The MONGOOSE rational arithmetic toolbox

Christopher Le, Leonid Chindelevitch

Abstract The modeling of metabolic networks has seen a rapid expansion following the complete sequencing of thousands of genomes. The constraint-based modeling framework has emerged as one of the most popular approaches to reconstructing and analyzing genome-scale metabolic models. Its main assumption is that of a quasi-steady-state, requiring that the production of each internal metabolite be balanced by its consumption. However, due to the multiscale nature of the models, the large number of reactions and metabolites, and the use of floating-point arithmetic for the stoichiometric coefficients, ensuring that this assumption holds can be challenging.

The MONGOOSE toolbox addresses this problem by using rational arithmetic, thus ensuring that models are analyzed in a reproducible manner and consistently with modeling assumptions. In this chapter we present a protocol for the complete analysis of a metabolic network model using the MONGOOSE toolbox, via its newly developed GUI, and describe how it can be used as a model-checking platform both during and after the model construction process.

Leonid Chindelevitch

Simon Fraser University, Burnaby, BC, Canada; e-mail: leonid@sfu.ca

Key words: Metabolic networks, constraint-based analysis, rational arithmetic, user interface, reproducibility

1 Introduction

The rapid explosion of genomic technologies has led to a proliferation of complete genome sequences for a variety of organisms, from Archaea to Zooplankton [1]. The metabolism of many of these organisms is important, for reasons ranging from antimicrobial target discovery to metabolic engineering [2]. Given the complexity of metabolic processes, strategies such as systematic gene knockouts may not always be feasible, requiring the development of computational methods for rapid hypothesis generation and prioritization [3]. For this reason, a unified modeling framework enabling a genome-scale analysis of metabolism in these organisms becomes critical.

Such a framework, typically called constraint-based metabolic modeling, emerged in the early 1990s [4, 5]. Its main premise is that accurate predictions can be made simply from the knowledge of the full complement of reactions available to an organism, without any detailed information on the kinetic parameters governing the rates of those reactions. By making the quasi-steady-state assumption, meaning that production and consumption balance one another for each internal metabolite, constraint-based models can be used to predict the distribution of reaction fluxes, as well as features such as growth rates, essential and synthetic lethal reactions, and minimal media [6]. This framework has been successfully used to analyze and make predictions about a number of organisms, in all the major domains of life [7].

However, many of the metabolic network models analyzed in the constraint-based framework have turned out to be poorly conditioned, meaning that a small perturbation to their coefficients could result in a large change in the model predictions [8]. Such perturbations are commonplace during analysis which is typically performed in floating-point arithmetic, an approximate representation of numbers on a computer that allows their fast manipulation. For this reason, the author has proposed an alternative way of analyzing these networks, using rational arithmetic, in which the coefficients of each reaction are represented as ratios of integers [8]. This representation, together with key insights into the structure of metabolic network models, form the foundation of the MONGOOSE toolbox for metabolic network analysis described in this chapter.

2 Materials

In this section we describe the computational tools that need to be in place in order to make use of the MOGNOOSE toolbox, and give detailed instructions on how to set them up. Since MONGOOSE was developed for a Unix-based environment, including Mac OS, we focus on those environments, and in particular use Mac OS as a running example. However, it is possible to run it in a Windows environment using a virtual machine - an approach based on a platform such as Cygwin is, however, unlikely to work.

2.1 Installing MONGOOSE and the GUI - the easy way

The easiest way to install MONGOOSE and its graphical user interface (GUI) is via the Docker [9] distribution available at `https://hub.docker.com/r/ctlevn/mongoose/`. If you follow the instructions on that page you will be able to get MONGOOSE up and running in no time. The only disadvantage is that the files you create will need to be transferred from the Docker container back to your local hard drive at the end of each session. Alternatively, if you know Docker well you can update your container to include the results of the work you have performed, and restart it based on the same container identifier the next time around.

However, if Docker does not work for you for whatever reason, you will need to follow the more complex path outlined in the rest of this section.

2.2 Installing Python

The latest version of the Python software [10] can always be downloaded from the Python website. The first version of the toolbox, MONGOOSE 1.1, has been developed under Python 2.6, and is still available for download from GitHub, `https://github.com/WGS-TB/MongooseGUI`, for those users who choose to use Python 2. However, there is now a new version, MONGOOSE 2.0, ported to Python 3.5, available at `https://github.com/WGS-TB/MongooseGUI3`. MONGOOSE 2.0 has new features such as a much faster analysis thanks to the Python binding for *QSOpt_ex* [11] (see subsection 2.5 below), which means that optimization problems can now be solved directly within Python instead of creating an input file, making a system call to run *QSopt_ex*, and then parsing the output file. For this reason, we use the Python 3.5 version in our running example in this section.

To install Python 3.5 on Mac OS, click on the appropriate Download button on the Python website, `https://www.python.org/downloads/`. This will download a .pkg file. Double-click on that file and follow the instructions to select the installation location (or use the default for simplicity) and any other options.

2.3 Installing additional Python modules

In order to make the most of the MONGOOSE toolbox you will need to install some additional Python modules, namely

- *setuptools* (needed to install some of the other modules listed below)
- *cython* [12] (needed for the Python interface to *QSopt_ex*)
- *python-libsbml* [13] (needed for reading metabolic models from SBML files)
- *xlrd* (needed for reading metabolic models from Excel files)
- *SIP* (needed for the GUI to MONGOOSE)
- *Qt* and *PyQt* [14] (needed for the GUI to MONGOOSE)

To install setuptools, simply open a Terminal window and run the command

```
pip3 install setuptools
```

To install *cython*, *python-libsbml*, *xlrd* and *SIP* you can proceed in exactly the same way:

```
pip3 install cython
pip3 install python-libsbml
pip3 install xlrd
pip3 install sip
```

Finally, to install *Qt*, first download an installer package from the Qt website, `https://www.qt.io/download-open-source/`, then run it by double-clicking it and following the instructions (see also Notes 1 and 2). The installation folder, for instance /Applications/Qt, will be used in the installation of PyQt; we refer to it as PATH below.

When the installation is complete, you can install *PyQt*. To do so, download its source code from the Riverbank website, `https://www.riverbankcomputing.com/software/pyqt/download` (if you are following our running example, select the OS X source). Unpack the zipped file and run the following commands from the Terminal window (where PATH is the directory in which you installed *Qt*):

```
cd PyQt-mac-gpl-4.11.4
python configure-ng.py --qmake
 ↪  PATH/5.7/clang_64/bin/qmake
make
make install
```

See Note 3 in case of any difficulties with installing *PyQt*.

2.4 Installing QSopt_ex

To install *QSopt_ex* in the most painless way possible, we highly recommend the fork of the project created by Jon Lund Steffensen, in support of the PSAMM toolbox [15].

Before you get it to work, however, you need to install several dependencies:

- *Libtool* [16] (needed to install *QSopt_ex* as a usable library)
- *GNU MP* [17] (needed to perform arbitrary precision computations)
- *libz* and *libbz2* (optional libraries to read and write compressed files)

This is actually a lot fewer dependencies relative to the original version, created by Daniel Espinoza [11]. Here is how to get them set up.

To get *Libtool*, the tool needed to install *QSopt_ex* as a library, download it from `http://mirror.jre655.com/GNU/libtool/libtool-2.4.6.tar.gz`. Unpack the zipped file and run the following commands from the Terminal window:

```
cd libtool-2.4.6
./configure --prefix /usr/local
make
make install
```

Also see Note 4 for a possible additional dependency.

To get *GNU MP*, the arbitrary precision arithmetic library, download it from `https://gmplib.org/download/gmp/gmp-6.0.0a.tar.bz2`. Unpack the zipped file and run the following commands from the Terminal window:

```
cd gmp-6.0.0
./configure
make
make check
make install
```

You are now ready to install *QSopt_ex*. You can obtain it from GitHub by opening a Terminal window and running the command

```
git clone https://github.com/jonls/qsopt-ex.git
```

Once that is done, run the following commands from the Terminal window:

```
cd qsopt-ex
./bootstrap
```

```
mkdir build && cd build
../configure --prefix /usr/local
make
make check
make install
```

See Note 5 for interpreting the results of the tests performed during installation.

2.5 Installing python-qsoptex

The Python module *python-qsoptex*, created by Jon Lund Steffensen in support of the PSAMM toolbox [15], provides an additional tool to facilitate creating, editing and solving linear programs in rational arithmetic. This saves a substantial amount of time compared to the original version of MONGOOSE, which required reading and writing files and using a system call for *QSopt_ex*.

To install *python-qsoptex*, start a Terminal window and run the commands

```
git clone https://github.com/jonls/python-qsoptex.git
cd python-qsoptex
python setup.py install
python test_qsoptex.py
```

If you get a line ending with “OK” at the end of this process, the installation was successful. If it fails during the installation process (penultimate step), you might be missing one of the dependencies and need to go back to one of the previous steps.

2.6 Installing MONGOOSE and its GUI

After all this preparation, the installation of MONGOOSE itself, together with its GUI, is quite simple. You just need to run the commands

```
git clone https://github.com/WGS-TB/MongooseGUI3.git
cd MongooseGUI3
python MONGOOSEgui.py
```

This will start the MONGOOSE GUI, which looks as shown in Figure 1 below.

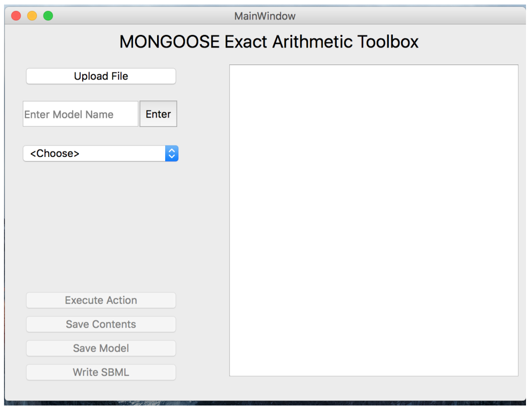


Fig. 1 The initial view of the MONGOOSE GUI

3 Methods

In this section we describe the protocol to be followed to identify the most common structural features – or bugs – that metabolic network models have, using the constraint-based framework. When the detected model behavior is in accordance with the modeler's expectations, the reconstruction process should proceed to the next stage (such as model refinement). However, if model behavior violates the modeler's expectations, especially if those expectations are based on prior biological knowledge, it might be necessary to modify the model until the behavior changes.

3.1 Parsing a metabolic network model

There are two formats supported by MONGOOSE, the mostly outdated Excel format and the newer SBML format [18]. As the vast majority of metabolic network models are published in the SBML format, we demonstrate the parsing on an SBML file, the one for *Acinetobacter baumannii* [19], abbreviated as AB1, chosen because it appears first in alphabetical order among the available reconstructions [20] and shows a number of interesting features detected by MONGOOSE. Unfortunately, since the parsing of a metabolic network model may require some flexibility and non-trivial user input, the parsing part of the MONGOOSE pipeline is not currently integrated with the GUI. However, we plan to integrate it in a future version.

MONGOOSE provides two parsing modes - the default mode is based on the original version of MONGOOSE, while the other one emulates the COBRA toolbox [21, 22]; the default is more suitable for model verification purposes, but both should give the same result on a properly designed model. The differences between the two modes are listed in Table 1 below. For more information, including details on how many of the existing models were affected by those differences, see the table at `http://groups.csail.mit.edu/cb/mongoose/`.

To parse a model in the default mode, start a Python session with the command

```
python -i ModelParsing.py
```

and then, once the input line starting with a ">>>" appears, run the command

Decision	Default mode	COBRA mode
Is the metabolite constrained?	not [external]	not [boundaryCondition or name ends in _b]
Is the reaction forward-only?	not [reversible]	[LOWER_BOUND $\geq$ 0]
Is the reaction reverse-only?	No[†]	[UPPER_BOUND $\leq$ 0]
Is the reaction the objective?	User-chosen[*]	[OBJECTIVE_COEFFICIENT == 1]

Table 1 Differences between the default mode and the COBRA mode for parsing. Legend: $*$ - among reactions with "biomass" in their lowercased name; † - except after structural analysis.

```
model = parseSBML('Acinetobacter Baumannii.xml')
```

If you want to parse the model in the COBRA mode instead, run the command

```
model = parseSBML('Acinetobacter Baumannii.xml', True)
```

The parser may generate warnings, which should be regarded as signs that something needs to be corrected in the model. Here are the main ones, both of which are encountered in parsing AB1:

```
Warning: no candidate for extra, outside found in the list
Warning: no candidate for biomass found in the list
```

The first warning means that no external compartment specification could be identified. This can be fixed easily if there is a systematic naming convention that distinguishes external metabolites from internal ones. For instance, if metabolites whose compartment is specified as 'e' are to be considered external, run the command

```
model.adjustExternal('e')
```

On the other hand, there is also a way to specify external metabolites based on their names; in the AB1 model we are using as the example, any metabolite whose name ends in ’xt’ is external, so we can run the command

```
model.adjustCompartments('xt', startPos = -2)
```

The second argument means that the position where the string we are looking for is expected to start is position -2, or 2 from the end, of the metabolite name.

The second warning means that no biomass reaction could be identified. This can also be fixed by specifying the particular reaction that should be used as the biomass reaction by searching for the reaction with the correct name or other defining properties. For instance, if the correct biomass reaction is reaction number 760, like in the AB1 model, run the command

```
model.biomassCoefficients[760] = 1
```

Another type of warning,

```
Warning: multiple candidates for biomass found in the list
```

occurs when multiple candidate reactions are identified; in this case, you will be given an explicit choice of reaction names and need to enter the number corresponding to the right one.

The internal representation that MONGOOSE uses for all parts of a metabolic network model are objects. For a reaction, this object consists of a name, a list of metabolite-coefficient pairs, a Boolean “reversible” value (determining whether it can proceed in both directions), and optional attributes. For a metabolite, this object contains its species, its compartment, a Boolean “external” value (determining whether it is subject to flux balance), and optional attributes. Multiple metabolites can share the same species, as long as they belong to different compartments. For the network as a whole, this object consists of a list of metabolites, a list of reactions, a stoichiometric matrix, and optional attributes. A lot of the optional attributes are created by the process of structurally analyzing the network, which we refer to as “reducing” it.

At the end of the parsing process, to avoid losing any of the changes you made, we recommend saving the model into a **shelve** file, by using the commands (recall that AB1 is an abbreviated name of the model organism)

```
s = shelve.open('MyModel')
s['AB1'] = model
s.close()
```

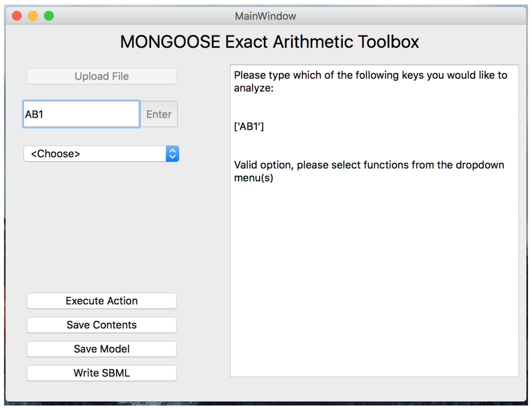
This will create a file storing a compressed representation of the model, which will be useful for future processing; in particular, this compressed representation can be directly loaded into and modified within the GUI, as we describe below.

3.2 *Inspecting, modifying and saving a model*

Once you have a shelve file containing a model, you can load it into the MONGOOSE GUI that you started by clicking on the Upload File button, then typing in the key corresponding to the model you want to load (in our example it will be AB1) and pressing the Enter button. Note that a shelve file may contain multiple models, and the full list of options will appear in the right-hand display.

Once your choice has been confirmed, the drop-down menu will become active. It allows you to inspect the following elements of the model:

- a metabolite (its species name, compartment, and reduction status if known)



Please type which of the following keys you would like to analyze:

['AB1']

Valid option, please select functions from the dropdown menu(s)

Fig. 2 Selecting the model in a file with the GUI

- a reaction (its name, metabolite-coefficient pairs, and reduction status if known)
- a reaction formula (in a readable format instead of metabolite-coefficient pairs)
- the biomass reaction (its number only)
- a reaction subset (if the network has been structurally analyzed; see section 3.7)

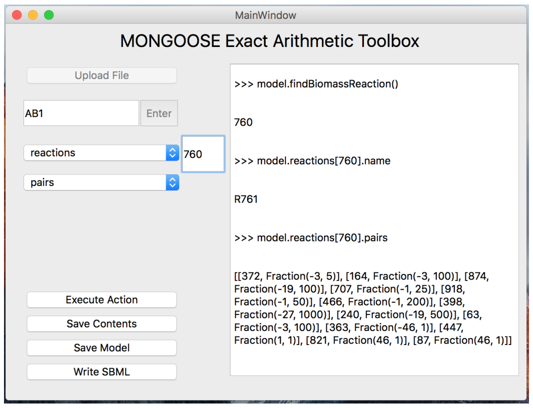
In the snapshot below, we successively:

1. determine the biomass reaction
2. display its name and metabolite-coefficient pairs
3. print out its formula
4. identify a metabolite involved in it, determine its species name and compartment

After making a choice from the drop-down menu, press the $\boxed{\text{Execute Action}}$ button to see its result on the right-hand side display.

For step 1, we simply select $\boxed{\text{findBiomassReaction}}$ from the drop-down menu and press the $\boxed{\text{Execute Action}}$ button, which displays 760 as the index of the biomass reaction.

For step 2, we select $\boxed{\text{reactions}}$ from the drop-down menu and enter the number 760 next to it, then select $\boxed{\text{name}}$ from the second drop-down menu. After we press the $\boxed{\text{Execute Action}}$ button, the name, “R761”, appears. We then repeat the same process, but select $\boxed{\text{pairs}}$ instead of $\boxed{\text{name}}$, which displays a list of pairs of the form [metabolite index, coefficient].



```
>>> model.findBiomassReaction()

760

>>> model.reactions[760].name

R761

>>> model.reactions[760].pairs

[[372, Fraction(-3, 5)], [164, Fraction(-3, 100)], [874,
Fraction(-19, 100)], [707, Fraction(-1, 25)], [918,
Fraction(-1, 50)], [466, Fraction(-1, 200)], [398,
Fraction(-27, 1000)], [240, Fraction(-19, 500)], [63,
Fraction(-3, 100)], [363, Fraction(-46, 1)], [447,
Fraction(1, 1)], [821, Fraction(46, 1)], [87, Fraction(46, 1)]]
```

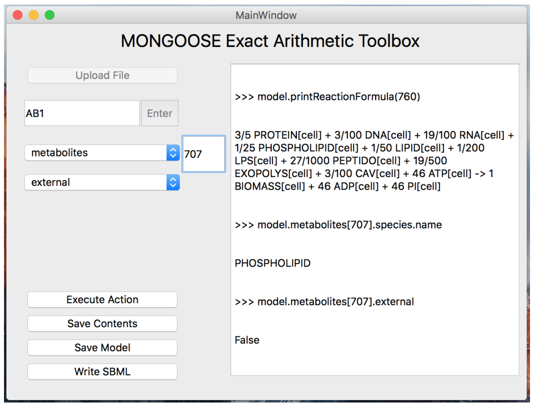
Fig. 3 The result of performing the first two queries described in the text

For step 3, we select $\boxed{\text{printReactionFormula}}$ from the drop-down menu without changing the reaction number, which produces a human-readable version of the biomass reaction.

For step 4, we select $\boxed{\text{metabolites}}$ from the drop-down menu, enter the number 707, and query its $\boxed{\text{species}}$ by $\boxed{\text{name}}$, which produces the output “PHOSPHO-LIPID”. We also check whether it is an $\boxed{\text{external}}$ metabolite, with the result - False - letting us know that it isn’t.

The GUI also allows you to modify a model, in the following ways:

- add a reaction
- delete one or more reactions
- add a metabolite
- delete one or more metabolites
- reduce the network by performing structural analysis on it [8]



AB1

metabolites

external

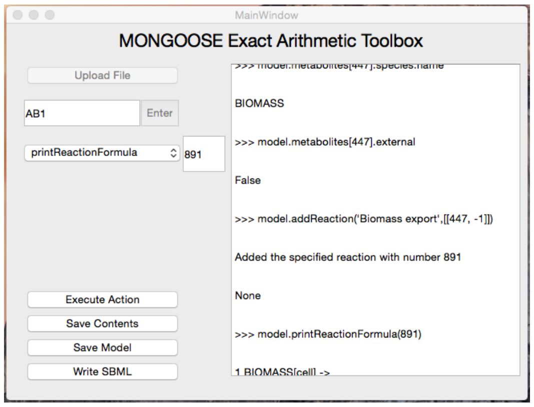
707

```
>>> model.printReactionFormula(760)

3/5 PROTEIN[cell] + 3/100 DNA[cell] + 19/100 RNA[cell] +
1/25 PHOSPHOLIPID[cell] + 1/50 LIPID[cell] + 1/200
LPS[cell] + 27/1000 PEPTIDO[cell] + 19/500
EXOPOLYS[cell] + 3/100 CAV[cell] + 46 ATP[cell] -> 1
BIOMASS[cell] + 46 ADP[cell] + 46 PI[cell]

>>> model.metabolites[707].species.name

PHOSPHOLIPID

>>> model.metabolites[707].external

False
```

Fig. 4 The result of performing the second two queries described in the text

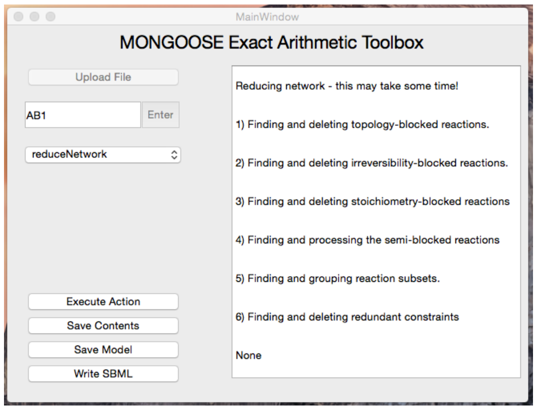
Here, we demonstrate the addition of a reaction (the deletion of reactions and the addition or deletion of metabolites work along similar lines), and discuss the network reduction in detail in the following sections.

For the model in our example, the biomass metabolite is only present in the biomass reaction, and thus will be a topologically blocked (dead-end) metabolite unless we add an export reaction for it. Just as we did for PHOSPHOLIPID in the previous example, we begin by listing its name to ensure that we have the correct one, and check that it is not external, so that the balance condition applies to it. Then we will add the reaction that exports it, giving it an appropriate name, and print out its formula to confirm that it is added correctly. Finally, we will perform a network reduction by using the $\boxed{\text{reduceNetwork}}$ command (note that this can take several minutes for large models, so please be patient and don't close the GUI while it is processing). The elements of structural analysis performed during network reduction will be explained in detail in sections 3.4 - 3.7, so we do not describe them here.



```
>>> model.metabolites[447].species.name

BIOMASS

>>> model.metabolites[447].external

False

>>> model.addReaction('Biomass export',[[447, -1]])

Added the specified reaction with number 891

None

>>> model.printReactionFormula(891)

1 BIOMASS[cell] ->
```

Fig. 5 The result of adding a biomass export reaction



```
Reducing network - this may take some time!

1) Finding and deleting topology-blocked reactions.

2) Finding and deleting irreversibility-blocked reactions.

3) Finding and deleting stoichiometry-blocked reactions

4) Finding and processing the semi-blocked reactions

5) Finding and grouping reaction subsets.

6) Finding and deleting redundant constraints

None
```

Fig. 6 The result of reducing the model

3.3 Checking elemental balance

A properly specified reaction needs to be elementally balanced, meaning that it has the same number of atoms of each type on the reactant side as on the product side [23]. For instance, the reaction $C_2H_5OH + 3O_2 \rightarrow 3H_2O + 2CO_2$ is elementally balanced.

MONGOOSE now provides two ways of checking elemental balance for the reactions in a model - an *internal* method based on the chemical formulas available for the metabolites in the model description, and an *external* method based on the consistency between different reactions. Let us explain the difference between them.

Suppose first that all - or some - of the metabolites come with a chemical formula describing their elemental composition. When the formula is available for all metabolites in a reaction, MONGOOSE checks whether the reaction is balanced with respect to every atom (this includes charges for any charged metabolites such as protons). When formulas are not available in the model description for some of the compounds (we call such compounds “unspecified”), MONGOOSE flags reactions in which one of the following occurs:

1. Only one metabolite is unspecified, but its inferred formula is not valid.
2. Only one side of the reaction contains unspecified metabolites, but has more atoms of a certain type than the other side, without the unspecified metabolites.
3. Only one side of the reaction contains unspecified metabolites, but there is less than one atom difference per unspecified metabolite between the two sides.
4. The gcd of stoichiometric coefficients of the unspecified metabolites does not divide the difference of atoms of a certain type between the two sides of a reaction.

Here, we say that a chemical formula is valid if it includes at least one atom and contains a positive integer multiplicity for every atom included in it. If any of the conditions above occurs, there cannot exist valid chemical formulas for the unspecified metabolites that make the reaction balanced. Note that MONGOOSE assumes that the molecules are described using a **chemical formula**, not an **empirical formula** which determines the ratios of each type of atom.

In addition to the internal method which relies on formulas specified in the model, MONGOOSE provides an external method which identifies groups of reactions, if any, for which a linear combination produces a “free lunch”, a reaction with an empty reactant side and a non-empty product side [24]. In other words, if the net reaction of a group of reactions, taken with positive integer multiplicities, is of this type, this shows that at least one of the reactions in the group is elementally unbalanced. MONGOOSE identifies and flags groups of reactions that can produce a free lunch, trying to make them as small and as disjoint from each other as possible.

Only the internal method is made available via the $\boxed{\text{checkElementalBalance}}$ command in the GUI at the moment; however, we plan to incorporate the external method into a future version of MONGOOSE. By default, only the internal reactions, not the exchange reactions, are tested. In the case of the AB1 model, since no formulas are available in the model description, this command will simply print the statement "No formulas found!"

3.4 Identifying blocked reactions

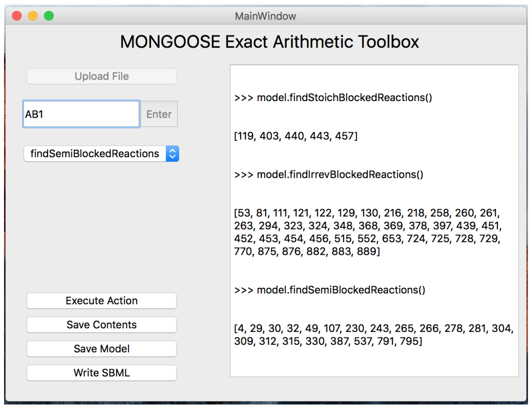
MONGOOSE is able to identify blocked reactions [25], those that are unable to carry any non-zero flux based on the model specifications, and diagnose the reason for their blockage. The box below describes the three types of blockage that MONGOOSE is able to uncover and diagnose.

MONGOOSE classifies blocked reactions in a network into three categories:

- **topology-blocked** reactions, whose blockage follows from the topology of the metabolic network (e.g. reactions containing a dead-end metabolite)
- **stoichiometry-blocked** reactions, whose blockage follows from the flux balance constraints on the internal metabolites
- **irreversibility-blocked** reactions, whose blockage follows from the addition of irreversibility constraints to the flux balance constraints

The first step of the network reduction process consists in identifying all the blocked reactions in the model, and deleting them. It also deletes all the dead-end metabolites identified in conjunction with topology-blocked reactions. This does not alter the set of possible flux distributions of the network, since the blocked reactions can only carry zero flux by definition, and the dead-end metabolites do not add any effective constraints.

In order to identify the status of a particular reaction, if it is suspected that it might be blocked, you may use the $\boxed{\text{reductionStatus}}$ option when querying the model about this reaction. However, there is also a way to access the list of all those reactions that suffer from a particular type of blockage - simply use one of the commands $\boxed{\text{findTopoBlockedReactions}}$, $\boxed{\text{findStoichBlockedReactions}}$ or $\boxed{\text{findIrrevBlockedReactions}}$ in the GUI. This will produce the list of all the reactions blocked in this particular way. The command $\boxed{\text{findSemiBlockedReactions}}$ produces the list of all semi-blocked reactions, discussed in section 3.6. In addition, you can also see the list of all metabolites that are eliminated due to topology via the $\boxed{\text{findTopoBlockedMetabolites}}$ command. See the figure below for an illustration on the AB1 model.



```
>>> model.findStoichBlockedReactions()

[119, 403, 440, 443, 457]

>>> model.findIrrevBlockedReactions()

[53, 81, 111, 121, 122, 129, 130, 216, 218, 258, 260, 261,
263, 294, 323, 324, 348, 368, 369, 378, 397, 439, 451,
452, 453, 454, 456, 515, 552, 653, 724, 725, 728, 729,
770, 875, 876, 882, 883, 889]

>>> model.findSemiBlockedReactions()

[4, 29, 30, 32, 49, 107, 230, 243, 265, 266, 278, 281, 304,
309, 312, 315, 330, 387, 537, 791, 795]
```

Fig. 7 The result of identifying stoichiometry-, irreversibility- and semi-blocked reactions

The presence of blocked reactions is in general undesirable, and this is why in section 3.5 we describe ways in which MONGOOSE can help unblock particular blocked reactions. However, the results of these automated methods cannot substitute for the expertise of someone with a detailed knowledge of the metabolism of the particular organism in question, and should be treated as suggestions.

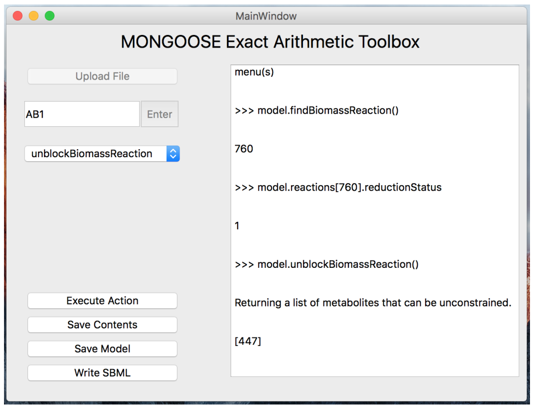
3.5 Unblocking blocked reactions

The topology-blocked reactions are typically the simplest kind of reactions to unblock. First, one needs to compute the causal chain that has led to the blockage of this reaction (this is not always a single metabolite that is not used elsewhere in the model, but could be the result of an iterative dead-end elimination process). Then, starting with the first element of the causal chain, one needs to identify the reason for the blockage. Often, though not always, this will be the a typo in the model specification, where the name of a metabolite that should be identical to another one is spelled differently somewhere in the model. Another reason for topology-blocked reactions can simply be the lack of another reaction that contains a metabolite present in the first reaction of the causal chain for this blockage.

In the case of a typo, the metabolite needs to be renamed, which can be done by identifying the reaction containing the incorrectly spelled metabolite name (the second link in the causal chain), adding a version of this reaction with the correct metabolite in place, deleting the original reaction, and deleting the incorrectly spelled metabolite. Once the changes have been made to the model, one should reduce the network once more and see if the problem has been resolved. In the case of a missing reaction, the blocked reaction can remain in the model, but a search should be initiated for a reaction that can produce (respectively, consume) the metabolite that the blocked reaction consumes (respectively, produces).

Unlike topology-blocked reactions, MONGOOSE can unblock stoichiometry- or irreversibility-blocked reactions by suggesting that certain model constraints be relaxed - flux balance constraints on a small number of metabolites in the former case, and irreversibility constraints on a small number of reactions in the latter case. Although these sets of constraints are typically quite small, there could be multiple solutions and they are not guaranteed to be the smallest possible (although they are always minimal in the sense that relaxing only some, but not all, of the constraints in them will not remove the blockage). For this reason, they should be treated as suggestions to the modeler rather than definitive changes to be made to the model [26].

The changes needed to unblock the biomass reaction can be identified using the $\boxed{\text{unblockBiomassReaction}}$ command in the GUI. In the case of the AB1 model, this command will produce the message “The biomass reaction is not blocked; nothing to do here.” However, if we run it on the parsed model before the addition of a biomass export reaction, it produces the list [447], containing the single BIOMASS metabolite, as an optional set of constraints to be relaxed (and indeed, the addition of a biomass export reaction effectively addresses the issue).



```
menu(s)

>>> model.findBiomassReaction()

760

>>> model.reactions[760].reductionStatus

1

>>> model.unblockBiomassReaction()

Returning a list of metabolites that can be unconstrained.

[447]
```

Fig. 8 The result of testing the reduction status of the biomass reaction and unblocking it

3.6 Identifying semi-blocked reactions

In addition to blocked reactions, MONGOOSE also identifies semi-blocked reactions [8], namely, reactions that are specified to be reversible, but for which only one direction (forward or reverse) is possible due to the model constraints. These are also potentially undesirable discrepancies between the model specification and its functionality, but may not be as severe as blocked reactions.

MONGOOSE identifies two types of semi-blocked reactions:

- **effectively forward** reactions, specified to be reversible but only able to carry positive (or zero) flux
- **effectively reverse** reactions, specified to be reversible but only able to carry negative (or zero) flux

Normally, any such reactions are labeled irreversible (and reversed in the case of effectively reverse reactions) by MONGOOSE before the next stage of the structural analysis. However, if the modeler strongly believes that some semi-blocked reaction should be reversible, MONGOOSE can help identify a small number of irreversibility constraints that could be relaxed in order to allow this reaction to carry flux in both directions (similar to what would be done to unblock an irreversibility-blocked reaction). Again, like in section 3.6, these results should be treated as suggestions. The current version of the MONGOOSE GUI does not allow the unblocking of a semi-blocked reaction, but the command $\boxed{\text{findSemiBlockedReactions}}$ does allow for a list of all such reactions to be displayed.

3.7 Identifying reaction subsets

A reaction subset (also called an enzyme subset in the literature [27]) is a set of reactions which are guaranteed to have proportional fluxes in any valid flux distribution. A reaction subset could correspond to the steps of a linear pathway, in which each reaction consumes a unique product of the previous reaction and produces a unique reactant of the following one. If this is the case, a reaction subset is a desired feature of the model. On the other hand, there can be reactions in a reaction subset that are unrelated to one another except implicitly, through the constraints of the model (the reactions in a reaction subset of size 2 may not even share any metabolites). In such a case, since inhibiting one model in a reaction subset inhibits all of the other ones, the behavior is undesirable and should be closely examined by the modeler.

MONGOOSE identifies two types of reaction subsets:

- **reversible** reaction subsets, in which all reactions can carry flux of either sign
- **irreversible** reaction subsets, in which all reactions can only carry positive flux

The advantage of identifying reaction subsets, from the point of view of network reduction, is that they can then be grouped into a single overall reaction without loss of information [27]. This is a helpful step in the subsequent computation of essential reactions, synthetic lethal pairs, and minimal media, discussed in sections 3.8 – 3.9.

The last stage of the network reduction process also deletes any metabolites that define redundant constraints. At the end of the process, the network contains the information about the reduction status for each of the reactions and metabolites in the model, as well as all the information required to convert a flux vector in the reduced network to one in the original network (via the `reactionSubsets` attribute, which in this case can also contain a single reaction).

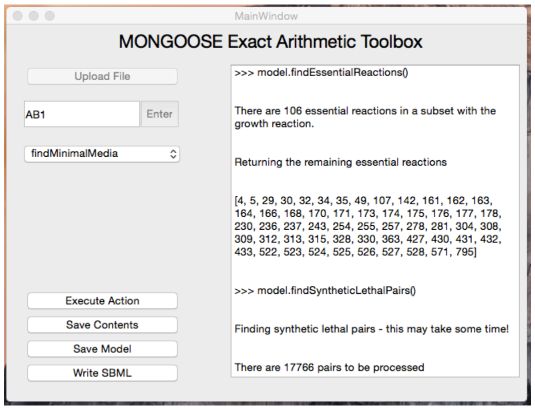
3.8 Identifying essential reactions and synthetic lethal pairs

The concept of a **minimal cut set** is a very fruitful way of analyzing metabolic network models. A cut set is a set of reactions whose inhibition (by setting their flux to 0) disables the production of a target reaction, typically the biomass reaction. A minimal cut set [28, 29] is one such that no reaction can be removed from it while preserving the property defining a cut set. Minimal cut sets correspond to inhibition strategies and can point the way to discovering new drug targets or selecting metabolic engineering interventions.

MONGOOSE identifies two types of minimal cut sets:

- **essential reactions**, single reactions whose inhibition disables biomass production
- **synthetic lethal pairs**, pairs of reactions that are not essential on their own, but whose simultaneous inhibition disables biomass production

MONGOOSE performs the search for essential and synthetic lethal reactions [6] by using the reduced network, since its size is substantially smaller. It also samples a valid flux vector for each reaction in the reduced network, in a pre-processing step in order to further speed up the computation (since only those reactions or pairs of reactions that have a possibility of being minimal cut sets then need to be tested). In the final stage, MONGOOSE expands the minimal cut sets it found back to the original (non-reduced) network. The commands `findEssentialReactions` and `findSyntheticLethalPairs` implement this functionality in the MOGNOOSE GUI.



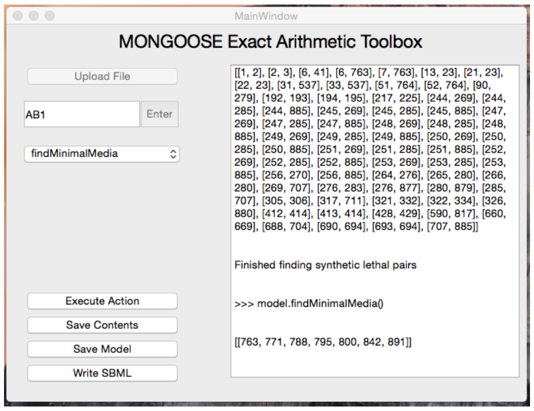
```
>>> model.findEssentialReactions()

There are 106 essential reactions in a subset with the
growth reaction.

Returning the remaining essential reactions

[4, 5, 29, 30, 32, 34, 35, 49, 107, 142, 161, 162, 163,
164, 166, 168, 170, 171, 173, 174, 175, 176, 177, 178,
230, 236, 237, 243, 254, 255, 257, 278, 281, 304, 308,
309, 312, 313, 315, 328, 330, 363, 427, 430, 431, 432,
433, 522, 523, 524, 525, 526, 527, 528, 571, 795]

>>> model.findSyntheticLethalPairs()

Finding synthetic lethal pairs - this may take some time!

There are 17766 pairs to be processed
```

Fig. 9 The result of identifying the essential reactions and synthetic lethal pairs

3.9 Identifying minimal media

A **medium** in a metabolic network model is a subset of the import reactions that enable biomass production. A minimal medium [30] is a medium from which no import reaction can be removed without disabling biomass production. The search for good minimal media is particularly informative for bacteria, where it can, for instance, lead to designing new growth media for culturing them in a laboratory setting.

MONGOOSE can identify a number of small minimal media based on the reduced network, then expand them to the original network. While these minimal media are not guaranteed to be as small as possible, they are typically quite small. The command $\boxed{\text{findMinimalMedia}}$ implements this functionality in the MOGNOOSE GUI.



```
[[1, 2], [2, 3], [6, 41], [6, 763], [7, 763], [13, 23], [21, 23],
[22, 23], [31, 537], [33, 537], [51, 764], [52, 764], [90,
279], [192, 193], [194, 195], [217, 225], [244, 269], [244,
285], [244, 885], [245, 269], [245, 285], [245, 885], [247,
269], [247, 285], [247, 885], [248, 269], [248, 285], [248,
885], [249, 269], [249, 285], [249, 885], [250, 269], [250,
285], [250, 885], [251, 269], [251, 285], [251, 885], [252,
269], [252, 285], [252, 885], [253, 269], [253, 285], [253,
885], [256, 270], [256, 885], [264, 276], [265, 280], [266,
280], [269, 707], [276, 283], [276, 877], [280, 879], [285,
707], [305, 306], [317, 711], [321, 332], [322, 334], [326,
880], [412, 414], [413, 414], [428, 429], [590, 817], [660,
669], [688, 704], [690, 694], [693, 694], [707, 885]]


Finished finding synthetic lethal pairs


>>> model.findMinimalMedia()


[[763, 771, 788, 795, 800, 842, 891]]
```

Fig. 10 The result of identifying the minimal media (includes output from the previous command)

3.10 Ending your session

At the end of a session using the MONGOOSE GUI, press the $\boxed{\text{Save Contents}}$ to save the history of the commands you ran on the model and their output, which will create a dialog box prompting you to enter a name and a location to save your file in. If you made changes to the model that you would like to save (in particular, if you used the $\boxed{\text{reduceNetwork}}$ command and would like to save time by not having to run it again in a future analysis), you can also use the $\boxed{\text{Save Model}}$ button. Please note that this will overwrite the model in your original shelve file, so make sure that you are happy with the changes you made and want to overwrite the original model. It is always a good idea to create a backup version of the original shelve file that you can revert to if you change your mind later. Finally, the $\boxed{\text{Write SBML}}$ button enables you to write out the model into a text file in SBML format [18], making it suitable for analysis with most other toolboxes for metabolic network models.

We hope that you find MONGOOSE useful, and always welcome feedback, which will help us decide on the directions to pursue in developing it further.

4 Notes

1. If you have a fast Internet connection, it will be faster to use the offline installer for *Qt* than the online one, which can be slow due to the use of a mirror server.

2. When installing *Qt*, you may get this error: "You need to install Xcode version 5.0.0. Download Xcode from `https://developer.apple.com/xcode`". If you get this error you might need to install Xcode - not version 5.0.0, but the latest version that works on your operating system - go to the suggested website, download and install it. You will also need to accept the license via the command

   ```
   sudo /usr/bin/xcodebuild
   ```

   from a Terminal window. Otherwise you may get an error when installing *PyQt*.

3. Another problem that may arise during the installation of *PyQt* is a failure to locate SIP. In that case, instead of using *pip3* to install SIP, follow `http://pyqt.sourceforge.net/Docs/sip4/installation.html`'s instructions and specify SIP's location via the `--sip` option when installing *PyQt* (by default, it is /Library/Frameworks/Python.framework/Versions/3.5/bin/sip).

4. When installing *QSopt_ex*, you may encounter a problem if you do not have the *Autoconf* [31] tool. It can be installed in essentially the same way as *Libtool*, from `http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.gz`. In this case, the first two commands in the Terminal window should be replaced by

   ```
   tar -xvf autoconf-2.69.tar.gz
   cd autoconf-2.69
   ```

5. When installing *GNU MP* and *QSopt_ex*, as you run *make check* before the final step, pay close attention to the result of the tests. If any of them fail, the software may not work correctly; however, warnings are (usually) perfectly fine.

Acknowledgements The authors would like to thank Dan Park for creating the MONGOOSE website. In addition, the authors would like to acknowledge the invaluable input of Bonnie Berger, Aviv Regev, and Jason Trigg, as well as the help of Daniel Espinoza and Dan Steffy. This work is supported by an NSERC Discovery Grant as well as an Alfred P. Sloan Fellowship.

References

1. Benson D A, Clark K, Karsch-Mizrachi I et al (2015) GenBank. Nucleic Acids Research 43:D30–D35
2. Zhang C, Hua Q (2015) Applications of Genome-Scale Metabolic Models in Biotechnology and Systems Medicine. Frontiers in Physiology 6:413
3. Long M R, Ong W K, Reed J L (2015) Computational methods in metabolic engineering for strain design. Current Opinion in Biotechnology 34:135–141
4. Varma A, Palsson B (1994) Metabolic flux balancing: basic concepts, scientific and practical use. Nature Biotechnology 12:994–998
5. Varma A, Palsson B (1994) Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type Escherichia coli W3110. Appl Environ Microbiol 60:3724–3731
6. Suthers P F, Zomorrodi A, Maranas C D (2009) Genome-scale gene/reaction essentiality and synthetic lethality analysis. Molecular Systems Biology 5:1
7. Gottstein W, Olivier B G, Bruggeman F J, Teusink B (2016) Constraint-based stoichiometric modelling from single organisms to microbial communities. J. R. Soc. Interface 13(124):20160627

8. Chindelevitch L, Trigg J, Regev A, Berger B (2014) An exact arithmetic toolbox for a consistent and reproducible structural analysis of metabolic network models. Nature Communications 5:4893
9. Boettiger C (2015) An introduction to Docker for reproducible research, with examples from the R environment. ACM SIGOPS Operating Systems Review, Special Issue on Repeatability and Sharing of Experimental Artifacts. 49(1):71–79
10. van Rossum G (1995) Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam
11. Applegate D, Cook W, Dash S, Espinoza D (2007) Exact solutions to linear programming problems. Oper. Res. Lett. 35:693-699
12. Behnel S, Bradshaw R, Citro C, Dalcin L et al (2011) Cython: The Best of Both Worlds. Computing in Science and Engineering 13:31–39
13. Bornstein B J, Keating S M, Jouraku A, Hucka M (2008) LibSBML: An API Library for SBML. Bioinformatics 24(6):880–881
14. PyQt whitepaper. Riverbank Computing. `http://www.riverbankcomputing.com/static/Docs/PyQt4/pyqt-whitepaper-a4.pdf` Accessed November 21, 2017
15. Steffensen J L, Dufault-Thompson K, Zhang Y (2016) PSAMM: A Portable System for the Analysis of Metabolic Models. PLOS Computational Biology 12(2):e1004732
16. Matzigkeit G, Oliva A, Tanner T, Vaughan G V (2015) GNU Libtool Reference Manual. Samurai Media Limited, United Kingdom
17. Granlund T and the GMP development team (2016) GNU MP: The GNU Multiple Precision Arithmetic Library. `http://gmplib.org/`. Accessed November 21, 2017
18. Hucka M, Finney A, Sauro H M, Bolouri H et al (2003) The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. Bioinformatics 19(4):524-531
19. Kim H U, Kim T Y, Lee S Y (2010) Genome-scale metabolic network analysis and drug targeting of multi-drug resistant pathogen Acinetobacter baumannii AYE. Mol. BioSyst. 6(2):339–348
20. In Silico Organisms (2016) UCSD Systems Biology group. `http://systemsbiology.ucsd.edu/InSilicoOrganisms/OtherOrganisms`. Accessed November 21, 2017
21. Hyduke D, Schellenberger J, Que R, Fleming R et al (2011) COBRA Toolbox 2.0. Protoc Exch doi:10.1038/protex.2011.234

22. Ebrahim A, Lerman JA, Palsson BO, Hyduke DR (2013) COBRApy: COnstraints-Based Reconstruction and Analysis for Python. BMC Syst Biol 7(74). doi: 10.1186/1752-0509-7-7
23. Ravikrishnan A, Raman K (2015) Critical assessment of genome-scale metabolic networks: the need for a unified standard. Brief Bioinform 16(6):1057–1068
24. Chindelevitch L A (2010) Extracting information from biological networks. Dissertation, Massachusetts Institute of Technology. `http://hdl.handle.net/1721.1/64607`
25. Schuster R, Schuster S (1991) Detecting strictly detailed balanced subnetworks in open chemical reaction networks. J. Math. Chem. 6:17-40
26. Ponce-de-León M, Montero F, Peretó J (2013) Solving gap metabolites and blocked reactions in genome-scale models: application to the metabolic network of *Blattabacterium cuenoti*. BMC Systems Biology 7(114)
27. Gagneur J, Klamt S (2004) Computation of elementary modes: a unifying framework and the new binary approach. BMC Bioinformatics 5(175)
28. Klamt S, Gilles E (2004) Minimal cut sets in biochemical reaction networks. Bioinformatics 20(2):226-234
29. Acuña V, Chierichetti F, Lacroix V, Marchetti-Spaccamela A et al (2009) Modes and cuts in metabolic networks: complexity and algorithms. BioSystems 95(1):51-60
30. Suthers P F, Dasika M S, Kumar V S, Denisov G et al (2009) A Genome-Scale Metabolic Reconstruction of Mycoplasma genitalium, iPS189. PLOS Computational Biology 5(2):e1000285
31. Calcote J (2010) Autotools: A Practioner's Guide to GNU Autoconf, Automake, and Libtool. No Starch Press, San Francisco, CA, USA