

# Supplementary Material

## Efficient enzyme coupling algorithms identify functional pathways in genome-scale metabolic models.

Dikshant Pradhan<sup>1</sup>, Jason A. Papin<sup>2</sup>, and Paul A. Jensen<sup>1\*</sup>

<sup>1</sup>Department of Bioengineering and Carl R. Woese Institute for Genomic Biology,  
University of Illinois at Urbana-Champaign, Urbana, IL, USA

<sup>2</sup>Department of Biomedical Engineering and Division of Infectious Disease and International  
Health, University of Virginia School of Medicine, Charlottesville, VA, USA

---

\*Correspondence to [pjens@illinois.edu](mailto:pjens@illinois.edu)

**Table S1.** Caching reduces the number of optimizations and runtime when identifying coupled enzyme sets. "Brute force" is the number of optimizations required to test all pairs without the shortcuts in FCF. "cachedFCF" uses only local caching. "cachedFCF + correlation" uses local and global caching.

| Model                | Optimizations |                      |                    |                         | Runtime [minutes] |           |                         |
|----------------------|---------------|----------------------|--------------------|-------------------------|-------------------|-----------|-------------------------|
|                      | Brute Force   | FCF                  | cachedFCF          | cachedFCF + correlation | FCF               | cachedFCF | cachedFCF + correlation |
| <i>P. aeruginosa</i> | 1,010,331     | 630,213<br>(62.4%)   | 385,690<br>(38.1%) | 2,285<br>(0.2%)         | 304.1             | 176.1     | 4.3                     |
| <i>S. mutans</i>     | 193,131       | 115,283<br>(59.7%)   | 44,599<br>(23.1%)  | 1,096<br>(0.6%)         | 10.8              | 0.8       | 0.7                     |
| <i>S. cerevisiae</i> | 11,589,705    | 1,558,818<br>(13.4%) | 868,876<br>(7.5%)  | 4,594<br>(0.04%)        | 1,789.6           | 927.8     | 15.5                    |

**Data:** MILP COBRA model

**Result:** Boolean matrix indicating coupling between variables

$N_{\text{vars}}$  = number of variables in model;

$c_{\text{thresh}}$  = threshold for correlation between variables;

$x_{\text{max}}^{\text{global}}[N_{\text{vars}}]$  = maximum observed values for each variable. Initialized to 0;

$x_{\text{min}}^{\text{global}}[N_{\text{vars}}]$  = minimum observed values for each variable. Initialized to 0;

$x_{\text{max}}^{\text{local}}[N_{\text{vars}}]$  = maximum observed values when a single variable is fixed. Initialized to Infinity;

$x_{\text{min}}^{\text{local}}[N_{\text{vars}}]$  = minimum observed values when a single variable is fixed. Initialized to -Infinity;

Cache[ $N_{\text{vars}}, N_{\text{cache}}$ ] = matrix of cached solutions. Initialized to 0;

**Routine** update(solution[ $N_{\text{vars}}$ ]) :

```

for  $i \in 1 : N_{\text{vars}}$  do
     $x_{\text{min}}^{\text{global}}[i] = \min\{x_{\text{min}}^{\text{global}}[i], \text{solution}[i]\};$ 
     $x_{\text{max}}^{\text{global}}[i] = \max\{x_{\text{max}}^{\text{global}}[i], \text{solution}[i]\};$ 
     $x_{\text{min}}^{\text{local}}[i] = \min\{x_{\text{min}}^{\text{local}}[i], \text{solution}[i]\};$ 
     $x_{\text{max}}^{\text{local}}[i] = \max\{x_{\text{max}}^{\text{local}}[i], \text{solution}[i]\};$ 
    if Cache is full then
        | add solution to Cache;
    else
        | replace oldest Cache entry with new solution;

```

Coupled[ $N_{\text{vars}}, N_{\text{vars}}$ ] = coupling matrix. If Coupled[ $i, j$ ] is true, variables  $i$  and  $j$  are coupled;

isBlocked[ $N_{\text{vars}}$ ] = vector identifying blocked variables. Initialized to false;

isCoupled[ $N_{\text{vars}}$ ] = vector identifying variables that are not yet coupled. Initialized to false;

```

for  $i$  in model.variables do
    reinitialize  $x_{\text{min}}^{\text{local}}$  and  $x_{\text{max}}^{\text{local}}$ ;
    if model.ub[ $i$ ]  $\neq 0$  then
        | solution  $\leftarrow$  maximize variable  $i$ ;
        | update(solution);
    if model.lb[ $i$ ]  $\neq 0$  then
        | solution  $\leftarrow$  minimize variable  $i$ ;
        | update(solution);
    if  $x_{\text{max}}^{\text{global}}[i] = x_{\text{min}}^{\text{global}}[i] = 0$  then
        | isBlocked[ $i$ ] = true;
    else
        | Coupled[ $i, i$ ] = true;
    isCoupled[ $i$ ] = false;
    choose  $\epsilon \neq 0 \in (x_{\text{min}}^{\text{global}}[i], x_{\text{max}}^{\text{global}}[i])$ ;
    fix variable  $i$  at  $\epsilon$  (model.lb[ $i$ ] =  $\epsilon$  and model.ub[ $i$ ] =  $\epsilon$ );
    for  $j$  in model.variables[ $i + 1$ :end] do
        | if isBlocked[ $j$ ] or isCoupled[ $j$ ] then next  $j$ ;
        | if  $x_{\text{max}}^{\text{local}} \neq x_{\text{min}}^{\text{local}}$  then next  $j$ ;
        | if correlation(Cache[:, $i$ ], Cache[:, $j$ ])  $\leq c_{\text{thresh}}$  then next  $j$ ;
        | solution  $\leftarrow$  maximize variable  $j$ ;
        | update(solution);
        | if  $x_{\text{max}}^{\text{local}} = x_{\text{min}}^{\text{local}}$  then
            | solution  $\leftarrow$  minimize variable  $j$ ;
            | update(solution);
        | if  $x_{\text{max}}^{\text{local}} = x_{\text{min}}^{\text{local}}$  then
            | Coupled[ $i, j$ ] = true;
            | isCoupled[ $j$ ] = true;
    reset model.lb[ $i$ ] and model.ub[ $i$ ];

```

return Coupled;

**Algorithm 1:** Cached Flux Coupling Finder (cachedFCF).

## Converting COBRA models into FALCON models.

We define three classes of reactions based on the degeneracy of their enzyme associations.

- **Class Ia** reactions are associated with enzymes that are exclusive to the reaction – the enzymes are not associated with any other reaction. Additionally, the DNF representation of the GPR rule has only a single clause (see below). This is equivalent to having no *or*'s in the GPR rule.
- **Class Ib** reactions also associated with enzymes that are exclusive to the reaction. Unlike Class Ia reactions, the DNF representation of the GPR rules for Class Ib reactions can contain multiple clauses.
- **Class II** are associated with enzymes that are also associated with another (Class II) reaction.

Adding enzyme associations requires the boolean GPR rules be factored into the disjunctive normal form (DNF). A DNF rule is a series of one or more *clauses* joined by *or*'s. Each clause contains one or more enzyme joined by *and*'s. For example, the GPR rule

$$e_1 \text{ and } (e_2 \text{ or } e_3)$$

is not a DNF rule. It can be factored into the equivalent DNF rule

$$(e_1 \text{ and } e_2) \text{ or } (e_1 \text{ and } e_3)$$

For GPR rules, each clause in the DNF represents a set of enzymes that are sufficient to catalyze the enzyme. FALCON creates a separate reaction for each clause in the DNF. These reactions are linked to enforce mass balance constraints and prevent futile cycles. Algorithm 2 presents the complete FALCON method for linking enzyme activities to reactions in COBRA models.

**Data:** LP COBRA model

**Result:** MILP FALCON model

**Routine** coupleEnzymesToReaction(reaction, enzymes, reversible) :

```
┌   if reversible then
├     ┌ add enzymes as pseudo-reactants to reaction;
├     └
├   else
├     ┌ split reaction into forward and reverse reactions;
├     └ add enzymes as pseudo-reactants to both reactions;
└
```

**Routine** addEnzymeClausesToReaction(reaction, clauses, reversible, split) :

```
┌   add new metabolite "rxnActivity" to model;
├   for clause in clauses do
├     ┌ if split then
├     │   ┌ add two reactions (forward and reverse) that consume enzymes in clause and produce rxnActivity;
├     │   └
├     │   else
├     │   ┌ add one reversible reaction that consumes enzymes in clause and produces rxnActivity;
├     │   └ addEnzymesToReaction(reaction, enzymes=rxnActivity, reversible=reversible);
├     └
└   add indicator variable so only forward or reverse reactions can carry flux not both;
```

**for** reaction in Class Ia **do**

```
┌   addEnzymesToReaction(reaction, enzymes(reaction), split=false);
```

**for** reaction in Class Ib **do**

```
┌   addEnzymesClausesToReaction(reaction, DNF(enzymes(reaction)), reversible=true, split=true);
```

**for** reaction in Class II **do**

```
┌   addEnzymesClausesToReaction(reaction, DNF(enzymes(reaction)), reversible=false, split=true);
```

**Algorithm 2:** Creating a FALCON model.