Practical: Identification of metabolic pathways involved in AMR in *E.coli* using GSMM

Pareena Verma

September 19, 2024

In this practical, we will investigate the GSM of *E.coli*. We will integrate the experimental data (as discussed in the presentation) into GSM and indentify reactions, hence pathways involved in resistance against antibiotics.

Note: DO NOT COPY PASTE THE CODE AND REMEMBER PYTHON IS A CASE-SENSITIVE LANGUAGE.

1. Get the model.

   cd into your working directory and download the archive containing the model and extract the files by this command on the terminal :

   ```
   $ tar -xvf Practical_PV.tar
   ```

2. Open the model.

   Start ScrumPy in the folder containing the .spy files and load the top-level model file Model.spy to create a model object. Note that the model is created in a modular fashion, and the top-level file will load the different components of the model.

   ```
   >>> m = ScrumPy.Model("Model.spy")
   ```

3. Check the model (Beginner)

   (a) Examine how the model is created. How many modules are loaded? Can you tell what each module contains?

   (b) How many reactions and metabolites are present?

   (c) How many media transporters are present in the model? (Hint: all media transporters have suffix "_mc_tx".)

   (d) How many biomass transporters are present in the model? (Hint: all biomass transporters have suffix "_bm_tx".)

4. Check the model (Expert)

   Check your model for energy and mass conservation. BuildLP is a module for model curation, which has the functions to check energy and mass conservation.

   (a) Import the module named BuildLP using this command:

   ```
   >>> import BuildLP
   ```

   (b) Check for energy consistency by:

   ```
   >>> BuildLP.ATPaseCheckLP(m)
   >>> BuildLP.NADPHCheckLP(m)
   >>> BuildLP.NADHCheckLP(m)
   ```

   You should get the result as "undefined", which means that model is not generating energy out of nothing.

   (c) Check for biomass formation by:

   ```
   >>> BuildLP.CheckBiomass(m)
   ```

   You should get the result as "optimal", which means that model is able to form all biomass components.

(d) Check number of reactions involved with biomass formation. Read the documentation about LP here : `https://mudshark.brookes.ac.uk/ScrumPy/Doc/LinProg`. Exercise 6 is complementary to the documentation here but for the purpose of this exercise, generate lp object as follows because BuildLP generates lp in the background of biomass formation.

```
>>> lp = BuildLP.BuildLP(m)
>>> lp.Solve()
>>> sol = lp.GetPrimSol()
>>> len(sol)
```

5. Let's find some drug targets.

(a) Import the module named "Gene2Reac" by using command:

```
>>> from Analysis import Gene2Reac
```

(b) Extract the reactions by:

```
>>> reacs = Gene2Reac.ExtractReacs()
```

How many reactions are present?

(c) Extract the metabolic reactions which are present in the model by:

```
>>> model_reacs = Gene2Reac.ModelReacs(m, reacs)
```

Check the difference between number of reacs and model_reacs. Remember from the presentation, not all genes are associated with metabolism, therefore model_reacs will have only the metabolic reactions present in the model.

(d) Knockout analysis : Now, we will check the impact of knocking out these reactions one by one in the model by:

```
>>> Gene2Reac.KnockOut(m, model_reacs)
```

Check the results and compare with the slides. Notice the reactions with optimal solution and remember from elementary mode analysis part of the lecture. See next exercise for detailed explanation.

6. Knock out two reactions associated with one gene and find out if the biomass component is being formed or not.

   (a) Generate lp object by:

   ```
   >>> lp = m.GetLP()
   ```

   (b) Set objective as minimisation of total flux by:

   ```
   >>> lp.SetObjective(m.sm.cnames)
   ```

   (c) Set fixed flux for production of 1 unit of LPS, i.e., flux of -1.

   ```
   >>> lp.SetFixedFlux({'lps_bm_tx':-1})
   ```

   (d) Knockout the first reaction associated with the gene (refer to solution from previous problem or slides from the lecture for the name of reactions and replace 'reac1' with actual name.)

   ```
   >>> lp.SetFixedFlux({'reac1':0})
   ```

   (e) Solve the lp. You should get a solution, i.e., optimal.

   ```
   >>> lp.Solve()
   ```

   (f) Knockout the second reaction, in the same manner as first.
   Note : This is the *in silico* knockout of one gene as you are knocking out all reactions associated with that gene.

   ```
   >>> lp.SetFixedFlux({'reac2':0})
   ```

(g) Solve the lp again. If you don't get a solution, it means this gene is essential for production of the biomass component.

```
>>> lp.Solve()
```