HPC/Exascale Centre of Excellence in Personalised Medicine

SYSTEMATISING COMPLEX AND COMBINED METABOLIC ANALYSES WITH COBREXA.JL

MIREK KRATOCHVÍL (UNI.LU), DANIEL THOMAS LOPEZ (EMBL EBI)

Nov 15TH 2022

The PerMedCoE project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement Nº951773

0

Preparatory session

Check-list

- Julia is installed
- COBREXA is installed
  1. start Julia
  2. press `]` to activate packaging mode
  3. type `add COBREXA`
  4. return to normal mode using backspace
- at least one solver is installed (preferably GLPK)
- packages compile all right

Test data

`http://bigg.ucsd.edu/static/models/e_coli_core.json`

(search engines: 'BiGG E coli Core JSON')

Test

```
using COBREXA, GLPK

download(
  "http://bigg.ucsd.edu/static/models/e_coli_core.json",
  "e_coli_core.json")

model = load_model("e_coli_core.json")

fluxes = flux_balance_analysis_dict(model, GLPK.Optimizer)
```

See you on Nov 15TH!

Systematising complex and combined metabolic analyses with COBREXA.jl

Mirek Kratochvíl (Uni.Lu), Daniel Thomas Lopez (EMBL EBI)

Nov 15TH 2022

Follow us in social media:

linkedin.com/company/permedcoe

@permedcoe

The PerMedCoE project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement Nº951773

PerMedCoE

HPC/Exascale Centre of Excellence in Personalised Medicine

www.permedcoe.eu

1

What's and Why's

Constraint-based modeling (recap)

1. Have some biochemical knowledge about processes in organisms
2. Summarize the knowledge as a linear constrained model
   - reactions convert metabolites to other metabolites
   - atoms are preserved
3. Gather observations about your model using linear optimization machinery

Main systematic problems:

- What is the best knowledge representation?
- How precise is the knowledge?

$$\begin{aligned} \text{find} \quad & \arg\max_x \; c^T x \\ \text{such that} \quad & Sx = 0 \\ & x \geq u \\ & x \leq l \end{aligned}$$

COBRA TOOL DEVELOPMENT

Old times Build the model from what you remember from school&lab; directly use the LP solver.

Boxed analyses Use precise model formats and several streamlined reconstruction procedures, enjoy some analyses implemented in COBRA toolboxes (COBRApy, matlab, …).

Extensibility Use arbitrary model formats, construct any procedures and analyses from sensible reusable building blocks.

Main topics for today

- Avoid long fragile analysis scripts
- Learn to 'package' and 'compose' things that happen in metabolic modeling
- Gain some side software-engineering benefits (efficiency, parallelization&reentrancy, reproducibility, repurposability, transparency).

2

COBREXA.jl Design Primer

What is a model?

Traditional answer: "Representation of an organism that may or may not have genes, reactions, metabolites, biopolymers, annotations, gene-reaction associations, enzyme amounts, cell signaling state, …, preferably reflecting some reality and solving just right"

COBREXA.jl view: "Anything we can convert to an annotated linear model"

Model types

Idea: Let's not care about the actual model type

Specialized model types:

- `SBMLModel`
- `MATModel`
- `JSONModel`
- `HDF5Model`
- `StandardModel` (COBRApy-like)
- `CoreModel` (core linear-algebraic model)
- `GeckoModel`
- `SMomentModel`
- …

Interfaces (abstract):

- `MetabolicModel`
- `ModelWrapper`

Accessors that work on all model types:

- `reactions`, `metabolites`
- `stoichiometry`
- `bounds`
- `coupling`, `coupling_bounds`
- …

Let's do a small demo of the method-based overloading mechanism[1] that makes this possible.

[1] Not specific to Julia, you might know the same from C++, C# TypeScript, Swift, Java, …

Systematization of Modifications and Analyses

- **Model variants** are functions that takes a `MetabolicModel` and produces another `MetabolicModel` with some specific change applied.
  Possible examples: `with_added_reaction`, `with_removed_metabolite`, `knockout_gene`, `with_changed_bounds`, `with_enzyme_crowding_bounds`, `with_enzyme_usage_bounds` …
- **Analysis functions** take a `MetabolicModel`, convert it into the optimizer language in some analysis-specific way, and run the analysis.
  Examples: `flux_balance_analysis`, `parsimonious_flux_balance_analysis`, `minimize_metabolic_adjustment`, …
- **Optimizer modifications** are an additional concept added for efficiency, functions that take a model and its representation in the linear optimizer, and adjust the optimizer to do something differently.
  Possible examples: `knockout`, `add_enzyme_usage_constraints`, `change_objective`, `set_quadratic_objective`, `set_optimizer_parameter`, `add_loopless_constraints`, …
- **Meta-analysis functions** run analysis functions in multiple steps for a more complicated goal.
  Main example: `screen`, derived examples: `objective_envelope`, `flux_variability_analysis`, …

Systematization of Modifications and Analyses

Simplified view

- **Model variants**

  `MetabolicModel` → `MetabolicModel`

- **Analysis functions**

  `MetabolicModel` → result

- **Optimizer modifications**

  (`MetabolicModel`, `Optimizer`) → adjustments to the optimizer

- **Meta-analysis functions**

  (`MetabolicModel`, array of variants, analysis) → array of results

Systematization of Modifications and Analyses

Simplified view

- **Model variants** — can be chained without any danger

  `MetabolicModel` → `MetabolicModel`

- **Analysis functions** — applied at the end of chain

  `MetabolicModel` → result

- **Optimizer modifications**

  (`MetabolicModel`, `Optimizer`) → adjustments to the optimizer

- **Meta-analysis functions**

  (`MetabolicModel`, array of variants, analysis) → array of results

Systematization of Modifications and Analyses

Simplified view

- **Model variants**

  `MetabolicModel` → `MetabolicModel`

- **Analysis functions**

  `MetabolicModel` → result

- **Optimizer modifications**

  (`MetabolicModel`, `Optimizer`) → adjustments to the optimizer

- **Meta-analysis functions** — apply to 'fork' the chain

  (`MetabolicModel`, array of variants, analysis) → array of results

Systematization of Modifications and Analyses

Simplified view

- **Model variants**

  `MetabolicModel` → `MetabolicModel`

- **Analysis functions**

  `MetabolicModel` → result

- **Optimizer modifications** — must be chained carefully

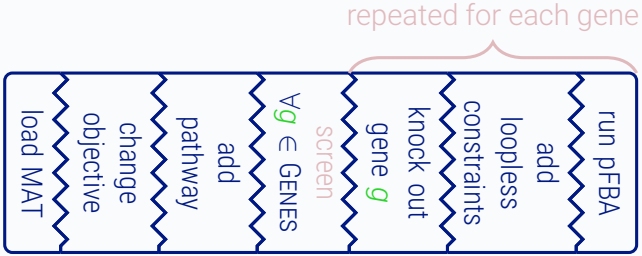
  (`MetabolicModel`, `Optimizer`) → adjustments to the optimizer

- **Meta-analysis functions**

  (`MetabolicModel`, array of variants, analysis) → array of results

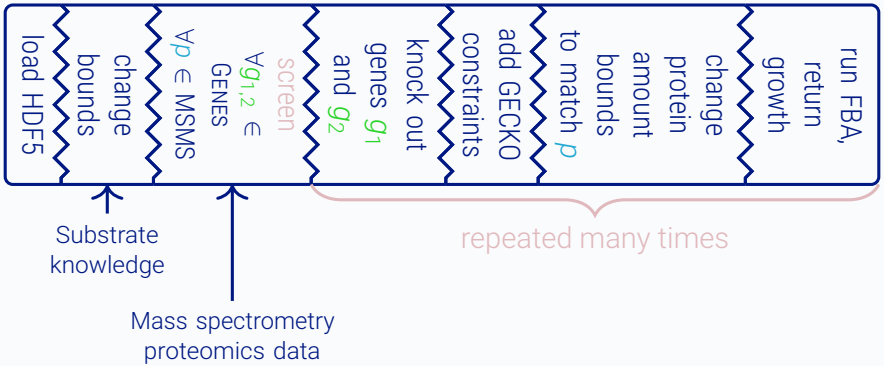
Systematization example in a picture

A complicated scheme of knockout screening



Systematization example in a picture

Backfitting data to lab measurements



The main goals today

1. Get familiar with this system
2. Use it for realistic tasks
3. Write your own model variants/modifications

3

Exercises — Basics

Start up!

Load the E. coli toy model (or the big one, it doesn't really matter), and find optimal production.

You need:

- `load_model`
- `flux_balance_analysis_dict`

Start up!

Manual vs. named model modification

1. `convert` the model to a `StandardModel`, restrict oxygen intake and see what happens.
2. Restrict oxygen intake using `change_constraint`
3. Can the model live completely without oxygen? (glucose?)

Start up!

Modifications combine easily

How much CO_2 can the model produce given just $0.5 \frac{\text{mmol}}{g_{DW} h}$ oxygen?

(use `change_objective`)

What if we restrict some other intakes too?

Simple meta-analyses

- Use `flux_variability_analysis_dict` to explore variability in the reaction fluxes.
- …ensuring the model produces at least at 90% of the original production.
- …given the oxygen intake is restricted.
- …and there is no glucose to ingest.
- …and gene with ID `b0727` (*sucB*) is knocked out. (use `knockout`)

What *are* these modification functions?

What *are* these modification functions? Functions that return another functions.

- Functions are the best (and likely only) means to freely *parametrize the functionality* of code.
- Think about them as small recipes that the target function reads and knows what to do.
- You can easily make your own recipes to implement more functionality!

Exercise: `bounds` parameter of FVA is actually also a 'recipe' — FVA gives it the optimal objective value it found, and expects to receive the bounds it should place on the objective. Try using `objective_bounds` or `gamma_bounds` manually.

Task: run a FVA that explores model production at *precisely* 66% of the optimum capacity.

Custom optimizer modification

Make a modification that uses `change_constraint` to reduce limits of all reactions that require a certain gene by a given factor.

Example use:

```
flux_balance_analysis_dict(m, GLPK.Optimizer, modifications = [
  choke_gene_reactions("b0727", 0.5),
])
```

4

Exercises — Screening

Screening

`screen` function "just" runs an analysis over a huge amount of models generated from a single original model.

We found that pretty useful:

- parallelizes well
- you don't need to write possibly complicated cycles manually
- prevents programming errors

Screening

Parameters of `screen()`

- model

`analysis=` a function to run on the model (possibly many times)

`args=` array of arguments to be passed to all function invocations

`variants=` array of variant chains to be applied to the base model before it is given to the analysis function

Screening

Use `screen` and `change_constraint`[2] to see how much biomass can be produced depending on variable oxygen availability.[3]

[2] you may use also `change_bound`, but that only works with mutable models

[3] normally we would use `objective_envelope`, which is optimized for this purpose

Use `reactions` and `change_constraint` to disable reactions, see what happens.

- what happens after choking the ATPM reaction?
- …and the biomass reaction?

If everything went all right[4], we should have a break sometime around now.

[4] the plan was naive and this timing is improbable

Use `knockout` to see what the model would look like without important genes.

- make a list of genes that make the organism unable to grow
- …and that make the model completely infeasible

Run a double gene knockout screening on the model:

- make a matrix of gene combinations (let's not worry about duplicities now)
- careful about `knockout` — it is not associative!

Screening

Choking all possible genes

Use the previously created modification to see what happens if you choke each of the genes by 10%, 20%, 30%, …

A QUICK NOTE ABOUT PARALLELIZATION

COBREXA.jl was originally intended as a HPC runner of large screening-type analyses. Speeding up your analysis is thus quite trivial:

1. Add a few Julia "worker" processes to form a cluster
   - locally, `using Distributed; addprocs(10);`
   - on HPC, you can e.g.: `using Distributed,ClusterManagers; addprocs_slurm(100);`

   …you may check the available workers IDs by `workers`
2. load COBREXA and the solver everywhere: `@everywhere using COBREXA, GLPK`
3. pass in argument `workers` to parallelizable meta-analysis functions, such as:

   ```
   flux_variability_analysis(m, GLPK.Optimizer, workers=workers())
   screen(m, analysis=..., args=..., workers=workers())
   ```

   Since the problems are moreless embarassingly parallel, the analysis times should get (roughly) divided by worker count.

5

MODIFYING THE MODIFICATIONS

How to make a model variant?

1. Make a new structure to hold your model
2. Load any information you need from parameters and the wrapped model
3. Overload the accessors to provide the modified information

Efficiency concerns:

- Source the least amount of stuff
- Modify/reconstruct only the data you really need to
- You can cache some precomputed items locally (models are immutable!)

ModelWrapper

`ModelWrapper` is a subtype of `MetabolicModel` that you can use to easily derive models from others.

```
struct MyTrivialModel <: ModelWrapper
  inner :: MetabolicModel
end

COBREXA.unwrap_model(x::MyTrivialModel) = x.inner

m = MyTrivialModel(load_model(...))
flux_balance_analysis(m, GLPK.Optimizer)
```

Overriding parts of models

```
struct ScaledModel <: ModelWrapper
  inner::MetabolicModel
  ratio::Float64
end

COBREXA.unwrap_model(x::ScaledModel) = x.inner
COBREXA.bounds(x::ScaledModel) =
  let (lbs, ubs) = bounds(x.inner)
    (x.ratio .* lbs, x.ratio .* ubs)
  end
COBREXA.balance(x::ScaledModel) = x.ratio * balance(x.inner)

m = ScaledModel(load_model(...), 0.5)
flux_balance_analysis(m, GLPK.Optimizer)
```

Helpful bonus: Piping

```
m = ScaledModel(load_model(...), 0.5)
```

…is the same as…

```
m = load_model(...) |> ScaledModel(0.5)
```

6

Exercises — Custom variants

Reminder

Why do we want to make model wrappers rather than optimizer modifications?

- **Models can be converted to SBML/JSON and saved.**
- **Well-constructed model wrappers chain indefinitely without any risks.**

Make a *model wrapper* where ATP is constantly "injected" into otherwise normal model.

Possible approaches:

- change the balance vector
- add a reaction

(What are the benefits and downsides of both approaches?)

Make a model where glucose and oxygen uptake is constrained to be the same (by molar amount) as carbon dioxide output.

Possible approaches:

- add an extra reaction
- use coupling

How to structure the community models?

Let's ferment some example bread

Make a model that contains several copies of the wrapped model, each with a different variant applied (e.g., with a knockout).

- simplifications:
  - start with just N same models
  - we can ignore genes, annotations and other properties for now
- community members can be otherwise completely independent

Combine the reaction-rate balancing wrapper with the community wrapper to make a biomass-stable community.

- All submodels should produce the same amount of biomass.
- Bonus exercise: All submodels have their O_2 vs. CO_2 exchanges balanced (individually).

Which glucose vs. ammonia intake ratio is the best for our E. coli?

That's all for today.

Time for questions?

Where to go next?

- There is more functionality ready or being implemented.
  Check out COBREXA.jl docs and examples!
- If you make a useful wrapper, you can easily share it with the community.
- Try a HPC (perhaps one from EuroHPC's) for a really huge analysis.

THANK YOU!

SYSTEMATISING COMPLEX AND COMBINED METABOLIC ANALYSES WITH COBREXA.JL

MIREK KRATOCHVÍL (UNI.LU), DANIEL THOMAS LOPEZ (EMBL EBI)

Nov 15TH 2022

HPC/Exascale Centre of Excellence in Personalised Medicine

www.permedcoe.eu

Follow us in social media:

linkedin.com/company/permedcoe

@permedcoe

The PerMedCoE project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement Nº951773