

gcFront Documentation

gcFront: a tool for determining a Pareto front of growth-coupled cell factory designs

Laurence Legon^{1,2}, Christophe Corre², Declan G. Bates^{1,*} and Ahmad A. Mannan^{1,*}

¹ Warwick Integrative Synthetic Biology Centre, School of Engineering, University of Warwick, Coventry, UK

² Warwick Integrative Synthetic Biology Centre, School of Life Sciences, University of Warwick, Coventry, UK

* Corresponding authors: ahmad.mannan@warwick.ac.uk, d.bates@warwick.ac.uk

Table of Contents

| | |
|---|---|
| A. PRELIMINARIES, DEPENDENCIES, AND INSTALLATION..... | 2 |
| B. OVERVIEW OF THE GCFRONT ALGORITHM..... | 2 |
| (I) INPUTS..... | 2 |
| (II) PRE-PROCESSING..... | 5 |
| (III) GCFRONT SOLVING THE MULTI-OBJECTIVE OPTIMIZATION PROBLEM..... | 5 |
| (IV) POST-PROCESSING AND OUTPUT OF DESIGNS | 7 |
| REFERENCES..... | 7 |

A. Preliminaries, dependencies, and installation

We developed gcFront as a toolbox for use in MATLAB, but it is dependent on the following:

- MATLAB's Global Optimization toolbox for solving the multiobjective optimization problem.
- COBRA toolbox (Heirendt *et al.*, 2019) for the import and analysis of a COBRA-compatible GSM in SBML/XML format, such as those of the BiGGs database (King *et al.*, 2016). The full COBRA toolbox package can be installed from GitHub here: <https://github.com/opencobra/cobratoolbox>, and documentation and installation instructions can be found here: <https://opencobra.github.io/cobratoolbox/stable/installation.html>, and at their webpage here: <https://opencobra.github.io/cobratoolbox/stable/>.
- An LP solver for analysis of the GSM models, including flux balance analysis, flux variability analysis, and to enable gcFront to produce production envelopes, such as glpk or Gurobi (Gurobi Optimization, 2021) (recommended and free with an academic licence) (downloadable from here <https://www.gurobi.com/>). This is a requirement of COBRA toolbox anyway, so any LP that successfully functions for the COBRA toolbox solver will also work for gcFront. The help guide for installation and testing of LP solvers with COBRA can be found in the section titled "Solver installation" in the COBRA installation webpage here: <https://opencobra.github.io/cobratoolbox/stable/installation.html>.
- gcFront was developed with MATLAB 2017A/2021A, COBRA 3.1 and Gurobi 8.1.1. Other versions of these software packages and other linear algebra solvers should still function, but have not been extensively tested.

Once gcFront is downloaded from GitHub, it is to be run using MATLAB. There are two approaches to run gcFront:

1. Change MATLAB's current directory to the gcFront folder downloaded and run it by typing "gcFront" into the command window.
2. Alternatively, add the gcFront folder to your MATLAB path, then you can call the function "gcFront" from within your own script or command window. The algorithm's inputs can be supplied by following the prompt windows that appear or can be supplied as arguments to the function.

B. Overview of the gcFront algorithm

See Fig. 1 illustrating the gcFront workflow. gcFront is composed of four key steps: inputs, pre-processing of GSM, solving the multiobjective optimization problem, post-processing, and output of designs.

(i) Inputs

The gcFront function: [designTable, reducedModel, options] = gcFront(model, prodTarget, options).

Inputs:

- model = a genome-scale constraint-based model compatible with the MATLAB COBRA toolbox (XML/SBML). The model can be imported directly as a .mat file, or as a character array that specifies the file path to the model. If this input is left blank, users can browse to select the file from which the model should be loaded.
- prodTarget = Character array of the reaction or product/metabolite name in the GSM to be growth- coupled and overproduced. If a metabolite is specified, the target will be set to its exchange reaction.
- options = A MATLAB data structure with fields of the optional input parameter names, of the form "options.Parametername". The parameter names and default values are detailed in Table 1 below, but values of the parameters specified by the user will override the defaults. The order in which they are specified does not matter.

Outputs:

- designTable = a table of all Pareto optimal designs, including a column of the KO combination suggested, performance metrics (growth rate, minimum synthesis flux, coupling strength), and Euclidean distance to the ideal point normalized to maximum values to each objective value.
- reducedModel = MATLAB data structure of the COBRA GSM as reduced in the pre-processing.
- options = list of the values of the optional inputs used for this search.

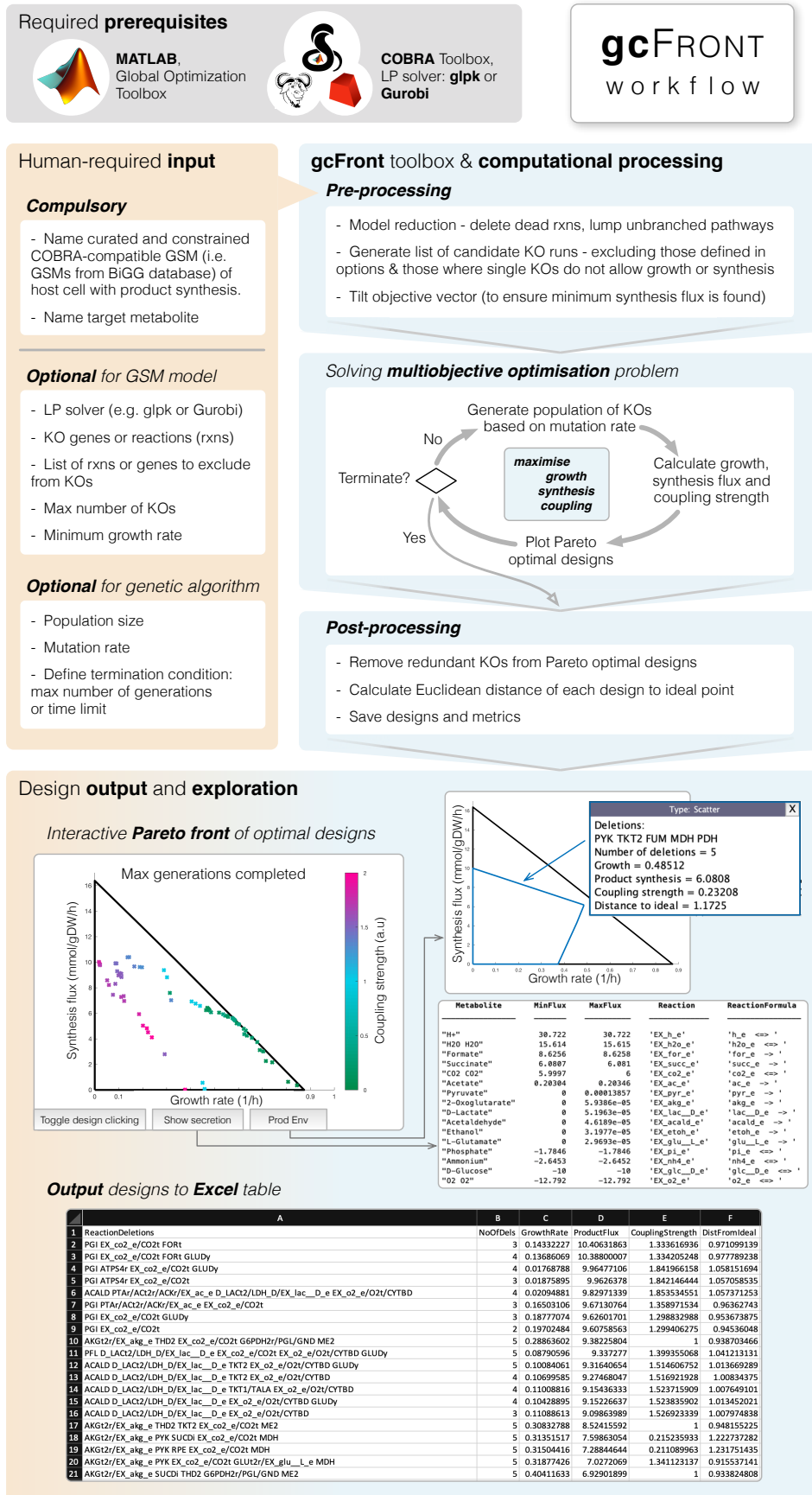


Fig. 1. The gcFRONT workflow. A flow chart detailing (i) the required prerequisites for gcFRONT to run (gray box); (ii) the human required input of the COBRA-compatible and defined SBML model of the GSM of the cell host and product synthesis of interest, and options to adjust default parameters concerning GSM model and genetic algorithm (left orange box); (iii) what gcFRONT does “under-the-hood” (blue boxes), i.e. model reduction, iterative search for Pareto front of gc-designs that maximise growth, product synthesis flux, and coupling strength; and (iv) snapshots of the interactive plot of designs and performance metrics or option to save to Excel table (bottom orange box).

Table 1. Optional inputs. A list of all the optional inputs, with default values defined.

| Parameter name | Description | Default | Format |
|---------------------|--|--|---|
| solver | The LP solver used to carry out FBA | Current COBRA toolbox LP solver. If none, the solver set by “initCobraToolbox” is used | Char, such as ‘glpk’ or ‘gurobi’, for example |
| biomassrxn | The name of the pseudoreaction that represents cell growth | Reaction with highest coefficient in objective vector | Char |
| tol | Tolerance to mathematical errors. Differences in flux smaller than this will be ignored | 10^{-8} | Positive double |
| tiltval | Coefficient used for objective vector tilting (so minimum product synthesis is identified) | 10^{-4} | Positive double |
| shiftval | Size of the change in growth rate that is used when manually calculating shadow price. | 10^{-5} | Positive double |
| skipreduction | Parameter that controls whether to reduce model size by removing inactive reactions and pooling linear pathways | False | Logical |
| mingrowth | Minimum growth threshold- designs with growth below this are ignored | 10^{-3} | Positive double |
| minprod | Minimum product threshold- any deletion that lowers product synthesis will not be considered when finding designs | 10^{-3} | Positive double |
| removedundancy | Parameter that controls whether designs should be tested to see if they contain redundant deletions that do not contribute to the design | True | Logical |
| saveresults | Defining if designs should be saved and exported to a csv file, and parameters are saved as .mat on termination | True | Logical |
| newredundantremoval | Parameter that controls whether redundant KOs are removed using the old method (as was used to generate data for this paper) or using a new, more efficient methodology. | True | Logical |
| maxreductionsizes | Maximum number of KOs in designs that will have redundant KOs removed from them (helps avoid reduction of huge designs, which can take a long time). Not compatible with the old reduction method. | 15 | Positive double |
| maxknockouts | Maximum number of knockouts that a design may have | Infinite | Positive integer double |
| deletegenes | Parameter that controls whether the algorithm searches for gene or reaction knockouts | False | Logical |
| ignorelistrxns | List of reactions that should be knocked out | Empty vector | Cell (contains characters) |
| ignorelistgenes | List of genes that should not be KO’d. If reaction KOs have been selected, reactions that can only be KO’d if these genes are KO’d are excluded from consideration | Empty vector | Cell (contains characters) |
| dontkoess | Parameter that controls if reaction KOs are ignored if they cannot be KO’d without a KO of a gene that is essential <i>in silico</i> . | True | Logical |
| onlykogeneassoc | Parameter that controls if reactions are only considered if they are gene associated | True | Logical |
| mutationrate | Average number of changes to a design during the GA mutation step | 1 | Positive double |

| | | | |
|-------------------|---|---|-------------------------|
| popsiz | Size of the GA population | 200 | Positive integer double |
| genlimit | Maximum number of generations that the GA will be run for | 10000 | Positive integer double |
| timelimit | Maximum number of seconds that the GA will be run for | 86400 (i.e. 1 day) | Positive double |
| fitnesslimit | GA will terminate if a design is found with a product synthesis that exceeds this value | Infinite | Double |
| spreadchangelimit | How low the change in the spread of designs must be before the algorithm starts to terminate. See MATLAB's 'gamultiobj' documentation for more details. | 10^{-4} | Positive double |
| stallgenlimit | How many generations the change in spread of the designs must be below spreadchangelimit before the algorithm is terminated | Same as genlimit (i.e. this will not lead to termination) | Positive integer |
| plotinterval | Number of generations that must pass before the plot of the current designs is updated | 1 | Positive integer |

(ii) Pre-processing

- First, all reactions in the model are subject to flux variability analysis (FVA) to see if they can carry non-zero flux, and are removed if they do not, i.e. are dead reactions.
- The algorithm then searches for reactions that form unbranched pathways and pools them into a composite reaction that represents flux through that pathway.
- If the algorithm has been set to search for gene knockouts, then the gene-reaction rules of the model are analysed to find sets of gene knockouts that are functionally equivalent, and to find sets of gene knockouts that can only have an effect if they are knocked out simultaneously. These genes are then pooled into their sets to reduce the size of the gene search space.
- Essential reactions are identified by determining single gene/reaction knockouts that reduce maximum growth rate or product synthesis flux below a user-defined minimum threshold ("mingrowth" parameter in Table 1).
- A list of potential knockouts is then constructed, excluding essential, pooled, and dead reactions. Further reactions can also be excluded if the vector of cells listing those is added as the optional input parameter "ignorelistrxns", such as non-gene-associated reactions (including exchange reactions) or reactions that are disabled if an essential gene is knocked out, for example.
- Finally, the model objective vector (usually *modelname.c* in the COBRA model data structure) is "tilted", similar to (Feist *et al.*, 2010). Tilting is done by setting the value of the element of the c vector corresponding to the biomass reaction (growth rate) to +1, and the value of the element corresponding to the target reaction to a very small negative value ($-1 \cdot \text{tiltval}$ in Table 1, e.g. -10^{-4}). This ensures that the product synthesis rate found by flux balance analysis that maximises growth rate is the minimum product synthesis rate.

(iii) gcFront solving the multi-objective optimization problem

Once executed, gcFront then moves to solve the multiobjective optimisation problem defined as follows, using MATLAB's Global Optimization toolbox function *gamultiobj*:

$$\begin{aligned}
 & \max_{\underline{r}} (J_1, J_2, J_3), \\
 & J_1 = \lambda; \quad J_2 = r_p; \quad J_3 = c_s, \\
 & \text{subject to} \\
 & S \cdot \underline{r} = \underline{0}, \\
 & \underline{k} \circ \underline{\text{lb}} \leq \underline{r} \leq \underline{k} \circ \underline{\text{ub}}, \text{ where } \sum_{i=1}^R k_i \geq R - I, \quad k_i \in \{0,1\},
 \end{aligned}$$

for vector \underline{r} of R reaction fluxes of the GSM, excluding essential reactions, the biomass reaction, exchange reactions defining the growth media, and other user-specified reactions. S is the stoichiometric matrix of the GSM (in reduced form after the preprocessing step), I is the maximum allowable number of simultaneous knockouts (not an identity matrix), and \underline{k} is a vector where $k_i = 0$ defines KO of the i^{th} reaction. This is implemented in the flux constraints as a Hadamard product of vector \underline{k} and the vectors of the reaction fluxes' lower and upper bounds, \underline{lb} and \underline{ub} . We solve this multiobjective optimization problem using the genetic algorithm function `gamultiobj`, from MATLAB's Global Optimization toolbox. Such metaheuristic methods have been shown to be less computationally demanding (Patil *et al.*, 2005; Rocha *et al.*, 2008), and although the solutions found cannot be guaranteed to be globally optimal this would have been computationally intractable anyway, given the massive search space. It is important to note that if searching KOs over genes rather than reactions, the logical statement of how genes are associated to each reaction and a gene-reaction mapping matrix must exist in the GSM to enable KOs of the respective reactions to be made.

The steps involved in solving the multiobjective optimisation problem are as follows:

- An initial random population is created of size as specified in the optional input parameter “popsize” (Table 1). Each member is defined by a different combination of KOs, as selected from the list of potential KOs generated during pre-processing, where KOs are implemented by fixing both the lower and upper bound flux constraints of those reactions to zero.
- Each member is then subject to flux balance analysis (FBA) and the three performance objectives are measured: growth rate, product synthesis flux, and coupling strength. Recall that the “tilted” objective vector determines both the maximum possible growth rate and minimum product synthesis flux possible at the maximum growth rate. The coupling strength (c_s) is calculated from the formula:

$$c_s = \begin{cases} c_s^{\text{nc}} = \frac{\lambda_{\max} - \max(\lambda(r_p = \delta))}{\delta}, & \text{when } \min(r_p(\lambda_{\max})) = 0, \\ c_s^{\text{gc}} = \frac{\lambda_{\max} - \max(\lambda(r_p = 0))}{\lambda_{\max}} + \min\left(1, \frac{\min(r_p(\lambda = 0))}{\min(r_p(\lambda_{\max}))}\right), & \text{when } \min(r_p(\lambda_{\max})) > 0, \end{cases}$$

as discussed in detail in Supplementary Notes 2B of the main paper and illustrated in Fig. 2 below:

Concept of growth-synthesis coupling-strength

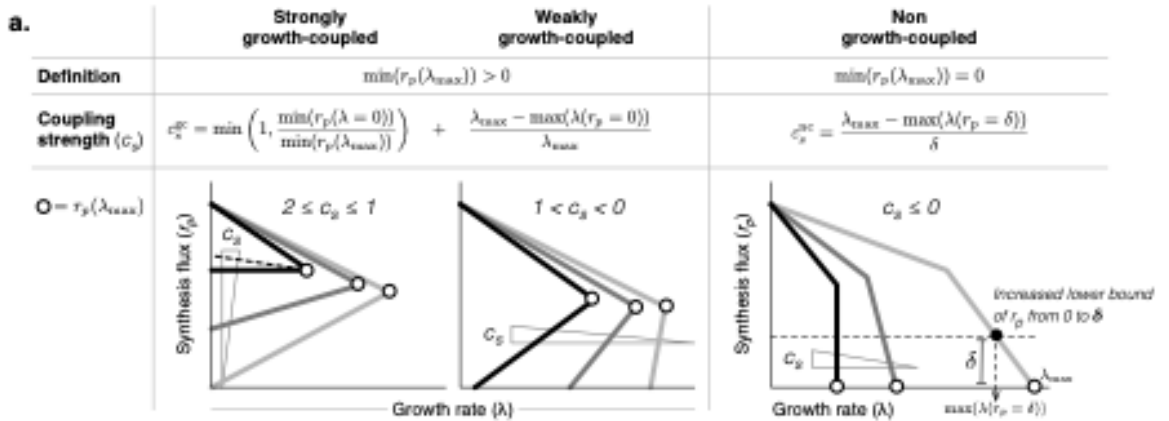


Fig. 2. Proposed measure of growth-coupling strength to ease navigation of the design space. (a). Illustration of the proposed coupling strength (c_s) measure, and how it is calculated from the production envelope, for designs (from right to left) that are non-, weakly-, and strongly-growth-coupled.

- Each population member is then ranked based on their dominance (i.e. if they have an equal or better score for every metric), and gcFront plots the current population on the growth-synthesis plane, where each design is coloured by coupling strength. The production envelope of the parent model (i.e. with no deletions) is also plotted to help visualise how the growth and product synthesis of each design compares to that which is theoretically possible of the wild-type organism. If, however, no coupled designs have yet been discovered in the current generation, a scatter of growth rate vs coupling strength is plotted instead.

- A binary selection tourney is used to select designs of the population based on their rank and similarity to other designs, and those are then used to create new designs by mutation (adding/taking away random KOs) or crossover (taking random combinations of KOs from two parent designs).
- These steps are then repeated over many generations until the user specified termination condition is met (as defined in Table 1), the default being whichever of “genlimit” or “timelimit” is hit first. Other optional termination conditions that can be set include “fitness limit”, “spreadchangelimit”, or “stallgenlimit”.

(iv) *Post-processing and output of designs*

- Post-processing: On termination, the non-dominated gc-designs that make up the Pareto front are listed. Some of the optimal designs found may include KOs that are not contributing to performance. If desired (via optional input “removedundancy”), gcFront removes such redundancy by testing each KO that makes up a design to see if the same growth, product synthesis and coupling strength can be achieved without it, removing the KO if it is not contributing to performance, and then repeating the process until no single deletion can be removed without moving the design off the Pareto front.. As a simple metric to identify designs that lie in an optimal trade-off region, gcFront also measures the Euclidean distance (D) of the point of each design (defined by the three performance objective values) to the ideal point, after normalising it to the ideal point, i.e.

$$D = \sqrt{\sum_{i=1}^3 \left(1 - \frac{\text{obj}_i^{\text{design}}}{\text{obj}_i^{\text{ideal}}}\right)^2}$$
. The ideal point lies at the point of maximum growth and maximum product synthesis rates, as calculated from FBA of the model with no deletions (the parent strain), and at the maximum possible coupling strength (i.e. 2).

- Output: All Pareto optimal gc-designs are then outputted in three formats simultaneously:
 - As a table onto the command window, listing all Pareto optimal designs, along with their KOs, the three performance measures, and distance to the ideal point.
 - As a table, exported to a .csv file, if the user specifies via optional input “saveresults”.
 - As an interactive plot on the growth-product synthesis flux plane, with variations in coupling strength illustrated as the variation in colors of designs (dots), defined by the colorbar to the right of the plot. This plot helps visualise the inherent trade-offs between the objectives. Since this is an interactive plot, each design can be clicked to display their set of proposed KOs and performance metrics. Tabs at the bottom of the window also enable one to plot the production envelope of the respective design and a table of other side-products predicted to be secreted for that design.

For an example work-through, the users are referred to the tutorial document (also available in Supplementary Note 3 of the main paper).

REFERENCES

-
- Feist,A.M. *et al.* (2010) Model-driven evaluation of the production potential for growth-coupled products of *Escherichia coli*. *Metab. Eng.*, **12**, 173–186.
- Gurobi Optimization,L. (2021) Gurobi Optimizer Reference Manual.
- Heirendt,L. *et al.* (2019) Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0. *Nat. Protoc.*, **14**, 639–702.
- King,Z.A. *et al.* (2016) BiGG Models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic Acids Res.*, **44**, D515–D522.
- Patil,K.R. *et al.* (2005) Evolutionary programming as a platform for in silico metabolic engineering. *BMC Bioinformatics*, **6**, 1–12.
- Rocha,M. *et al.* (2008) Natural computation meta-heuristics for the in silico optimization of microbial strains. *BMC Bioinformatics*, **9**, 1–16.