e_: 0.145316144053
 EX_glyclt_e_: 0.0415
 EX_urea_e_: 0.03486
```

Or we can retrieve the computationally predicted labeling distribution or the experimentally measured one:

```
In[11]:
TSresult.EMUlabel['pep']
```

```
Out[11]:
 array([ 0.63083, 0.15962, 0.06024, 0.14931])
```

```
In[12]:
TSresult.fragDict['pep'].mdv
```

```
Out[12]:
 array([ 0.624, 0.165, 0.06 , 0.151])
```

An important test to make sure that the assumptions used in the 2S-^{13}C MFA properly hold is the External Labeling Variability Analysis (ELVA) [1] test. This test checks that the reactions for which no carbon transition information was available do not significantly distort the flux solution obtained.

```
In[13]:
resultELVA = TSmodel.ELVA(TSresult)
```

ELVA results can be plotted in an *x*–*y* graph showing the experimentally determined isotope labeling which defines a confidence interval that represents the maximum possible difference in labeling that could be attributed to non-core reactions for the current solution. The reactions that contribute an unacceptable amount of uncertainty are then added to the core set and the procedure can be repeated as necessary, until a core set of reactions is found which fully justifies the two-scale approximation. In this example, all reactions have only small fluctuations in predicted computational labeling.

```
In[14]:
resultELVA.plotExpvsCompLabelxvsy(titleFig="WT",outputFileNam
e="ELVAComparisonWT.txt",save="ELVA-W.eps")
Out[14]:
```

See Fig. 4.

The error bars in the *y* axis (computational error) are of the same order of magnitude as the experimental error, hence justifying the two-scale assumption [1].

5. Visualizing flux profiles

Once the metabolic fluxes have been calculated they can be understood visually via their plotting on a flux map. In the jQMM library fluxes can be plotted via the commands:

```
In[15]:
TSresult.drawFluxes('wt.svg',svgInFileName='TOYAexp.svg',
norm='EX_glc_e_')

Out[15]:
 svgin:
 /scratch/david.ando/quantmodel/code/core/TOYAexp.svg
```

where 'TOYAexp.svg' is the base flux map contained in the jQMM library [4]. The *drawFluxes()* method will indicate the flux magnitude on the base flux map in two ways: visually by changing the flux arrow width according to the flux magnitude through a reaction, and also numerically by showing the net flux value (with confidence intervals) next to the reaction:

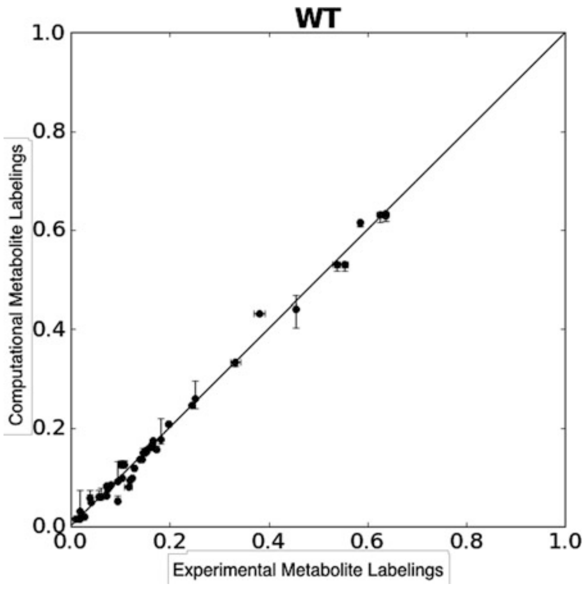

Fig. 4 ELVA plot which shows an *x–y* graph of the experimentally determined isotope labeling versus the computationally predicted labeling. The *vertical error bars* define the computational error that represent the maximum possible difference in labeling that could be attributed to non-core reactions for the current solution. In this example, all reactions have only small fluctuations in predicted computational labeling, confirming that non-core reactions do not significantly contribute to core metabolite labeling [1]

The command ‘SVG’ displays the flux map in the Jupyter notebook which is contained in the svg file which was saved locally.

```
In[16]:
SVG(filename='wt.svg')

Out[16]:
```

See Fig. 5.

In the near future, one will also be able to display fluxes using the web browser-based flux plotting library Arrowland (http://public-arrowland.jbei.org).

6. Predicting which genes to knock out using MoMA and ROOM

So far we have used targeted metabolomic data from ^{13}C labeling experiments to infer the underlying internal metabolic fluxes in the cell. We will now use these inferred fluxes along with two Constraint-Based Reconstruction and Analysis (COBRA) methods to predict which genes to knock out in order to increase the production of acetate. These methods are MoMA (Minimization of Metabolic Adjustment) and ROOM (Regulatory On/Off Minimization). MoMA provides an approximate solution for a suboptimal growth flux state after a knockout has been made to an organism, which is nearest in flux distribution to the unperturbed state [9]. On the other hand, ROOM aims to minimize the number

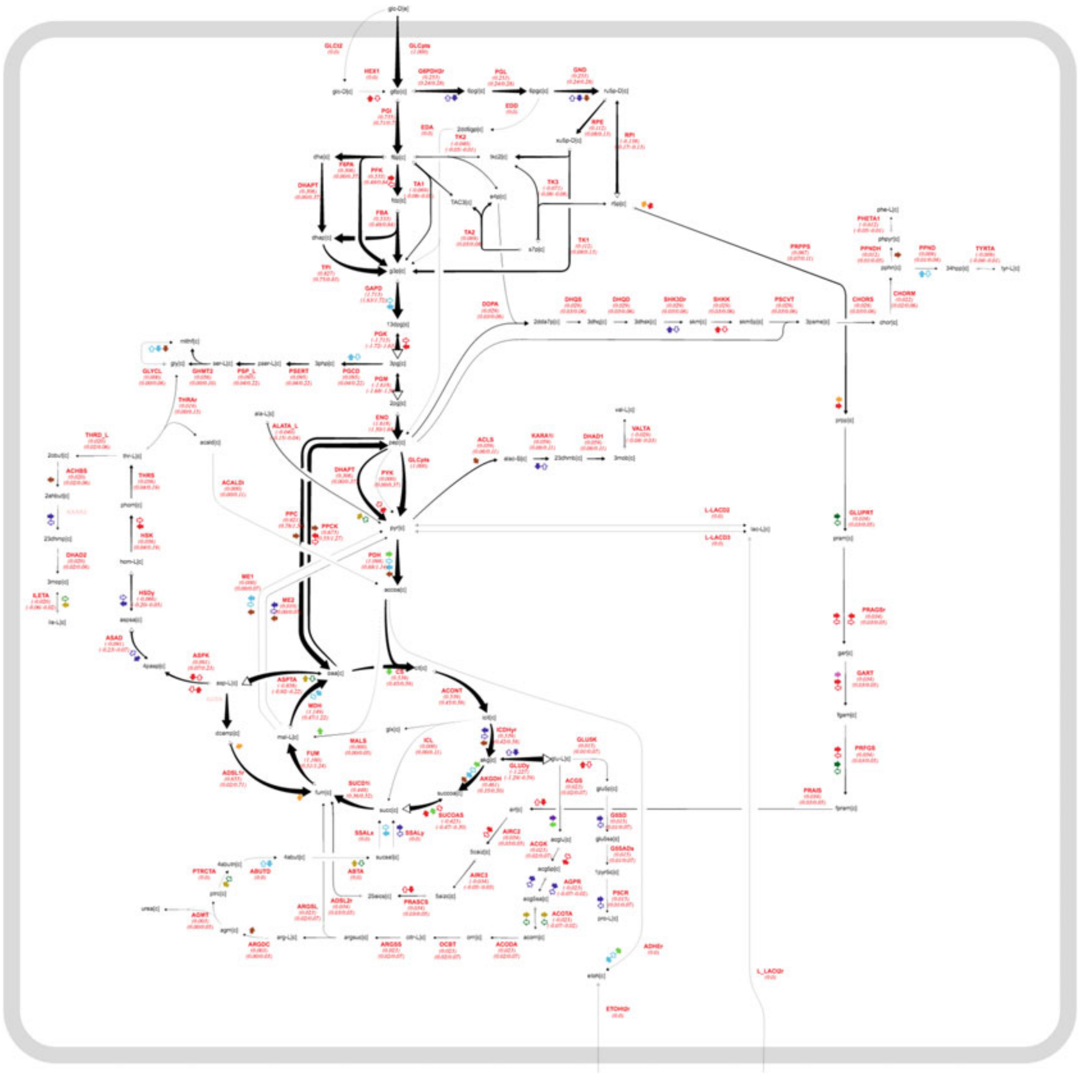

Fig. 5 Plot of 2S-^{13}C metabolic fluxes in the jQMM library using the TOYAexp.svg base flux map for a wild-type strain of *E. coli* from Toya et al. [12]. This map can be interactively studied at Arrowland (https://public-arrowland.jbei.org/, wt5h)

of significant flux changes with respect to the wild type to predict resultant fluxes from a knockout of a reaction [10].

First we need to specify flexible flux bounds for the final solution in order to avoid biasing the knockout predictions:

```
In[17]:
reactionNetwork = copy.deepcopy(TSmodel.reactionNetwork)
reactionNetwork.changeFluxBounds('GLCpts' ,core.fluxBounds
(0, 25 ,False)[1])
reactionNetwork.changeFluxBounds('EX_glc_e_' ,core.fluxBounds
(-15,-0 ,True,True)[1])
reactionNetwork.changeFluxBounds('BiomassEcoli' ,core.flux
Bounds( 0, 25 ,False)[1])
reactionNetwork.changeFluxBounds('EX_ac_e_' ,core.fluxBounds
(0, 25 ,True,True)[1])
```

Then we can calculate the base flux profiles for MoMA and ROOM:

```
In[18]:
TSresult = TSmodel.findFluxesStds(Nrep=30,Nrand=10)
```

We then specify a list of reactions to knock out and determine resultant fluxes:

```
In[19]:
KOs = ['RPE','RPI']
```

For reference, we determine the amount of acetate production in the base WT strain:

```
In[20]:
fluxDict = TSresult.reactionNetwork.reactionList.getReaction
Dictionary()
print 'predicted acetate flux = ',fluxDict['EX_ac_e_'].flux.
net.best
```

```
Out[20]:
 predicted acetate flux = 4.3
```

Perform MOMA and ROOM predictions over the set of specified knockouts:

```
In[21]:
for KO in KOs:
 print KO,'knockout:'
  TS13CMOMA = predictions.predict(TSresult, KO, 'MOMA', reac
tionNetwork.getSBMLString())
  TS13CROOM = predictions.predict(TSresult, KO, 'ROOM', reac
tionNetwork.getSBMLString())
  fluxDict = TS13CMOMA.reactionNetwork.reactionList.getReac
tionDictionary()
 print ' MoMA predicted acetate flux = ',fluxDict['EX_ac_e_'].
flux.net.best
  fluxDict = TS13CROOM.reactionNetwork.reactionList.getReac
tionDictionary()
 print ' ROOM predicted acetate flux = ',fluxDict['EX_ac_e_'].
flux.net.best
 print '———————————————————'
 print ''
```

```
Out[21]:
RPE knockout:
 MoMA predicted acetate flux = 4.56474059324
 ROOM predicted acetate flux = 6.87162731209
__________________
RPI knockout:
 MoMA predicted acetate flux = 3.9427989398
 ROOM predicted acetate flux = 4.17
__________________
```

As can be observed, knocking out the gene corresponding to the RPE reaction is predicted to increase acetate production by 6.2% according to the MoMA methodology and by 60.0% when using the ROOM methodology. As can be seen with an RPI knockout, both MoMA and ROOM predict a decline in acetate production.

```
-- Jupyter Notebook End --
```

4 Notes

1. Proper quality control of MDV data is crucial to proper determination of fluxes. Experiments should be designed to include internal controls, and should include several biological and technical replicates.
2. Free open-source software tools, such as the jQMM, provide for universal accessibility and unlimited modification and customization. Overall, we wish that the community can support the jQMM's further development by submitting bug fixes to the github repo (https://github.com/JBEI/jqmm) and including any functional extensions that different research groups have achieved.

Acknowledgments

This work was part of the DOE Joint BioEnergy Institute (http://www.jbei.org) supported by the U. S. Department of Energy, Office of Science, Office of Biological and Environmental Research, through contract DE-AC02-05CH11231 between Lawrence Berkeley National Laboratory and the U. S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

1. H. G. Martín, V. S. Kumar, D. Weaver, and A. Ghosh, A method to constrain genome-scale models with ^{13}C labeling data, 2015, 11(9): e1004363.
2. Kitney R, Freemont P (2012) Synthetic biology - the state of play. FEBS Lett 586 (15):2029–2036
3. Nielsen J, Keasling JD (2016) Engineering cellular metabolism. Cell 164(6):1185–1197
4. Birkel G, Ghosh A, Vinay K, Weaver D, Ando D, Arkin A, Keasling JD, Martin HG (2017) The JBEI quantitative metabolic modeling library (jQMM): a Python library for modeling microbial metabolism. BMC Bioinformatics 18 (1):205
5. Wiechert W (2001) ^{13}C metabolic flux analysis. Metab Eng 3(3):195–206
6. Lewis NE, Nagarajan H, Palsson BO (2012) Constraining the metabolic genotype-phenotype relationship using a phylogeny of in silico methods. Nat Rev Microbiol 10 (4):291–305
7. Ghosh A, Ando D, Gin J, Runguphan W, Denby C, Wang G, Baidoo E, Shymansky C, Keasling J, Garcia Martin H (2016) ^{13}C Metabolic flux analysis for systematic metabolic engineering of S. cerevisiae for overproduction of fatty acids. Front Bioeng Biotechnol 4:76
8. Schellenberger J, Que R, Fleming RMT, Thiele I, Orth JD, Feist AM, Zielinski DC, Bordbar A, Lewis NE, Rahmanian S, Kang J, Hyduke DR, Palsson BØ (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. Nat Protoc 6(9):1290–1307
9. Segrè D, Vitkup D, Church GM (2002) Analysis of optimality in natural and perturbed metabolic networks. Proc Natl Acad Sci U S A 99 (23):15112–15117
10. Shlomi T, Berkman O, Ruppin E (2005) Regulatory on/off minimization of metabolic flux changes after genetic perturbations. Proc Natl Acad Sci U S A 102(21):7695–7700
11. Suthers PF, Burgard AP, Dasika MS, Nowroozi F, Van Dien S, Keasling JD, Maranas CD (2007) Metabolic flux elucidation for large-scale models using ^{13}C labeled isotopes. Metab Eng 9(5–6):387–405
12. Toya Y, Ishii N, Hirasawa T, Naba M, Hirai K, Sugawara K, Igarashi S, Shimizu K, Tomita M, Soga T (2007) Direct measurement of isotopomer of intracellular metabolites using capillary electrophoresis time-of-flight mass spectrometry for efficient metabolic flux analysis. J Chromatogr A 1159(1–2):134–141
13. Zamboni N, Fendt S-M, Rühl M, Sauer U (2009) ^{13}C-Based metabolic flux analysis. Nat Protoc 4(6):878–892
14. Reed JL, Vo TD, Schilling CH, Palsson BO (2003) An expanded genome-scale model of Escherichia coli K-12 (iJR904 GSM/GPR). Genome Biol 4(9):R54
15. Orth JD, Conrad TM, Na J, Lerman JA, Nam H, Feist AM, Palsson BØ (2011) A comprehensive genome-scale reconstruction of Escherichia coli metabolism–2011. Mol Syst Biol 7(535):535
16. Feist AM, Henry CS, Reed JL, Krummenacker M, Joyce AR, Karp PD, Broadbelt LJ, Hatzimanikatis V, Palsson BØ (2007) A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information. Mol Syst Biol 3(121):121
17. Antoniewicz MR, Kelleher JK, Stephanopoulos G (2007) Elementary metabolite units (EMU): a novel framework for modeling isotopic distributions. Metab Eng 9(1):68–86