Creation and simulation of personalized microbiota models through metagenomic data integration

Author: Federico Baldini, Molecular Systems Physiology Group, University of Luxembourg.

INTRODUCTION

This tutorial shows the steps that MgPipe automatically performs to create and simulate personalized microbiota models trough metagenomic data integration. Please note that this tutorial uses as an example a small dataset (4 columns and 30 rows) with the purpose of demonstrating the functionalities of the pipeline. We recommend using high-performance computing clusters when assembling and simulating from bigger datasets.

The pipeline is divided into 3 parts:

1. **[PART 1]** Analysis of individuals' specific microbes abundances is computed. Individuals' metabolic diversity in relation to microbiota size and disease presence, as well as, classical multidimensional scaling (PCoA) on individuals' reaction repertoire are examples.
2. **[PART 2]**: 1 Constructing a global metabolic model (setup) containing all the microbes listed in the study. 2 Building individuals' specific models integrating abundance data retrieved from metagenomics. For each organism, reactions are coupled to their objective function.
3. **[PART 3]** A specific range of growth is imposed for each microbiota model and Simulations under specific diet regimes are carried. Set of standard analysis to apply to the personalized models. PCA of computed MNPCs of individuals as for example.

USAGE

Normally, once provided all the input variables in the driver (StartMgPipe), the only action required is to run the driver itself. However, for this tutorial, we will disable the autorun functionality and compute each section manually.

DRIVER

This file has to be modified by the user to launch the pipeline and to define inputs and outputs files and locations.

```
% We first set the paths to input and output files
initCobraToolbox(0);
```

```
% path to where the COBRA Toolbox is located
global CBTDIR

% path to microbe models (download AGORA models from https://www.vmh.life/#downloadview)
modPath = [getenv('HOME') filesep 'AGORA' filesep 'mat'];

% path where to save results
resPath = [CBTDIR filesep '.tmp']

% path to and name of the file with dietary information. Here,
% we will use an "Average European" diet that is located in the
% DietImplementation folder.
dietFilePath=[CBTDIR filesep 'papers' filesep '2018_microbiomeModelingToolbox' filesep 'resources' filesep 'AverageEuropeanDiet'];
```

Then we set the path and the name of the file from which to load the abundances. For this tutorial, to reduce the time of computations, we will use a reduced version of the example file (normCoverageReduced.csv) provided in the folder Resources: only 4 individuals and 30 strains will be considered. Plese, note that abundances are normalized to a total sum of one.

```
abunFilePath=[CBTDIR filesep 'tutorials' filesep 'analysis' filesep 'microbiomeModelingToolbox' filesep 'normCoverageReduced.csv'];
```

Next inputs will define:

1. name of the objective function of organisms
2. format to use to save images
3. number of cores to use for the pipeline execution
4. if to enable automatic detection and correction of possible bugs
5. if to enable compatibility mode
6. if stratification criteria are available
7. if to simulate also a rich diet
8. if to use an external solver and save models with diet
9. the type of FVA function to use to solve

The following setting should work for almost any system, but please check carefully to be sure these options are valid for you. A more detailed description of these variables is available in the documentation.

The same inputs need to be set in the driver file StartMgPipe when running mgPipe outside of this tutorial or directly in the "initMgPipe" function.

```
% name of the objective function of organisms
objre={'EX_biomass(e)'};

% the output is a vectorized picture, change to '-dpng' for .png
figForm = '-depsc';

% number of cores dedicated for parallelization
numWorkers = 2;

% autofix for names mismatch
autoFix = true;

% if outputs in open formats should be produced for each section
compMod = false;

% if documentation (.csv) on stratification criteria is available
indInfoFilePath='none';
```

```
% to enable also rich diet simulations
rDiet = false;

% if to use an external solver and save models with diet
extSolve = false;

% the type of FVA function to use to solve
fvaType = true;

% to turn off the autorun to be able to manually execute each part of the pipeline
autorun = false;

[init,modPath,~,resPath,dietFilePath,abunFilePath,indInfoFilePath,objre,figForm,numWorkers,autoFix,compMod,rDiet,extSolve,fvaType,autorun]=
```

PIPELINE: [PART 1]

The number of organisms, their names, the number of samples and their identifiers are automatically detected from the input file.

```
[patNumb, sampName, strains] = getIndividualSizeName(abunFilePath)
```

Now we detect from the content of the results folder if PART1 was already computed: if the associated file is already present in the results folder its execution is skipped else its execution starts

```
[mapP] = detectOutput(resPath, 'mapInfo.mat');

if ~isempty(mapP)
    s= 'mapping file found: loading from resPath and skipping [PART1] analysis';
    disp(s)
    load(strcat(resPath,'mapInfo.mat'))
end
```

In case PART 1 was not computed we will compute it now. We will first load the models and create a cell array containing them. This cell array will be used as input by many functions in the pipeline. Any possible constraint from each model reactions will be removed. Moreover we will run and subsequentially plot the results of some analysis that are computed. The main outputs are:

1. **Metabolic diversity** The number of mapped organisms for each individual compared to the total number of unique reactions (extrapolated by the number of reactions of each organism).Please, note that bigger circles with a number inside represent overlapping individuals for metabolic diversity.
2. **Classical multidimensional scaling of each individual reactions repertoire**

Other outputs computed during this phase are saved together with the previous ones into the **.mat** file called **mapInfo.mat**. If the **compMod** option is enabled (disabled here and by default in the **mgPipe** pipeline) these results are outputted as different **.csv** files. For simplicity reasons we will not discuss these additional outputs in this tutorial: for a description of them, please refer to the documentation.

```
[mapP] = detectOutput(resPath,'mapInfo.mat')
```

```
if isempty(mapP)
    % Loading models
    models = loadUncModels(modPath,strains,objre);

    % Computing genetic information
    [reac,micRea,binOrg,patOrg,reacPat,reacNumb,reacSet,reacTab,reacAbun,reacNumber] = getMappingInfo(models,abunFilePath,patNumb);
    writetable(cell2table(reacAbun,'VariableNames',['Reactions';sampName]'),strcat(resPath,'reactions.csv'));

    % Plotting genetic information
    [PCoA] = plotMappingInfo(resPath,patOrg,reacPat,reacTab,reacNumber,indInfoFilePath,figForm,sampName,strains);

    if compMod == 1
       mkdir(strcat(resPath,'compfile'))
       writetable([array2table(reac),array2table(reacTab,'VariableNames',sampName')],[resPath 'compfile' filesep 'ReacTab.csv'])
       writetable(cell2table(reacSet,'VariableNames',sampName'),[resPath 'compfile' filesep 'reacSet.csv'])
       writetable([array2table(strains),array2table(reacPat,'VariableNames',sampName')],[resPath 'compfile' filesep 'ReacPat.csv'])
       csvwrite(strcat(resPath,'compfile/PCoA_tab.csv'),PCoA)
    end

    %Save all the created variables

    %Create tables and save all the created variables
    reacTab=[array2table(reac),array2table(reacTab,'VariableNames',sampName')],[resPath 'compfile' filesep 'ReacTab.csv'];
    reacSet=cell2table(reacSet,'VariableNames',sampName');
    reacPat=[array2table(strains),array2table(reacPat,'VariableNames',sampName')];

    save(strcat(resPath,'mapInfo.mat'))
end
%end of trigger for Autoload
```

PIPELINE: [PART 2.1]

Checking consistency of inputs: if autofix == 0 halts execution with error msg if inconsistencies are detected, otherwise it really tries hard to fix the problem and continues execution when possible.

```
[autoStat,fixVec,strains]=checkNomenConsist(strains,autoFix);
```

Now we detect from the content of the results folder If PART2 was already computed: if the associated file is already present in the results folder its execution is skipped else its execution starts

```
[mapP]=detectOutput(resPath,'Setup_allbacs.mat');

if isempty(mapP)
```

```
    modbuild = 1;
else
    modbuild = 0;
    s= 'global setup file found: loading from resPath and skipping [PART2.1] analysis';
    disp(s)
end
%end of trigger for Autoload
```

A model joining all the reconstructions contained in the study will be created in this section. This model will be later used, integrating abundances coming from the metagenomic sequencing, to derive the different microbiota models. The result of this section will be automatically saved in the results folder.

```
if modbuild == 1
   setup=fastSetupCreator(models, strains, {},objre)
   setup.name='Global reconstruction with lumen / fecal compartments no host';
   setup.recon=0;
   save(strcat(resPath,'Setup_allbacs.mat'), 'setup')
end

if modbuild==0
    load(strcat(resPath,'Setup_allbacs.mat'))
end
```

PIPELINE: [PART 2.2]

Now we will create the different microbiota models integrating the given abundances. Coupling constraints and personalized "cumulative biomass" objective functions are also added. Models that are already existent will not be recreated, and new microbiota models will be saved in the results folder.

```
[createdModels]=createPersonalizedModel(abunFilePath,resPath,setup,sampName,strains,patNumb)
```

PIPELINE: [PART 3]

In this phase, for each microbiota model, a diet, in the form of set constraints to the exchanges reactions of the diet compartment, is integrated. Flux Variability analysis for all the exchange reactions of the diet and fecal compartment is also computed and saved in a file called "simRes". Specifically what computed and saved are:

1. **ID** a vector containing the names of metabolites for which FVA of exchange reactions was computed
2. **fvaCt** a cell array containing min flux trough uptake and max trough secretion exchanges (later used for computing NMPCs)
3. **nsCT** a cell array containing max flux trough uptake and min trough secretion exchanges
4. **presol** an array containing the value of objectives for each microbiota model with rich and selected diet
5. **inFesMat** cell array containing the names of the microbiota models that reported an infeasible status when solved for their objective

```
[ID,fvaCt,nsCt,presol,inFesMat]=microbiotaModelSimulator(resPath,setup,sampName,dietFilePath,rDiet,0,extSolve,patNumb,fvaType)
```

Finally, NMPCs (net maximal production capability) are computed in a metabolite resolved manner and saved in a comma delimited file in the results folder. NMPCs indicate the maximal production of each metabolite and are computed as the absolute value of the sum of the maximal secretion flux with the maximal uptake flux. The similarity of metabolic profiles (using the different NMPCs as features) between individuals is also evaluated with classical multidimensional scaling.

```
[Fsp,Y]= mgSimResCollect(resPath,ID,sampName,rDiet,0,patNumb,indInfoFilePath,fvaCt,figForm);
```

Additionally, it is possible to retrieve and export, comprehensively, all the results (fluxes) computed during the simulations for a specified diet. Since FVA is computed on diet and fecal exchanges, every metabolite will have four different values for each individual, values corresponding min and max of uptake and secretion.

```
[finRes] = extractFullRes(resPath, ID, 'sDiet', sampName, fvaCt, nsCt);
```