Minimal media design: biology, algorithms and implementation

Problem definition, modelling frameworks and software

Once a genome scale reconstruction was constructed the question arises how can it be used to generate new hypothesis or enhance particular aspects of a micro-organism in its environment. In this section we pose the question: "is it possible to design a minimal media that will enable a specific yield?" In this context we define growth as biomass yield per unit substrate uptake and a minimal media as the simplest set of substrates necessary to achieve a target yield.

We use is constraint based modelling (CBM) as our modelling framework and in the discussion that follows a media component is represented by an uptake flux e.g. if substrate X is required for growth in the predicted media, this is modelled as a minimal required flux through the X transporter. In all examples that follow the GSR model has been set-up such that the optimal biomass yield (reaction *R_BP_biomass_130704*) is set equal to one. In addition transport reactions, typically modelled as reversible, are split into individual irreversible import and export reactions. This allows the specific targeting of import reactions which only carry a positive flux.

All modelling and algorithms described here were developed and implemented using the Open Source software PySCeS CBMPy 0.7.0 using IBM ILOG CPLEX™ 12.5.1 for optimization. CBMPy is a Python based modelling environment and the scripts used to generate the results shown below are available on request (???ref: PySCeS, CBMPy???). The GSR model is available in the Systems Biology Markup Language Level 3 with Flux Balance Constraints (SBML3FBC) and will be made available on request and in the Biomodels Database (???ref: SBML, SBML3FBC???).

Defining a minimal media

As mentioned earlier a minimal media is one that contains fewest components, however, in order to model this we need to be able to quantify "fewer". One classical approach, generally applied to internal fluxes, is that of "flux minimization" which assumes that a cell has to produce more protein (enzyme) in order to carry a larger flux through a reaction. Therefore for a cell to be efficient, for any particular optimal objective function (e.g. max growth), there exists a solution which minimizes the maximal flux of each reaction needed to achieve that optima. Practically, this is implemented so that the sum of absolute fluxes in minimized which allows this to be applied to reversible reactions without bias towards the reverse reaction. Generally flux minimizations are relatively simple to calculate using a linear program (LP) as described in Equation 1 and is implemented in most CBM/FBA software.

Modifying this method slightly to only minimize a selected set of transporters (???200??? potential uptake reactions) does result in a minimal media, however, it is observed that the algorithm resulted in a significant number of transporters each carrying a small flux. If one considers that, in general, transporters are large, multimeric protein complexes and are expressed in relative low numbers (relative to metabolic enzymes) that are not easily saturated with substrate, then it seems highly unlikely that a "large number of low flux carrying transporters" strategy would be adopted by the cell.

As an alternative we propose the hypothesis that, with respect to transport reactions, it is more likely that the cell minimizes the number of active transporters used to achieve a specific objective rather than the combined flux passing through them. A minimal media is therefore defined as a media which supports the minimum number of active transporters (uptake) needed to satisfy a particular objective, in this case biomass production.

Media design: minimizing the number of active transporters

The question is then: "can we algorithmically generate such a minimal media?" In order to compute this we make use of a mixed integer linear program (MILP) based on the standard FBA LP formulation (constraints and flux bounds) but in addition introduce binary variables to control whether a flux is on (a non-zero flux) or off and a constraint that fixes the objective to a specific target value. This MILP is described in Equation 2 and is analogous to an approach used in (???ref: segre???).

$$\text{minimize: } \sum_{1}^{i} abs(J_i)$$

such that:

$$NJ = 0$$

$$J_{opt} \geq optarg$$

$$lb_n \leq J_n \leq ub_n$$

Equation 2: LP minimizing the sum of absolute fluxes. J_n represents the model fluxes, *N* the stoichiometric matrix, J_{opt} is the FBA objective function flux, *optarg* is the target objective function value, *lb* and *ub* are constants representing the lower and upper bounds of each flux. The objective function minimizes the sum of the absolute value of each of the target fluxes, in this case the set of fluxes represented by J_i

$$\text{minimize: } \sum_{1}^{i} z_i$$

such that:

$$NJ = 0$$

$$J_{opt} \geq optarg$$

$$z_i = 0 \rightarrow J_i = 0$$

$$lb_n \leq J_n \leq ub_n$$

$$z_i \in \{0 : 1\}$$

Equation 1: MILP minimizing the number of non-zero fluxes. J_n represents the variable model fluxes, *N* the stoichiometric matrix, J_{opt} is the FBA objective function flux, *optarg* is the target objective function value, *lb* and *ub* are constants representing the lower and upper bounds of each flux and *z* is a binary indicator variable which is zero if the constraint it represents is active. The objective function minimizes the number of non-zero indicator variables i.e. non-zero fluxes, in this case the set of fluxes represented by J_i

There are various ways of implementing the MILP shown in Equation 2, the conventional approach being to use a so called Big M formulation. However, Big M methods which can be sensitive to the values of the artificial parameters used to turn on or off a constraint and therefore may lead to problems involving numerical instability (???ref: ibm cplex???). By developing a custom implementation using the IBM ILOG CPLEX™ optimizer and formulating the problem using *indicator constraints*, it was possible to avoid an explicit Big M formulation thus exploiting CPLEX's ability to minimize these side-effects (???ref: cplex indicator constraints???).

This implementation proved to be both fast and efficient in calculating the a minimal media and answering our initial question: "can we generate a minimal media?" While a single solution provides the

minimal number of active fluxes as well as a single possible solution there may, of course, be many solutions (media) that satisfy this same optimality criteria (number of fluxes). Knowing as many as possible of these potential alternatives soon became of prime importance and led to the question, "is it possible to generate some, or all, minimal media combinations?"

In general, the enumeration of MILP solution spaces is a computational hard problem yet the ability to calculate multiple minimal media would be extremely useful. We pursued this using a dual strategy.

The first strategy leveraged CLPEX's extension to its branch-and-cut MILP algorithm that allows it to generate and store pools of alternate optimal solutions (???ref: cplex solution pools???). Through a judicious choice of solution pool parameters i.e. *solution gap*, *pool replacement strategy* and *search intensity* it is possible to use CPLEX to enumerate optimal solution spaces. However, depending on the structure and size of the problem, complete enumeration is not guaranteed (and should not be expected). Fully implemented in CBMPy this "Optimized MILP" method turned out to be fast and efficient with the results of a typical analysis shown in Table 1 where 6 different environments were enumerated. These results show the number of *media* generated and minimal *size* predicted for each environment as well as the total number of media generated and total time taken (from model load to data output) to analyse all environments.

	Optimized MILP		**CustomSearch**	
	media	*size*	*media*	*size*
AA	2	9	2	9
All	2	9	2	9
C	44	11	44	11
N	2	11	12	11
NC	2	9	2	9
S	13	10	14	10
Total media	65		76	
Total time (s)	85		275	

Table 1: Comparing the performance of two minimal media generating algorithms using different nutrient sets. ??? (A), ??? (All), ??? (C), ??? (N), ??? (NC) and ??? (s). m*edia* is the number of minimal media generated set using each method for each nutrient. Minimal s*ize* is the minimal number of components required, out of a potential target 200, to achieve a biomass yield of 1.0. All simulations performed using CBMPy 0.7.1 with IBM CPLEX 12.5.1 running on Microsoft Windows 7 with an Intel i7-2720QM 2.2 GHz quad core CPU and 8 GB RAM.

while the Optimize MILP approach efficiently generated, at least, some of the solutions it was clear that in certain cases (e.g. environment N) that certain biologically relevant and mathematical feasible yet difficult to find solutions were not being found. It is significant that even detecting such omissions requires an in depth understanding of the underlying microbial biochemistry. This lead to the subsequent development of a second strategy.

In formulating our second strategy we were able to use the information obtained using strategy one to develop our own media enumeration algorithm. Critically, this includes knowing the minimum number of components in a minimal media as well as an initial solution. By limiting the scope of our search only to combinations of transporters of a specific length, the search space is reduced so as to become

computationally feasible. The algorithm named CustomSearch is essentially composed of three parts each represented as pseudo-code in Figure 1.

```
function Subsearch(model, target, ignore):
  optimize(model)
  if model is infeasible:
    return None
  minimize_active_flux_set(model, target)
  if number of active fluxes is greater than global minimum:
    return None
  select testable non-zero fluxes in target not in ignore
  if testable is empty:
    return None
  for each flux in testable:
    set flux bounds equal to zero
    for each remaining flux in testable:
      set flux lower bound to a small non-zero value
      add flux to new ignore list
    Subsearch(model, target, new ignore)
    add Subsearch results to output
  return output

function CustomSearch(model):
  initialise model exchange reactions
  initialise minimization target reaction set
  minimize_active_flux_set(model, target)
  minimum number of active fluxes stored as global minimum
  flux_variability_analysis(model) on target reactions
  ignore required reactions with FVA minimum greater than zero
  Subsearch(model, target, required)
  return medias

function ConstructMedia(model, ignore, media):
  set target reaction bounds to zero
  set required reaction upper bound to infinity
  for each flux in media:
    set flux upper bound to infinity
    minimize_sum_absolute_fluxes(model)
    store target flux values as media and add to minimal media
  return minimal media
```

Figure 1: Pseudo-code representation of CustomSearch minimal media generating algorithm. A targeted, recursive media search algorithm is implemented in three parts. A *CustomSearch* driver method that initialises the model, determines the initial search parameters and initiates a recursive *Subsearch* which through a process of elimination determines minimal viable media combinations. Finally in *ConstructMedia* minimal media combinations are unrolled and quantified to construct minimal media sets. A full implementation of this algorithm is implemented CBMPy 0.7.1

To begin with the *CustomSearch* initialises the model, sets up the external environment and performs both an active flux minimization and flux variability analysis (FVA) on the target reactions (uptake reactions). This provides the global minimum number of reaction (media components) as well as a set of required reactions defined as: reactions that have an FVA minimum greater than zero or in other words reactions that are required to carry a flux in all solutions. It then calls the *Subsearch* function recursively.

The *Subsearch* function is the core of the algorithm and takes as arguments a model instance, minimization target and list of reactions to ignore. Essentially *Subsearch* computes minimal number of active flux solutions and will exit if the model is infeasible or the number of active solutions is larger than the global minimum. It then finds the set of non-zero fluxes, as specified by the target list and not in the ignore list (and will exit if this set is empty) and labels them as testable.

For each flux in testable, clone the model, set its bounds to zero and all other fluxes in testable to a small positive number, add them to a new ignore list and call *Subsearch* recursively using the new model, target and ignore parameters. When *Subsearch* fails any of the tests mentioned earlier it returns either an empty solution or the testable flux it was evaluating (set to zero) which results in a breadth-first search of the minimal media solution space.

The final part of the algorithm *ConstructMedia* unrolls the recursive data structures and regenerates the qualitative media composition (in terms of components) it then uses this information coupled with a absolute flux minimization approach to quantify the minimal required flux required to be taken up to achieve the target objective. For this particular model, while not formally proved, this algorithm appears to enumerate all minimal media required to satisfy a target objective.

Final thoughts

In Table 1 we compare the results of the *CustomSearch* to those achieved by the *Optimized MILP* for a selection of different environments. While both algorithms correctly predicted the minimal length *Optimized MILP* generated 85% of the *CustomSearch* solutions in 33% of the time. While the total running time in both cases is in this case insignificant it does suggest a combined strategy may be the most effective with *Optimized MILP* being used for exploratory purposes and *CustomSearch* for full enumeration.

By combining these two algorithms we propose a comprehensive method for the design of minimal media for GSR steady-state models and we look forward to making this facility available to the broader constraint based modelling community.

References

PySCeS

B. G. Olivier, J. M. Rohwer, J.-H. S. Hofmeyr, Modelling cellular systems with PySCeS. *Bioinformatics* **21**, 560 (2005).

CBMPy

B. G. Olivier, PySCeS CBMPy. <*http://cbmpy.sourceforge.net*> (2012).

IBM CPLEX

IBM, IBM ILOG CPLEX™. <*http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/*> (2013).

SBML

M. Hucka *et al.*, The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* **19**, 524 (2003).

SBML3FBC

B. G. Olivier, F. T. Bergmann, Flux Balance Constraints Version 1 Release 1. <http://identifiers.org/combine.specifications/sbml.level-3.version-1.fbc.version-1.release-1> (2013).

IBM CPLEX Solution Pools

http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r5/index.jsp?topic=%2Filog.odms.cplex.help%2FCPLEX%2FUsrMan%2Ftopics%2Fdiscr_optim%2Fsoln_pool%2F01_soln_pool_title_synopsis.html

IBM CPLEX Indicator Constraints

http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r5/index.jsp?topic=%2Filog.odms.cplex.help%2FCPLEX%2FUsrMan%2Ftopics%2Fdiscr_optim%2Findicator_constr%2F01_indicators_title_synopsis.html

SEGRE

N. Klitgord, D. Segrè, Environments that induce synthetic microbial ecosystems. *PLoS computational biology* **6**, (2010).