Pipeline for the identification of metabolic engineering strategies and the suggestion of Biobrick parts

# **Description**

This pipeline is a Python package that allow users to find strain design strategies in a genome scale metabolic model of interest. The pipeline suggests engineering strategies to make a chassis organism able to grow on atypical carbon sources (e.g. pollutants such as methane) and to use them as the substrates for the production of targets (e.g. high-value compounds). It is therefore the goal of the pipeline to combine stains optimizations aiming at bioremediation with engineering strategies that contribute to the idea of a biobased economy.

# **Target users**

The package has been developed on the basis of [CobraPy package](https://github.com/opencobra/cobrapy/tree/devel/src/cobra), thus the user should be familiar with the principles of constraint-based analysis. Given the absence of a graphic user interface, the user should ideally have some basic knowledge of coding in Python. This facilitate flexibility in the usage of the functions of the pipeline from Jupyter notebooks as demonstrated in the [tutorials](link)

# **Installation**

The [installation of Anaconda](https://docs.anaconda.com/anaconda/install/) is suggested to facilitate the usage of the pipeline. This application provides an easy-to-use graphic user interface to create new working environments specific for the usage of a software. Python is automatically installed with Anaconda together with most of the libraries needed for the usage of the pipeline.

In order to use the appropriate version o python it is advice to create a new environment which can easily be don within the Anaconda Navigator or by typing the following command via the Command Prompt:

conda create –name <insert name of the new environment> python=3.6.10

## **Requirements – Python libraries**

Within the environment look for the following required packages:

* COBRApy [1] version 0.18.1
* The solver used throughout the whole pipeline is GLPK [2], the one selected default by COBRApy, except for Optknock [3] analysis, which requires Gurobi solver [4] version 9.0.3. this can be freely installed and an academic licence can be requested from [Gurobi’s website](https://www.gurobi.com/gurobi-and-anaconda-for-windows/) after signing up for free.
* The scripts for querying the wikibase of Biobricks uses wikidataintegrator package wikidateintegrator [5]

Other required packages are:

* Codonharmonizer [9] v1.0.1
* cplex v12.10.0.3
* equilibrator-api v0.3.1
* equilibrator-cache v0.3.1
* equilibrator-pathway [6] v0.3.1
* pandas v1.0.4
* numpy v1.18.5
* tqdm v4.48.2
* matplotlib v3.3.0
* sbtab v0.9.73
* wget v3.2

Some additional packages might be required depending on your operating system, such as the following:

* shexer
* bio

These are imported by the scripts of the pipeline, therefore they should be installed. It can be done by looking them up in the Anaconda Navigator within the newly created environment or by using pip install from the command line:

conda activate <environment name>

pip install <package name>

## **Jupyter notebook**

Jupyter notebook can be downloaded from Anaconda too. From the home of the Anaconda Navigator it is possible to see the applications available in each environments. By selecting the new environment with the correct python version it is possible to see if Jupyter notebook has been downloaded already or not and in that case in stall it. Once the installation is completed Jupyter notebook can be launched from the same page in the Anaconda Navigator.

It is advised to launch the Jupyter notebook and create a repository dedicated to the analysis done with the pipeline.

## **Download of the pipeline**

The scripts that constitute the pipeline can be downloaded from the [GitLab repository](https://gitlab.com/DeliP/cattlelyst-software/-/tree/master/pipeline) of the WUR iGEM project 2021. The directory called pipeline should be placed in the same directory created in the [previous step](#_Jupyter_notebook). The following sub-repositories should be contained inside pipeline: inputs, scripts, outputs, tests, tutorials. Once this step is finished it is possible to use the functions of the pipeline by importing them inside the Jupyter notebooks. Use the tutorials 0 – 6 to get started with using the pipeline

To run the pipeline via command line download directory called `pipeline-cmd` from the GitLab repository. This directory should be placed in the location as in the [previous step](#_Jupyter_notebook)s and the call from command line should be done as follows:

conda activate <environment name>

cd <path to pipeline-cmd repository>

python call\_full\_analysis.py -r "./inputs" -i "./inputs/<input file name in .csv> "

For the command above to work the input file should be placed in the repository called “inputs” together with the BiGG model of the reference organism. For more detailed information on the usage see the paragraph called “usage”.

# **Features of the pipeline**

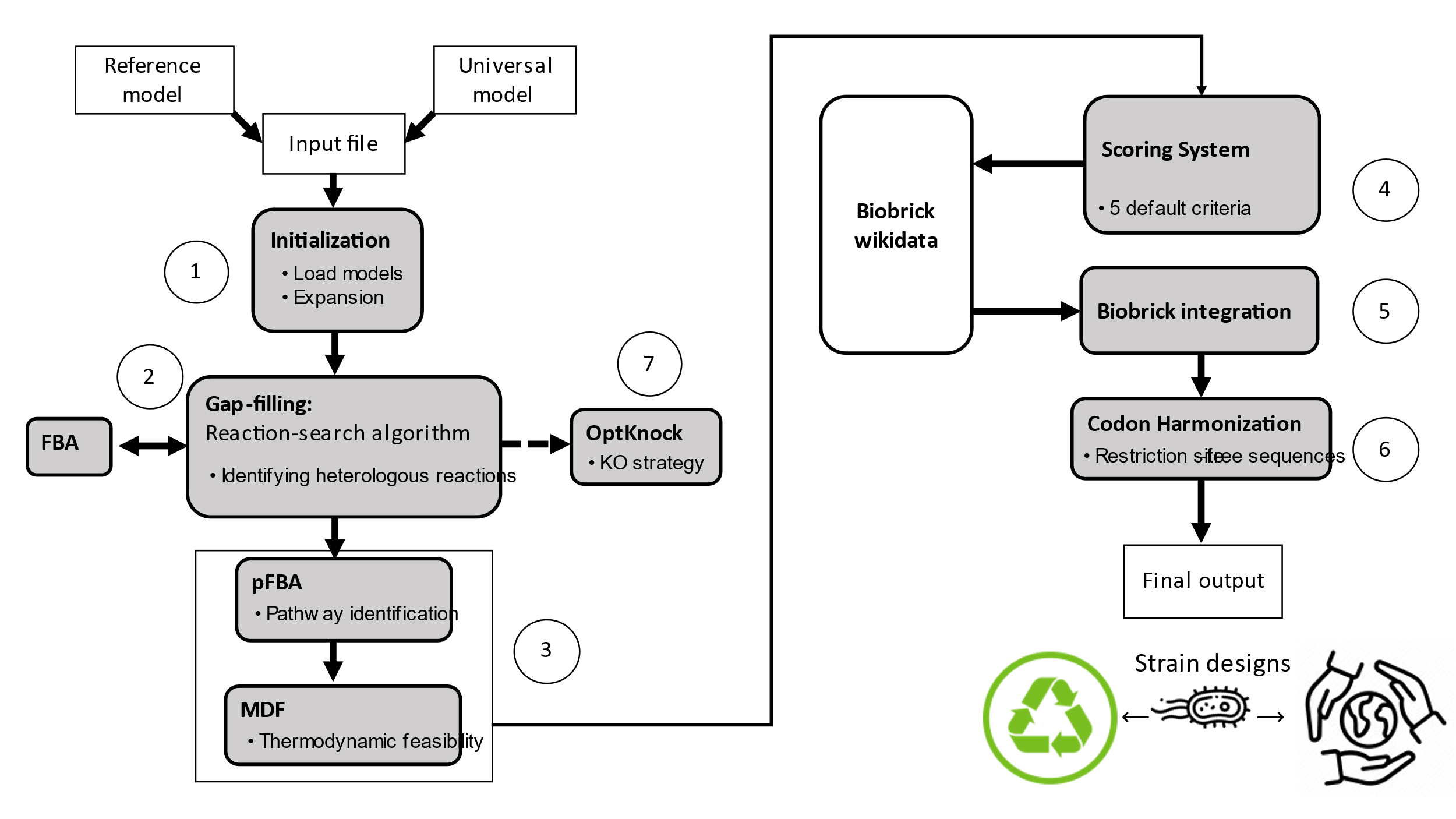
## **Innovative aspects**

The pipeline has three innovative aspects:

1. the prioritization of the assimilation of an uncommon substrate as the sole carbon source, which is particularly interesting for bioremediation purposes;
2. the identification of Biobrick parts corresponding to heterologous reactions and
3. their codon harmonisation, which is a step usually kept separated from strain design tools.

The pipeline takes a user-filled template file indicating the specifics of the analysis as input. See tutorial number 0 on how to fill the template input file.

## **Flowchart**



The pipeline's workflow is shown above.

1. Initialization involves loading the model, adding the exchange reactions and expanding the reaction database (here called universal model);
2. Reaction search with the Gapfilling algorithm suggests reaction to add to the reference model (here indicated as knock -ins). It is performed if the model does not already grow on the selected substrate and/or cannot produce the target compound;
3. Pathway definition and thermodynamic feasibility assessment evaluates the reaction knock-in strategies by calculating the pathway’s Max-min driving force value [7];
4. Comparison of the knock-in strategies is done by calculating the scores of each alternative by using an additive function;
5. Biobricks associated to the reactions knock-ins are identified querying the Biobrick wikibase [8];
6. The restriction site-free codon harmonised coding sequence of a Biobrick is obtained with an expanded version of Codon Harmonizer tool [9];
7. Optionally strategies involving the removal of reactions (here indicated as knock-outs) can be identified using the Optknock algorithm [3]

# **Usage**

The pipeline can be used via command line call. Two arguments are taken: the path to the repository with the input data, which should correspond to ./pipeline/inputs, and the path to the .csv input file. Usage details can be displayed with ‘call\_full\_analysis.py --help’. Tutorial number 7 explains how to use the pipeline’s workflow from command line.

Below you can find an example of call from command line.

conda activate <environment name>

cd <path to pipeline-cmd repository>

python call\_full\_analysis.py -r "./inputs" -i "./inputs/<input file name in .csv> "

The pipeline’s functions can be used from any other Python IDE. Jupyter Notebook is the one used for the tests that can be found in pipeline/tests/ directory. The procedure of the installation of Jupyter notebooks from Anaconda is described in [this paragraph](#_Jupyter_notebook). To use any function from the pipeline, once you’re in the repository for the analysis making use of the pipeline you can create a script there and import the functions by using the following structure:

from pipeline.scripts.<module name> import <function name>

This follows the import rules of Python and here is an example:

from pipeline.scripts.analysis import analysis\_gf\_sol

For a more detailed explanation on the usage of the main functions of the pipeline the user can refer to the tutorials 0 – 6.

# **Functions**

Below there is a description of input and output of the most important function of the pipeline’s modules.

## **analysis.py**

get\_biomass\_equation(*model*):

**Usage**

Find the reaction corresponding to the biomass equation

**Argument**

model--cobra.Model reference model in BiGG namespace

**Return**

biomass\_r\_list[0] or reaction: cobra.Reaction for biomass equation

set\_bounds\_obj(*input\_file*, *model*, *universal*):

**Usage**

Set the upper bound of the biomass reaction and the lb of the uptake reaction for optimization of consumption. It uses the float value of the growth rate on the preferred carbon source (resulting from get\_max\_growth\_on\_preferred\_c) and function as upper bounds of biomass. Additionally, the lb of the exchange reaction of the selected carbon source is set to -1000.

**Arguments**

input\_file--str input file in .csv format dictionary like

model--cobra.Model reference model in BiGG namespace

universal--cobra.Model universal model in BiGG namespace

analysis\_gf\_sol(*input\_file*, *model*, *universal*)

**Usage**

The function sets the right constraint in the model of the chassis and then calls the COBRApy’s function for [gapfilling analysis](https://cobrapy.readthedocs.io/en/latest/gapfilling.html" \t "_blank). In particular, *analysis\_gf\_sol*  calls functions (including set\_bounds\_obj ) to set the right model constraints for the analysis of the reactions to be added and to actually call the analysis with Gapfilling algorithm. It also prints the evaluation of the results of the analysis in an output file in .csv.

**Arguments**

input\_file--str input file in .csv format dictionary like

model--cobra.Model reference model in BiGG namespace

universal--cobra.Model universal model in BiGG namespace

**Return**

sol\_for\_csv: dict with the information on unique solutions for optimized consumption.

production\_analysis(*input\_file, model, universal*):

**Usage**

Similarly to analysis\_gf\_sol it calls the function for GapFilling analysis with target production as objective      
This function

    1) changes the objective of the model to the exchange of the

    target compound(s)

    2) Calls Gapfilling function and creates a dict with the solutions

    resulting from the iterations for each target compound

**Arguments:**

input\_file--str input file in .csv format dictionary like

model--cobra.Model reference model in BiGG namespace

universal--cobra.Model universal model in BiGG namespace

**Return:**

dict\_products: dict with the reactions resulting from GapFilling

dict\_prod\_sol(input\_file, sol\_for\_csv, model, universal):

**Usage**

Call functions for optimization of production and store information

**Arguments**

input\_file--str input file in .csv format dictionary like

    sol\_for\_csv--dict as output from analysis\_gf\_sol function

    model--cobra.Model reference model in BiGG namespace

    universal--cobra.Model universal model in BiGG namespace

**Return**

dict\_sol\_prod: dict with the solutions of GapFilling for production associated to each model viariant growing on the desired substrate

cons\_prod\_dict(*input\_file, model, universal, sol\_cons, sol\_prod*)

**Usage**

Create a dictionary with the reactions knock-ins for both optimizations. If the model can already grow on the selected carbon source and or produce the indicate target, the dictionary will be generated with the fluxes derived from the optimization without the need of the addition of any reaction.

**Arguments**

    input\_file--str input file in .csv format dictionary like

    model--cobra.Model reference model in BiGG namespace

    universal--cobra.Model universal model in BiGG namespace

    sol\_cons--dict retured by analysis\_gf\_sol function

    sol\_prod--dict retured by dict\_prod\_sol function

**Return**

    candp: dict with comprehensive information on reactions additions

        and fluxes for each model veraint growing on the indicated

        substrate and producing the target.

        Keys of candp Python dictionary

- consumption\_n where n = number in range(len(sol\_cons)

- production\_n where n = number in range(len(sol\_cons))

## **call\_Optknock.py**

full\_knock\_out\_analysis(*input\_file, output\_consumption, output\_con\_and\_prod, model, universal*):

**Usage**

Find knockout strategies and evaluate them with production envelope. It combines the two functions run\_optknock\_analysis and production\_env\_KO\_eval which respectively run Optknock algorithm and evaluate the solutions by calculating the production envelope on the model variant without and with the reaction knock outs.

**Arguments**

    input\_file--str input file in .csv format dictionary like

    output\_consumption--dict as output from analysis\_gf\_sol function

    output\_con\_and\_prod--dict returned by cons\_prod\_dict function

    model--cobra.Model reference model in BiGG namespace

    universal--cobra.Model universal model in BiGG namespace

**Return**

    evalKOs: dict with the pandas.DataFrame objects of the production

        envelope of each model variants and the correspondant KO

        mutants, as returned by production\_env\_KO\_eval function

## **codon\_harmonizer\_RSs.py**

This module was made by Shyam Saladi in May 2020. Delielena Poli from WUR iGEM team 2021 modified this function by adding two new function in order to make the algorithm able to exclude restriction sites from the harmonized sequence.

RS\_check(*seq, seq\_harmonized, source\_freq, target\_freq*):

**Usage**

Generate a restriction site-free harmonized sequence Given an harmonized sequence, it checks for the presence of restriction sites and if any is   
found it substitute the first codon involved in the site with the codon with the next lower   
impact on CHI difference.

**Arguments**

    seq--str, native sequence

    seq\_harmonized--str harmonized sequence

    source\_freq--pd.DataFrame with the relative frequencies and counts of all the codons   
 from the source organism

    target\_freq--pd.DataFrame with the relative frequencies and counts of all the codons from   
 the target organism

**Return**

    seq\_harmo: harmonized sequences without restriction sites

## **import\_models.py**

get\_reference\_model(*data\_dir, infile*)

**Usage**

Load the model of the organism that has to be engineered

**Arguments**

data\_dir--str path to be read by os.path.join

    infile--str input file in .csv format dictionary like

**Return**

    model: cobra.Model reference model of a specific organism

get\_universal\_main(data\_dir, infile)

**Usage**

Load the universal model, which is one of the 5 versions of BiGG database contained in CarveMe [Gitlab repository](https://github.com/cdanielmachado/carveme/tree/master/carveme/data/generated):

The reactions from all bacteria species in BiGG

A special universal reaction database with reactions specific for Gram negative and one for Gram positive bacetia

Another special reaction database specific for archaea

Finally a set of reactions that inlcudes specific cyanobacteria reactions

**Arguments**

data\_dir--str path to be read by os.path.join

infile--str csv file dictionary like

**Return**

universal: cobra.Model universal model

## **input\_parser.py**

main(*input\_file, universal, model*)

**Usage**

The main function is used to read the information in the input file and check for the presence of exchange reactions and transporters in the model of the organism of interest. Additionally it adds to the universal model any reactions that the user might have indicated as possibly involved in the conversion/production pathway.

**Arguments:**

    input\_file--str input file in .csv format dictionary like

    universal--cobra.Model universal model

    model--cobra.Model reference model

**Retrun**

    model: cobra.Model reference model with the right settings.

## **path\_definition\_mdf.py**

whole\_procedure\_path\_definition(*input\_file, model, raw\_tsv\_file\_name, input\_mdf\_filename*)

**Usage**

Calls the functions involved in finding the minimal set of reactions for the production of the target, in particular:

1. adds the free balancing reactions
2. remove maintenance
3. constrains substrate and product using the stoichiometric coefficients
4. checks the carbon source to be the expected one
5. set the target as model objective
6. checks the metabolism
7. gets reactions list from pFBA
8. processes the raw list to get the minimal reaction set
9. generates SBtab file with the results ready for MDF calculation

**Arguments**

    input\_file--str input file in .csv format dictionary like

    model--cobra.Model reference model in BiGG namespace

    raw\_tsv\_file\_name--str of the intermediate .tsv SBtab file

    input\_mdf\_filename--str of the .tsv SBtab file for MDF analysis

**Return**

    new\_r\_list:list of cobra.Reaction objects, pruned, returned by reaction\_list\_pruning   
 function

mdf\_analysis(*input\_file, model, raw\_tsv\_file\_name, input\_mdf\_filename*)

**Usage**

Combines all the functions for identifying the pathway and calculating MDF

Calls the function for the identification of the minimal set of reactions active in the conversion and uses the generated SBtab for MDF calculation. If anything in the process goes wrong and MDF can't be calculated or the pathway is not identified the values for pathway length and MDF are set to None.

**Arguments**

    input\_file--str input file in .csv format dictionary like

    model--cobra.Model reference model in BiGG namespace

    raw\_tsv\_file\_name--str of the intermediate .tsv SBtab file

    input\_mdf\_filename--str of the .tsv SBtab file for MDF analysis

**Returns**

    pathways\_MDF:mdf value (float) and its associated magnitude returned

        by get\_mdf\_value or NoneType

    path\_len: int, length of the list with the minimal reaction set as

        obtained from whole\_procedure\_path\_definition function

## **scoring\_system.py**

scores\_evaluations(*input\_file, output\_consumption, output\_con\_and\_prod*)

**Usage**

Rank the different engineering strategies. This function parses the output of generate\_scores function and ranks the model variants representing different engineering strategies by total scores in decreasing order. The model at position 1 is the one considered the best based on the "metabolic criteria": consumption and production rate, number of knock-ins, MDF, pathway length.

**Arguments**

    input\_file--str, input file in .csv format dictionary like

    output\_consumption--dict returned by analysis\_gf\_sol function

    output\_con\_and\_prod--dict returned by cons\_prod\_dict function

**Return**

    final\_output\_scores: dict with the ranked variant. The rank

        positions are the keys, while the values are tuples with

        model name and its finals score.

score\_BB\_presence(*input\_file, to\_wiki, from\_wiki, scores\_output*)

**Usage**

Get percentage of matching biobrick per model variant. It counts the number of knock ins involved in each different engineering strategy (model variant) and it counts the number of biobrick matching the reaction function were found by querying the wikidata biobrick database. It results the percentage of matching biobricks.

**Arguments**

    input\_file--str, input file in .csv format dictionary like

    to\_wiki--list of lists with information on EC number

        and KEGG ID of each heterologous reaction

    from wiki--dict type (nested dictionary) returned from

        wikidata query function.

    scores\_output--dict with the ranked variant. The rank

        positions are the keys, while the values are tuples with

        model name and its finals score.

**Return**

    BB\_scores: list of tuples in which the object with index 0 is

        the name of the model variant, while index 1 indicate

        the percentage of biobricks

## **to\_and\_from\_biobrick\_wikidata.py**

Module for interacition with Biobrick wikidata page and output evaluation.

Contains functions to query the biobrick wikidata and parse the output.

The output is used to:

1) get the percentage of biobrick matching the heterologous reactions of a pathway

2) generate codon harmonized and restriction sites free sequences of the gene(s) in the matching biobricks.

output\_for\_biobrick\_search(*universal, scores, output\_con\_and\_prod*):

**Usage**

Create output for the query of the biobrick wikidata. It reads the results of the metabolic optimization and create output for the query of the biobrick wikidata. The information needed for the query is the EC number of the heterologous reactions found with gapfilling optimization.

**Arguments**

    universal--cobra.Model universal model in BiGG namespace

    scores--dict with the ranked model variants and associated scores.

    output\_con\_and\_prod--dict with comprehensive information on

        reactions additions and fluxes for each model veraint

        growing on the indicated substrate and producing the target.

**Return**

    for\_biobrick\_search: list of lists with information on EC number

        and KEGG ID of each heterologous reaction

run\_harmonization\_per\_model(*output\_wikidata, input\_file*):

**Usage**

Read wikidata output and add harmonized sequences. For each model and each EC number (which are associated to the reaction) it obtains the coding sequences of native (with wget package) and expression host and it derives the codon frequences.

The codon harmonization is done on the fasta sequence of the gene(s) encoded by the matching biobrick. The sequence is also provided in the wikidata output. It returns a dictionary with the codon-harmonized and restriction-free sequences for each model.

**Arguments**

    output\_wikidata--dict type (nested dictionary) returned from

        wikidata query function.

    input\_file--str input file in .csv format dictionary like

**Return**

    harmonized\_seq\_per\_model: nested dictionary with the model variants

        as keys, and the EC numbers of the heterologous reactions as

        keys of the nested dictionary, while the values are tuples.

# **Tutorials**

List of the tutorials that can be accessed through the GitLab repository.

0. How to fill the input file

This tutorial gives a detailed description on how to fill the input file in Comma Separate Value (.csv) format.

1. How to load the model and prepare them for further analysis

An example of how to open the model of the organism of interest and the reaction database for the analysis. Additionally, in this tutorial it is show how the information in the input file are used for setting up part of the analysis.

2.1 Analysis for reactions addition allowing growth on uncommon substrates

In this example a model of *Escherichia coli* is used to generate strains able to grow on methane. Formate is indicated as target product as it is an intermediate of methane oxidation pathway.

2.2 Analysis for reactions addition allowing growth on uncommon substrates and production of a target compound

In this tutorial a model of *Escherichia coli* is used to generate strains growing on methane and producing itaconate.

3. Using the pipeline together with COBRApy functions to studying auxotrophy

In this tutorial the pipeline will suggest the reactions to knock-in to allow *E. coli* auxotrophic for Tryptophan to grow on methane and produce Arginine.

4. How to use the pipeline when production of the target is the only objective

This tutorial makes use of the functions of the pipeline to generate a strain of *E. coli* growing on glucose and producing itaconic acid.

5. Running the thermodynamic analysis using the functions of the pipeline

In the following tutorial some of the functions for the thermodynamic analysis are used individually on one of the combination of the strains of *E. coli* growing on methane.

6. How to find reaction knock-outs

In this tutorial uses the same example case as the tutorial number 4. The aim is to generate a strain of *E. coli* growing on glucose and producing itaconic acid.

# **Status**

The author of the pipeline is Delielena Poli. delielena.poli@gmail.com

The pipeline is under development. The step *v* in the flowchart has to be finalized and implemented in the pipeline.

# **Similar tools**

If the user is interested in similar tools below there is a short list of them:

* Cameo
* OptStrain
* OptCouple

[Cameo](http://cameo.bio/) is a COBRApy based library [Z]. [OptStrain](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC525696/) [X]and the recent [OptCouple](https://www.sciencedirect.com/science/article/pii/S2214030118300373) [Y] are two examples of tools considering both addition and deletion of reactions to achieve product formation in a growth-coupled manner. OptStrain uses a two-steps optimization approach in which non-native reactions are found first and reaction knock-outs are successively identified with the Optknock algorithm [4]. OptCouple has been developed to look for reaction insertions, deletions and medium supplements simultaneously. Both tools exist in implementations which are under development. OptStrain as independent Matlab functions (<https://github.com/Ernir/optstrain>), OptCouple as a function in Python based Cameo tool.

# **References**

[1] A. Ebrahim, J. A. Lerman, B. O. Palsson, and D. R. Hyduke, “COBRApy: COnstraints-Based Reconstruction and Analysis for Python,” *BMC Syst. Biol.*, vol. 7, no. 1, p. 74, Aug. 2013, doi: 10.1186/1752-0509-7-74.

[2] GLPK, “GLPK.”Available: http://www.gnu.org/software/glpk.

[3] A. P. Burgard, P. Pharkya, and C. D. Maranas, “OptKnock: A Bilevel Programming Framework for Identifying Gene Knockout Strategies for Microbial Strain Optimization,” 2003, doi: 10.1002/bit.10803

[4] A. P. Burgard, P. Pharkya, and C. D. Maranas, “OptKnock: A Bilevel Programming Framework for Identifying Gene Knockout Strategies for Microbial Strain Optimization,” 2003, doi: 10.1002/bit.10803.

[5] A. Waa, “A protocol for adding knowledge to Wikidata, a case report,” bioRxiv, p. 2020.04.05.026336, Apr. 2020, doi: 10.1101/2020.04.05.026336.

[6] A. Flamholz, E. Noor, A. Bar-Even, and R. Milo, “eQuilibrator-the biochemical thermodynamics calculator,” Nucleic Acids Res., vol. 40, 2012, doi: 10.1093/nar/gkr874

[7] E. Noor, A. Bar-Even, A. Flamholz, E. Reznik, W. Liebermeister, and R. Milo, “Pathway Thermodynamics Highlights Kinetic Obstacles in Central Metabolism,” PLoS Comput. Biol., vol. 10, no. 2, p. e1003483, Feb. 2014, doi: 10.1371/journal.pcbi.1003483.

[8] I. Fundation, “iGEM foundation.” https://igem.org/Main\_Page (accessed Oct. 24, 2020).

[9] N. J. Claassens et al., “Improving heterologous membrane protein production in Escherichia coli by combining transcriptional tuning and codon usage algorithms,” PLoS One, vol. 12, no. 9, Sep. 2017, doi: 10.1371/journal.pone.0184355

[Z] J. G. R. Cardoso *et al.*, “Cameo: A Python Library for Computer Aided Metabolic Engineering and Optimization of Cell Factories,” *ACS Synth. Biol.*, vol. 7, no. 4, pp. 1163–1166, Apr. 2018, doi: 10.1021/acssynbio.7b00423.

[X] P. Pharkya, A. P. Burgard, and C. D. Maranas, “OptStrain: A computational framework for redesign of microbial production systems,” Genome Res., vol. 14, no. 11, pp. 2367–2376, Nov. 2004, doi: 10.1101/gr.2872004.

[Y] K. Jensen, V. Broeken, A. S. L. Hansen, N. Sonnenschein, and M. J. Herrgård, “OptCouple: Joint simulation of gene knockouts, insertions and medium modifications for prediction of growth-coupled strain designs,” Metab. Eng. Commun., vol. 8, p. e00087, Jun. 2019, doi: 10.1016/j.mec.2019.e00087.