Tutorial 5 – Reconstruct and Refine a GEM from KEGG

From the tutorial file:

> This exercise is about creating a model from KEGG, based on protein sequences in a FASTA file, and doing some functionality checks on the model. The example case is for the yeast Saccharomyces cerevisiae. This tutorial is more of a showcase and its main purpose is to serve as a scaffold to reconstruct a GEM for any organism. Open tutorial5.m in MATLAB to begin this exercise.

In [ ]:
```
clear;
setRavenSolver('gurobi');
addpath('./source');
tolerance = 10^-7;
```

1. Metabolic reconstruction from KEEG and protein homology

Check the wiki Use Pre Trained HMMs for more details.

The tutorial uses:

- getKEGGModelForOrganism

Reconstructs a genome-scale metabolic model based on protein homology to the orthologies in KEGG. If the target species is not available in KEGG, the user must select a closely related species. It is also possible to circumvent protein homology search (see fastaFile parameter for more details)

❗ I cannot execute the function in the Notebook. I got this error message:

```
Error connecting to MATLAB. Check the status of MATLAB by clicking the "Open MATLAB" button. Retry after ensuring MATLAB is running successfully
```

Seems there is an issue with the MATLAB Kernel and calls that take long to complete; So, I excecuted the following call in console, and saved the output:

```
model= getKEGGModelForOrganism('sce', ...
    './tutorial_data/sce.fa', ... % fastaFile
    'euk90_kegg105', ... % Dataset: <phylogeny><% CD-HIT Identity>_kegg<release>
```

```
    './output/_ignore_tutorial_5', ... % outDir
    false, ... % keepSpontaneous. Label "spontaneous".
    false, ... % keepUndefinedStoich. "n A <=> n+1 A" form.
    false, ... % keepIncomplete. Label "incomplete", "erroneous" or "unclear".
    false, ... % keepGeneral. Label "general reaction". WARNING: this script cannot remove all.
    10^-30, ... % cutOff. significance score from HMMer needed to assign genes to a KO
    0.8, ... % minScoreRatioKO
    0.3, ... % minScoreRatioG
    -1); % maxPhylDist
save('./output/tutorial_5_model.mat', 'model');
```

In [ ]:

```
model = load('./output/tutorial_5_model.mat').model;
disp(model);
```

```
           id: 'sce'
         name: 'Automatically generated from KEGG database'
         rxns: {1589x1 cell}
     rxnNames: {1589x1 cell}
      eccodes: {1589x1 cell}
   subSystems: {1589x1 cell}
   rxnMiriams: {1589x1 cell}
     rxnNotes: {1589x1 cell}
            S: [1600x1589 double]
         mets: {1600x1 cell}
          rev: [1589x1 double]
           ub: [1589x1 double]
           lb: [1589x1 double]
            c: [1589x1 double]
            b: [1600x1 double]
        genes: {836x1 cell}
   rxnGeneMat: [1589x836 double]
     metNames: {1600x1 cell}
  metFormulas: {1600x1 cell}
      inchis: {1600x1 cell}
   metMiriams: {1600x1 cell}
        comps: {'s'}
    compNames: {'System'}
     metComps: [1600x1 double]
      grRules: {1589x1 cell}
```

2. Validating mass balance

❗ Redox and charge balance can be very tricky since formulas depends on conditions such as pH.

In [ ]:
```
% It is not neccesary, but I preffer tables to display information.
nonBalancedReactionsTable = makeUnablancedMetabolitesTable(model);
head(nonBalancedReactionsTable)
```

```
Potential problematic reactions: 615.
```

	balanceStatus	C	N	O	S	P	H	F	Cl	Fe	Se	R
R00021	-1	0	0	0	0	0	0	0	0	0	0	0
R00025	0	0	0	0	0	0	-1	0	0	0	0	0
R00093	0	0	0	0	0	0	-1	0	0	0	0	0
R00094	0	0	0	0	0	0	-1	0	0	0	0	0
R00100	-1	0	0	0	0	0	1	0	0	0	0	0
R00111	0	0	0	0	0	0	1	0	0	0	0	0
R00114	0	0	0	0	0	0	-1	0	0	0	0	0
R00115	0	0	0	0	0	0	-1	0	0	0	0	0

❗ to use makeSomething and consumeSomething functions, it is recomendable to ensure that there is not open exchange reactions:

In [ ]:
```
[~, exchangeRxnsIndexes] = getExchangeRxns(model);
fprintf('There are %i exchange reactions:\n', length(exchangeRxnsIndexes));
disp(table(model.rxns(exchangeRxnsIndexes), ...
    model.lb(exchangeRxnsIndexes), ...
    model.ub(exchangeRxnsIndexes), ...
    'VariableNames', {'ID', 'LB', 'UB'}));
```

```
There are 0 exchange reactions:
```

2.1. Free production

- makeSomething

`function [solution, metabolite] = makeSomething(model, ignoreMets, isNames, minNrFluxes, allowExcretion, params, ignoreIntBounds)`
Tries to excrete any metabolite using as few reactions as possible. The intended use is when you want to make sure that you model cannot synthesize anything from nothing. It is then a faster way than to use checkProduction or canProduce

In [ ]:
```
% First, check reactions producing species from nothing:
% From the nonBalancedReactionsTable table we already known there are issues with H+. Ignore it to avoid noise.
ignoreMets = {'H+'};
[solutionFlux, metaboliteIndex] = makeSomething(model, ignoreMets, true);
metaboliteName = model.metNames(metaboliteIndex);
reactionsWithFluxIndexes = find(abs(solutionFlux) >= tolerance);
fprintf('There are %i reactions with flux. The involved metabolite is %s', ...
    length(reactionsWithFluxIndexes), metaboliteName{:});
```

```
There are 35 reactions with flux. The involved metabolite is H2O
```

In [ ]:
```
% Assess which of these reactions have problems with their equations
% Get the reactions that have flux AND appears in the table of problematic reactions
carryingFluxProblematicReactions = intersect(...
    nonBalancedReactionsTable.Properties.RowNames, ...
    model.rxns(reactionsWithFluxIndexes));
reactions2check = nonBalancedReactionsTable(carryingFluxProblematicReactions, :);
fprintf('Reactions to check: %i\n', height(reactions2check));
disp(reactions2check);
```

```
Reactions to check: 7
             balanceStatus    C     N    O     S    P     H     F    Cl    Fe    Se    R
             _____________    __    _    __    _    _    ___    _    __    __    __    _

    R00941           0          0    0     0    0    0     -1    0    0     0     0     0
    R01058           0          0    0     0    0    0     -1    0    0     0     0     0
    R01061           0          0    0     0    0    0     -1    0    0     0     0     0
    R01221           0          0    0     0    0    0     -1    0    0     0     0     0
    R01655           0          0    0     0    0    0     -1    0    0     0     0     0
    R02110           0         -6    0    -5    0    0    -10    0    0     0     0     0
    R02413           0          0    0     0    0    0     -1    0    0     0     0     0
```

❗ R02110 reaction looks very suspicious, C H O are unbalanced.

In [ ]:
```
% Print fluxes from the problematic reactions
flux2showIndexes = getIndexes(model, carryingFluxProblematicReactions, 'rxns');
```

```
flux2show = zeros(size(solutionFlux));
flux2show(flux2showIndexes) = solutionFlux(flux2showIndexes);
printFluxes(model, flux2show, false, [], [], ...
    '▪ %flux\n%rxnID (%rxnName):\n\t%eqn\n');
```

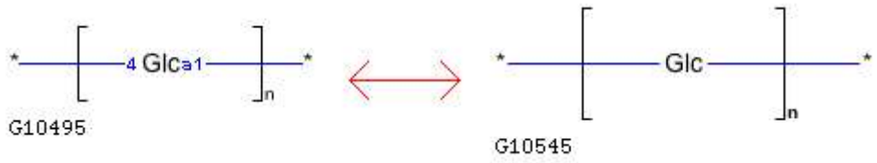
```
FLUXES:
▪ -0.085106
R00941 (10-formyltetrahydrofolate:NADP+ oxidoreductase):
        H2O[s] + NADP+[s] + 10-Formyltetrahydrofolate[s] <=> NADPH[s] + CO2[s] + H+[s] + Tetrahydrofolate[s]
▪ 0.42553
R01058 (D-glyceraldehyde 3-phosphate:NADP+ oxidoreductase):
        H2O[s] + NADP+[s] + D-Glyceraldehyde 3-phosphate[s] <=> NADPH[s] + H+[s] + 3-Phospho-D-glycerate[s]
▪ 0.042553
R01061 (D-glyceraldehyde-3-phosphate:NAD+ oxidoreductase (phosphorylating)):
        NAD+[s] + Orthophosphate[s] + D-Glyceraldehyde 3-phosphate[s] <=> NADH[s] + H+[s] + 3-Phospho-D-glyceroyl phosphate[s]
▪ -0.042553
R01221 (glycine:NAD+ 2-oxidoreductase (tetrahydrofolate-methylene-adding);):
        NAD+[s] + Glycine[s] + Tetrahydrofolate[s] <=> NADH[s] + CO2[s] + Ammonia[s] + H+[s] + 5,10-Methylenetetrahydrofolate[s]
▪ -0.085106
R01655 (5,10-Methenyltetrahydrofolate 5-hydrolase (decyclizing)):
        H2O[s] + 5,10-Methenyltetrahydrofolate[s] <=> H+[s] + 10-Formyltetrahydrofolate[s]
▪ 0.40426
R02110 (1,4-alpha-D-Glucan:1,4-alpha-D-glucan 6-alpha-D-(1,4-alpha-D-glucano)-transferase):
        Amylose[s] <=> Starch[s]
▪ -0.25532
R02413 (Shikimate:NADP+ 3-oxidoreductase):
        NADP+[s] + Shikimate[s] <=> NADPH[s] + H+[s] + 3-Dehydroshikimate[s]
```

The reaction involves two polysacharides. I want to remark the following from KEEG COMPOUND Database:

> "**Biosynthetic Codes**
> The structures of DNA, RNA, and proteins are determined by template-based syntheses of replication, transcription, and translation with the genetic code. In contrast, the structures of glycans, lipids, polyketides, nonribosomal peptides, and various secondary metabolites are determined by biosynthetic pathways. KEGG COMPOUND, as well as KEGG GLYCAN, is a resource for understanding such biosynthetic codes and for inferring chemical repertoires of these diverse substances from genomic information."

❗ R02110 is not an interconversión but a polymerization reaction, adding a glucose polymer Amylose (C6H10O5)n to another glucose polymer Starch (C12H20O10)n:

From KEGG GLYCAN Database, R02110 is the same as R06186:



In [ ]:
```
% Looking for al starch reactions
printFluxes(model, solutionFlux, false, [], [], ...
    '▪ %flux\n%rxnID (%rxnName):\n\t%eqn\n', {'Starch'});
```

```
FLUXES:
▪ 0.40426
R02110 (1,4-alpha-D-Glucan:1,4-alpha-D-glucan 6-alpha-D-(1,4-alpha-D-glucano)-transferase):
        Amylose[s] <=> Starch[s]
▪ 0.40426
R02111 (1,4-alpha-D-Glucan:orthophosphate alpha-D-glucosyltransferase):
        Orthophosphate[s] + Starch[s] <=> D-Glucose 1-phosphate[s] + Amylose[s]
```

❗ Why is there Starch in a yeast in first place? 😉

The tutorial removes only R02110, but, unless it is a genetically engineered yeast, I will consider to remove also R02111.

In [ ]:
```
model=removeReactions(model,'R02110');
```

In [ ]:
```
[solutionFlux, metaboliteIndex] = makeSomething(model, ignoreMets, true);
metaboliteName = model.metNames(metaboliteIndex);
reactionsWithFluxIndexes = find(abs(solutionFlux) >= tolerance);
fprintf('There are %i reactions with flux. The involved metabolite is %s', ...
    length(reactionsWithFluxIndexes), metaboliteName{:});
```

```
There are 0 reactions with flux. The involved metabolite is
```

2.2. Free internal consumption

- consumeSomething

```
function [solution, metabolite] = consumeSomething(model, ignoreMets, isNames, minNrFluxes, params, ignoreIntBounds)
```

Tries to consume any metabolite using as few reactions as possible. The intended use is when you want to make sure that you model cannot consume anything without producing something. It is intended to be used with no active exchange reactions.

In [ ]:
```
[solutionFlux, metaboliteIndex] = consumeSomething(model,{'H+'},true);
metaboliteName = model.metNames(metaboliteIndex);
reactionsWithFluxIndexes = find(abs(solutionFlux) >= tolerance);
fprintf('There are %i reactions with flux. The involved metabolite is %s', ...
    length(reactionsWithFluxIndexes), metaboliteName{:});
```

```
There are 0 reactions with flux. The involved metabolite is
```

No free consumption of internal metabolites 👌

2.3. Network connectivity - Gap filling

In [ ]:
```
% get the correct IDs for the metabolites.
% Notice that was used "Ammonia" instead of "NH3".
expression = '^(D-Glucose|H2O|Orthophosphate|Oxygen|Ammonia|Sulfate)$';
matchTable = findMetaboliteByName(model, expression);
disp(matchTable)
```

```
    metID           metName
    _________    ________________

    {'C00001'}    {'H2O'          }
    {'C00007'}    {'Oxygen'       }
    {'C00009'}    {'Orthophosphate'}
    {'C00014'}    {'Ammonia'      }
    {'C00031'}    {'D-Glucose'    }
    {'C00059'}    {'Sulfate'      }
```

In [ ]:
```
% added new inputs
[model, addedRxns ] = addExchangeRxns(model, 'in', matchTable.metID);
disp(addedRxns)
```

checkProduction

Checks which metabolites that can be produced from a model using the specified constraints.

```
function [notProduced, notProducedNames, neededForProductionMat, minToConnect, model] = checkProduction(model, checkNeededForProduction,
excretionFromCompartments, printDetails)
```

The function is intended to be used to identify which metabolites must be connected in order to have a fully connected network. It does so by first identifying which metabolites could have a net production in the network. Then it calculates which other metabolites must be able to have net production in order to have production of all metabolites in the network. So, if a network contains the equations A[external]->B, C->D, and D->E it will identify that production of C will connect the metabolites D and E.

In [ ]:
```
[notProduced, notProducedNames, neededForProductionMat, minToConnect, ~ ] = checkProduction(model, false, model.comps, false);
```

In [ ]:
```
fprintf('There are %i non-produced metabolites from %i (%3.2f %%)\n', ...
    length(notProduced), length(model.mets), 100*length(notProduced)/length(model.mets));
fprintf('Minimal number of metabolites that need to be connected: %i\n', length(minToConnect));
```

```
There are 1008 non-produced metabolites from 1600 (63.00 %)
Minimal number of metabolites that need to be connected: 0
```

❗ I am not sure what means an empty minToConnect array.

2.3.1. Template model preparation

From getModelFromKEGG documentation:

> NOTE: The model output from getModelFromKEGG can be used as template for fillGaps. In that case, ensure that the genes and rxnGeneMat fields are removed before parsing: model=rmfield(model,'genes'), etc.

In [ ]:
```
% keggPath (opt, default 'RAVEN/external/kegg')
keggModel = getModelFromKEGG([],false,false,false,false);
% following the documentation instruction for gap filling:
keggModel=rmfield(keggModel,'genes');
keggModel=rmfield(keggModel,'rxnGeneMat');
```

In [ ]:
```
% remove unbalanced reactions
balanceStructure = getElementalBalance(keggModel);
% Notice that removeUnusedMets and removeUnusedGenes are set to true
keggModel = removeReactions(keggModel, balanceStructure.balanceStatus~=1, true, true);
```

2.3.2. Gap filling

- fillGaps

```
function [newConnected, cannotConnect, addedRxns, newModel, exitFlag] = fillGaps(model, models,allowNetProduction, useModelConstraints, supressWarnings, rxnScores, params)
```

This method works by merging the model to the reference model(s) and checking which reactions can carry flux. All reactions that can't carry flux are removed (cannotConnect).

If **useModelConstraints is false** it then solves the MILP problem of minimizing the number of active reactions from the reference models that are required to have flux in all the reactions in model. This requires that the input model has exchange reactions present for the nutrients that are needed for its metabolism. **If useModelConstraints is true** then the problem is to include as few reactions as possible from the reference models in order to satisfy the model constraints.

The intended use is that the user can attempt a general gap-filling using useModelConstraint=false or a more targeted gap-filling by setting constraints in the model structure and then use useModelConstraints=true. Say that the user want to include reactions so that all biomass components can be synthesized. He/she could then define a biomass equation and set the lower bound to >0. Running this function with useModelConstraints=true would then give the smallest set of reactions that have to be included in order for the model to produce biomass.

❗ The tutorial makes the following configuration:

```
params.relGap=0.6; %Lower number for a more exhaustive search
params.printReport=true;
```

The fillGaps documentation is not much clear about what "params" is:

> params: parameter structure as used by getMILPParams (opt)

Checking the code, fillGaps uses params in the getMinNrFluxes call, which then uses it to call optimizeProb:

> params: solver specific parameters (optional)

I am not sure to which solver *relGap* and *printReport* are valid parameters 😳. I checked the RAVEN GLPK (Matlab MEX interface), the Gurobi (Parameters) and COBRA MILP Solver, without results. Then, I will just omit those parameters 😉.

> Since I am using Gurobi, worth to check this:
>
> - The params argument
> - MATLAB Parameter Examples

In [ ]:

```
[newConnected, cannotConnect, addedRxns, newModel, exitFlag] = fillGaps(...
    model, keggModel, true, false, false);
```

```
MILP detected.
Set parameter Username
Set parameter TimeLimit to value 1000
Set parameter FeasibilityTol to value 1e-09
Set parameter IntFeasTol to value 1e-09
Set parameter MIPGap to value 1e-12
Set parameter OptimalityTol to value 1e-09
Set parameter DisplayInterval to value 1
Set parameter Presolve to value 2
Set parameter Seed to value 1
Academic license - for non-commercial use only - expires 2024-07-25
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (win64)

CPU model: Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads

Optimize a model with 18294 rows, 44936 columns and 101985 nonzeros
Model fingerprint: 0x6576949b
Variable types: 33049 continuous, 11887 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+03]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e-01, 1e+03]
  RHS range        [0e+00, 0e+00]
Presolve removed 9054 rows and 29838 columns (presolve time = 1s) ...
Presolve removed 9054 rows and 29838 columns
Presolve time: 1.13s
Presolved: 9240 rows, 15098 columns, 46355 nonzeros
Variable types: 8190 continuous, 6908 integer (6908 binary)
Found heuristic solution: objective 2401.0000000
Root relaxation presolve removed 6899 rows and 7144 columns
Root relaxation presolved: 2341 rows, 7954 columns, 31915 nonzeros


Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.4000000e+01   8.334406e+04   0.000000e+00      2s
    1717    2.7003350e+01   0.000000e+00   0.000000e+00      2s
    1719    2.7003350e+01   0.000000e+00   0.000000e+00      2s
Extra simplex iterations after uncrush: 2
```

```
Root relaxation: objective 2.700335e+01, 1719 iterations, 0.22 seconds (0.09 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   27.00335    0   25 2401.00000   27.00335  98.9%     -    2s
H    0     0                      52.0000000   27.00335  48.1%     -    2s
     0     0   40.00175    0   18   52.00000   40.00175  23.1%     -    2s
     0     0   43.00190    0   20   52.00000   43.00190  17.3%     -    2s
     0     0   45.00100    0   11   52.00000   45.00100  13.5%     -    2s
     0     0   45.00100    0   13   52.00000   45.00100  13.5%     -    2s
     0     0   46.00090    0   10   52.00000   46.00090  11.5%     -    2s
     0     0   47.00080    0   11   52.00000   47.00080  9.61%     -    3s
     0     0   48.00070    0    8   52.00000   48.00070  7.69%     -    3s
     0     0   48.00070    0    8   52.00000   48.00070  7.69%     -    3s
     0     2   48.00070    0    8   52.00000   48.00070  7.69%     -    3s
    77    12   49.00030    3    5   52.00000   49.00030  5.77%   4.1    4s

Cutting planes:
  Cover: 1
  Implied bound: 3
  Clique: 3
  Flow cover: 21
  Network: 18
  Relax-and-lift: 1

Explored 137 nodes (3375 simplex iterations) in 4.30 seconds (1.71 work units)
Thread count was 4 (of 4 available processors)

Solution count 2: 52 2401

Optimal solution found (tolerance 1.00e-12)
Best objective 5.200000000000e+01, best bound 5.200000000000e+01, gap 0.0000%
```

In [ ]:
```
fprintf('Connected %i reactions by adding %i new reactions.\n', length(newConnected), length(addedRxns));
```

```
Connected 29 reactions by adding 52 new reactions
```

In [ ]:
```
save('./output/tutorial_5_fillGapOutput.mat', 'newConnected', 'cannotConnect', 'addedRxns', 'newModel');
```

2.3.3. Manual curation of added reactions (not done here)

It is necessary to check that the added reactions exist in yeast since the template is a general model.

2.3.4. Further connectivity improving (iterative, not finished here)

In [ ]:
```
fprintf('Remaining unconnected reactions: %i (%3.2f)%%.\n', ...
    length(cannotConnect), 100 * length(cannotConnect)/length(newModel.rxns));
```

```
Remaining unconnected reactions: 512 (31.11)%.
```

- gapReport

Performs a gap analysis and summarizes the results

```
function [noFluxRxns, noFluxRxnsRelaxed, subGraphs, notProducedMets, minToConnect,neededForProductionMat, canProduceWithoutInput,
canConsumeWithoutOutput,connectedFromTemplates, addedFromTemplates] = gapReport(model,  templateModels)
```

Since gapReport can takes hours to run, I will perform this step in console and save the results:

```
newModel = load('./output/tutorial_5_fillGapOutput.mat').newModel;
[noFluxRxns, noFluxRxnsRelaxed, subGraphs, notProducedMets, minToConnect,...
    neededForProductionMat]=gapReport(newModel);
clear newModel
save('./output/tutorial_5_gapReport01.mat');
```

Output log:

```
Gap analysis for MERGED -

***Overview
483 out of 1646 reactions cannot carry flux (483 if net production of all metabolites is allowed)
691 out of 1619 metabolites are unreachable (691 if net production of all metabolites is allowed)

***Isolated subnetworks
A total of 6 isolated sub-networks are present in the model
        1. 1607 metabolites
        2. 4 metabolites
        3. 2 metabolites
```

4. 2 metabolites
5. 2 metabolites
6. 2 metabolites

***Metabolite connectivity

To enable net production of all metabolites, a total of 305 metabolites must be connected
Top 10 metabolites to connect:
1. Retinyl ester[s] (connects 66 metabolites)
2. Thiamin diphosphate[s] (connects 17 metabolites)
3. Selenomethionyl-tRNA(Met)[s] (connects 15 metabolites)
4. [Dihydrolipoyllysine-residue succinyltransferase] S-glutaryldihydrolipoyllysine[s] (connects 14 metabolites)
5. G00003[s] (connects 14 metabolites)
6. NAD+[s] (connects 13 metabolites)
7. Androstenediol[s] (connects 13 metabolites)
8. 1-Alkyl-2-acylglycerol[s] (connects 10 metabolites)
9. G00146[s] (connects 10 metabolites)
10. Progesterone[s] (connects 9 metabolites)

In [ ]:
```
gapReportOutput = load('./output/tutorial_5_gapReport01.mat');
disp(gapReportOutput);
fprintf('■ There are %i isolated subnetworks.\n', size(gapReportOutput.subGraphs, 2));
for index = 1:size(gapReportOutput.subGraphs, 2)
    fprintf('\t- %i metabolites\n', sum(gapReportOutput.subGraphs(:, index)));
end
fprintf('■ Top 10 metabolites to connect:\n');
for index = 1:10
    fprintf('\t%i. %s.\n', index, string(gapReportOutput.minToConnect(index, 1)))
end
```

```
            minToConnect: {305x1 cell}
   neededForProductionMat: [952x952 logical]
               noFluxRxns: {483x1 cell}
         noFluxRxnsRelaxed: {483x1 cell}
           notProducedMets: [952x1 double]
                 subGraphs: [1619x6 logical]

■ There are ■ There are 6 isolated subnetworks.
        - 1607 metabolites
        - 4 metabolites
        - 2 metabolites
        - 2 metabolites
        - 2 metabolites
        - 2 metabolites
■ Top 10 metabolites to connect:
        1. Retinyl ester[s] (connects 66 metabolites).
        2. Thiamin diphosphate[s] (connects 17 metabolites).
        3. Selenomethionyl-tRNA(Met)[s] (connects 15 metabolites).
        4. [Dihydrolipoyllysine-residue succinyltransferase] S-glutaryldihydrolipoyllysine[s] (connects 14 metabolites).
        5. G00003[s] (connects 14 metabolites).
        6. NAD+[s] (connects 13 metabolites).
        7. Androstenediol[s] (connects 13 metabolites).
        8. 1-Alkyl-2-acylglycerol[s] (connects 10 metabolites).
        9. G00146[s] (connects 10 metabolites).
        10. Progesterone[s] (connects 9 metabolites).
```

In [ ]:

```
expression = '^(4-Aminobenzoate|Riboflavin|Thiamine|Biotin|Folate|Nicotinate|Zymosterol|Choline)$';
matchTable = findMetaboliteByName(model, expression);
fprintf('Number of matches: %i\n', height(matchTable));
disp(matchTable);
```

```
Number of matches: 8
8
      metID             metName
    ___________    ____________________

    {'C00114'}    {'Choline'         }
    {'C00120'}    {'Biotin'          }
    {'C00253'}    {'Nicotinate'      }
    {'C00255'}    {'Riboflavin'      }
    {'C00378'}    {'Thiamine'        }
    {'C00504'}    {'Folate'          }
    {'C00568'}    {'4-Aminobenzoate'}
    {'C05437'}    {'Zymosterol'      }
```

In [ ]:
```
[model, addedRxns ] = addExchangeRxns(model, 'in', matchTable.metID);
disp(addedRxns)
```

```
    {'EXC_IN_C00114'}
    {'EXC_IN_C00120'}
    {'EXC_IN_C00253'}
    {'EXC_IN_C00255'}
    {'EXC_IN_C00378'}
    {'EXC_IN_C00504'}
    {'EXC_IN_C00568'}
    {'EXC_IN_C05437'}
```

🔁 Run again *gapReport* and repeat 🔁