contrabass

Release 0.2.1-3-gc0c8415

Alex Oarga

Feb 03, 2022

Contents

1 1. Introduction

`CONTRABASS` is a tool for the detection of vulnerabilities in metabolic models. The main purpose of the tool is to compute chokepoint and essential reactions by taking into account both the topology and the dynamic information of the model. In addition to the detection of vulnerabilities, CONTRABASS can compute essential genes, compute and remove dead-end metabolites, compute different sets of growth-dependent reactions, and update the flux bounds of the reactions according to the results of Flux Variability Analysis.

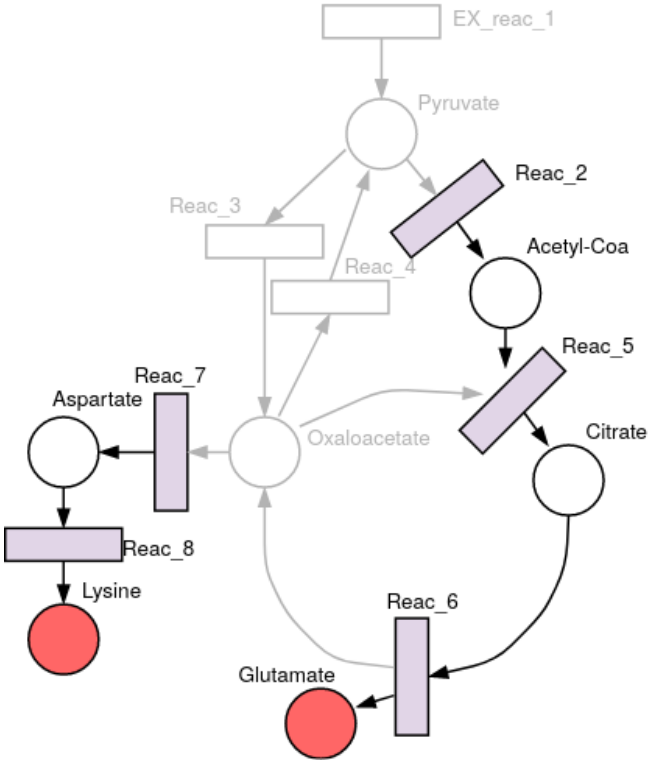
CONTRABASS takes as input the SBML file of a metabolic model, and provides as output a spreadsheet file and an html file reporting the obtained results. CONTRABASS accounts for the following sets of reactions and metabolites:

- **Chokepoint reactions:** A reaction is a chokepoint if it is the unique consumer or the only producer of a given metabolite.
- **Essential Reactions:** A reaction is essential if its deletion, or equivalently, restricting its flux to zero, causes a significant decrease in the objective function (e.g. cellular growth).
- **Dead-End Metabolites (DEM):** A metabolite is a dead-end metabolite if it is not produced or not consumed by any reaction.
- **Essential reactions for optimal growth:** A reaction is essential for optimal growth if its deletion, or equivalently, restricting its flux to zero, causes a decrease in the objective function.
- **Dead reactions:** A reaction is dead is its upper flux bound and its lower flux bound are equal to zero.
- **Blocked reactions:** A reaction is blocked if its flux is necessarily zero at any possible steady state of the model.

- **Reversible reactions:** A reaction is reversible if its upper flux bound is strictly positive and its lower flux bound is strictly negative.
- **Non-reversible reactions:** A reaction is non-reversible if it is not dead and not reversible.
- **Essential genes:** A gene is essential if the objetive function (e.g. cellular growth) is zero when it is knocked down.

1.1 1.1. Example

Chokepoint reactions and dead-end metabolites example:



Chokepont reactions

Reaction	Metabolite	Type
Reac_2	Acetyl-Coa	Unique producer
Reac_5	Acetyl-Coa	Unique consumer
Reac_5	Citrate	Unique producer
Reac_6	Citrate	Unique consumer
Reac_6	Glutamate	Unique producer
Reac_7	Aspartate	Unique producer
Reac_8	Aspartate	Unique consumer
Reac_8	Lysine	Unique producer

Dead-End Metabolites

Lysine
Glutamate

The computation of vulnerabilities can also be exploited programmatically via the Low Level API (see core) which is based on COBRApy[1].

2 2. Installation

`contrabass` can be installed with **pip** package manager:

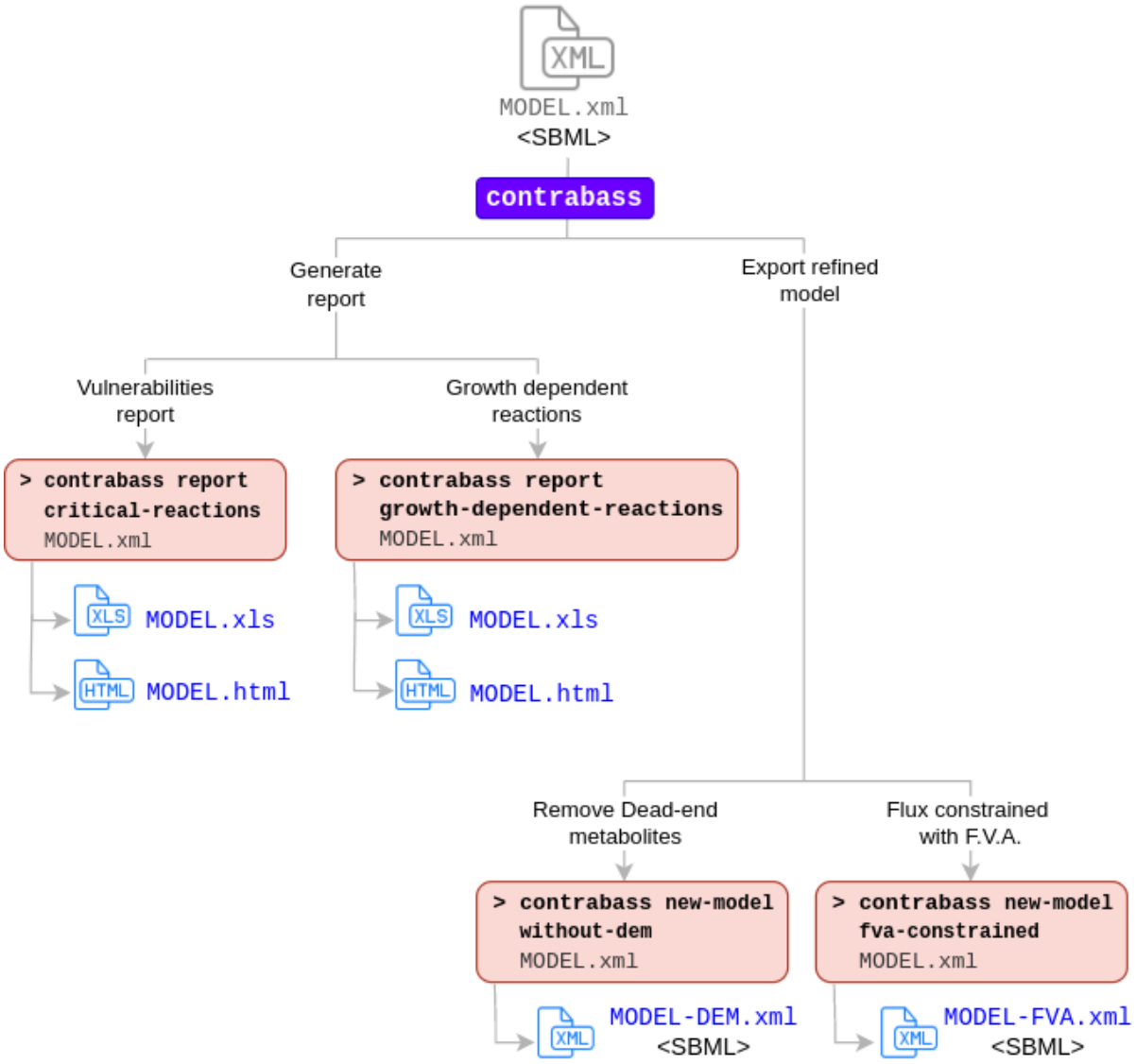
```
$ pip install contrabass
```

[1] https://github.com/opencobra/cobrapy

3 3. Tool commands

3.1 3.1 Flowchart

The next flowchart provides a graphical description of the available operations that can be performed with CONTRABASS and their respective commands, along with the output each command produces:



3.2 3.2 Commands and parameters

More information about the parameters of the tool can be obtained by executing `contrabass -h`.

```
$ contrabass -h

Usage: contrabass [OPTIONS] COMMAND [ARGS]...

  Compute vulnerabilities on constrained-based models

Options:
  -h, --help     Show this message and exit.
```

(continues on next page)

(continued from previous page)

```
  -V, --version  Show the version and exit.

Commands:
  new-model  Export refined constrained-based model.
  report     Compute vulnerabilities on constrained-based models.
```

Two options are available regarding the production of vulnerabilities reports:

```
$ contrabass report -h

Commands:
  critical-reactions
  growth-dependent-reactions
```

Three options are available when exporting a new model:

```
$ contrabass new-model

Commands:
  fva-constrained              Export a new model with its (maximum and...
  fva-constrained-without-dem  Export a new model with its (maximum and...
  without-dem                  Export a new model where Dead-End...
```

4 4. Compute vulnerabilities

Given a genome-scale model in SBML format, contrabass computes chokepoints, dead-end metabolites, essential reactions, and essential genes, and saves the results in a spreadsheet report and a html report.

4.1 4.1. Pseudocode

This section presents the pseudocode with procedures on how each vulnerability is computed:

- Find chokepoints on a model

```
function find_chokepoints(model)
    chokepoint_list = empty list
    for reaction in model
        if reaction upper flux bound not equal 0 and lower flux bound not␣
↪equal 0
            for reactant in reaction
                if reaction is the only consumer of reactant
                    chokepoint_list = chokepoint_list + (reaction, reactant)
            for product in reaction
                if reaction is the only producer of product
                    chokepoint_list = chokepoint_list + (reaction, product)
    return chokepoint_list
```

- Find essential reactions

```
function find_essential_reactions(model)
    essential_reactions = empty list
```

(continues on next page)

(continued from previous page)

```
    for reaction in model
        knock out reaction
        if flux_balance_analysis on model is 0
            essential_reactions = essential_reactions + reaction
        undo knock out
    return essential_reactions
```

- Find and remove dead-end metabolites

  See *6. Remove Dead-End Metabolites*

- Refine model with FVA

  Besides for computing the previous vulnerabilities, CONTRABASS constraints model fluxes to a certain growth (optimal growth by default) and computes again vulnerabilities for comparison. See *7. Refine model with FVA*

4.2 4.2. Command

Vulnerabilities can be computed on the model with the following command, where *MODEL.xml* is the file with the SBML model.:

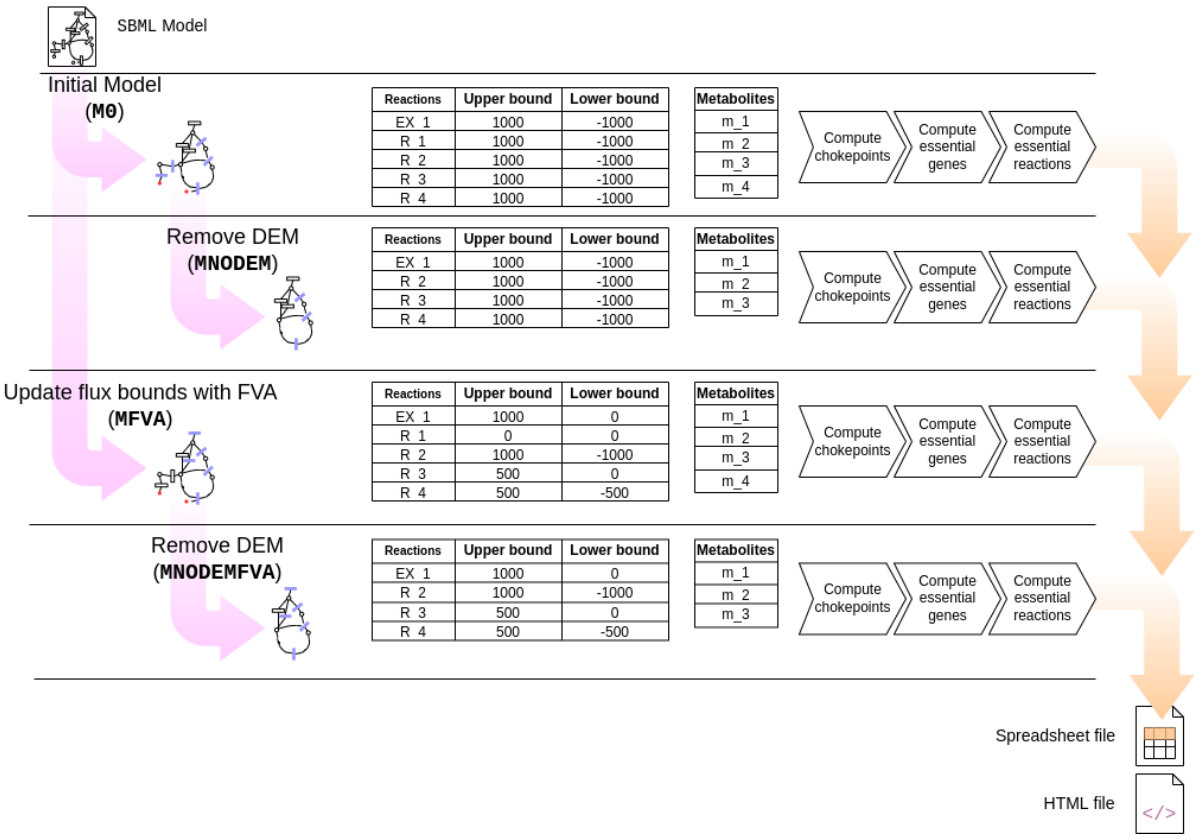
```
$ contrabass report critical-reactions MODEL.xml
```

.The prevoius command computes vulnerabilities using an optimal growth constraint. To use suboptimal growth constraints use the `fraction` to provide the fraction of optimal growht to use when constraining the model to a certain growth. See using a fraction of 95% growth:

```
$ contrabass report critical-reactions MODEL.xml --fraction 0.95
```

. The following figure shows the pipeline of the vulnerabilities computation process. For a given SBML file, computations are performed on 4 models with the following acronyms:

1. Model in the SBML file (`MO`).
2. Model without DEM (`MNODEM`).
3. Model refined with FVA, i.e. with flux bounds updated according to FVA (`MFVA`);
4. Model refined with FVA and without DEM (`MFVANODEM`).



4.2.1 4.2.1. Spreadsheet data

The previous command produces a spreadsheet file containing the following sheets:

- `model_info`: General model information.
- `reactions`: List of reactions of the model
- `metabolites`: List of metabolites of the model
- `genes`: List of genes of the model
- `reactions_FVA`: Upper and lower flux bound of each reaction obtained with Flux Variability Analysis.
- `metabolites_FVA`: Upper and lower flux bound of each reaction obtained with Flux Variability Analysis grouped by metabolite.
- `reversible_reactions`: List of reversible reactions of the model before and after FVA refinement.
- `chokepoints`: Chokepoint reactions and the metabolite/s they produce/consume. Chokepoints are computed in 4 different models:
  1. Input model in the SBML file.
  2. Model without DEM.
  3. Model refined with FVA.
  4. Model refined with FVA and without DEM.
- `dead-end`: Dead-end metabolites before and after FVA refinement.

- `essential genes`: List of essential genes of the model. Essential genes are computed in the 4 previously listed models.
- `essential reactions`: List of essential reactions of the model. Essential reactions are computed in the 4 previously listed models.
- `comparison`: Comparison of chokepoint, essential reactions and essential gene reactions in the 4 previously listed models.
- `summary`: Comparison the size of the previous sets and their intersections.

5 5. Compute growth dependent reactions

. contrabass can be used to compute, among other sets of reactions, Growth Dependent Chokepoints[2] as follows:

1. the flux of the objective function is set to a given fraction of its optimal value;
2. FVA is run to compute lower and upper flux bounds of the reactions (see *7. Refine model with FVA*);
3. the flux bounds are used to identify reversible, non reversible and dead reactions (see *1. Introduction*);
4. this directionality of reactions is used to determine consumer and producer reactions, and in turn, chokepoints. The tool produces a spreadsheet and an html report showing how the size of the set of chokepoints varies with the fraction of the optimal value set to the objective function.

```
$ contrabass report growth-dependent-reactions MODEL.xml
```

5.1 5.1. Procedure pseudocode

The pseudocode of this task is included below:

```
model = read_model()
initial_reversible_reactions     = all reactions with upper flux bound > 0 and␣
↪lower flux bound < 0
initial_dead_reactions           = all reactions with both upper and lower flux␣
↪bound equal to 0
initial_non_reversible_reactions = (model.reactions - reversible_reactions) -␣
↪dead_reactions
initial_chokepoint_reactions     = find_chokepoints(model)

for fraction in [0,0 ... 1,0]

    model = read_model()
    model = update_flux_bounds_with_fva(model, fraction)

    reversible_reactions[fraction]     = all reactions with upper flux bound > 0␣
↪and lower flux bound < 0
    dead_reactions[fraction]           = all reactions with both upper and lower␣
↪flux bound equal to 0
    non_non_reversible_reactions[fraction] = (model.reactions - reversible_
↪reactions) - dead_reactions
```

(continues on next page)

[2] https://doi.org/10.1007/978-3-030-60327-4_6

(continued from previous page)

```
    compute chokepoints on the model
    compute essential reactions on the model
```

6 6. Remove Dead-End Metabolites

6.1 6.1. Pseudocode

This sections presents the pseudocode of the procures that imply finding and removing dead-end metabolites:

- Find dead-end metabolites on a model

```
function find_dead_end_metabolites(model)
  dem_list = empty list
  for metabolite in model
      if length(metabolite.consumers) == 0 or length(metabolite.producers) == 0
          dem_list = dem_list + metabolite
  return dem_list
```

- Remove dead-end metabolites on a model

```
function remove_dead_end_metabolites(model)

    do while previous_metabolites != current_metabolites:

        previous_metabolites = model metabolites
        delete all metabolites in find_dead_end_metabolites(model)
        for reaction that produced or consumed dead-end metabolites:
            if reaction produces or consumes 0 metabolites [and is not exchange␣
↪nor demand]:
                delete reaction on model
        current_metabolites = model metabolites

    return model
```

6.2 6.2. Command

The following command exports a new generated model without Dead-End Metabolites from an input SBML model.

```
$ contrabass new-model without-dem MODEL.xml
```

7 7. Refine model with FVA

`contrabass` can generate a new model in which the flux bounds of the reactions have been updated with the values obtained in the computation of FVA . In this way the model can receive a different topology and the number of chokepoints, essential reactions or dead reactions, among others, can vary. The pseudocode of this operation is presented below:

7.1 7.1. Pseudocode

- Update model flux bounds with Flux Variability Analysis

```
function update_flux_bounds_with_fva(model, fraction_of_optmimum_growth)
    max_fva, min_fva = flux_variability_analysis(model, fraction_of_optmimum_
↪growth)
    for reaction in model
        reaction.upper_flux_bound = max_fva[reaction]
        reaction.lower_flux_bound = min_fva[reaction]
    return model
```

7.2 7.2. Command

Apart from computing vulnerabilities and generating reports, CONTRABASS can also produce a new model with refined fluxes (procedure explained above). This can be achieved with command:

```
$ contrabass new-model fva-constrained MODEL.xml
```

Alternatively a new model can be generated refined with FVA and with DEMs removed after.

```
$ contrabass new-model fva-constrained-without-dem MODEL.xml
```

.

Note that the previous command is equivalent as first constraining the model with FVA (running command `contrabass new-model fva-constrained`) and then removing DEM on the output model (running command `contrabass new-model without-dem`):

```
$ contrabass new-model fva-constrained MODEL.xml
$ mv output.xml MODEL_FVA.xml
$ contrabass new-model without-dem MODEL_FVA.xml
```

8 8. Low Level API

8.1 Try it with Binder

Recall you can also try these examples with Binder[3].

[3] https://mybinder.org/v2/gh/openCONTRABASS/CONTRABASS/HEAD?labpath=docs%2Fsource%2FCORE.ipynb

8.2 1. core

8.2.1 Import core modules

In order to use **contrabass** library you have to import the following core module.

[1]:
```
from contrabass.core import CobraMetabolicModel
```

8.2.2 Download example model

Download a genome-scale model on which vulnerabilitites will be computed.
This downloads 'Staphylococcus aureus' model and saves it to 'aureus.xml' file.

[8]:
```
import urllib.request

FILE = "aureus.xml"
MODEL_URL = 'https://raw.githubusercontent.com/openCONTRABASS/CONTRABASS/main/
 ↪docs/source/aureus.xml'

urllib.request.urlretrieve(MODEL_URL, FILE)
```

[8]:
```
('aureus.xml', <http.client.HTTPMessage at 0x7f21ef540510>)
```

8.2.3 Read metabolic model

Read input metabolic model in Systems Biology Markup Language (SBML)[4] format. SBML is an XML-based standard for systems biology model's exchange.

Allowed file formats are: * xml * json * yml

[7]:
```
model = CobraMetabolicModel("aureus.xml")
```

8.2.4 Get model info

The following methods allow us to print on the command line data from the model.

8.2.4.1 Model info

[3]:
```
model.print_model_info()
```

```
MODEL INFO
------------------------------------------------------
MODEL:  MODEL1507180070
REACTIONS:  743
METABOLITES:  655
GENES:  619
COMPARTMENTS:  c
               e
```

[4] http://sbml.org

8.2.4.2 Metabolites

[4]:
```
model.print_metabolites()
```

```
MODEL:  MODEL1507180070  - NUMBER OF METABOLITES:  655
METABOLITE  |  COMPARTMENT      |  REACTION ID
-----------------------------------------------------
10fthf_c    |  c                |  MTHFC
            |                   |  biomass_SA_7b
            |                   |  GARFT
            |                   |  biomass_SA_8a
            |                   |  FTHFL
            |                   |  biomass_SA_7a
            |                   |  AICART
12dgr_EC_c  |  c                |  biomass_SA_3a
            |                   |  biomass_SA_lipids_only
            |                   |  biomass_SA_2a
            |                   |  biomass_SA_3b
            |
```

8.2.4.3 Reactions

[5]:
```
model.print_reactions()
```

```
MODEL:  MODEL1507180070  - NUMBER OF REACTIONS:  743
REACTION ID | UPPER BOUND | LOWER BOUND | REACTION
-----------------------------------------------------
3M2OBLOXRD  |   999999.0  |  -999999.0  |  3mob_c + h_c + lpam_c <=> 2mpdhl_c +␣
 ↪co2_c
3M2OPLOXRD  |   999999.0  |  -999999.0  |  3mop_c + h_c + lpam_c <=> 2mbdhl_c +␣
 ↪co2_c
4M2OPLOXRD  |   999999.0  |  -999999.0  |  4mop_c + h_c + lpam_c <=> 3mbdhl_c +␣
 ↪co2_c
6PGALSZ     |   999999.0  |  0.0        |  h2o_c + lac6p_c --> dgal6p_c + glc_
 ↪DASH_D_c
6PHBG       |   999999.0  |  0.0        |  h2o_c + salc6p_c --> 2hymeph_c + g6p_c
ABTAr       |   999999.0  |  0.0        |  4abut_c + akg_c --> glu_DASH_L_c +␣
 ↪sucsal_c
ACACT1r     |   999999.0
```

8.2.4.4 Genes

[6]:
```
model.print_genes()
```

```
MODEL:  MODEL1507180070  - NUMBER OF GENES:  619
GENE ID     |  GENE NAME   |  REACTION ID |  GPR RELATION
-----------------------------------------------------
SA0008      |  SA0008      |  HISDr       |  SA0008
SA0009      |  SA0009      |  SERTRS      |  SA0009
SA0011      |  SA0011      |  HSERTA      |  SA0011
SA0016      |  SA0016      |  ADSS        |  SA0016
```

(continues on next page)

(continued from previous page)

```
SA0036      |  SA0036      |  GPDDA2      |  SA0036 or SA0820 or SA1542 or SA0969␣
↪or SA0220
            |              |  GPDDA5      |  SA0036 or SA0820 or SA1542 or SA0969␣
↪or SA0220
            |              |  GPDDA4      |  SA0036 or SA0820 or SA1542 or SA0969␣
↪or SA0220
            |
```

8.2.5 Dead-end metabolites

Dead-end metabolites[5] are those metabolites which are not consumed or not produced by any reaction of a given compartment of the model, including exchange reactions.

8.2.5.1 Finding Dead-end metabolites

Dead-end metabolites of the model are calculated with the method `find_dem()`. Method `dem()` returns a python dict[6] with the following content:
- **key**: string with compartment name.
- **value**: list with cobra.core.metabolites[7].

[7]:
```
model.find_dem()
model.dem()
```

[7]:
```
{'c': [<Metabolite uppg1_c at 0x7f288f6b0110>,
  <Metabolite mi1p_DASH_D_c at 0x7f288f67a110>,
  <Metabolite malm_c at 0x7f288f6741d0>,
  <Metabolite nal2a6o_c at 0x7f288f67e250>,
  <Metabolite ssaltpp_c at 0x7f288f69e390>,
  <Metabolite trnaala_c at 0x7f288f6a43d0>,
  <Metabolite trnaarg_c at 0x7f288f6a44d0>,
  <Metabolite adcobdam_c at 0x7f2858f96510>,
  <Metabolite trnaasn_c at 0x7f288f6a4590>,
  <Metabolite 2ombz_c at 0x7f2858fe2610>,
  <Metabolite trnaasp_c at 0x7f288f6a4650>,
  <Metabolite trnacys_c at 0x7f288f6a4750>,
  <Metabolite pala_SA_c at 0x7f28b43a8810>,
  <Metabolite 2pglyc_c at 0x7f2858fe28d0>,
  <Metabolite trnagly_c at 0x7f288f6a48d0>,
  <Metabolite sucr_c at 0x7f288f69e8d0
```

Dead-end metabolites can be printed on the command line with `print_dem()` method.

[8]:
```
model.print_dem()
```

```
MODEL:  MODEL1507180070  - NUMBER OF DEM:  2  - COMPARTMENT:  ALL
METABOLITE  |  COMPARTMENT      |  REACTION ID
-----------------------------------------------------
```

(continues on next page)

[5] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0075210
[6] https://docs.python.org/2/tutorial/datastructures.html#dictionaries
[7] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Metabolites

(continued from previous page)

```
uppg1_c      |  c                |  UPPDC2
mi1p_DASH_D_c  |  c                |  MI1PP
malm_c       |  c                |  PYRZAM
nal2a6o_c    |  c                |  NALN6
ssaltpp_c    |  c                |  SHCHCS2
trnaala_c    |  c                |  ALATRS
trnaarg_c    |  c                |  ARGTRS
adcobdam_c   |  c                |  ADCYRS
trnaasn_c    |  c                |  ASNTRS
2ombz_c      |  c                |  URFGTT
trnaasp_c    |  c                |  ASPTRS
trnacys_c    |  c                |  CYSTRS
pala_SA_c    |  c
```

Printed dead-end metabolites can also be limited to a specific compartment. Available compartments are given by `model.compartments()` method.

[9]:
```
model.print_dem(compartment="e")
```

```
MODEL:  MODEL1507180070  - NUMBER OF DEM:  2  - COMPARTMENT:  e
METABOLITE  |  COMPARTMENT      |  REACTION ID
-----------------------------------------------------
zn2_e       |  e                |  EX_zn2_e
            |                   |  ZNabc
spmd_e      |  e                |  EX_spmd_e
            |                   |  SPMDabc
salcn_e     |  e                |  EX_salcn_e
            |                   |  SALCpts
tre_e       |  e                |  EX_tre_e
            |                   |  TREpts
mn2_e       |  e                |  EX_mn2_e
            |                   |  MNabc
            |                   |  MNt2
ura_e       |  e                |  URAt2
            |
```

8.2.5.2 Removing dead-end metabolites

Method `remove_dem()` removes dead-end metabolites from the model. Once dead-end metabolites are deleted, some reactions might not produce a metabolite anymore. This method deletes also these reactions and loops again until no dead-end metabolite is found:

```
function remove_dead_end_metabolites(model)

    do while previous_metabolites != current_metabolites:

        previous_metabolites = model metabolites
        delete all metabolites in find_dead_end_metabolites(model)
        for reaction that produced or consumed dead-end metabolites:
            if reaction produces or consumes 0 metabolites [and is not exchange␣
↪nor demand]:
                delete reaction on model
```

(continues on next page)

(continued from previous page)

```
        current_metabolites = model metabolites

    return model
```

Reactions that are deleted with this method can be asjusted with the following params:

- **keep_all_exchange_demand_reactions** :
  - True: (default): A reaction is considered exchange or demand if it initially does not produce or consume any metabolite. This means the reaction represents an input or output of the metabolic network. If the parameter is true, none of these reactions will be removed during the execution.
  - False: If a reaction is a cobra boundary reaction[8] (calculated with heuristics) that reaction will not be deleted during the execution. Note that exchange or demand reactions that are not identified by the cobrapy framework as boundary reactions will be deleted.
- **delete_exchange** : Caution: this parameter is only intended for closed models without exchange or demand reactions, that is, models without reactions that do not produce or do not consume one metabolites on one of its sides. Note that most models do not match this property. Note also that this type of model will always yield o with Flux Balance Analysis.
  - True: all the reactions that produce or consume o metabolites are deleted whether they are exchange/demand or not.
  - False: (default) deleted according to 'keep_all_incomplete_reactions' param.

[10]:
```
print("Metabolites: ", len(model.metabolites()), "\nReactions: ", len(model.
↪reactions()))
```

```
Metabolites:  655
Reactions:  743
```

[11]:
```
model.remove_dem()
```

[12]:
```
print("Metabolites: ", len(model.metabolites()), "\nReactions: ", len(model.
↪reactions()))
```

```
Metabolites:  486
Reactions:  739
```

8.2.6 Dead reactions

Dead reactions are those reactions with upper and lower flux equal to zero.

[8] https://cobrapy.readthedocs.io/en/latest/media.html#Boundary-reactions

8.2.6.1 Finding Dead reactions

Dead reactions of the model are calculated with method `dead_reactions()`, which returns a list of cobra.core.reactions[9] representing dead reactions.

[13]:
```
model.dead_reactions()
```

[13]:
```
[<Reaction SA_biomass_1a at 0x7f288f110ad0>,
 <Reaction biomass_SA_2a at 0x7f288f06c750>,
 <Reaction biomass_SA_2b at 0x7f288f06c2d0>,
 <Reaction biomass_SA_3a at 0x7f288f06c810>,
 <Reaction biomass_SA_3b at 0x7f288f06cf90>,
 <Reaction biomass_SA_4a at 0x7f288eff60d0>,
 <Reaction biomass_SA_5a at 0x7f288eff6050>,
 <Reaction biomass_SA_6a at 0x7f288eff6fd0>,
 <Reaction biomass_SA_6b at 0x7f288f06cd50>,
 <Reaction biomass_SA_7a at 0x7f288f01de90>,
 <Reaction biomass_SA_7b at 0x7f288f01df10>,
 <Reaction biomass_SA_lipids_only at 0x7f288f06cc10>,
 <Reaction biomass_SA_nuc_only at 0x7f288f06c7d0>,
 <Reaction biomass_SA_only_AA at 0x7f288f06c690>]
```

8.2.7 Chokepoint reactions

Chokepoint reactions[10] are those reactions that are the only consumer or producer of a given metabolite that is not a dead-end metabolite.

8.2.7.1 Finding chokepoint reactions

Chokepoint reactions are calculated with method `find_chokepoints()`.
Method `chokepoints()` returns a list of tuples of types (cobra.core.reactions[11], cobra.core.metabolites[12]) representing chokepoint reactions and the metabolite of which they are the only consumer/producer.

[14]:
```
# Read initial model again
model = CobraMetabolicModel("aureus.xml")

model.find_chokepoints()
model.chokepoints()
```

[14]:
```
[(<Reaction PAPA_SA at 0x7f288ec42750>,
  <Metabolite 12dgr_SA_c at 0x7f288f683790>),
 (<Reaction PROD2 at 0x7f288f4637d0>, <Metabolite 1pyr5c_c at 0x7f2858f9da50>),
 (<Reaction DHDPRy at 0x7f288ee48a50>,
  <Metabolite 23dhdp_c at 0x7f288f67a150>),
 (<Reaction DHDPS at 0x7f288ee48a90>, <Metabolite 23dhdp_c at 0x7f288f67a150>),
 (<Reaction DHAD1 at 0x7f288ee48cd0>, <Metabolite 23dhmb_c at 0x7f288f67a350>),
 (<Reaction KARA1i at 0x7f288edae690>,
```

(continues on next page)

[9] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Reactions
[10] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2174424/
[11] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Reactions
[12] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Metabolites

(continued from previous page)

```
 <Metabolite 23dhmb_c at 0x7f288f67a350>),
(<Reaction DHAD2 at 0x7f288ee48910>, <Metabolite 23dhmp_c at 0x7f288f67a750>),
(<Reaction KARA2i at 0x7f288edae890>,
 <Metabolite 23dhmp_c at 0x7f288f67a750>),
(<Reaction DHPPDA at 0x7f288ee4bcd0>,
 <Met
```

Dead reactions can be excluded from the computation of chokepoints with parameter `exclude_dead_reactions=True`. This is strongly suggested as dead reactions are considered forward reactions by default an this can lead to misinterpretations.

[15]:
```
model.find_chokepoints(exclude_dead_reactions=True)
model.chokepoints()
```

[15]:
```
[(<Reaction PAPA_SA at 0x7f288ec42750>,
  <Metabolite 12dgr_SA_c at 0x7f288f683790>),
 (<Reaction PROD2 at 0x7f288f4637d0>, <Metabolite 1pyr5c_c at 0x7f2858f9da50>),
 (<Reaction DHDPRy at 0x7f288ee48a50>,
  <Metabolite 23dhdp_c at 0x7f288f67a150>),
 (<Reaction DHDPS at 0x7f288ee48a90>, <Metabolite 23dhdp_c at 0x7f288f67a150>),
 (<Reaction DHAD1 at 0x7f288ee48cd0>, <Metabolite 23dhmb_c at 0x7f288f67a350>),
 (<Reaction KARA1i at 0x7f288edae690>,
  <Metabolite 23dhmb_c at 0x7f288f67a350>),
 (<Reaction DHAD2 at 0x7f288ee48910>, <Metabolite 23dhmp_c at 0x7f288f67a750>),
 (<Reaction KARA2i at 0x7f288edae890>,
  <Metabolite 23dhmp_c at 0x7f288f67a750>),
 (<Reaction DHPPDA at 0x7f288ee4bcd0>,
  <Met
```

Chokepoint reactions can also be printed on the command line with `print_chokepoints`.

[16]:
```
model.print_chokepoints()
```

```
MODEL:  MODEL1507180070  - NUMBER OF CHOKEPOINTS:  448
METABOLITE ID |  METABOLITE NAME                          | REACTION ID |␣
 ↪REACTION NAME
-------------------------------------------------------------
12dgr_SA_c    |  1,2-Daicylglycerol (Saureus)             |  PAPA_SA    | ␣
 ↪Phosphatidate phosphatase
1pyr5c_c      |  1-Pyrroline-5-carboxylate                |  PROD2      | ␣
 ↪Proline dehydrogenase
23dhdp_c      |  2,3-Dihydrodipicolinate                  |  DHDPRy     | ␣
 ↪dihydrodipicolinate reductase (NADPH)
23dhdp_c      |  2,3-Dihydrodipicolinate                  |  DHDPS      | ␣
 ↪dihydrodipicolinate synthase
23dhmb_c      |  (R)-2,3-Dihydroxy-3-methylbutanoate      |  DHAD1      |
```

8.2.8 Flux Balance Analysis

Flux Balance Analysis (FBA)[13] is a mathematical procedure to estimate the fluxes of reactions in a metabolic model.

Method `get_growth()` calculates the objective value (growth rate) that maximizes the objective function. This method uses cobra.core.model.slim_optimize[14] and was obtained from cobra.flux_analysis.deletion[15].

The objective value obtained with FBA can be accessed with `objective_value()`.

[17]:
```
model.get_growth()
model.objective_value()
```

[17]:
```
0.1580502916027849
```

The objective function that is maximized during FBA can be accessed with `objective()`

[18]:
```
model.objective()
```

[18]:
```
'1.0*biomass_SA_8a - 1.0*biomass_SA_8a_reverse_4ce77'
```

This objective function can be changed by another reaction of the model with `set_objective()`. This method receives the id of the reaction that will be set as the new objective value.

[19]:
```
model.set_objective("DHAD1")
model.objective()
```

[19]:
```
'1.0*DHAD1 - 1.0*DHAD1_reverse_39dca'
```

8.2.9 Flux Variability Analysis

Flux Vatiability Analysis (FVA)[16] is a mathematical procedure used to calculate the ''minimum and maximum flux for reactions in the network while maintaining some state of the network, e.g., supporting 90% of maximal possible biomass production rate".

The method `fva()` runs Flux Variability Analysis on the model. This method runs cobra.flux_analysis.variability[17] and as so, it allows the same parameters (see the previous link):

- **loopless**: (default False) return only loopless solutions.

- **threshold**: (float default None. In cobrapy 'fraction_of_optimum'): Requires that the objective value is at least the fraction times maximum objective value.

- **pfba_factor**: (float default None) the total sum of absolute fluxes must not be larger than this value times the smallest possible sum of absolute fluxes.

An extra parameter:

- **verbose**: (default False) if True prints on the command line the result of FVA while running the analysis.

[13] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3108565/

[14] https://cobrapy.readthedocs.io/en/latest/autoapi/cobra/core/model/index.html?highlight=slim#cobra.core.model.Model.slim_optimize

[15] https://cobrapy.readthedocs.io/en/latest/_modules/cobra/flux_analysis/deletion.html#_get_growth

[16] https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-489

[17] https://cobrapy.readthedocs.io/en/latest/autoapi/cobra/flux_analysis/variability/index.html?highlight=flux_varia#cobra.flux_analysis.variability.flux_variability_analysis

Method `fva()` returns an error list if there was an error while running FVA or an empty list `[]` otherwise.

[20]:
```
model.fva(threshold=0.95)
```

[20]:
```
[]
```

[21]:
```
model.fva(threshold=0.95, verbose=True)
```

```
FLUX VARIABILITY ANALYSIS:  MODEL1507180070
REACTION:  3-Methyl-2-oxobutanoate:lipoamide oxidoreductase(decarboxylating and␣
 ↪acceptor-2-methylpropanoylating)
    fva ranges:   [ 0.0         ,  0.0         ]
REACTION:  3-Methyl-2-oxopentanoate:lipoamide oxidoreductase(decarboxylating and␣
 ↪acceptor-2-methylpropanoylating)
    fva ranges:   [ 0.0         ,  0.0         ]
REACTION:  4-Methyl-2-oxopentanoate:lipoamide oxidoreductase(decarboxylating and␣
 ↪acceptor-2-methylpropanoylating)
    fva ranges:   [ 0.0         ,  0.0         ]
REACTION:  6-phospho-beta-galactosidase
    fva ranges:   [ 0.0         ,  0.0         ]
REACTION:  6-phospho-beta-glucosidase
    fva ranges:   [ 0.0         ,  0.0         ]
RE
```

The result obtained with FVA can be accessed with `get_fva()`. This method returns a list of tuples: (cobra.core.reaction[18], [float] maximum flux, [float] minimum flux)

[22]:
```
model.get_fva()
```

[22]:
```
[(<Reaction 3M2OBLOXRD at 0x7f288eeab950>, 0.0, 0.0),
 (<Reaction 3M2OPLOXRD at 0x7f288eeab990>, 0.0, 0.0),
 (<Reaction 4M2OPLOXRD at 0x7f288eeabc10>, 0.0, 0.0),
 (<Reaction 6PGALSZ at 0x7f288eeabad0>, 0.0, 0.0),
 (<Reaction 6PHBG at 0x7f288eeabb10>, 0.0, 0.0),
 (<Reaction ABTAr at 0x7f288eead890>, 0.0, 0.0),
 (<Reaction ACACT1r at 0x7f288eeadd50>,
  0.00032511320071648854,
  -2065.6249999999704),
 (<Reaction ACACT2r at 0x7f288eeadcd0>, 0.0, -2065.624999999973),
 (<Reaction ACACT3r at 0x7f288eead710>, 0.0, -2065.624999999973),
 (<Reaction ACACT4r at 0x7f288eead150>, 0.0, -2065.624999999973),
 (<Reaction ACACT5r at 0x7f288eeadd10>, 0.0, -2065.624999999973),
 (<Reaction ACACT6r at 0x7f288eead8
```

[18] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Reactions

8.2.9.1 Updating flux of reactions with FVA

contrabass adds an extra parameter to this method which is `update_flux`. If `True` the method `fva()` updates the fluxes of reactions with the maximum and minimum flux values obtained with FVA (see the example below).

[23]:
```
# Print initial reactions of the model with initial flux values.
model.print_reactions()
```

```
MODEL:  MODEL1507180070  - NUMBER OF REACTIONS:  743
REACTION ID | UPPER BOUND | LOWER BOUND | REACTION
-----------------------------------------------------
3M2OBLOXRD  |   999999.0  |  -999999.0  |  3mob_c + h_c + lpam_c <=> 2mpdhl_c +␣
 ↪co2_c
3M2OPLOXRD  |   999999.0  |  -999999.0  |  3mop_c + h_c + lpam_c <=> 2mbdhl_c +␣
 ↪co2_c
4M2OPLOXRD  |   999999.0  |  -999999.0  |  4mop_c + h_c + lpam_c <=> 3mbdhl_c +␣
 ↪co2_c
6PGALSZ     |   999999.0  |  0.0        |  h2o_c + lac6p_c --> dgal6p_c + glc_
 ↪DASH_D_c
6PHBG       |   999999.0  |  0.0        |  h2o_c + salc6p_c --> 2hymeph_c + g6p_c
ABTAr       |   999999.0  |  0.0        |  4abut_c + akg_c --> glu_DASH_L_c +␣
 ↪sucsal_c
ACACT1r     |   999999.0
```

[24]:
```
# Update reactions flux values with FVA
model.fva(update_flux=True)
```

[24]:
```
[]
```

[25]:
```
# Print reactions of the model with constrained reactions flux values.
model.print_reactions()
```

```
MODEL:  MODEL1507180070  - NUMBER OF REACTIONS:  743
REACTION ID | UPPER BOUND | LOWER BOUND | REACTION
-----------------------------------------------------
3M2OBLOXRD  |   0.0       |  0.0        |  3mob_c + h_c + lpam_c --> 2mpdhl_c +␣
 ↪co2_c
3M2OPLOXRD  |   0.0       |  0.0        |  3mop_c + h_c + lpam_c --> 2mbdhl_c +␣
 ↪co2_c
4M2OPLOXRD  |   0.0       |  0.0        |  4mop_c + h_c + lpam_c --> 3mbdhl_c +␣
 ↪co2_c
6PGALSZ     |   0.0       |  0.0        |  h2o_c + lac6p_c --> dgal6p_c + glc_
 ↪DASH_D_c
6PHBG       |   0.0       |  0.0        |  h2o_c + salc6p_c --> 2hymeph_c + g6p_c
ABTAr       |   0.0       |  0.0        |  4abut_c + akg_c --> glu_DASH_L_c +␣
 ↪sucsal_c
ACACT1r     |   -3.26642
```

8.2.10 Essential genes

Essential genes[19] are those genes that cause a zero growth rate when knocked out.

8.2.10.1 Finding essential genes

Essential genes are calculated with the method `find_essential_genes_1()`. This method returns an error list if there was an error during the computing or an empty list `[]` otherwise.

[26]:
```
model.find_essential_genes_1()
```

[26]:
```
[]
```

If essential genes were computed with the method above, they can be accessed with `essential_genes()`. This method return a list of cobra.core.gene[20] with essential genes.

[27]:
```
model.essential_genes()
```

[27]:
```
{<Gene SA0183 at 0x7f2858fe2b10>,
 <Gene SA0512 at 0x7f288eff6310>,
 <Gene SA0728 at 0x7f288eea6890>,
 <Gene SA0729 at 0x7f288eea6950>,
 <Gene SA0823 at 0x7f288ee87d10>,
 <Gene SA0910 at 0x7f288ee89890>,
 <Gene SA0911 at 0x7f288ee89950>,
 <Gene SA0912 at 0x7f288ee89a10>,
 <Gene SA0913 at 0x7f288ee89ad0>,
 <Gene SA0937 at 0x7f288eea5650>,
 <Gene SA0938 at 0x7f288eea5710>,
 <Gene SA0994 at 0x7f288ee8d350>,
 <Gene SA0995 at 0x7f288ee8d410>,
 <Gene SA0996 at 0x7f288ee8d4d0>,
 <Gene SA1088 at 0x7f288ee8b4d0>,
 <Gene SA1089 at 0x7f288ee8b590>,
 <Gene SA1244 at 0x7f288ee91ad0>,
 <Gene SA1245 at 0x7f288ee91b90>,
 <Gene SA1255 at 0x7f288ee91dd0>,
 <Gene SA1521 at 0x7f288ee93410>,
 <Gene SA1585 at 0x7
```

8.2.11 Essential reactions

Essential reactions are those genes that cause a zero growth rate when knocked out.

[19] https://link.springer.com/chapter/10.1007/978-3-642-36546-1_43
[20] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Genes

8.2.11.1 Finding essential reactions

Essential reactions are calculated with the method `compute_essential_reactions()`. This method returns an error list if there was an error during the computing, or an empty list `[]` otherwise.

[31]:
```
model.compute_essential_reactions()
```

If essential reactions were computed with the method above, they can be accessed with `essential_reactions()`. This method return a `dict` with keys cobra.core.reaction[21] with all the reactions of the model, and values `float` with the result of computing FBA with the reaction knocked-out.

[32]:
```
model.essential_reactions()
```

[32]:
```
[<Reaction CO2t at 0x7f288ee40750>,
 <Reaction KARA1i at 0x7f288edae690>,
 <Reaction DHAD1 at 0x7f288ee48cd0>,
 <Reaction PDHcr at 0x7f288ec4cc10>,
 <Reaction EX_pro_DASH_L_e at 0x7f288edf5350>,
 <Reaction SUCOAS at 0x7f288eead690>,
 <Reaction PFK at 0x7f288ec4cf50>,
 <Reaction FBA at 0x7f288edfb3d0>,
 <Reaction AKGDa at 0x7f288ee34c90>,
 <Reaction GLCpts at 0x7f288ee25590>,
 <Reaction PGI at 0x7f288f537490>,
 <Reaction FUM at 0x7f288ee0a650>,
 <Reaction EX_co2_e at 0x7f288ee5f850>,
 <Reaction PROD2 at 0x7f288f4637d0>,
 <Reaction EX_glc_DASH_D_e at 0x7f288ee67550>,
 <Reaction ACLS at 0x7f288ee84d50>,
 <Reaction AKGDb at 0x7f288ee34d50>,
 <Reaction EX_o2_e at 0x7f288ee6dcd0>,
 <Reaction TPI a
```

8.2.12 Essential genes reactions

Essential genes reactions are those reactions that are knocked-out when an essential gene is knocked-out.

8.2.12.1 Finding essential genes reactions

Essential genes reactions are calculated with the method `find_essential_genes_reactions()`. This method returns an error list if there was an error during the computing or an empty list `[]` otherwise.

[33]:
```
model.find_essential_genes_reactions()
```

[33]:
```
[]
```

If essential genes reactions were computed with the method above, they can be accessed with `essential_genes_reactions()`. This method returns a `dict` with keys cobra.core.reaction[22] with all the reactions of the model, and values a `list` of cobra.core.genes[23] with the essential genes that cause the knock-out of the reaction.

[21] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Reactions
[22] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Reactions
[23] https://cobrapy.readthedocs.io/en/latest/getting_started.html#Genes

[34]:
```
model.essential_genes_reactions()
```

[34]:
```
{<Reaction PGK at 0x7f288f53c950>: [<Gene SA0728 at 0x7f288eea6890>],
 <Reaction CYTBD at 0x7f288ee42e90>: [<Gene SA0910 at 0x7f288ee89890>,
  <Gene SA0911 at 0x7f288ee89950>,
  <Gene SA0912 at 0x7f288ee89a10>,
  <Gene SA0913 at 0x7f288ee89ad0>,
  <Gene SA0937 at 0x7f288eea5650>,
  <Gene SA0938 at 0x7f288eea5710>],
 <Reaction PROD2 at 0x7f288f4637d0>: [<Gene SA1585 at 0x7f288ee97110>],
 <Reaction TPI at 0x7f288eb48190>: [<Gene SA0729 at 0x7f288eea6950>],
 <Reaction DHAD2 at 0x7f288ee48910>: [<Gene SA1858 at 0x7f288ee83a10>],
 <Reaction DHAD1 at 0x7f288ee48cd0>: [<Gene SA1858 at 0x7f288ee83a10>],
 <Reaction AKGDb at 0x7f288ee34d50>: [<Gene SA1244 at 0x7f288ee91ad0>],
 <Reaction ILETA at 0x7f2
```

8.3 2. core.Facade core.FacadeUtils

Modules `FacadeUtils` and `Facade` from `contrabass` provide specific methods for the generation of reports from the computation of critical reactions taking into account flux constraints.

8.3.1 Import FacadeUtils modules

Import the module

[35]:
```
from contrabass.core import Facade
from contrabass.core import FacadeUtils
```

8.3.2 Generate vulnerabilities summary spreadsheet

The method `run_summary_model` generates a spreadsheet file from a model file with the following data:

- List of metabolites of the model.
- List of reactions of the model.
- List of genes of the model.
- Upper and lower flux bound of each reaction obtained with Flux Variability Analysis.
- Upper and lower flux bound of each reaction obtained with Flux Variability Analysis grouped by metabolite.
- List of reversible reactions of the model before and after Flux Variability Analysis refinement (see *refinement*).
- List of Dead End Metabolites before and after FVA refinement.
- For the following models (each one listed along with its acronym):
    * (**Mo**) Initial model.
    * (**MDEM**) Model without DEM.
    * (**MFVA**) Model refined with FVA.
    * (**MFVADEM**) Model refined with FVA and with DEM removed.
- The following set of assets is computed for each and included in one sheet:
    * List of chokepoint reactions with the metabolite they produce/consume.
    * List of essential genes of the models.
    * List of essential reactions of the models.

    * Comparison of chokepoint reactions, essential reactions and essential gene reactions.
- Summary comparing the size of the previous sets and their intersections.

Method declaration:

`run_summary_model(self, model_path, print_f, arg1, arg2, objective=None, fraction=1.0)`

Parameters:

- `model_path`: Path of the SBML metabolic model file.
- `print_f`: Callback function to inform of the computation progression. The function must have a declaration like the following:
- `custom_function_name(message, arg1, arg2)`
- `arg1`: First parameter to pass to the callback function if any.
- `arg2`: Second parameter to pass to the callback function if any.
- `objective`: Reactions id to be used as objective function.
- `fraction`: Fraction of optmimum to be used with FVA.

[36]:
```
def callback_print_ignore(message, arg1, arg2):
    pass

def callback_print_logger(message, arg1, arg2):
    logger = arg1
    logger.info(message)

def callback_print(message, arg1, arg2):
    print(arg1 + message)

facadeUtils = FacadeUtils()
spreadsheet = facadeUtils.run_summary_model("aureus.xml", callback_print, "LOG:",␣
↪None, fraction=0.95)
facadeUtils.save_spreadsheet("output.xls", spreadsheet)
```

```
LOG:Reading model...
LOG:Generating models...
LOG:Searching Dead End Metabolites (D.E.M.)...
LOG:Searching chokepoint reactions...
LOG:Searching essential reactions...
LOG:Searching essential genes...
LOG:Searching essential genes reactions...
LOG:Removing Dead End Metabolites (D.E.M.)...
LOG:Searching essential reactions...
LOG:Searching new chokepoint reactions...
LOG:Searching essential genes...
LOG:Searching essential genes reactions...
LOG:Running Flux Variability Analysis...
LOG:Searching Dead End Metabolites (D.E.M.)...
LOG:Searching new chokepoint reactions...
LOG:Searching essential genes...
LOG:Searching essential genes reactions...
LOG:Searching essential reactions...
LOG:Removing
```

8.3.3 Generate growth-dependent reactions

The method `generate_growth_dependent_report` of `Facade` computes and returns growth dependent reactions.

The output can be saved in a spreadsheet file or html report. The resulting files include a report of the effect that varying values of optimal growth with FVA has on the folloing sets: - Reversible Reactions (RR) - Non-reversible Reactions (NR) - Dead Reactions (DR) - Chokepoint reactions (CP)

Methods declaration:

`generate_growth_dependent_report(self, config)`

Parameters:

- `config`: `contrabass.core.utils.GrowthDependentCPConfig`

class `contrabass.core.utils.GrowthDependentCPConfig`

- `model_path`: Path of the SBML metabolic model file.
- `print_f`: Callback function to inform of the computation progression. The function must have a declaration like the following: `custom_function_name(message, arg1, arg2)`
- `arg1`: First parameter to pass to the function if any.
- `arg2`: Second parameter to pass to the function if any.
- `objective`: Reactions id to be used as objective function.
- `fraction`: Fraction of optmimum to be used with FVA.
- `output_path_spreadsheet`: Output path to save spreadsheet report. If None no file is generated.
- `output_path_html`: Output path to save html report. If None no file is generated.

[37]:
```
# import config class
from contrabass.core.utils.GrowthDependentCPConfig import GrowthDependentCPConfig


def callback_print(message, arg1, arg2):
    print(message)

config = GrowthDependentCPConfig()
config.print_f                 = callback_print
config.model_path              = "aureus.xml"
config.output_path_spreadsheet = "output.xls"
config.output_path_html        = "report.html"

facade = Facade()
facade.generate_growth_dependent_report(config)
```

```
Reading model...
Computing essential reactions...
Computing optimal growth essential reactions
Computing growth dependent essential reactions
Running Flux Variability Analysis with fraction: 0.0
```

(continues on next page)

(continued from previous page)

```
Running Flux Variability Analysis with fraction: 0.1
Running Flux Variability Analysis with fraction: 0.2
Running Flux Variability Analysis with fraction: 0.3
Running Flux Variability Analysis with fraction: 0.4
Running Flux Variability Analysis with fraction: 0.5
Running Flux Variability Analysis with fraction: 0.6
Running Flux Variability Analysis with fraction: 0.7
Running Flux Variability Analysis with fraction: 0.8
Running Flux Variability Analysis with fraction: 0.9
Running Flux Variability Ana
```

[ ]:

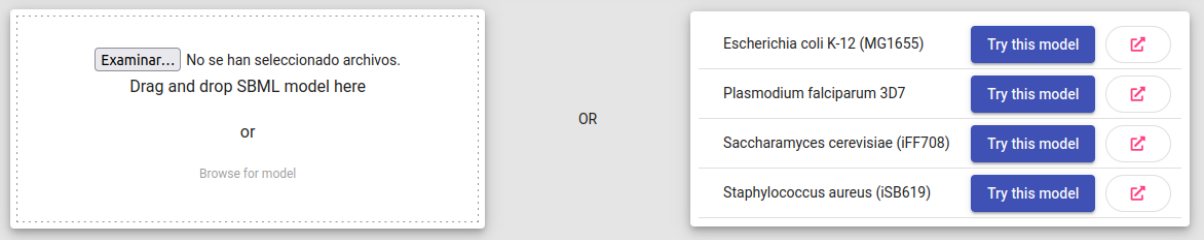
9 9. Online Web App

CONTRABASS can also be executed though the online web application at CONTRABASS-Online[24].

9.1 9.1. Usage

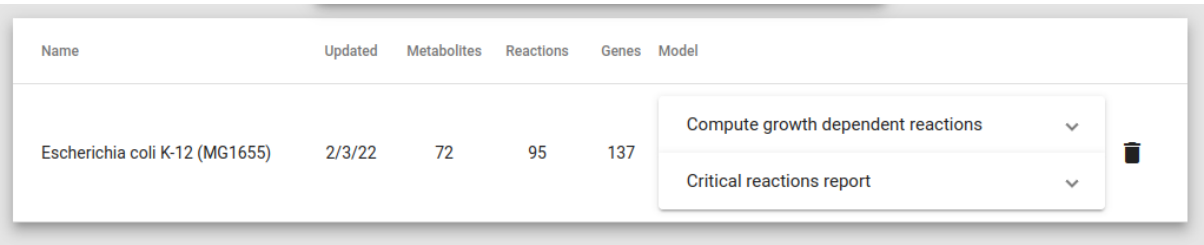
9.1.1 9.1.1. Submit model

To load a model in the web app drag-and-drop a valid SBML model to the box on the left (see image below). CONTRABASS also comes with 4 sample models to try out the tool. To use one of the 4 sample models, simply click on one of the `Try this model` buttons in the box on the right (see image below).



9.1.2 9.1.2. Run analysis

If the model is correctly formatted, CONTRABASS will read the model and notify when it is ready. When the read completes successfully, a row similar to the following one with the model data will appear on the page:

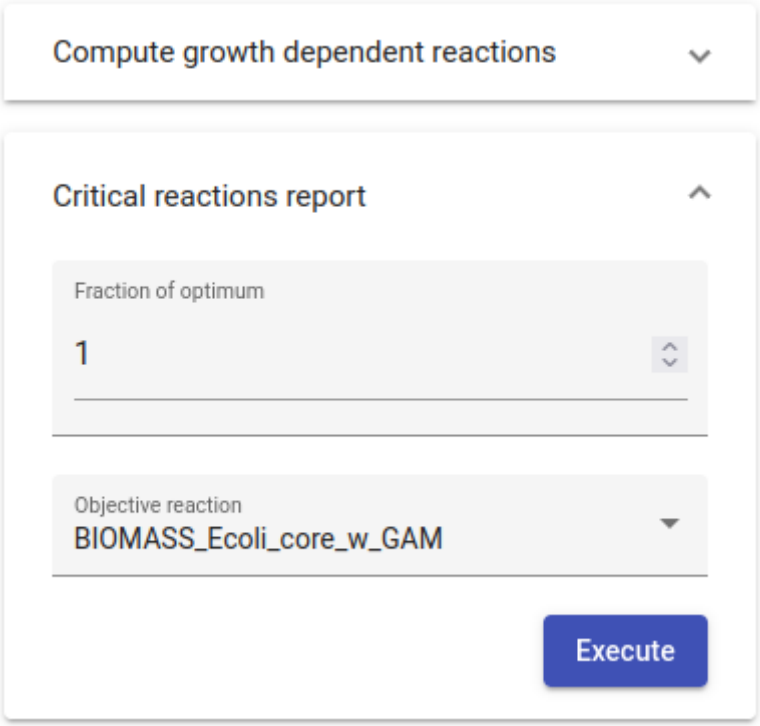


[24] http://contrabass.unizar.es

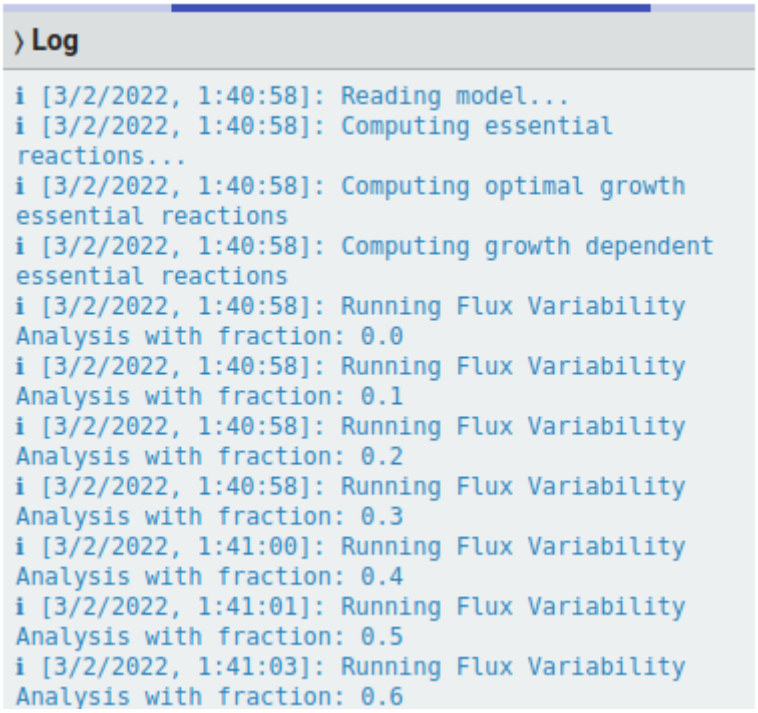
. CONTRABASS allows you to execute the two types of analysis explained above in the documentation, that is:

- **Compute growth dependent reactions**: produce an overview of the critical reactions in the model and explore how they vary with the growth constraints (See *5. Compute growth dependent reactions*).
- **Compute critical reactions**: compute sets of critical reactions and their intersections for a given growth rate (See *4. Compute vulnerabilities*).

If you click on one of the 2 accordions showing the 2 possible analysis, they will expand showing the optional parameters that you can adjust before starting the execution (see image below). Once the parameters are adjusted, to start the execution simply click the `Execute` button.



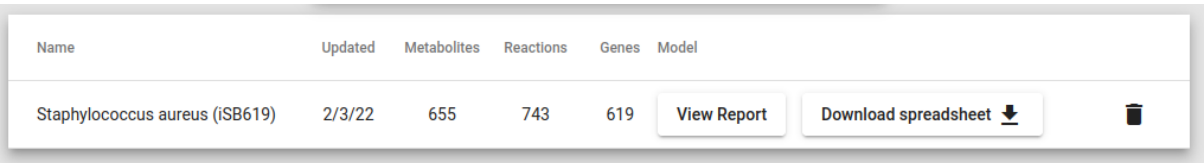
While the selected analysis is being executed, the page will show feedback on the computations performed:



9.1.3 9.1.3. Download reports

Once the selected analysis ends its execution, the web will display 2 buttons (see image below) to download different reports:

- **Download spreadsheet**: Download the spreadsheet report of the computation.
- **View report**: View and online interactive report with the result of the computation.



10 10. License

CONTRABASS is released under GPLv3[25].

[25] https://github.com/openCONTRABASS/CONTRABASS/blob/master/LICENSE