reconstruction_tutorial

August 22, 2019

1 Metabolic Network Reconstruction Tutorial

In the following tutorial you will learn how to manipulate a (genome-scale) constraint-based metabolic reconstruction with cobrapy such that you can conform with the recommendations of **Box 2** in the manuscript on community standards.

From the caption of **Box 2** itself:

> Proposed minimum standardized content for a metabolic network reconstruction. We propose that modelers use this list as a guide to help standardize accessibility, content, and quality; however, more comprehensive documentation and more interpretablee and accessible information can only improve the usability and biological relevance of the shared reconstruction.

Throughout this tutorial we will look at and improve the example reconstruction `iPfal19.xml` for *Plasmodium falciparum* 3D7 provided with the publication, as well as building a minimal example from scratch where you can easily see the generated SBML. Each section will show:

1. Python code needed for the inspection and manipulation of a metabolic reconstruction.
2. Excerpts from the resulting SBML that is generated by cobrapy when saving your model.

A Note on Terminology

> In the COBRA community, usually a distinction is made between a *reconstruction* and a *model*. A *reconstruction* represents the bare metabolic network of biochemical reactions that may occur in an organism due to the catalytic action of enzymes encoded in the genome plus some spontaneous reactions. A *model* is then a specification (or parametrization) of a reconstruction by, for example, defining certain medium conditions, i.e., restricting uptake rates; by fixing the directionality of certain reactions to fit those conditions; and by introducing specific energetic maintenance costs of the organism in those conditions. Other forms of model parametrization exist but these are some typical examples.
>
> This is the consensus within the community, however, both the SBML element to encode the metabolic network and the cobrapy class are called *model*. They do not make an explicit distinction between those two representations. Within this tutorial we will therefore loosely call everything a model.

[1]:
```
import logging
```

[2]:
```
logging.basicConfig(level="INFO")
```

[3]:
```
import cobra
from cobra.io import read_sbml_model, write_sbml_model
```

The tutorial was created with the following software versions.

[4]:
```
cobra.show_versions()
```

```
System Information
==================
OS                      Linux
OS-release 4.15.0-58-generic
Python                  3.6.8

Package Versions
================
cobra                          0.16.0
depinfo                         1.5.1
future                         0.17.1
numpy                          1.17.0
optlang                         1.4.4
pandas                         0.25.0
pip                            19.2.1
python-libsbml-experimental 5.18.0
ruamel.yaml                    0.16.1
setuptools                     41.0.1
six                            1.12.0
swiglpk                        4.65.0
wheel                          0.33.4
```

1.1 Model

Load the existing metabolic model.

[5]:
```
p_falciparum = read_sbml_model("iPfal19.xml")
```

Create an empty model that lets us easily inspect generated SBML.

[6]:
```
bare = cobra.Model("bare", name="Empty Demo Model")
```

Simply saving an empty model with an identifier and name to SBML

```
write_sbml_model(bare, "bare.xml")
```

will generate the following SBML element:

```
<model metaid="meta_bare" id="bare" name="Empty Demo Model" fbc:strict="true">
```

Please be aware that, by convention, genome-scale metabolic network models are expected to have at least two compartments: the cytosol and extracellular space. At the moment, support for compartments in cobrapy is a bit weak. You can only create them by referencing them from a metabolite or by loading an existing SBML model. Better support for compartments is in preparation.

In their simplest form, compartments are defined as follows in SBML:

```
<compartment id="c" name="cytosol" constant="true"/>
```

You can inspect existing compartments with cobrapy as follows.

[7]:
```
p_falciparum.compartments
```

[7]:
```
{'c': 'cytosol',
 'e': 'extracellular',
 'ap': 'apicoplast',
 'm': 'mitochondria',
 'v': 'vacuole'}
```

1.1.1 Recognized naming convention

The existing model `iPfal17` follows the recommended practice for model identifiers. Quoted from **Box 2**:

> recommended approach: `i + species indicator + iteration identifier`, e.g., iPfal17 for P. falciparum published in 2017

Let us inspect this from Python.

[8]:
```
p_falciparum.id
```

[8]:
```
'plata_orig_xml'
```

[9]:
```
p_falciparum.name
```

[9]:
```
''
```

We can see that the SBML model identifier does not correspond to the recommendation and that the model has no name. Let us correct this.

[10]:
```
p_falciparum.id = "iPfal17"
```

[11]:
```
p_falciparum.name = "Plasmodium falciparum 3D7"
```

1.1.2 Machine-readable reference to organism and species embedded via MIRIAM annotation

We can find good information about a sequenced species at NCBI. *Plasmodium falciparum* 3D7 is described here.

[12]:
```
p_falciparum.annotation
```

[12]:
```
{}
```

So far the existing model has not been annotated at all. We will update this information in the following steps on the both models so that both the existing model is corrected and you can easily navigate the resulting SBML.

[13]:
```
p_falciparum.annotation["taxonomy"] = "36329"
bare.annotation["taxonomy"] = "36329"
```

This ensures that the reconstruction's taxonomy is annotated. The `annotation` attribute is a Python dictionary whose key-value pairs are automatically converted to MIRIAM compatible URIs. In order for this to work correctly, please first verify the exact spelling of the registry key on identifiers.org.

```
<annotation>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://pu
    <rdf:Description rdf:about="#meta_bare">
      <bqbiol:is>
        <rdf:Bag>
          <rdf:li rdf:resource="https://identifiers.org/taxonomy/36329"/>
        </rdf:Bag>
      </bqbiol:is>
    </rdf:Description>
  </rdf:RDF>
</annotation>
```

Important elements here are the description which is about the `model` element due to the `metaid` (compare with section Section 1.1).

```
<rdf:Description rdf:about="#meta_bare">
```

The next important element is the biological qualifier

```
<bqbiol:is>
```

you can find out more about its meaning at combine and BioModels.

Finally, there is one MIRIAM compliant annotation that was generated by cobrapy.

```
<rdf:li rdf:resource="https://identifiers.org/taxonomy/36329"/>
```

1.1.3 NCBI reference genome

The genome assembly is described here.

[14]:
```
p_falciparum.annotation["insdc.gca"] = "GCA_000002765.2"
bare.annotation["insdc.gca"] = "GCA_000002765.2"
```

Which leads to the following new SBML element:

```
<rdf:li rdf:resource="https://identifiers.org/insdc.gca/GCA_000002765.2"/>
```

[15]:
```
p_falciparum.annotation["genedb"] = "Plasmodium_falciparum_3D7"
bare.annotation["genedb"] = "Plasmodium_falciparum_3D7"
```

This generates the following familiar element:

```
<rdf:li rdf:resource="https://identifiers.org/genedb/Plasmodium_falciparum_3D7"/>
```

1.1.4 Author(s) contact information embedded

Author information should be encoded as `VCards`. You can find more information in sections 6.6 and 6.7 in the SBML level 3, release 2 specification. At the time of writing this tutorial, cobrapy lacks direct support to encode author information as VCards, however, if you manually edit the SBML as per the specification, cobrapy will respect and maintain this information.

Additionally, you can write more free-form text into the model's notes field.

[16]:
```
bare.notes["Authors"] = "Tricia McMillan, Marvin, God"
```

1.2 Metabolite

A Note on Terminology

> A tangent on the term *metabolite*: In the field of metabolic modeling four different terms are often used without clear distinction. There are compounds, chemicals, metabolites, and species. For the purpose of this document, we say 'metabolite' or 'species' to mean the most representative (most common) tautomer at the given pH. Implicitly, this acknowledges that interconvertible groups of tautomers may participate in a reaction. We will use 'compound' or 'chemical' to mean an exact chemical representation only.

An excellent resource for metabolites is MetaNetX. It merges information from several source databases (among them KEGG, ChEBI, BiGG) and aims to provide consistent cross references. At the time of writing, there is information on around 700 k compounds in MetaNetX.

For this tutorial, we will look at 1-dodecanoyl-sn-glycerol 3-phosphate. On the referenced page you can see that a lot of information that we will need is directly provided for us.

When creating a metabolite identifier, we should be aware of several conventions. As noted before, every metabolite must be allocated to a compartment. Here, we reference the cytosol by its short identifier `"c"`. Since the same metabolite can appear in multiple compartments, it is common practice to add the compartment as a suffix to the identifier of the metabolite. Thus making the identifier unique within the model.

In principle, a metabolite identifier can be any string that satisfies the SBML constraints (which can be expressed by the following regular expression `[a-zA-Z_][a-zA-Z_0-9]*`). However, many modelers choose BiGG identifiers because they often resemble their common names and are thus easier to reason about quickly. We will use this convention here.

[17]:
```
metabolite = cobra.Metabolite(id="1ddecg3p_c", compartment="c")
```

Please note that the metabolite identifier starts with a digit which is against the SBML specification. Luckily for us, cobrapy uses a general `M_` prefix for all metabolites when writing SBML to prevent these cases. In most cases, this is the right default choice but you may prefer different behavior. In those cases you will have to manually adjust the default replacement functions of `cobra.io.write_sbml_model`.

[18]:
```
bare.add_metabolites([metabolite])
```

When we add the metabolite to the model it produces the following SBML:

```
<species id="M_1ddecg3p_c" compartment="c" hasOnlySubstanceUnits="false" boundaryCondition="fal
```

You can check the existance of a metabolite in a given model in two ways

[19]:
```
p_falciparum.metabolites.get_by_id("1ddecg3p_c")
```

[19]:
```
<Metabolite 1ddecg3p_c at 0x7f5e352c0240>
```

If the metabolite identifier conforms with the above regular expression, there is also a short-hand version.

[20]:
```
p_falciparum.metabolites.glc__D_c
```

[20]:
```
<Metabolite glc__D_c at 0x7f5e35219b38>
```

1.2.1 Human readable, descriptive name

The full names are used when displaying more information about a metabolite. This is very helpful when the identifier is rather hard to guess, as is the case for our example, and it will often be the only identifying piece of information that biologists can work with.

[21]:
```
metabolite.name = "1-dodecanoyl-sn-glycerol 3-phosphate"
```

We have seen above that the names within the `iPfal17` model are formatted a bit strangely. A curation task for the model would be to clean up those names, for example, by using information from MetaNetX.

1.2.2 Charge

The metabolite charge is defined in the SBML package flux-balance constraints (fbc). In most cases the charge should be an integer although the upcoming version 3 of the fbc also allows real numbers for the charge to cover certain edge cases.

[22]:
```
metabolite.charge = -2
```

1.2.3 Chemical formula

The formula is also an fbc extension attribute.

[23]:
```
metabolite.formula = "C15H29O7P"
```

1.2.4 InChI strings

The InChI is a very information rich, unique description of a compound. In cobrapy we can provide it as an annotation to the metabolite.

[24]:
```
metabolite.annotation["inchi"] = "InChI=1S/C15H31O7P/
↪c1-2-3-4-5-6-7-8-9-10-11-15(17)21-12-14(16)13-22-23(18,19)20/
↪h14,16H,2-13H2,1H3,(H2,18,19,20)/p-2/t14-/m1/s1"
```

1.2.5 At least one database identifier from a reliable resource

It might seem annoying and boring work but really: the more the merrier. There are tools that can help you automate this process! (They might also introduce subtle mistakes.) More cross references are better because:

1. There are no one-to-one mappings of identifiers between identifiers and the more you use the better determined your metabolite is.
2. Other users of your model will have data in a myriad of formats. They will thank you deeply, if the identifier namespace of their data already exists in the model.

[25]:
```
metabolite.annotation["bigg.metabolite"] = "1ddecg3p"
metabolite.annotation["chebi"] = "CHEBI:62840"
metabolite.annotation["hmdb"] = "HMDB62319"
metabolite.annotation["seed.compound"] = "cpd15325"
metabolite.annotation["metacyc.compound"] = "CPD0-2200"
```

After adding all of this information, the metabolite SBML definition looks like:

```
<species metaid="meta_M_1ddecg3p_c" id="M_1ddecg3p_c" name="1-dodecanoyl-sn-glycerol 3-pl
  <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http
      <rdf:Description rdf:about="#meta_M_1ddecg3p_c">
        <bqbiol:is>
          <rdf:Bag>
            <rdf:li rdf:resource="https://identifiers.org/inchi/InChI=1S/C15H31O7P/c1-2-3
            <rdf:li rdf:resource="https://identifiers.org/bigg.metabolite/1ddecg3p"/>
            <rdf:li rdf:resource="https://identifiers.org/chebi/CHEBI:62840"/>
            <rdf:li rdf:resource="https://identifiers.org/hmdb/HMDB62319"/>
            <rdf:li rdf:resource="https://identifiers.org/seed.compound/cpd15325"/>
            <rdf:li rdf:resource="https://identifiers.org/metacyc.compound/CPD0-2200"/>
          </rdf:Bag>
        </bqbiol:is>
```

```
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</species>
```

1.2.6 SBO

The systems biology ontology (SBO) provides terms that can help specify the role of and allow reasoning about an element within the model. For metabolites we recommend to at least use the term SBO:0000247 for 'simple chemical' but other terms like polysaccharide might be more appropriate and informative.

[26]:
```
metabolite.annotation["sbo"] = "SBO:0000247"
```

Annotating an SBO term will add the the following attribute to the `species` element.

```
sboTerm="SBO:0000247"
```

1.3 Biochemical reaction

Similarly to metabolites, MetaNetX is a great resource for biochemical reactions. Likewise, the identifiers easiest to interpret for human beings are BiGG symbols.

We will use phosphofructokinase as an example.

[27]:
```
reaction = cobra.Reaction("PFK")
```

[28]:
```
bare.add_reactions([reaction])
```

```
<reaction metaid="meta_R_PFK" id="R_PFK" reversible="false" fast="false" fbc:lowerFluxBound="c
```

As you can see, by default, the reaction identifier is prefixed with `R_`.

1.3.1 Human readable, descriptive name

[29]:
```
reaction.name = "phosphofructokinase"
```

Similarly to metabolites, you can also inspect existing reactions on models.

[30]:
```
p_falciparum.reactions.PFK
```

[30]:
```
<Reaction PFK at 0x7f5e35050ac8>
```

1.3.2 Reaction formula

We create a few metabolites just for the purpose of this example reaction.

[31]:
```
bare.add_metabolites([
    cobra.Metabolite("adp_c", compartment="c"),
    cobra.Metabolite("h_c", compartment="c"),
    cobra.Metabolite("fdp_c", compartment="c"),
    cobra.Metabolite("atp_c", compartment="c"),
    cobra.Metabolite("f6p_c", compartment="c"),
])
```

```
atp_c + f6p_c   adp_c + fdp_c + h_c
```

[32]:
```
reaction.reaction = "atp_c + f6p_c <=> adp_c + fdp_c + h_c"
```

You can see above for the existing PFK reaction that it was parametrized differently: it is irreversible (proceeding only in one direction.

A reaction formula is automatically translated into stoichiometric coefficients and flux bounds. This can be modified at any point before, during, or after creation of the reaction.

[33]:
```
reaction.metabolites
```

[33]:
```
{<Metabolite atp_c at 0x7f5e34da0748>: -1,
 <Metabolite f6p_c at 0x7f5e34da0780>: -1,
 <Metabolite adp_c at 0x7f5e34da02e8>: 1,
 <Metabolite fdp_c at 0x7f5e34da0710>: 1,
 <Metabolite h_c at 0x7f5e34da0358>: 1}
```

[34]:
```
reaction.bounds
```

[34]:
```
(-1000.0, 1000.0)
```

1.3.3 At least one database identifier from a reliable resource

Unfortunately, there is rarely a one-to-one relation between identifiers.

[35]:
```
reaction.annotation["bigg.reaction"] = "PFK"
reaction.annotation["rhea"] = ["16109", "16110", "16111", "16112"]
```

1.3.4 EC number

[36]:
```
reaction.annotation["ec-code"] = "2.7.1.11"
```

After adding all of this information, the metabolite SBML definition looks like:

```
<reaction metaid="meta_R_PFK" id="R_PFK" name="phosphofructokinase" reversible="true" fas
  <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http
      <rdf:Description rdf:about="#meta_R_PFK">
```

```
          <bqbiol:is>
            <rdf:Bag>
              <rdf:li rdf:resource="https://identifiers.org/bigg.reaction/PFK"/>
              <rdf:li rdf:resource="https://identifiers.org/rhea/16109"/>
              <rdf:li rdf:resource="https://identifiers.org/rhea/16110"/>
              <rdf:li rdf:resource="https://identifiers.org/rhea/16111"/>
              <rdf:li rdf:resource="https://identifiers.org/rhea/16112"/>
              <rdf:li rdf:resource="https://identifiers.org/ec-code/2.7.1.11"/>
            </rdf:Bag>
          </bqbiol:is>
        </rdf:Description>
      </rdf:RDF>
    </annotation>
    <listOfReactants>
      <speciesReference species="M_atp_c" stoichiometry="1" constant="true"/>
      <speciesReference species="M_f6p_c" stoichiometry="1" constant="true"/>
    </listOfReactants>
    <listOfProducts>
      <speciesReference species="M_adp_c" stoichiometry="1" constant="true"/>
      <speciesReference species="M_fdp_c" stoichiometry="1" constant="true"/>
      <speciesReference species="M_h_c" stoichiometry="1" constant="true"/>
    </listOfProducts>
  </reaction>
```

1.3.5 SBO

SBO terms for reactions are extremely useful in order to clearly distinguish a few categories of reactions without having to rely on naming conventions.

- Typical biochemical reactions should be annotated with SBO:0000176 or better yet with one of the more specific child terms.
- Transport reactions should receive SBO:0000655 or a more specific term. This obviates the need to append a `t` to a reaction identifier, as is often done for BiGG reactions such as PHEMEt.
- Exchange reactions should be annotated with SBO:0000627 rather than solely relying on an `EX_` identifier prefix.
- Demand reactions should be annotated with SBO:0000628 rather than solely relying on a `DM_` identifier prefix.
- Sink reactions should be annotated with SBO:0000632 rather than solely relying on an `SK_` identifier prefix.
- The ATP maintenance reaction should be labelled with SBO:0000630.
- All biomass reactions if any exist should be annotated with SBO:0000629.

1.4 Gene

Gene resources depend a lot on your organism. You may find information on MetaCyc, KEGG, NCBI, or more specialized databases. Many automatic reconstruction pipelines will take a genome

identifier and include genes for you in the draft reconstruction.

Genes are also defined in the fbc SBML package. The corresponding element is called `geneProduct`. We will use one of the genes encoding for PFK as an example.

[37]:
```
gene = cobra.Gene("PF3D7_1128300")
```

[38]:
```
bare.genes.append(gene)
```

This leads to the creation of the following SBML element in the `fbc:listOfGeneProducts`:

```
<fbc:geneProduct metaid="meta_G_PF3D7_1128300" fbc:id="G_PF3D7_1128300" fbc:label="G_PF3D7_1128
```

1.4.1 Name and/or identifier

[39]:
```
gene.name = "6-phosphofructokinase"
```

1.4.2 DNA sequence ID

[40]:
```
gene.annotation["refseq"] = "NC_037282.1"
gene.annotation["ncbigene"] = "810841"
```

1.4.3 Protein sequence ID

[41]:
```
gene.annotation["ncbiprotein"] = "XP_001347965.1"
```

1.4.4 Position (including chromosome, if applicable)

[42]:
```
gene.notes["Location"] = "Chromosome: 11; NC_037282.1 (1098167..1103555,␣
↪complement)"
```

The full `geneProduct` definition now looks like:

```
<fbc:geneProduct metaid="meta_G_PF3D7_1128300" fbc:id="G_PF3D7_1128300" fbc:name="6-phosp
  <notes>
    <html xmlns="http://www.w3.org/1999/xhtml">
      <p>Location: Chromosome: 11; NC_037282.1 (1098167..1103555, complement)</p>
    </html>
  </notes>
  <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http
      <rdf:Description rdf:about="#meta_G_PF3D7_1128300">
        <bqbiol:is>
          <rdf:Bag>
```

```
            <rdf:li rdf:resource="https://identifiers.org/refseq/NC_037282.1"/>
            <rdf:li rdf:resource="https://identifiers.org/ncbigene/810841"/>
            <rdf:li rdf:resource="https://identifiers.org/ncbiprotein/XP_001347965.1"/>
          </rdf:Bag>
        </bqbiol:is>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</fbc:geneProduct>
```

1.4.5 SBO

There is only one relevant term for genes SBO:0000243 but other elements such as mRNA also have terms.

1.4.6 Associated reactions (GPR)

In cobrapy, the association between gene, protein, and reaction (GPR) is set on the reaction object. This is currently set as a Boolean rule of gene identifiers. We will generate a rule here that encodes two isozymes (two independent proteins that can catalyze the reaction) in order to show a simple Boolean rule.

[43]:
```
reaction.gene_reaction_rule = "PF3D7_1128300 or PF3D7_0915400"
```

[44]:
```
gene.reactions
```

[44]:
```
frozenset({<Reaction PFK at 0x7f5e34db16a0>})
```

Adding the GPR to the PFK reaction expands its SBML definition by the following:

```
<fbc:geneProductAssociation>
  <fbc:or>
    <fbc:geneProductRef fbc:geneProduct="G_PF3D7_1128300"/>
    <fbc:geneProductRef fbc:geneProduct="G_PF3D7_0915400"/>
  </fbc:or>
</fbc:geneProductAssociation>
```

1.5 Save Model

[45]:
```
write_sbml_model(bare, "bare.xml")
```

You can now inspect the full SBML definition for our minimal model. Generating all the annotations manually required quite a bit of online research in different databases. We would like to emphasize that a good reconstruction tool will provide you with a lot of this information thus saving you a lot of tedious annotation work. However, if you ever get tired of annotating your model, please consider that you doing it once correctly for your reconstruction will provide great value to the countless researchers applying your model in other scenarios.