cameo Documentation

Release 0.1.0

Nikolaus Sonnenschein, João Cardoso

July 14, 2015

Contents

Warning: These pages are under construction. Feel free to look around ...

cameo is a python library for computer-aided metabolic engineering and optimization of microbes. It caters to different audiences as it provides a high-level interface that allows computer savvy bench biologists to predict heterologous pathways and enumerate engineering strategies and a modular framework for computational modelers that facilitates methods development and the construction of custom analysis workflows and prediction tools.

cameo is based on cobrapy, the community standard for constraint-based modeling of genome-scale metabolic models, and uses and extends the same data structures. For efficiency reasons, however, it differs quite significantly from cobrapy in the way optimization problems are solved and defined. Nevertheless, cameo models remain 100% compatible with cobrapy. If you are already a cobrapy user you might want to start reading here to get an overview of the main differences.

CHAPTER 1

Table of Contents

1.1 Installation

1.1.1 Setting up a virtual environment first

We highly recommended installing cameo inside a virtual environment (virtualenv). virtualenvwrapper tremendously simplifies using virtualenv and can easily be installed using virtualenv-burrito. Once you installed virtualenv and virtualenvwrapper, run

```
$ mkvirtualenv cameo  # or whatever you'd like to call your virtual environment
$ workon cameo
```

and then continue with the installation instructions described below.

1.1.2 Non-python dependencies

cameo relies on optlang to solve optimization problems. Currently, optlang supports either glpk (open source) or cplex (academic licenses available), which are not python tools. At least one of them has to be installed before one can proceed with the cameo installation.

GLPK

Using cameo with glpk also requires swig to be installed (in order to generate python bindings). On ubuntu (or other similar linux platforms) we recommend using `apt-get`:

```
$ sudo apt-get install libglpk-dev glpk-utils swig
```

On macs we recommend using homebrew.

```
$ brew install swig
$ brew install glpk
```

CPLEX

The cplex contains a python directory (similar to `IBM/ILOG/CPLEX_Studio1251/cplex/python/x86-64_osx`). Inside this directory run

```
$ python setup.py install
```

to install the python bindings.

1.1.3 Normal installation

Warning: cameo is still under heavy development. We recommend installing the development version (see below) if you would like to stay up-to-date with the latest changes.

cameo can be installed using *pip*.

```
$ pip install cameo
```

1.1.4 Development setup

pip can also be used to install cameo directly from the github repository.

```
$ pip install -e git+https://github.com/biosustain/cameo.git@devel#egg=cameo
```

Alternatively, you can clone the repository (or your fork) and then run

```
$ python setup.py install
```

From withing the cameo directory.

1.2 Quickstart

```
import pandas
pandas.options.display.max_rows = 15
from cameo import load_model
```

```
# model = load_model('Ecoli core Model')
model = load_model('iJO1366')
```

```
solution = model.solve()
```

```
solution.to_frame()
```

1.3 High-level interface for users

Users primarily interested in using cameo as a tool for enumerating metabolic engineering strategies have access to cameo's advanced programming interface via :mod:`cameo.api` that provides access to potential products (:mod:`cameo.api.products`), host organisms (:mod:`cameo.api.hosts`) and a configurable design function (:attr:`cameo.api.design`). Running :func:`cameo.api.design` requires only minimal input.

```
from cameo import api
report = api.design(product='L-serine')
```

```
Found 5 compounds that match query `L-serine'
                     name     formula charge     mass  MNXM53           L-serine     C3H7NO
MNXM114         L-proline     C5H9NO2      0   115.13
MNXM3635       L-mimosine   C8H10N2O4      0  198.176
MNXM89905      L-allysine    C6H11NO3      0  145.156
MNXM384    L-saccharopine  C11H19N2O6     -1  275.278

                                                      InChI  MNXM53     InChI=1S/C3H7NO3/c
MNXM114    InChI=1S/C5H9NO2/c7-5(8)4-2-1-3-6-4/h4,6H,1-3H...
MNXM3635   InChI=1S/C8H10N2O4/c9-5(8(13)14)3-10-2-1-6(11)...
MNXM89905  InChI=1S/C6H11NO3/c7-5(6(9)10)3-1-2-4-8/h4-5H,...
MNXM384    InChI=1S/C11H20N2O6/c12-7(10(16)17)3-1-2-6-13-...

                                                    SMILES       source  MNXM53
MNXM114                              [O-]C(=O)[C@@H]1CCC[NH2+]1  chebi:60039
MNXM3635           [NH3+][C@@H](CN1C=CC(=O)C(O)=C1)C([O-])=O  chebi:29063
MNXM89905                    [H]C(=O)CCC[C@H]([NH3+])C([O-])=O  chebi:58321
MNXM384    [NH3+][C@@H](CCCC[NH2+][C@@H](CCC([O-])=O)C([O...  chebi:57951

           search_rank
MNXM53               0
MNXM114              1
MNXM3635             2
MNXM89905            3
MNXM384              4
Choosing best match (L-serine) ... please interrupt if this is not the desired compound.
```

```
/Users/niko/Arbejder/Dev/cameo/cameo/api/products.py:75 [1;31mSettingWithCopyWarning[0m:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
```

```
<IPython.core.display.Javascript at 0x114c52910>
```

```
<IPython.core.display.Javascript at 0x114c52950>
```

```
<IPython.core.display.Javascript at 0x114c52d10>
```

```
<IPython.core.display.Javascript at 0x114c52e10>
```

```
<IPython.core.display.Javascript at 0x11bbcaa90>
```

```
<IPython.core.display.Javascript at 0x11bbcaa50>
```

```
<IPython.core.display.Javascript at 0x1048f3e50>
```

```
<IPython.core.display.Javascript at 0x11aa711d0>
```

```
<IPython.core.display.Javascript at 0x11c280cd0>
```

```
<IPython.core.display.Javascript at 0x11aa71150>
```

```
<IPython.core.display.Javascript at 0x11aa71090>
```

```
<IPython.core.display.Javascript at 0x11aa71110>
```

```
<IPython.core.display.Javascript at 0x11aa5eed0>
```

```
<IPython.core.display.Javascript at 0x11aa5ee90>
```

```
<IPython.core.display.Javascript at 0x11aa5ef10>
```

```
<IPython.core.display.Javascript at 0x11aa5ef50>
```

```
<IPython.core.display.Javascript at 0x114c52e90>
```

```
<IPython.core.display.Javascript at 0x11934d650>
```

```
<IPython.core.display.Javascript at 0x125185510>
```

```
<IPython.core.display.Javascript at 0x1251852d0>
```

```
<IPython.core.display.Javascript at 0x126889f10>
```

```
<IPython.core.display.Javascript at 0x1251851d0>
```

```
<IPython.core.display.Javascript at 0x125185250>
```

```
<IPython.core.display.Javascript at 0x125185190>
```

```
<IPython.core.display.Javascript at 0x125172e10>
```

```
<IPython.core.display.Javascript at 0x125172050>
```

```
<IPython.core.display.Javascript at 0x125185210>
```

```
<IPython.core.display.Javascript at 0x125172d90>
```

1.3.1 IPython notebook

Click here to download this page as an IPython notebook.

1.4 Low-level interface for developers

```
import pandas
pandas.options.display.max_rows = 8
from cameo import load_model
model = load_model('iJO1366')
```

1.4.1 cameo vs. cobrapy

Importing a model

cobrapy (load a model in SBML format):

```
from cobra.io import read_sbml_model
cobrapy_model = read_sbml_model('../tests/data/EcoliCore.xml')
```

cameo (load models from different formats):

```
from cameo import load_model
# read SBML model
model = load_model('../tests/data/EcoliCore.xml')
# ... or read a pickled model
model = load_model('../tests/data/iJO1366.pickle')
# ... or just import a model by ID from http://darwin.di.uminho.pt/models
iAF1260 = load_model('iAF1260')
```

Solving models

When optimizing models with cobrapy, the status of a returned solution needs to be checked to figure out if the solver found and optimal solution.

```
cobrapy_solution = cobrapy_model.optimize()
if cobrapy_solution.status == 'optimal':
    pass  # do something
```

cameo will raise an exception if non-optimal solution was found.

```
print model.objective
```

```
Maximize
1.0*Ec_biomass_iJO1366_core_53p95M
```

```
model.solve()  # no exception
cp_model = model.copy()
# demand a very high biomass production rendering the model infeasible
cp_model.reactions.Ec_biomass_iJO1366_core_53p95M.lower_bound = 1000
cp_model.solve()
```

```
---------------------------------------------------------------------------
Infeasible                                Traceback (most recent call last)

<ipython-input-13-286e284739c2> in <module>()
      2 cp_model = model.copy()
      3 cp_model.reactions.Ec_biomass_iJO1366_core_53p95M.lower_bound = 1000  # demand an unrealistic
----> 4 cp_model.solve()


/Users/niko/Arbejder/Dev/cameo/cameo/solver_based_model.pyc in solve(self, *args, **kwargs)
    881                 raise exceptions._OPTLANG_TO_EXCEPTIONS_DICT.get(status, SolveError)(
    882                     'Solving model %s did not return an optimal solution. The returned soluti
--> 883                         self, status))
    884             return solution
    885         else:


Infeasible: Solving model iJO1366 did not return an optimal solution. The returned solution status is
```

Therefore, :code:try-except-else-finally statements should be used in code that utilizes :py:meth:cameo.solver_based_model.SolverBasedModel.solve.

```
try:
    solution = model.solve()
except cameo.exceptions.SolverError:
    print "A non-optimal solution was returned by the solver"
else:
    pass  # do something
```

It is important to note that cameo models maintain `optimize` to maintain compatibility with cobrapy but we discourage its use.

Solution object

```
solution
```

1.4.2 Formulating and solving optimization problems

http://optlang.readthedocs.org/en/latest/

1.4.3 Importing models

cameo has a single function (`load_model`) that can import models stored in different formats.

SBML

Models available on the web

cameo provides an interface to models hosted online by the University of Minho.

```
from cameo.webmodels import index_models
index_models()
```

If `load_model` is provided with a valid model name, it will automatically download the model in SBML format and import it.

```
load_model('iMM904')
```

Strain-design

All strain-design methods have been coded as python classes t

1.4.4 Predicting heterologous pathways

```
from cameo.strain_design.pathway_prediction import PathwayPredictor
```

```
PathwayPredictor(model)
```

```
pathway_predictor = PathwayPredictor(model)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-4-9e8d88b43feb> in <module>()
----> 1 pathway_predictor = PathwayPredictor(model)


NameError: name 'model' is not defined
```

1.4.5 Deterministic methods

1.4.6 Heuristic methods

1.4.7 Avoid copies at all costs

Copying models is expensive!

```
%time model_copy = model.copy()
```

```
CPU times: user 3.17 s, sys: 84.7 ms, total: 3.26 s
Wall time: 3.25 s
```

To avoid excessive copying of models, we suggest the following utility included in cameo. For example, temporary reaction deletion, solve the model and magically returns to it original state.

```
from functools import partial
from cameo.util import TimeMachine
with TimeMachine() as tm:
    tm
```

TimeMachine performs `do` steps immediately and stores all corresponding `undo` actions in order. Running all `undo` steps can be achieved with `TimeMachine.reset()` (wich is run) We recommend using `TimeMachine` in conjunction with `with`, as in case of an unforseen exception all `undo` steps will be run and manipulated models will return to their original states. This is more elegant than using explicit `try-except-else-finally` statements.

1.4.8 IPython notebook

Click here to download this page as an IPython notebook.

1.5 Parallelization

Most methods in cameo can be parallelized using views.

1.6 cameo vs. cobrapy

1.6.1 Importing a model

cobrapy (load a model in SBML format):

```
from cobra.io import read_sbml_model
model = read_sbml_model('path/to/model.xml')
```

cameo (load models from different formats):

```
from cameo import load_model
# read SBML model
model = load_model('path/to/model.xml')
# ... or read a pickled model
model = load_model('path/to/model.pickle')
# ... or just import a model by ID from http://darwin.di.uminho.pt/models
iAF1260 = load_model('iAF1260')
```

1.6.2 Solving models

cobrapy:

```
solution = model.optimize()
if solution.status == 'optimal':
    # proceed
```

```
try:
    solution = model.solve()
except cameo.exceptions.SolverError:
    print "A non-optimal solution was returned by the solver"
else:
    # proceed
```

It is important to note that cameo models maintain *optimize* to maintain compatibility with cobrapy but we discourage its use.

1.7 How to ...

- ... run differential flux-variability analysis
- ... perform a heuristic gene knockout optimization (single-objective)
- ... perform a heuristic gene knockout optimization (multi-objective)

1.8 API

1.8.1 cameo package

Subpackages

cameo.api package

Submodules

cameo.api.designer module

cameo.api.hosts module

cameo.api.products module

Module contents

cameo.core package

Submodules

cameo.core.reaction module

cameo.core.solution module

cameo.core.solver_based_model module

Module contents

cameo.flux_analysis package

Submodules

cameo.flux_analysis.analysis module

cameo.flux_analysis.simulation module

Module contents

cameo.strain_design package

Subpackages

cameo.strain_design.deterministic package

Submodules

cameo.strain_design.deterministic.flux_variability_based module

Module contents

cameo.strain_design.heuristic package

Subpackages

cameo.strain_design.heuristic.multiprocess package

Submodules

cameo.strain_design.heuristic.multiprocess.migrators module

cameo.strain_design.heuristic.multiprocess.observers module

cameo.strain_design.heuristic.multiprocess.optimization module

cameo.strain_design.heuristic.multiprocess.plotters module

Module contents

Submodules

cameo.strain_design.heuristic.archivers module

cameo.strain_design.heuristic.decoders module

cameo.strain_design.heuristic.generators module

cameo.strain_design.heuristic.genomes module

cameo.strain_design.heuristic.metrics module

cameo.strain_design.heuristic.objective_functions module

cameo.strain_design.heuristic.observers module

cameo.strain_design.heuristic.optimization module

cameo.strain_design.heuristic.plotters module

cameo.strain_design.heuristic.stats module

cameo.strain_design.heuristic.variators module

Module contents

cameo.strain_design.pathway_prediction package

Module contents

Module contents

Submodules

cameo.config module

cameo.exceptions module

cameo.io module

cameo.parallel module

cameo.stuff module

cameo.util module

cameo.visualization module

cameo.webmodels module

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search