Tutorial 4 – Fix an Erroneous Model

The whole point of GEMs is that they are large. It is by incorporating the entire known metabolism of any given organism that complexity arises. However, this makes it almost certain that all models will contain errors. This is true regardless of whether one builds the model or if one uses a model from someone else. One of the issues is that if one group publishes a model for some specific purpose it is likely to function well in that specific part of metabolism, but it may not function at all for some problems. It is therefore a good idea to perform a round of error checking even if it is a published model one uses.

Model validation is an iterative process because some errors might not have an effect until some other errors have been fixed. It is not uncommon that the model "works" well in the beginning of the reconstruction process because there are errors that let it cheat on things like redox or energy balance. The model then works worse and worse as the errors are dealt with until all or most errors are fixed, after which it will start to work again. As RAVEN developers, we believe that it is much more important to try to make the model do something it should not be allowed to do rather than to test for the stuff it should do.

There is a version of the small yeast model with errors inserted (smallYeastBad.xlsx). The task for this exercise is to find and fix them. Some errors will be obvious (it is rather difficult to introduce errors in such a small model, because there is extraordinarily little redundancy in it), but it is strongly recommended not to fix them until they are "found" during the following steps, because otherwise one might get unpredictable results. Most of the stuff done here can be done with the gapReport function, but it is strongly recommended to do them step by step.

In [ ]:
```
clear
setRavenSolver('gurobi')
% values lower than tolerance are considered zero
tolerance = 10^-7;
```

1. The first thing to check for is that the model cannot make something from nothing, i.e. no metabolites should be produced if one does not give the model access to any carbon sources (this should be done for all elements, but carbon is the most important). A simple way to do this would be to optimize for the sum of all the producing exchange reactions, while keeping the consuming reactions closed. Any solution other than 0 would then be bad. Try that.

In [ ]:
```
% import and basic inspection
model = importExcelModel("./tutorial_data/smallYeastBad.xlsx");
printModelStats(model);
```

```
NOTE: DEFAULT LOWER not supplied. Uses -1000
NOTE: DEFAULT UPPER not supplied. Uses 1000
WARNING: No objective function found. This might be intended, but results in FBCv2 non-compliant SBML file when exported

WARNING: The following InChI strings are associated to more than one unique metabolite name:
        1S/C6H14O12P2/c7-4-3(1-16-19(10,11)12)18-6(9,5(4)8)2-17-20(13,14)15/h3-5,7-9H,1-2H2,(H2,10,11,12)(H2,13,14,15)/t3-,4-,5+,6-/m1/s1

NOTE: DEFAULT UPPER not supplied. Uses 1000
WARNING: No objective function found. This might be intended, but results in FBCv2 non-compliant SBML file when exported

WARNING: The following InChI strings are associated to more than one unique metabolite name:
        1S/C6H14O12P2/c7-4-3(1-16-19(10,11)12)18-6(9,5(4)8)2-17-20(13,14)15/h3-5,7-9H,1-2H2,(H2,10,11,12)(H2,13,14,15)/t3-,4-,5+,6-/m1/s1

Network statistics for smallYeastBad: Central carbon metabolism for yeast
Genes*                          61
        cytosol 52
        mitochondria    17

Reactions*                      54
        cytosol 46
        mitochondria    19
Unique reactions**      54

Metabolites                     52
        cytosol 35
        mitochondria    17
Unique metabolites      45

* Genes and reactions are counted for each compartment if any of the corresponding metabolites are in that compartment. The sum may therefore not add up to th
e total number.
** Unique reactions are defined as being biochemically unique (no compartmentalization)
```

In [ ]:
```
exchangeIndexes = getIndexes(model, getExchangeRxns(model), 'rxns');
disp(table(model.rxnNames(exchangeIndexes), ...
         model.lb(exchangeIndexes), model.ub(exchangeIndexes), ...
        'VariableNames',  {'ExchangeReaction', 'LB', 'UB'}));
```

```
      ExchangeReaction          LB     UB
    ____________________________    __    ____

    ____________________________    __    ____

    {'Production of acetate' }    0     1000
    {'Production of biomass' }    0     1000
    {'Production of CO2'     }    0     1000
    {'Production of ethanol' }    0     1000
    {'Production of glycerol'}    0     1000
    {'Uptake of glucose'     }    0     1000
    {'Uptake of O2'          }    0     1000
```

In [ ]:
```
% close uptake reactions
model = setParam(model, 'eq', getExchangeRxns(model, 'in'), 0);
% set optimization to max the sum of production reactions
model = setParam(model, 'obj', getExchangeRxns(model, 'out'), 1);
% Check if the flux is zero
solution = solveLP(model);
disp(solution);
printFluxes(model, solution.x, false, tolerance);
```

```
        x: [54x1 double]
        f: 0
     stat: 1
      msg: 'Optimal solution found'
   sPrice: [52x1 double]
    rCost: [54x1 double]

FLUXES:
```

2. The previous step did not provide any non-zero solutions, right? That is good, but there could be other factors that prevent the error from showing its ugly face. Maybe it costs energy or redox power for example. Or maybe the necessary reactions are in different compartments. It is generally a good idea to relax as many constraints as possible when searching for errors. For instance, one can include a temporary reaction like “ATP + H2O ⇔ ADP + Pi” and similar reactions with NADH and NADPH. Remember that the aim here is to try to “provoke” the model to show the errors. Add these reactions and try again.

❗ **Personal note**: seems that getIndexes do not use regex (except for 'metcomps'), then you need to know the EXACT name or ID. This is a strong limitation, what if I am exploring a model I am not familiar with?

For example:

In [ ]:
```
% Example of the commented limitation
disp(getIndexes(model, {'atp', 'AT'}, 'metnames'));
disp(getIndexes(model, {'atp', 'AT'}, 'mets'));
```

```
   {0x1 double}
   {0x1 double}
```

```
Error using getIndexes
Could not find object 'atp' in the model
```

In [ ]:
```
% Workaround 🤓
% regex for looking forr ATP, NADH or NADPH
pattern = '^(a[dt]p|nadp?h?|pi|h|h2o).*_c$';
% Get cells that match the query, then filter them to get the cells that have matches
indexesToCheck = find(~cellfun(@isempty, regexpi(model.mets, pattern, 'match')'));
tableToCheck = table(model.mets(indexesToCheck), ...
    model.metNames(indexesToCheck), ...
    'VariableNames', {'ID', 'metName'});
disp(tableToCheck);
```

```
      ID            metName
   ___________    _____________

   ___________    _____________

   {'ADP_c'  }    {'ADP'      }
   {'ATP_c'  }    {'ATP'      }
   {'NAD_c'  }    {'NAD(+)'   }
   {'NADH_c' }    {'NADH'     }
   {'NADP_c' }    {'NADP(+)'  }
   {'NADPH_c'}    {'NADPH'    }
   {'PI_c'   }    {'phosphate'}
```

- addRxns

```
function newModel=addRxns(model,rxnsToAdd,eqnType,compartment,allowNewMets,allowNewGenes)
```

Adds reactions to a model

In [ ]:
```
% cell array with unique strings that identifies each reaction
freeReactions.rxns = {
    'FREE_ATP';
    'FREE_NADH';
    'FREE_NADPH'
    };

% cell array with equation strings. Decimal coefficients are expressed as "1.2". Reversibility is indicated by "<=>" or "=>"
freeReactions.equations = {
    'ATP[c] <=> ADP[c] + phosphate[c]';
    'NAD(+)[c] <=> NADH[c]';
    'NADP(+)[c] <=> NADPH[c]'
    };

% eqnType 3: The metabolites are written as "metNames[comps]". Only compartments in model.comps are allowed
model = addRxns(model, freeReactions, 3);
printModel(model, freeReactions.rxns );
```

```
FLUXES:
FREE_ATP ()
        ATP[c] <=> ADP[c] + phosphate[c] [-1000 1000]
FREE_NADH ()
        NAD(+)[c] <=> NADH[c] [-1000 1000]
FREE_ATP ()
        ATP[c] <=> ADP[c] + phosphate[c] [-1000 1000]
FREE_NADH ()
        NAD(+)[c] <=> NADH[c] [-1000 1000]
FREE_NADPH ()
        NADP(+)[c] <=> NADPH[c] [-1000 1000]
```

❗ Pay attention to "minFLux" option:

In [ ]:
```
% minFlux 1: the sum of abs(fluxes) is minimized. This is the fastest way of getting rid of loops
modelSolution = solveLP(model, 1);
disp(modelSolution);
```

```
      x: [57x1 double]
      f: -1000
   stat: 1
    msg: 'Optimal solution found'
```

3. Did one get the production of ethanol? If so, print the resulting fluxes and see if it is possible to find the error. GEMs are normally very underdetermined, which means that there are infinite numbers of solutions to any given problem. When one solves using solveLP(model) one just gets a random solution which meets the objective and satisfies the constraints. These solutions often contain loops and are therefore difficult to interpret. One can read more about the solveLP function by typing "help solveLP" in MATLAB, but here it is chosen to solve using solveLP(model,1). This minimizes the sum of fluxes to have more easily interpreted results. Find and fix the error and rerun.

> Question 2: what modification is needed to prevent the of ethanol from nothing?

In [ ]:
```
% onlyExchange  false:  only print exchange fluxes
printFluxes(model, modelSolution.x, false);
```

```
FLUXES:
ethOUT  (Production of ethanol):        999.999
ethOUT  (Production of ethanol):        999.999
ADH1    (Alcohol dehydrogenase):        999.999
ADH2    (Alcohol dehydrogenase rev):    999.999
FREE_NADH       ():     999.999
FREE_NADPH      ():     -999.999
```

Aditional to ethOUT and the added testing reactions, there are two with infinite flux: ADH1 and ADH2

In [ ]:
```
reactionsToEvalue = {'ADH1', 'ADH2'};
printModel(model, reactionsToEvalue);
```

```
FLUXES:
ADH1 (Alcohol dehydrogenase)
        acetaldehyde[c] + NADH[c] => 2 ethanol[c] + NAD(+)[c] [0 1000]
ADH2 (Alcohol dehydrogenase rev)
        ethanol[c] + NADP(+)[c] => acetaldehyde[c] + NADPH[c] [0 1000]
ADH1 (Alcohol dehydrogenase)
        acetaldehyde[c] + NADH[c] => 2 ethanol[c] + NAD(+)[c] [0 1000]
ADH2 (Alcohol dehydrogenase rev)
        ethanol[c] + NADP(+)[c] => acetaldehyde[c] + NADPH[c] [0 1000]
```

The ethanol stoichiometric coefficient between the two reactions are inconsistent. At least one of the reactions is unbalanced. Let's check it:

```
function balanceStructure=getElementalBalance(model,rxns,printUnbalanced,printUnparsable)
```
Checks a model to see if the reactions are elementally balanced.

In [ ]:
```
getElementalBalance(model, reactionsToEvalue, true, true);
```

```
WARNING: The reaction ADH1 is not balanced with respect to carbon

WARNING: The reaction ADH1 is not balanced with respect to oxygen

WARNING: The reaction ADH1 is not balanced with respect to oxygen

WARNING: The reaction ADH1 is not balanced with respect to hydrogen
```

- changeRxns

```
function model=changeRxns(model,rxns,equations,eqnType,compartment,allowNewMets)
```

Modifies the equations of reactions

In [ ]:
```
rxnsToChange.rxns = {'ADH1'}%, 'ADH2'};
rxnsToChange.equations = {
    % acetaldehyde[c] + NADH[c] + H+[c] => ethanol[c] + NAD(+)[c]
    'acetaldehyde[c] + NADH[c] => ethanol[c] + NAD(+)[c]',
    % ethanol[c] + NADP(+)[c] => acetaldehyde[c] + NADPH[c] + H+[c]
    };
% update model
model = changeRxns(model, rxnsToChange.rxns, rxnsToChange.equations, 3);
% verify balance

disp(table(reactionsToEvalue', ...
        getElementalBalance(model, reactionsToEvalue, true, true).balanceStatus, ...
        'VariableNames', {'reaction', 'isBalanced'} ...
        ));
% optimize again
modelSolution = solveLP(model, 1);
disp(modelSolution);
printFluxes(model, modelSolution.x, false);
```

Out[ ]:
```
rxnsToChange = struct with fields:
  rxns: {'ADH1'}
```

```
    reaction    isBalanced
    ________    __________

    ________    __________

    {'ADH1'}         1
    {'ADH2'}         1

       x: [57x1 double]
       f: 0
    stat: 1
     msg: 'Optimal solution found'

FLUXES:
```

4. In GEMs it is normal to have excretion of only a few metabolites while having very many internal metabolites. A common case is that one has an error that would like to produce something from nothing, but to do so it also must produce some other metabolite for which there is no exchange reaction. A convenient way to test this is to allow all metabolites to be excreted. One can do this by changing the model.b structure. Normally it is always a vector of zeros, but if one adds a second column RAVEN will interpret it as lower and upper bound on the equality constraints. So if one puts model.b=[model.b inf(numel(model.b),1)]; one can now excrete anything. Do this and see if the model can produce anything. For instance, one should get ethanol, glycerol, and CO2. Look at the fluxes and find the error. One can get a clue by looking at the warnings from SBMLFromExcel. Since this is a problem that comes from reactions being unbalanced, the problematic ones must be in one of the warnings. Which was the metabolite that had to be excreted for the error to appear? Do this step two times to find both errors.

   > Question 3: what two modifications are needed to fix the warnings?

In [ ]:
```
% In RAVEN: if size(b) = [n, 1], it means Sx= b. If size(b) = [n, 2], it means b(:,1) <= S <= b(:,2)
model.b = [model.b, 1000 * ones(length(model.b),1)];
modelSolution = solveLP(model, 1);
disp(modelSolution);
printFluxes(model, modelSolution.x, false, tolerance,[], ...
    '▪ %flux\n%rxnID (%rxnName):\n\t%eqn\n\n');
```

```
       x: [57x1 double]
       f: -2.2083e+03
    stat: 1
     msg: 'Optimal solution found'

FLUXES:
▪ 500
co2OUT (Production of CO2):
        CO2[c] =>

▪ 708.3311
ethOUT (Production of ethanol):
        ethanol[c] =>

▪ 1000
glyOUT (Production of glycerol):
        glycerol[c] =>

▪ -500
PGI (Glucose-6-phosphate isomerase):
        alpha-D-glucose 6-phosphate[c] <=> beta-D-fructofuranose 6-phosphate[c]

▪ 1000
PFK (Phosphofructokinase):
        ATP[c] + beta-D-fructofuranose 6-phosphate[c] => ADP[c] + 2 beta-D-fructofuranose 1,6-bisphosphate[c]

FLUXES:
▪ 500
co2OUT (Production of CO2):
        CO2[c] =>

▪ 708.3311
ethOUT (Production of ethanol):
        ethanol[c] =>

▪ 1000
glyOUT (Production of glycerol):
        glycerol[c] =>

▪ -500
PGI (Glucose-6-phosphate isomerase):
        alpha-D-glucose 6-phosphate[c] <=> beta-D-fructofuranose 6-phosphate[c]
```

▪ 1000
PFK (Phosphofructokinase):
	ATP[c] + beta-D-fructofuranose 6-phosphate[c] => ADP[c] + 2 beta-D-fructofuranose 1,6-bisphosphate[c]

▪ 583.3378
FBP (Fructose-1,6-bisphosphatase):
	beta-D-fructofuranose 1,6-bisphosphate[c] => 2 beta-D-fructofuranose 6-phosphate[c] + phosphate[c]

▪ 1000
FBA (Fructose-bisphosphate aldolase):
	beta-D-fructofuranose 1,6-bisphosphate[c] <=> D-glyceraldehyde 3-phosphate[c] + glycerone phosphate[c]

▪ 1000
GLD (Triosephosphate dehydrogenase):
	D-glyceraldehyde 3-phosphate[c] + NAD(+)[c] + phosphate[c] <=> 3-phospho-D-glyceroyl phosphate[c] + NADH[c]

▪ 708.3311
PGK (Phosphoglycerate kinase):
	3-phospho-D-glyceroyl phosphate[c] + ADP[c] <=> 3-phospho-D-glycerate[c] + ATP[c]

▪ 708.3311
GPM (Phosphoglycerate mutase):
	3-phospho-D-glycerate[c] <=> 2-phospho-D-glycerate[c]

▪ 708.3311
ENO (Enolase):
	2-phospho-D-glycerate[c] <=> phosphoenolpyruvate[c]

▪ 708.3311
PYK (Pyruvate kinase):
	ADP[c] + phosphoenolpyruvate[c] => ATP[c] + pyruvate[c]

▪ 500
ZWF (Glucose-6-phosphate 1-dehydrogenase):
	alpha-D-glucose 6-phosphate[c] + NADP(+)[c] => 6-O-phosphono-D-glucono-1,5-lactone[c] + NADPH[c]

▪ 500
PGL (6-phosphogluconolactonase):
	6-O-phosphono-D-glucono-1,5-lactone[c] => 6-phospho-D-gluconate[c]

▪ 500

GND (6-phosphogluconate dehydrogenase, decarboxylating):
	6-phospho-D-gluconate[c] + NADP(+)[c] => CO2[c] + D-ribulose 5-phosphate[c] + NADPH[c]

▪ 166.6622
RPI (ribose 5-phosphate isomerase):
	D-ribulose 5-phosphate[c] <=> D-ribose 5-phosphate[c]

▪ 333.3245
RPE (Ribulose-phosphate 3-epimerase):
	D-ribulose 5-phosphate[c] <=> D-xylulose 5-phosphate[c]

▪ 166.6622
TKI1TKI2a (Transketolase):
	D-ribose 5-phosphate[c] + D-xylulose 5-phosphate[c] <=> D-glyceraldehyde 3-phosphate[c] + sedoheptulose 7-phosphate[c]

▪ 166.6622
TAL1 (Transaldolase):
	D-glyceraldehyde 3-phosphate[c] + sedoheptulose 7-phosphate[c] <=> beta-D-fructofuranose 6-phosphate[c] + D-erythrose 4-phosphate[c]

▪ 166.6622
TKI1TKI2b (Transketolase):
	D-erythrose 4-phosphate[c] + D-xylulose 5-phosphate[c] <=> beta-D-fructofuranose 6-phosphate[c] + D-glyceraldehyde 3-phosphate[c]

▪ 1000
DAR (glycerol-3-phosphate dehydrogenase):
	glycerone phosphate[c] + NADH[c] => glycerol monophosphate[c] + NAD(+)[c]

▪ 1000
GPP (sn-glycerol-3-phosphate phosphohydrolase):
	glycerol monophosphate[c] => glycerol[c] + phosphate[c]

▪ 708.3311
PDC (Pyruvate decarboxylase):
	pyruvate[c] => acetaldehyde[c]

▪ 708.3311
ADH1 (Alcohol dehydrogenase):
	acetaldehyde[c] + NADH[c] => ethanol[c] + NAD(+)[c]

▪ 416.6622
ATPX (ATP hydrolysis):
	ATP[c] => ADP[c] + phosphate[c]

- 708.3311

```
FREE_NADH ():
        NAD(+)[c] <=> NADH[c]
```

- -1000

```
FREE_NADPH ():
        NADP(+)[c] <=> NADPH[c]
```

❗ The import function did not raise the warnings the tutorial refers to; then, I will verify mass balance by each non-exchange reaction with non-zero flux (exchange reactions do not need to be mass balanced).

In [ ]:

```
% get reactions with non zero flux, then
% filter out exchange reactions and fake generating free energy and reductor power reactions
reactionsToEvalue = model.rxns(modelSolution.x >= tolerance);
reactionsToEvalue = setdiff(reactionsToEvalue, ...
    [getExchangeRxns(model); freeReactions.rxns]);
% check unbalanced reactions, organice data in a table and filter out balanced equations
balanceCheck = getElementalBalance(model, reactionsToEvalue, false, true);
printModel(model, reactionsToEvalue(balanceCheck.balanceStatus == 0));
```

```
FLUXES:
PFK (Phosphofructokinase)
        ATP[c] + beta-D-fructofuranose 6-phosphate[c] => ADP[c] + 2 beta-D-fructofuranose 1,6-bisphosphate[c] [0 1000]
FBP (Fructose-1,6-bisphosphatase)
        beta-D-fructofuranose 1,6-bisphosphate[c] => 2 beta-D-fructofuranose 6-phosphate[c] + phosphate[c] [0 1000]
ENO (Enolase)
        2-phospho-D-glycerate[c] <=> phosphoenolpyruvate[c] [-1000 1000]
PFK (Phosphofructokinase)
        ATP[c] + beta-D-fructofuranose 6-phosphate[c] => ADP[c] + 2 beta-D-fructofuranose 1,6-bisphosphate[c] [0 1000]
FBP (Fructose-1,6-bisphosphatase)
        beta-D-fructofuranose 1,6-bisphosphate[c] => 2 beta-D-fructofuranose 6-phosphate[c] + phosphate[c] [0 1000]
ENO (Enolase)
        2-phospho-D-glycerate[c] <=> phosphoenolpyruvate[c] [-1000 1000]
PGL (6-phosphogluconolactonase)
        6-O-phosphono-D-glucono-1,5-lactone[c] => 6-phospho-D-gluconate[c] [0 1000]
TAL1 (Transaldolase)
        D-glyceraldehyde 3-phosphate[c] + sedoheptulose 7-phosphate[c] <=> beta-D-fructofuranose 6-phosphate[c] + D-erythrose 4-phosphate[c] [-1000 1000]
TKI1TKI2b (Transketolase)
        D-erythrose 4-phosphate[c] + D-xylulose 5-phosphate[c] <=> beta-D-fructofuranose 6-phosphate[c] + D-glyceraldehyde 3-phosphate[c] [-1000 1000]
GPP (sn-glycerol-3-phosphate phosphohydrolase)
        glycerol monophosphate[c] => glycerol[c] + phosphate[c] [0 1000]
PDC (Pyruvate decarboxylase)
        pyruvate[c] => acetaldehyde[c] [0 1000]
ATPX (ATP hydrolysis)
        ATP[c] => ADP[c] + phosphate[c] [0 1000]
```

Let's organize a bit of the data.

The sign indicates which side has more mass. Unbalanced elements hint at the nature of the compound. For example, in ATPX, O: -1 and H: -2 clearly indicate water; in FBP, the stoichiometry suggests a carbohydrate and a phosphate compound, and the latter cannot be an adenosine compound since N is balanced.

In [ ]:
```
balanceCheckTable = array2table(balanceCheck.leftComp - balanceCheck.rightComp, ...
        'VariableNames', balanceCheck.elements.abbrevs, ...
        'RowNames', reactionsToEvalue);
unbalancedTable = balanceCheckTable(...
        reactionsToEvalue(balanceCheck.balanceStatus == 0), :);
fprintf('Table rows: %i.\n\n', height(unbalancedTable));
disp(unbalancedTable);
```

```
Table rows: 9.

9.
```

	C	N	O	P	H
ATPX	0	0	-1	0	-2
ENO	0	0	1	0	2
FBP	-6	0	-16	-3	-17
GPP	0	0	-1	0	-2
PDC	1	0	2	0	0
PFK	-6	0	-9	-1	-13
PGL	0	0	-1	0	-2
TAL1	0	0	-3	-1	-1
TKI1TKI2b	0	0	-3	-1	-1

The model is missing water:

In [ ]:
```
pattern = 'h2o';
indexesToCheck = find(~cellfun(@isempty, ...
    regexpi(model.mets, pattern, 'match')'));
fprintf('Found indexes for %s: %i\n', pattern, length(indexesToCheck));
% add water
metsToAdd.mets = 'H2O_c';
metsToAdd.metNames = 'H2O';
metsToAdd.compartments = 'c';
metsToAdd.metFormulas = 'H2O';
model = addMets(model,metsToAdd);
% verify it is added
disp(model.mets(getIndexes(model, metsToAdd.mets, 'mets')));
```

```
Found indexes for h2o: h2o: 0
    {'H2O_c'}
```

The equations for TAL1 (R08575) and TKI1TKI2b (R01067) are correct. Both have the same elemental imbalance (HPO3) and share a compound, the beta-D-fructofuranose 6-phosphate. Likely the compound has a incorrect formula.

beta-D-fructofuranose 6-phosphate have the formula C6H13O9P (KEGG: C05345)

In [ ]:
```
compoundName = 'beta-D-fructofuranose 6-phosphate';
compoundFormula = 'C6H13O9P';
compoundIndex = getIndexes(model, compoundName, 'metnames');
formulaToCheck = model.metFormulas(compoundIndex);
boolStr = {'false', 'true'};
fprintf('The formula %s is correct? %s.', ...
    formulaToCheck{:}, ...
    boolStr{strcmp(formulaToCheck, compoundFormula) + 1});
```

```
The formula C6H14O12P2 is correct? C6H14O12P2 is correct? false.
```

In [ ]:
```
% replace and verify
model.metFormulas(compoundIndex) = {compoundFormula};
formulaToCheck = model.metFormulas(compoundIndex);
fprintf('The formula %s is correct? %s.', ...
    formulaToCheck{:}, ...
    boolStr{strcmp(formulaToCheck, compoundFormula) + 1});
```

```
The formula C6H13O9PC6H13O9P is correct? true.
```

Now correct the equations:

In [ ]:
```
rxnsToChange.rxns = {'ATPX'; 'ENO'; 'FBP'; 'GPP'; 'PDC'; 'PFK'; 'PGL'};

rxnsToChange.equations = {
    'ATP[c] + H2O[c] => ADP[c] + phosphate[c]'; % ATPX
    '2-phospho-D-glycerate[c] <=> phosphoenolpyruvate[c]  + H2O[c]'; % ENO
    'beta-D-fructofuranose 1,6-bisphosphate[c] + H2O[c] => beta-D-fructofuranose 6-phosphate[c] + phosphate[c]'; % FBP
    'glycerol monophosphate[c] + H2O[c] => glycerol[c] + phosphate[c]'; % GPP
    'pyruvate[c] => acetaldehyde[c] + CO2[c]'; % PDC
    'ATP[c] + beta-D-fructofuranose 6-phosphate[c] => ADP[c] + beta-D-fructofuranose 1,6-bisphosphate[c]', % PFK
    '6-O-phosphono-D-glucono-1,5-lactone[c] + H2O[c] => 6-phospho-D-gluconate[c]'; % PGL
    };

model = changeRxns(model, rxnsToChange.rxns, rxnsToChange.equations, 3);
```

In [ ]:
```
% verify the new formulas work OK
balanceCheck = getElementalBalance(model, reactionsToEvalue, false, true);
balanceCheckTable = array2table(balanceCheck.leftComp - balanceCheck.rightComp, ...
        'VariableNames', balanceCheck.elements.abbrevs, ...
        'RowNames', reactionsToEvalue);
```

```
unbalancedTable = balanceCheckTable(reactionsToEvalue(balanceCheck.balanceStatus == 0), :);
fprintf('Table rows: %i', height(unbalancedTable));
disp(unbalancedTable);
```

```
Table rows: 00
```

In [ ]:
```
% Verify with solve if flux is zero
modelSolution = solveLP(model, 1);
disp(modelSolution);
printFluxes(model, modelSolution.x, false, tolerance,[],'▪ %flux\n%rxnID (%rxnName):\n\t%eqn\n\n');
```

```
      x: [57x1 double]
      f: 0
   stat: 1
    msg: 'Optimal solution found'

FLUXES:
FLUXES:
```

5. The same thing done in step 4 can be done with the function canProduce. There is a sister function called canConsume. It checks which metabolites can be consumed by the model. Change so that no production is allowed and run canConsume. One should see that 12 metabolites could be consumed even though the model is not allowed to produce anything. Pick one of them, force uptake of it by setting the lower bound to non-zero. If one does this one may not be able to get a feasible solution. That is because the problem solved by canConsume allows input of all metabolites, but the current model allows input for only O2 and glucose. Modify the model.b variable to allow for uptake of all metabolites.

> Question 4: study the fluxes and try to find the wrong one. What fix should be applied to the corresponding reaction?

Even if one fixes the problem one will see that the model can still get rid of O2. This is because of the reactions that were included for testing (NAD ⇔ NADH is not elementally or redox balanced). This part of the exercise is done, so those reactions can now be deleted.

- canConsume

```
function consumed=canConsume(model,mets)
```
Checks which metabolites that can be consumed by a model using the specified constraints.

In [ ]:
```
freeReactions.rxns
```

Out[ ]:
```
ans = 3x1 cell array
    {'FREE_ATP'  }
    {'FREE_NADH' }
    {'FREE_NADPH'}
```

In [ ]:
```
% first, close production. This include BOTH, exchange reactions, and the modification to the b vector
model.b = model.b(:, 1);
model = setParam(model, 'eq', getExchangeRxns(model, 'out'), 0);
% remove free ATP and electron carriers reactions
model = removeReactions(model, freeReactions.rxns);
```

In [ ]:
```
% second, use canConsume
consumed = canConsume(model);
disp(model.mets(consumed));
```

```
NOTE: The exchange reactions are assigned to the first compartment
```

I cannot get any of the 12 metabolites that the tutorial says. Checking the file 'tutorial4_solutions.m', I see that the expected result is the following:

```
{'MAL_m'}
    {'AKG_m'}
    {'ACA_c'}
    {'GLC_c'}
    {'CO2_c'}
    {'CO2_m'}
    {'ETH_c'}
    {'FUM_m'}
    {'O2_c' }
    {'OAA_m'}
    {'PYR_c'}
    {'SUC_m'}
```

I noticed that I made additional changes to the model in step 4 than those that appear in the solution file: adding water, balancing all problematic reactions, and fixing the chemical formula of some compounds. Apparently I already curated the chemical reaction the tutorial expect me to solve here 😅.

There is not point undoing my work. I will just copy that section of the solution file to study how they use the canConsume function.

```
I=canConsume(model);
disp(model.mets(I)); %These 12 metabolites can be consumed without any production
```

```
%Allow all uptake
model.b=[ones(numel(model.b),1)*-1000 model.b];

%Pick CO2 and force uptake of it
model=setParam(model,'eq',{'co2OUT'},-1); %Negative output means input
sol=solveLP(model);
printFluxes(model,sol.x,false,10^-5,[],'%rxnID (%rxnName):\n\t%eqn\n\t%flux\n'); %Now it works

%See that PDC converts pyruvate (3 carbons) to acetaldehyde (2 carbons)
%without any other products. If one googles, one may realize that CO2 is
%missing. This would be simpler to change in the Excel file (or using
%changeRxns), but one can change it here as an exercise. One therefore
%needs to find the index of the reactions and the index of cytosolic CO2 in
%order to change the reaction
Irxn=ismember(model.rxns,'PDC');
Imet=ismember(model.mets,'CO2_c');
model.S(Imet,Irxn)=1; %The coefficient is 1.0

%Display the new equation just to be sure
constructEquations(model,Irxn)

%The solution is now not feasible, meaning that it is no longer possible to
%force uptake of CO2 without any output
sol=solveLP(model);
```

In [ ]:
```
% The reaction was already curated in step 4
disp([rxnsToChange.rxns(5), rxnsToChange.equations(5)]);
```

```
    {'PDC'}    {'pyruvate[c] => acetaldehyde[c] + CO2[c]'}
```

6. Unbalanced reactions are a relatively small problem, since they are so easy to find. A much bigger problem is when metabolites are named differently even though they are meant to be the same. Use smallYeastBad2.xlsx from here on. A first check is to see which reactions can carry flux when one allows for all uptakes and outputs of exchange metabolites. There are several ways to check this but use the function simplifyModel here. The primary purpose of this function is to remove unnecessary stuff from a model to make it smaller, but since it removes “bad” reactions one can use it for error identification as well. If one runs it like it is in tutorial4.m one will see that there are about 20 metabolites and reactions that are dead ends. That is quite a lot, so take a look at the warnings from importExcelModel and see if it is possible to catch the obvious spelling error.

> Question 5: what correction should be applied to fix the spelling error?

In [ ]:
```
% although I set printWarnings to false, it still prints the warnings...
model = importExcelModel("./tutorial_data/smallYeastBad2.xlsx", true, false,true);
```

```
WARNING: The following metabolites already exist in the same compartment:
        6-O-phosphono-D-glucono-1,5-lactone[c]

WARNING: The following genes are not associated to a reaction:
        YPL061W

WARNING: No objective function found. This might be intended, but results in FBCv2 non-compliant SBML file when exported

WARNING: The following genes are not associated to a reaction:
        YPL061W

WARNING: No objective function found. This might be intended, but results in FBCv2 non-compliant SBML file when exported

WARNING: The following InChI strings are associated to more than one unique metabolite name:
        1/C3H7O6P/c4-1-3(5)2-9-10(6,7)8/h4H,1-2H2,(H2,6,7,8)/f/h6-7H
        1S/C6H14O12P2/c7-4-3(1-16-19(10,11)12)18-6(9,5(4)8)2-17-20(13,14)15/h3-5,7-9H,1-2H2,(H2,10,11,12)(H2,13,14,15)/t3-,4-,5+,6-/m1/s1
```

simplifyModel

```
function [reducedModel, deletedReactions, deletedMetabolites]=simplifyModel(model,deleteUnconstrained, deleteDuplicates, deleteZeroInterval, deleteInaccessible, deleteMinMax, groupLinear, constrainReversible, reservedRxns, suppressWarnings)
```

Simplifies a model by deleting reactions/metabolites

parameter	description
model	a model structure
deleteUnconstrained	delete metabolites marked as unconstrained (opt, default true)
deleteDuplicates	delete all but one of duplicate reactions (opt, default false)
deleteZeroInterval	delete reactions that are constrained to zero flux (opt, default false)
deleteInaccessible	delete dead end reactions (opt, default false)

In [ ]:

```
[reducedModel, deletedReactions, deletedMetabolites]=simplifyModel(model,[],[],[],true);
fprintf('Deleted reactions (%i):\n\n', length(deletedReactions));
deletedReactionsNames = model.rxnNames(getIndexes(model, deletedReactions, 'rxns'));
disp(table(deletedReactions, deletedReactionsNames));
fprintf('Deleted metabolites (%i):\n\n', length(deletedMetabolites));
deletedMetabolitesNames = model.metNames(getIndexes(model, deletedMetabolites, 'mets'));
disp(table(deletedMetabolites, deletedMetabolitesNames));
```

Deleted reactions (20):

deletedReactions	deletedReactionsNames
{'ACS'}	{'Acetyl-coenzyme A synthetase 1'}
{'DAR'}	{'glycerol-3-phosphate dehydrogenase'}
{'PGL'}	{'6-phosphogluconolactonase'}
{'ZWF'}	{'Glucose-6-phosphate 1-dehydrogenase'}
{'acOUT'}	{'Production of acetate'}
{'CAT2'}	{'Carnitine O-acetyltransferase'}
{'GND'}	{'6-phosphogluconate dehydrogenase, decarboxylating'}
{'GPP'}	{'sn-glycerol-3-phosphate phosphohydrolase'}
{'GROWTH'}	{'Growth'}
{'ADH2'}	{'Alcohol dehydrogenase rev'}
{'IDP1'}	{'Isocitrate dehydrogenase'}
{'MAE1'}	{'NAD-dependent malic enzyme'}
{'biomassOUT'}	{'Production of biomass'}
{'glyOUT'}	{'Production of glycerol'}
{'ACO'}	{'Aconitate hydratase, mitochondrial'}
{'KGD1KGD2'}	{'Alpha-ketoglutarate dehydrogenase'}
{'CIT'}	{'Citrate synthase, mitochondrial'}
{'LSC1LSC2'}	{'Succinyl-CoA ligase'}
{'PDH'}	{'Pyruvate dehydrogenase complex'}
{'CO2TRANS'}	{'CO2 transport'}

Deleted metabolites (21):

deletedMetabolites	deletedMetabolitesNames
{'G15L_c'}	{'6-O-phosphono-D-glucono-1,5-lactone'}
{'G15Lc'}	{'6-O-phosphono-D-glucono-1,5-lactone'}
{'AC_c'}	{'acetate'}
{'GLYP_c'}	{'glycerone phosphate'}
{'P6G_c'}	{'6-phospho-D-gluconate'}
{'ACCOA_c'}	{'acetyl-CoA'}
{'COA_c'}	{'coenzyme A'}
{'GP_c'}	{'glycerol monophosphate'}
{'BIOMASS_c'}	{'biomass'}
{'GLY_c'}	{'glycerol'}
{'NADP_c'}	{'NADP(+)'}

```
{'NADP_m'   }       {'NADP(+)'                          }
{'NADPH_c'  }       {'NADPH'                            }
{'NADPH_m'  }       {'NADPH'                            }
{'AKG_m'    }       {'2-oxoglutarate'                   }
{'ICI_m'    }       {'isocitrate'                       }
{'CI_m'     }       {'citrate'                          }
{'SUCCOA_m' }       {'succinyl-CoA'                     }
{'ACCOA_m'  }       {'acetyl-CoA'                       }
{'COA_m'    }       {'coenzyme A'                       }
{'CO2_m'    }       {'CO2'                              }
```

Same name, but different ID:

deletedMetabolites	deletedMetabolitesNames
{'G15L_c' }	{'6-O-phosphono-D-glucono-1,5-lactone'}
{'G15Lc' }	{'6-O-phosphono-D-glucono-1,5-lactone'}

In [ ]:
```
badMetaboliteIndex = getIndexes(model, 'G15Lc', 'mets');
okMetaboliteIndex = getIndexes(model, 'G15L_c', 'mets');
[~, reactionToCorrectIndexes] = find(model.S(badMetaboliteIndex, :))
% This is not the case, but it is possible that both species participate in the same reaction.
% The following aproach is only suitable if both species dont share reactions.
model.S(okMetaboliteIndex, reactionToCorrectIndexes) = model.S(badMetaboliteIndex, reactionToCorrectIndexes);
model=removeMets(model,'G15Lc');
```

Out[ ]:
```
reactionToCorrectIndexes =
  19
```

7. That did not help very much. Sometimes it is exceedingly difficult to find out where the root of the problem is. This is particularly true if it is in a region with many interconversions between metabolites and no clear input/output (Figure 3).

Figure 3. An example of pathway featuring many interconversions between metabolites and unclear input/output. If one reaction is wrong here it will be difficult to find since everything looks so connected because it is produced and consumed in many reactions.

A powerful but somewhat tricky function is checkProduction. It helps to identify metabolites needed to synthesize in order to to have the net synthesis of everything. Look at the suggestions from checkProduction if when running it like in the tutorial4.m. The function minToConnect tells that is needed to synthesize 12 metabolites to have the net synthesis of everything. However, 8 of them are co-factors or contain co-factors. Since there is no net synthesis of co-factors in this small model those are not interesting (coenzyme A or ATP are not synthesized from glucose). One should look at the top one that is not a co-factor. This one is a bit tricky, and one might want to look it up in KEGG.

> Question 6: what is the suspicious similarity between some metabolites?

In [ ]:
```
[reducedModel, deletedReactions, deletedMetabolites]=simplifyModel(model,[],[],[],true);
fprintf('Deleted metabolites (%i):\n\n', length(deletedMetabolites));
deletedMetabolitesNames = model.metNames(getIndexes(model, deletedMetabolites, 'mets'));
deletedMetabolitesTable = table(deletedMetabolites, deletedMetabolitesNames);
disp(deletedMetabolitesTable);
```

```
Deleted metabolites (2020):

    deletedMetabolites            deletedMetabolitesNames
    __________________    ____________________________________________

      {'AC_c'     }       {'acetate'                                 }
      {'GLYP_c'   }       {'glycerone phosphate'                     }
      {'ACCOA_c'  }       {'acetyl-CoA'                              }
      {'COA_c'    }       {'coenzyme A'                              }
      {'GP_c'     }       {'glycerol monophosphate'                  }
      {'BIOMASS_c'}       {'biomass'                                 }
      {'GLY_c'    }       {'glycerol'                                }
      {'NADP_c'   }       {'NADP(+)'                                 }
      {'NADP_m'   }       {'NADP(+)'                                 }
      {'NADPH_c'  }       {'NADPH'                                   }
      {'NADPH_m'  }       {'NADPH'                                   }
      {'AKG_m'    }       {'2-oxoglutarate'                          }
      {'G15L_c'   }       {'6-O-phosphono-D-glucono-1,5-lactone'}
      {'P6G_c'    }       {'6-phospho-D-gluconate'                   }
      {'ICI_m'    }       {'isocitrate'                              }
      {'CI_m'     }       {'citrate'                                 }
      {'SUCCOA_m' }       {'succinyl-CoA'                            }
      {'ACCOA_m'  }       {'acetyl-CoA'                              }
      {'COA_m'    }       {'coenzyme A'                              }
      {'CO2_m'    }       {'CO2'                                     }
```

checkProduction

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

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

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 [ ]:
```
[notProducedMets, ~, neededForProductionMat, minToConnect]=checkProduction(model,true,model.comps,false);
fprintf('mintToConnect size: (%i)\n', length(minToConnect));
disp(minToConnect);
```

```
NOTE: The exchange reactions are assigned to the first compartment
mintToConnect size: (10mintToConnect size: (10)
    {'NADP(+)[c] (connects 9 metabolites)'            }
    {'NADP(+)[m] (connects 4 metabolites)'            }
    {'glycerone phosphate[c] (connects 3 metabolites)'}
    {'succinyl-CoA[m] (connects 3 metabolites)'       }
    {'ADP[c] (connects 2 metabolites)'                }
    {'FADH2[m] (connects 2 metabolites)'              }
    {'NAD(+)[c] (connects 2 metabolites)'             }
    {'NAD(+)[m] (connects 2 metabolites)'             }
    {'acetate[c] (connects 1 metabolites)'            }
    {'biomass[c] (connects 1 metabolites)'            }
```

The top non-related to cofactor metabolite is {'glycerone phosphate[c] (connects 3 metabolites)'}.

KEGG Glycerone phosphate

- **Entry:**
  C00111
- **Name:**
  Glycerone phosphate
  Dihydroxyacetone phosphate
  3-Hydroxy-2-oxopropyl phosphate

- **Formula:**
  C3H7O6P

In [ ]:
```
% let's look for compounds with the same formula
sameFormulaIndex = find(strcmp(model.metFormulas, 'C3H7O6P'));
metabolitesToCheck = table(sameFormulaIndex, ...
    model.metFormulas(sameFormulaIndex), ...
    model.mets(sameFormulaIndex), ...
    model.metNames(sameFormulaIndex), ...
    'VariableNames', {'index', 'formula', 'ID', 'Name'});
disp(metabolitesToCheck);
```

```
   index      formula          ID                     Name
   _____    ___________    _________    ___________________________________

    25      {'C3H7O6P'}    {'GA3P_c'}    {'D-glyceraldehyde 3-phosphate'}
    35      {'C3H7O6P'}    {'DHAP_c'}    {'dihydroxyacetone phosphate'  }
    36      {'C3H7O6P'}    {'GLYP_c'}    {'glycerone phosphate'         }


   _____    ___________    _________    ___________________________________

    25      {'C3H7O6P'}    {'GA3P_c'}    {'D-glyceraldehyde 3-phosphate'}
    35      {'C3H7O6P'}    {'DHAP_c'}    {'dihydroxyacetone phosphate'  }
    36      {'C3H7O6P'}    {'GLYP_c'}    {'glycerone phosphate'         }
```

Notice that DHAP_c and GLYP_c are synonyms.

In [ ]:
```
badMetaboliteIndex = getIndexes(model, 'DHAP_c', 'mets');
okMetaboliteIndex = getIndexes(model, 'GLYP_c', 'mets');
[~, reactionToCorrectIndexes] = find(model.S(badMetaboliteIndex, :))
% This is not the case, but it is possible that both species participate in the same reaction.
% The following aproach is only suitable if both species dont share reactions.
model.S(okMetaboliteIndex, reactionToCorrectIndexes) = model.S(badMetaboliteIndex, reactionToCorrectIndexes);
model=removeMets(model,'DHAP_c');
```

Out[ ]:
```
reactionToCorrectIndexes = 1x2 double
   12   13
```

In [ ]:
```
[reducedModel, deletedReactions, deletedMetabolites]=simplifyModel(model,[],[],[],true);
fprintf('Deleted metabolites (%i):\n\n', length(deletedMetabolites));
deletedMetabolitesNames = model.metNames(getIndexes(model, deletedMetabolites, 'mets'));
deletedMetabolitesTable = table(deletedMetabolites, deletedMetabolitesNames);
disp(deletedMetabolitesTable);
```

```
Deleted metabolites (17):

17):

    deletedMetabolites            deletedMetabolitesNames
    __________________    ____________________________________

      {'AC_c'     }       {'acetate'                          }
      {'ACCOA_c'  }       {'acetyl-CoA'                       }
      {'COA_c'    }       {'coenzyme A'                       }
      {'BIOMASS_c'}       {'biomass'                          }
      {'NADP_c'   }       {'NADP(+)'                          }
      {'NADP_m'   }       {'NADP(+)'                          }
      {'NADPH_c'  }       {'NADPH'                            }
      {'NADPH_m'  }       {'NADPH'                            }
      {'AKG_m'    }       {'2-oxoglutarate'                   }
      {'G15L_c'   }       {'6-O-phosphono-D-glucono-1,5-lactone'}
      {'P6G_c'    }       {'6-phospho-D-gluconate'            }
      {'ICI_m'    }       {'isocitrate'                       }
      {'CI_m'     }       {'citrate'                          }
      {'SUCCOA_m' }       {'succinyl-CoA'                     }
      {'ACCOA_m'  }       {'acetyl-CoA'                       }
      {'COA_m'    }       {'coenzyme A'                       }
      {'CO2_m'    }       {'CO2'                              }
```

8. Still quite a bit of dead ends, and nothing that immediately looks like it would fix everything. It could be that some reactions are missing. One could try to include reactions from a set of other models to fill the gaps. This is a computationally expensive task for a large network, but for this small model it is easy. One could use any model structure, but here one can take the small yeast model from Tutorial 3. Run the code and include the suggested reaction(s). Run the previous tests to make sure that everything works.

In [ ]:
```
templateModel = importExcelModel('./tutorial_data/smallYeast.xlsx', [], false, true);
```

- fillGaps

Uses template model(s) to fill gaps in a model

```
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

In [ ]:
```
[newConnected, cannotConnect, addedRxns, newModel]=fillGaps(model,{templateModel});
```

```
WARNING: Removing dead-end reactions before removing exchange metabolites

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
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 115 rows, 311 columns and 748 nonzeros
Model fingerprint: 0x770939d1
Variable types: 248 continuous, 63 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 40 rows and 188 columns
Presolve time: 0.01s
Presolved: 75 rows, 123 columns, 333 nonzeros
Variable types: 82 continuous, 41 integer (41 binary)
Found heuristic solution: objective 17.0000000
Root relaxation presolve removed 42 rows and 43 columns
Root relaxation presolved: 33 rows, 80 columns, 247 nonzeros


Root relaxation: objective 1.004856e+00, 24 iterations, 0.01 seconds (0.00 work units)
```

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

     0     0    1.00486    0    1   17.00000    1.00486  94.1%     -    0s
H    0     0                       2.0000000    1.00486  49.8%     -    0s
     0     0    1.00486    0    1    2.00000    1.00486  49.8%     -    0s

Explored 1 nodes (24 simplex iterations) in 0.08 seconds (0.00 work units)
Thread count was 4 (of 4 available processors)

Solution count 2: 2 17

Optimal solution found (tolerance 1.00e-12)
Best objective 2.000000000000e+00, best bound 2.000000000000e+00, gap 0.0000%
```

In [ ]:
```
fprintf('Reaction added from the template(s) (%i):\n', length(addedRxns));
disp(addedRxns);
fprintf('Reaction that could be connected (%i):\n', length(newConnected));
disp(newConnected);
fprintf('Reactions that could not be connected (%i):\n', length(cannotConnect));
disp(cannotConnect);
```

```
Reaction added from the template(s) (22):
    {'ALD6'          }
    {'PFK_smallYeast'}

Reaction that could be connected (32):
    {'ACS'       }
    {'ADH1'      }
    {'ADH2'      }
    {'ATPX'      }
    {'CAT2'      }
    {'CIT'       }
    {'DAR'       }
    {'FADHX'     }
    {'GND'       }
    {'GPP'       }
    {'GROWTH'    }
    {'HXK'       }
    {'IDP1'      }
    {'KGD1KGD2'  }
    {'MAE1'      }
    {'NADHX'     }
    {'PCK'       }
    {'PDC'       }
    {'PDC_2'     }
    {'PDH'       }
    {'PGL'       }
    {'PYC'       }
    {'PYK'       }
    {'ShuttleX'  }
    {'ZWF'       }
    {'acOUT'     }
    {'biomassOUT'}
    {'co2OUT'    }
    {'ethOUT'    }
    {'glcIN'     }
    {'glyOUT'    }
    {'o2IN'      }

Reactions that could not be connected (19):
    {'ACO'      }
    {'CO2TRANS' }
    {'ENO'      }
```

```
{'FBA'      }
{'FBP'      }
{'FUM1'     }
{'GLD'      }
{'GPM'      }
{'LSC1LSC2' }
{'MDH1'     }
{'PFK'      }
{'PGI'      }
{'PGK'      }
{'RPE'      }
{'RPI'      }
{'TAL1'     }
{'TKI1TKI2a'}
{'TKI1TKI2b'}
{'TPI'      }
```

In [ ]:
```
[reducedModel, deletedReactions, deletedMetabolites]=simplifyModel(newModel,[],[],[],true);
fprintf('Deleted reactions (%i):\n\n', length(deletedReactions));
deletedReactionsNames = newModel.rxnNames(getIndexes(newModel, deletedReactions, 'rxns'));
disp(table(deletedReactions, deletedReactionsNames));
fprintf('Deleted metabolites (%i):\n\n', length(deletedMetabolites));
deletedMetabolitesNames = newModel.metNames(getIndexes(newModel, deletedMetabolites, 'mets'));
disp(table(deletedMetabolites, deletedMetabolitesNames));
```

```
WARNING: Removing dead-end reactions before removing exchange metabolites

Deleted reactions (0Deleted reactions (0):

Deleted metabolites (0):
```

Zero deleted reactions and metabolites. All OK 😳.

9. Finished! And do not forget that the gapReport function does all these things.

- gapReport

Performs a gap analysis and summarizes the results

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

In [ ]:
```
model = importExcelModel("./tutorial_data/smallYeastBad2.xlsx", true, false,true);
```

```
WARNING: The following metabolites already exist in the same compartment:
        6-O-phosphono-D-glucono-1,5-lactone[c]

WARNING: The following genes are not associated to a reaction:
        YPL061W

WARNING: No objective function found. This might be intended, but results in FBCv2 non-compliant SBML file when exported

WARNING: The following InChI strings are associated to more than one unique metabolite name:
        1/C3H7O6P/c4-1-3(5)2-9-10(6,7)8/h4H,1-2H2,(H2,6,7,8)/f/h6-7H
        1S/C6H14O12P2/c7-4-3(1-16-19(10,11)12)18-6(9,5(4)8)2-17-20(13,14)15/h3-5,7-9H,1-2H2,(H2,10,11,12)(H2,13,14,15)/t3-,4-,5+,6-/m1/s1

WARNING: The following genes are not associated to a reaction:
        YPL061W

WARNING: No objective function found. This might be intended, but results in FBCv2 non-compliant SBML file when exported

WARNING: The following InChI strings are associated to more than one unique metabolite name:
        1/C3H7O6P/c4-1-3(5)2-9-10(6,7)8/h4H,1-2H2,(H2,6,7,8)/f/h6-7H
        1S/C6H14O12P2/c7-4-3(1-16-19(10,11)12)18-6(9,5(4)8)2-17-20(13,14)15/h3-5,7-9H,1-2H2,(H2,10,11,12)(H2,13,14,15)/t3-,4-,5+,6-/m1/s1
```

In [ ]:
```
gapReport(model,{templateModel});
```

```
Gap analysis for smallYeastBad2 - Central carbon metabolism for yeast

***Overview
51 out of 53 reactions cannot carry flux (21 if net production of all metabolites is allowed)
50 out of 54 metabolites are unreachable (17 if net production of all metabolites is allowed)

***Isolated subnetworks
A total of 1 isolated sub-networks are present in the model
        1. 54 metabolites

***Metabolite connectivity
NOTE: The exchange reactions are assigned to the first compartment
To enable net production of all metabolites, a total of 11 metabolites must be connected
Top 10 metabolites to connect:
        1. NADP(+)[m] (connects 9 metabolites)
        2. glycerone phosphate[c] (connects 3 metabolites)
        3. NADP(+)[c] (connects 3 metabolites)
        4. succinyl-CoA[m] (connects 3 metabolites)
        5. 6-O-phosphono-D-glucono-1,5-lactone[c] (connects 2 metabolites)
        6. ADP[c] (connects 2 metabolites)
        7. FADH2[m] (connects 2 metabolites)
        8. NAD(+)[c] (connects 2 metabolites)
        9. NAD(+)[m] (connects 2 metabolites)
        10. acetate[c] (connects 1 metabolites)

***Automated gap-filling
WARNING: Removing dead-end reactions before removing exchange metabolites

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 116 rows, 311 columns and 745 nonzeros
Model fingerprint: 0xae909ed2
Variable types: 248 continuous, 63 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 46 rows and 197 columns
Presolve time: 0.00s
Presolved: 70 rows, 114 columns, 310 nonzeros
Variable types: 76 continuous, 38 integer (38 binary)
Found heuristic solution: objective 22.0000000
Root relaxation presolve removed 39 rows and 40 columns
Root relaxation presolved: 31 rows, 74 columns, 230 nonzeros


Root relaxation: objective 2.005056e+00, 23 iterations, 0.01 seconds (0.00 work units)

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

     0     0    2.00506    0    2   22.00000    2.00506  90.9%     -    0s
H    0     0                       4.0000000    2.00506  49.9%     -    0s
     0     0     cutoff    0         4.00000    4.00000  0.00%     -    0s

Cutting planes:
  Implied bound: 1
  Clique: 1
  Flow cover: 1

Explored 1 nodes (23 simplex iterations) in 0.06 seconds (0.00 work units)
Thread count was 4 (of 4 available processors)

Solution count 2: 4 22

Optimal solution found (tolerance 1.00e-12)
Best objective 4.000000000000e+00, best bound 4.000000000000e+00, gap 0.0000%
31 unconnected reactions can be connected by including 4 reactions from
smallYeast
```