modelBorgifier Manual

J. T. Sauls and J. M. Buescher

June 14, 2013

Contents

"We will add your biological and technological distinctiveness to our own. Resistance is futile."
– Borg hail.

1 Set-up

1.1 Installation

Assuming that the openCOBRA Toolbox is installed and functional on your system, installation only requires that you add the modelBorgifier directory to the Matlab path.

1.2 Dependencies

modelBorgifier relies on the openCOBRA Toolbox[6] for reading and writing functions (themselves dependent on the Matlab sbmltoolbox, included with the standard openCOBRA install), and any FBA analysis. Reading CSV files utilizes the csv2cell.m function from MatlabCental (http://www.mathworks.com/matlabcentral/fileexchange/20836-csv2cell). For machine learning, the LIBSVM library[2] (http://www.csie.ntu.edu.tw/~cjlin/libsvm/) is needed for SVM optimization. The script *findjobj.m* by Yair Altman (http://www.mathworks.com/matlabcentral/fileexchange/14317-findjobj-find-java-handles-of-matlab-graphic-objects) is used for GUI presentation. These scripts are located in the directory */modelBorgifier/dependencies* with accompanying licenses. The linear and exponential optimization techniques use functions available through the Matlab Optimization Toolbox.

1.3 Test script

The script *driveModelBorgifier.m* demonstrates the order of use and implementation of the functions in the modelBorgifier suite. It is designed to be a guide when adding a new model and uses the models iJO1366 and iBSU1103 as an example comparison and merging. It also utilizes the open databases from the Model SEED to add additional annotation information based on SEED IDs.

iJO1366[4] is a popular *E. coli* model which can be downloaded in SBML format from http://www.nature.com/msb/journal/v7/n1/extref/msb201165-s3.xml (paper: http://www.nature.com/msb/journal/v7/n1/full/msb201165.html; license http://creativecommons.org/licenses/by-nc-sa/3.0/ note this is a non-commercial license).

iBSU1103[3] is a *B. subtillus* model which uses IDs consistent with the Model SEED (model in SBML: http://genomebiology.com/content/supplementary/gb-2009-10-6-r69-s4.xml; paper: http://genomebiology.com/2009/10/6/R69; license: http://creativecommons.org/licenses/by/2.0/).

To use the test suite as is, download both the above models and place the SBML files in the */modelBorgifier/test* directory.

The Model SEED[5] (http://seed-viewer.theseed.org/) is a collection of stoichiometric models, and the way to access models created by the RAST[1] annotation server. Models in the Model SEED draw from two databases, for reactions (seed-viewer.theseed.org/ModelSEEDdownload.cgi?biochemistry=1) and metabolites

(seed-viewer.theseed.org/ModelSEEDdownload.cgi?biochemCompounds=1). These databases are open and freely available, and are included in */modelBorgifier/test/SEED_db* with the appropriate license for convenience. If you retrieve these databases yourself, convert them to CSVs, semicolon delimited, for them to accessible by modelBorgifier.

2 Procedure

To quickly demo the modelBorgifier, simply follow *driveModelBorgifier.m*. Additionally, all scripts have documentation located in the .m file that can be accessed via Matlab's help command. The following is a description of the comparison and modeling process.

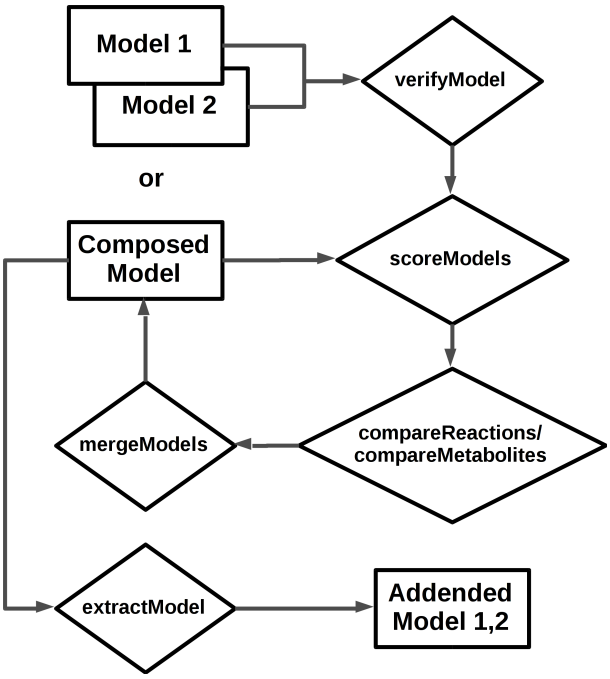

Figure 1: Program flow.

2.1 Load the comparison model.

Load the model you wish to compare (Cmodel) and merge with the template model (Tmodel), from its current format.

If the model is already in openCOBRA compatible SBML format, then use the openCOBRA function *readCbmodel.m*.

```
>> Cmodel = readCbModel(fileName);
```

If the model is in a less readily machine readable format, such as .xls, then use an appropriate custom written script (converting .xls to .csv may be necessary if you are not on a Windows machine). Some example scripts are located in */modelBorgifier/readModel* (should these be included). The model may be provided in SBML format, but an additional file contains useful information. If so, consider using using a script to extract information from this additional file(s).

2.1.1 Verify format

Use *verifyModel.m* to ensure that Cmodel has all relevant information and can be successfully compared against Tmodel for already existing reactions and metabolites.

```
>> Cmodel = verifyModel(Cmodel);
```

Verify model ensures all required information arrays are present, that all reactions are forward or reversible (not reverse), and that all reaction and metabolite names are unique. If reaction names are not unique, *verifyModel.m* will prompt you to put in new names, or you can use *checkCobraNamesUnique.m* from the openCOBRA toolbox to systematically rename all duplicate reactions. Finally, *verifyModel.m* formats reaction and metabolite names so they are lower case and do not contain obtuse characters, gives the fields a conserved order, and makes sure the model has a name.

2.1.2 Add information from the SEED database

Optional. If your model either came from the Model SEED or uses SEED identifiers (reaction names in the form rxnXXXXX and metabolites in the form cpdXXXXX), then the script *addSEEDInfo.m* usese those identifiers too add information to the model via the databases mentioned above.

```
>> Cmodel = addSEEDInfo(Cmodel,rxnDbFileName,cpdDbFileName);
```

This greatly aids comparison and matching, especially when the template model does not reference the SEED IDs otherwise. It also replaces the metabolite names with the appropriate abbreviations in the reaction equations. The databases should be in semicolon delimited CSV format. So I guess that's SCSV?

2.1.3 Verify model holds flux

Optional. Run a simple FBA simulation to see if the model holds flux. Make sure there is an objective function set, and that the bounds are reasonable. If the model doesn't hold flux before it is added to merged, than it is more difficult to determine if it was added correctly afterwards.

```
>> solverOkay = changeCobraSolver('glpk','LP');
>> CmodelSol = optimizeCbModel(Cmodel);
```

2.2 Load the template model

2.2.1 Load Tmodel from SBML

In this case, load and verify the model as above, optionally adding information from the SEED database and checking if the model holds flux. Then, convert the model to an appropriate template for comparison using *buildTmodel.m*.

```
>> Tmodel = buildTmodel(Tmodel);
```

2.2.2 Load Tmodel from .mat file

If previous models have been merged, then this composition model can be used as Tmodel. Simply load the Matlab workspace (.mat file) containing the composition model.

2.3 Compare models

Compare Cmodel to Tmodel by running the script *compareCbModels.m*.

```
>> [Cmodel,Tmodel,score,Stats] = compareCbModels(Cmodel,Tmodel);
```

This script first assembles additional information arrays in both models for the purpose of comparison, returning the information with the original models. It then compares Cmodel to Tmodel, pairwise by reaction, allocating a normalized score based on similarity. This may take ~4 hours for two models of around 2000 reactions each.

These scores are held in the 3D array score, which is of size [number of reactions in Cmodel x number of reactions in Tmodel x number of scoring parameters]. Stats contains information on best matches.

2.4 Match reactions and metabolites

Pair reactions and metabolites from Cmodel to Tmodel using *reactionCompare.m*. Matching is initiated with the following command.

```
>> [rxnList,metList,Stats] = reactionCompare(Cmodel,Tmodel,score);
```

reactionCompare.m calls a GUI that allows the user to choose a match for a given reaction in Cmodel, and also to match the metabolites for that reaction. *reactionCompare.m* gives auto-matching options described below. *reactionCompare.m* outputs rxnList and metList, which designate reactions and metabolites in Cmodel to the index of their match in Tmodel, or indicate them as new or need review. Stats contains weighting information in addition to best matches. Matching can be suspended at anytime, so subsequent calls to *reactionCompare.m* should include these output arguments.

```
>> [rxnList,metList,Stats] = reactionCompare(Cmodel,Tmodel,score, ...
rxnList,metList,Stats);
```

2.4.1 Optimizing scoring parameters

The initial comparison between models is based on forty parameters which are arbitrarily waited. The parameters cover a large swath of information available in genome-scale reconstructions, but do not necessarily emphasize the relevant information in the models you have at hand. For this reason it is important to optimize the weighting of the scoring parameters. This is accomplished through the *reactionCompare.m* GUI via one of three methods; support vector machines (SVM), linear optimization, or exponential optimization. It is recommended to check the optimize parameters box when using SVM.

All three methods use previously declared reactions as a training set to determine which parameters are pertinent, and weighs them accordingly. For example, if KEGG IDs are not available in one of the models, then that corresponding parameter will be down weighted (most likely to zero), as it holds no bearing on the matching process. Likewise, if reaction EC numbers prove to be particularly good at predicting matches, then it will be more heavily weighted.

2.4.2 Auto-matching based on scores

Optimizing the scoring parameters helps create scoring distance between probable matches and probable new reactions and metabolites. Thus, auto-matching based on cutoff scores can be done more confidently after optimization. Simply set a low cutoff to automatically declare new reactions when no match exists with a score above a certain value. The same applies for a high cutoff for automatic pairing. Additionally, a margin can be set, such that a reaction will only be automatically paired with a match, when the score for the match is better that the second best match by a certain margin. Auto-matching in the same manner exist for metabolites.

As metabolites become declared, the status of the reactions which contain those metabolites can be elucidated. For example, if all of a reaction's metabolites have matches in the template model, and the template model also contains a reaction with those exact metabolites, than the original reaction is automatically paired with the one from the template. Conversely, if a reaction contains a metabolite that does not exist in the template model, then it is declared as new.

2.4.3 Matching flow

The following is a general guide to effective and efficient matching.

1. Consider a small subset of reactions, ie 20-50. You can choose to consider reactions with at least one match with a score above a certain cutoff by adjusting the slider under the score frequency histogram in the GUI.

2. Declare these reactions as either having a match or new.

3. After each reaction is declared, a sub-GUI that allows the user to compare metabolites will start. Similarly, declare metabolites as having a match[1] or new.

4. Use one of the optimizing functions to partition scores, then use the auto-match function to declare for-sure matches and for-sure new reactions. Adjust the auto-match parameters based on the scores you observe. The metabolite comparison GUI will most likely re-launch for these newly declared reactions.

5. Repeat steps 1-4 until all reactions and metabolites have matches in Tmodel or are declared as new.

[1] Reactions that are given a match should also have matches for all of their metabolites.

3 Merge model with database

Use *mergeModels.m* to merge Cmodel with Tmodel. It returns the composition model, TmodelC, and Cmodel as extracted from the composition. Stats now additionally contains comparison information between models in the composition.

```
>> [TmodelC, Cspawn, Stats] = mergeModels(Cmodel, Tmodel, rxnList, metList, ...
Stats)
```

For reactions and metabolites for which matches have been found, the script adds information to the current entries in Tmodel. New entries are made for new reactions and metabolites from Cmodel. *mergeModels.m* calls *cleanToModel.m*, which may prompt the user to clarify duplicate reaction and metabolite names to ensure the composition only contains unique entries.

Merging is also done in a manner that ensures the original models can be retrieved in their original mathematical form. This may require that some metabolites be re-reviewed, and that some reactions are declared as new to maintain stoichiometric fidelity.

4 Appendix

4.1 Description of scoring parameters

The reaction scoring parameters are defined in *compareCbModels.m*. Most parameters come in a pair, one allotting points for a match, the other removing points for a miss between any to given reactions. String comparison supports multiple pieces of information contained within one field as separated by a pipe ('|').

rxnName	Reaction names match.
rxnNameL	Reaction long name match. Also cross checks to ID.
ec	Reaction EC number match.
rxnKEGG	Reaction KEGG ID match.
rev	Reaction reversibility match.
sub	Reaction subsystem match.
gr	Genes match.
metNum	Same number of metabolites.
metSto	Same stoichiometry.
metName	Metabolites have same name. Points alloted per metabolite.
metForm	Metabolites have same formula. Points alloted per metabolite.
metKEGG	Metabolites have same formula. Points alloted per metabolite.
metSEED	Metabolites have same SEED ID. Points alloted per metabolite.
reacNum	Same number of reactants.

reacSto Same reactant stoichiometry.

prodNum Same number of products.

prodSto Same product stoichiometry.

rxnComp Reactions involve the same compartments.

rxnSEED Reaction SEED ID match.

metBonus Bonus alloted if all the metabolite names, formulas, KEGG or SEED IDs match.

rxnNet Network similarity around reaction.

Metabolites are also scored individually on the following parameters: ID, compartment, name, formula, charge, KEGGID, SEEDID, ChEBIID, PubChemID, and InChIString.

4.2 Description of scripts

All scrips have usage descriptions located within the .m file itself. Scripts in */modelBorgifier/src* are core to the comparison process, while those in */modelBorgifier/tools* are mostly concerned with formatting data to aid comparison.

4.3 Important terms

Cmodel Compare Model. The model that is to be compared and merged to Tmodel.

Tmodel Template Model. The model to which Cmodel is compared and merged. This can simply be another model, or a composition of many previously combined models.

rxnList The array which pairs reactions from Cmodel to their match in Tmodel, or declares them as new.

metList Analogous array for metabolites.

References

[1] Aziz, R. K. et al. The RAST Server: rapid annotations using subsystems technology. BMC genomics 9, 75 (2008).

[2] C.-C. Chang and C.-J. Lin. LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27 (2011).

[3] Henry, C. S., Zinner, J. F., Cohoon, M. P. & Stevens, R. L. iBsu1103: a new genome-scale metabolic model of Bacillus subtilis based on SEED annotations. Genome biology 10, R69 (2009).

[4] Orth, J. D. et al. A comprehensive genome-scale reconstruction of Escherichia coli metabolism—2011. Molecular Systems Biology 7, (2011).

[5] Overbeek, R. et al. The subsystems approach to genome annotation and its use in the project to annotate 1000 genomes. Nucleic acids research 33, 5691–702 (2005).

[6] Schellenberger, J. et al. Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. Nature Protocols 6, 1290–1307 (2011).