AN ENUMERATIVE APPROACH TO COMPUTING CUT SETS IN METABOLIC NETWORKS

by

Daniel Salinas

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May, 2013

APPROVAL

of a thesis submitted by

Daniel Salinas

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to The Graduate School.

Dr. Brendan Mumey

Approved for the Department of Computer Science

Dr. John Paxton

Approved for The Graduate School

Dr. Ronald W. Larsen

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfullment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Daniel Salinas

May, 2013

DEDICATION

To those whose love and support have been lifelong: Mom and Dad.

ACKNOWLEDGEMENTS

I am grateful to my advisors, Dr. Jeff Heys and Dr. Brendan Mumey, who trusted me with their research, a privilige I was glad to accept.

I am also grateful to Prathish Rajamaran, whose company at the lab I would not have done without.

Funding Acknowledgment

This research was kindly supported by Montana State University's CoBRE Center for the Analysis of Cellular Mechanisms and Systems Biology.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF ALGORITHMS

Algorithm Page

ABSTRACT

The productivity of organisms used in biotechnology may be enhanced when certain parts of their metabolism are rendered inaccessible. This can be achieved with genetic modifications, but current techniques set a practical limit on number of modifications that can be applied. Taking advantage of this limit, we implement a brute force algorithm that can compute cut sets for any set of metabolites and reactions that is shown to perform better than alternative approaches. Also, an attempt is made to approximate a binary linear program with a quadratic program; this approximation is meant to be used when refining the growth model of organisms used in flux balance analysis. The approximation is shown to be less efficient that the original program. Finally, extensions to the brute force algorithm are proposed.

1

INTRODUCTION

This thesis presents work on metabolic networks, transforming them into hypergraphs to delve into the problem of generating cutsets, as well as using quadratic programming to generate flux vectors of low cost. The thesis begins by giving the reader the appropriate background, after which the algorithms developed will be presented with experimental results and relevant evaluation and discussion.

Context and Motivation

Metabolic networks are a set of interdependent biochemical reactions which, as part of an organism's metabolism, sequentially convert a set of initial metabolites into a set of final metabolites. They are the core model of several approaches to optimize the metabolism of an organism, usually bacteria, toward a desirable state. A desirable state is, in the most general sense, a profile of activity for all reactions in the network that achieves desirable properties. Usually this can be interpreted as a profile that has high activities for the reactions that produce desirable metabolites and corresponding low activities for reactions whose products are undesirable under a particular set of conditions.

Metabolic networks are a useful model because they abstract an organism's chemical processing capability to its intuitive components: chemical reactions and the metabolites they transform. This abstraction provides a framework to which well-known algorithms and theoretical results can be applied following a simple conversion into either hypergraphs or stoichiometric matrices.

Modeling metabolism as a hypergraph allows several theoretical results to be applied to problems in genetic engineering. Specifically, the problem of finding the minimal set of reactions whose removal would render the production of a target

metabolite (or the functioning of certain target reactions) impossible. *Removing a reaction from the network* often means deactivating a gene by means of a *gene knockout*, which can be complicated and costly.

Modern techniques set the practical amount of gene knockouts per organism to three genes [1]. Successfully avoiding the production of any target metabolite t that is produced by more than three reactions must therefore be done indirectly. That is, by removing reactions *upstream* of that metabolite, i.e. reactions whose products are required as reactants (or precursors to the reactants) of the reactions that produce t. As we will see, there are a variety of hypergraphs that can be derived from a metabolic network with corresponding theoretical questions to which this problem can be reduced.

While an organism's productive capabilities can be improved by modifying its genetic material, there are theoretical limits to its production, some of which may be elucidated via *flux balance analysis*. Performing flux balance analysis, from here on FBA, requires the reactions in the network to be put into the form,

$$r_i : \ c_1M_1 + c_2M_2 + ... = ... + c_{n-1}M_{n-1} + c_nM_n$$

where the left and right sides of the equation respectively represent the reactants and products of reaction r_i, and c_j is the stoichiometric coefficient for metabolite M_j. As will be discussed, a *stoichiometric matrix* is formed from the set of stoichiometric coefficients, and FBA is performed integrating the stoichiometric matrix as a constraint to a linear program, which, when solved, yields an optimal flux profile for the network. A flux profile can be interpreted as a vector of fluxes, where each flux in the vector corresponds to the relative rate of a reaction. For example, given a flux vector v = [1,2,4,0], we infer that r_1 occurs at one half the rate of r_2, at one fourth of the rate of r_3, and reaction r_4 does not occur at all. By setting the

linear program's objective as the flux for a single reaction, and setting the equality constraints from the stoichiometric matrix, flux balance analysis can be used to find the theoretical limit of a certain reaction's flux. This knowledge can be tested against modified networks (matrices) to ask questions about the network's capabilities with the addition or removal of certain reactions, *without* having to create the modified organisms in the lab.

Much richer analysis may be done by modifying the objective and constraints. This thesis attempts to tackle the problem of optimizing networks whose objective includes the total cost of a flux profile, given a cost function for each individual reaction. This application of a cost function can be used to reflect the adaptive nature of an organism's metabolism. Metabolic reactions most likely require enzymes, which naturally "cost" the organism energy and molecules to produce. This cost plays a role in why certain reactions are only active under certain environmental conditions: the payoff (in energy and other molecules) from these reactions must be enough to justify their investment.

Contents

The rest of this thesis is organized as follows: Chapter 2 introduces the reader to the analysis of metabolic networks using hypergraphs and flux balance analysis. Chapter 3 introduces using cutsets to remove elements from the network as well as our approach to generating them. Chapter 4 discusses coupling flux balance analysis with cost functions for each reaction in the network and introduces our approach to approximate an NP-hard problem instance. Chapter 5 details experiments on the approaches developed in Chapters 3 and 4 and presents the results. Chapter 6

discusses the potential of the approaches developed in this thesis. Chapter 7 outlines future work.

BACKGROUND

This chapter introduces the reader to hypergraphs and cutsets as well as stoichiometric matrices and linear and quadratic programming. The first section introduces the reader to metabolic networks as a set of reactions. The second section deals with hypergraphs and cutsets. The third section explains the derivation of stoichiometric matrices as well as relevant aspects of linear and quadratic programming. The fourth section introduces the concept of an elementary mode.

Bioengineering is an interdisciplinary field, both in theory as well as in application. In application it can be linked to materials design, pharmacology, agriculture, mechanical engineering, architecture, among others. This thesis is an example of an interdisciplinary approach to theoretical problems in the field. Diversification of theoretical aspects has come with an increased understanding of the mechanisms which can enable the effective control of an organism's useful properties. This increased understanding allows the biological processes which control an organism's functionality to be organized into roles and functions generalizable to similar problems in previously unrelated areas of research.

Escherichia coli, from here on *E.coli*, is one of the most widely used and understood organisms in bioengineering [2, 3]. *E.coli* is an ideal organism for bioengineering because it is relatively safe (it has low human virulence), its highly adaptable phenotype allows for straightforward introduction of recombinant DNA via plasmids, besides naturally producing a plethora of useful biochemicals [4]. Plasmids provide an ideal platform for genetically enhancing an organism because they are short sequences for which a small replication error rate can be achieved, and since they are independent of the bacterial host's genetic material they can be introduced into the host with a minimized risk of causing fatal mutations. *E.coli*, unlike more complicated organ-

isms, naturally utilizes plasmids to gain adaptations from its bacterial community. Therefore, naturally occurring *E.coli* is fully capable of accepting and using plasmids.

E.coli has been used to produce biofuels such as ethanol [5], essential amino acids and other basic nutrients [4, 6], and even in pharmaceuticals for which customized proteins are necessary [7]. Its metabolic network is amongst the most well characterized for bacterial organisms and organisms in general [3]. The metabolic networks used in the experiments of this thesis belong to *E.coli*. We refer the reader to [8] for a discussion on the reactions included in the *E.coli* network.

Metabolic Networks

Cells are an amalgam of chemical processes as complex as they are varied. To abstract their mechanism of growth, therefore, to graphs and simple arithmetic equations seems not only to oversimplify the system, but to do so in a way which renders the abstraction too limited to be of practical value. There are however, very simple principles which guide even most complicated of a cell's metabolic processes, and it is the responsibility of the researcher to recognize the potential as well as the limitations of considering only these principles to formulate the problem correctly.

A point in case is one dealt with specifically in this thesis, where, by recognizing that since a chemical reaction's activity is determined by the presence of its reactants,[1] a reaction r is dependent on any other reaction which produces the reactants of r. Thus, to determine this dependency it is only necessary to know the reactions in the network in terms of their participating metabolites.

[1] There are many other contributing factors to the activity (such as whether it is thermodynamically favored or there is an enzyme present) but *determined* here indicates whether the reaction occurs or not, all other circumstances permitting.

Therefore, for this thesis a metabolic network $\mathcal{N}$ can be formally defined as

$$\mathcal{N} := (M, R)$$

where M and R are the set of metabolites and the set of reactions, respectively. A reaction $R_i \in R$ is a chemical equation the form

$$R_i : aA + bB \rightarrow cC + dD$$

where this indicates that a moles of metabolite A and b moles of metabolite B combine to form c moles of C and d moles of D. Reversible reactions can be split into two reactions, forward and backward, which will form part of R instead. We can define two functions on R_i. They are defined as,

$$reactants(R_i) \Rightarrow \{A, B\}$$

$$products(R_i) \Rightarrow \{C, D\}$$

where for all $R_i \in R$, $reactants(R_i)$ and $products(R_i)$ must be subsets of M. Also, all $M_i \in M$ must belong to the set of reactants or products of at least one reaction. Throughout this chapter, we will be using an exemplary network $\mathcal{M}$ defined by,

$$M := \{M_1, M_2, ...M_{12}, M_{13}\}$$

where $R := \{R_1, R_2...R_5\}$ such that

$$R_1 : 8M_1 + 14M_2 \rightarrow 6M_3 + 9M_4$$

$$R_2 : 31M_5 + 25M_6 \rightarrow 9M_9 + 6M_{10} + 11M_{11}$$

$$R_3 : 4M_3 + 7M_4 \rightarrow 7M_7 + 78M_8$$

$$R_4 : 9M_9 + 3M_{10} + M_{11} \rightarrow 2M_{12} + 9M_6$$

$$R_5 : 3M_8 + 9M_{12} \rightarrow 3M_{13}$$

Note that none of the coefficients which accompany the metabolites in the reactions listed will be included in their graphical representation. This goes to the point referred to above where the researcher must recognize the limitations of the abstraction applied to the metabolic network, since a path through the metabolic network in graph form may not be stoichiometrically balanced. The next section will introduce hypergraphs and how to derive them from a list of reactions in the form above.

Hypergraphs

This is a brief introduction to hypergraphs and cut sets. Readers looking for a more thorough overview are referred to [9].

Definition

Hypergraphs are graphs that can be described by two sets: edges and vertices. In this thesis, this will be expressed as,

$$\mathcal{H} := (V, E)$$

where V is the set of vertices and E is the set of edges. Edges in a hypergraph are sets themselves. More precisely, an edge $e_i \in E$ is a non-empty subset of V. This defines edges in *undirected* hypergraphs. Edges in directed hypergraphs are also sets, but divide their vertices between a head and a tail. For a directed hypergraph,

$$\mathcal{H}_{directed} := (V', E')$$

an edge $e_i \in E'$ is defined as

$$e_i := (T, h)$$

where T and h are respectively the tail and head of the edge. $T \cup h$ cannot be empty. Finally, directed edges are considered to go in direction from the tail to the head.

A graph of network $\mathcal{M}$ is as follows,

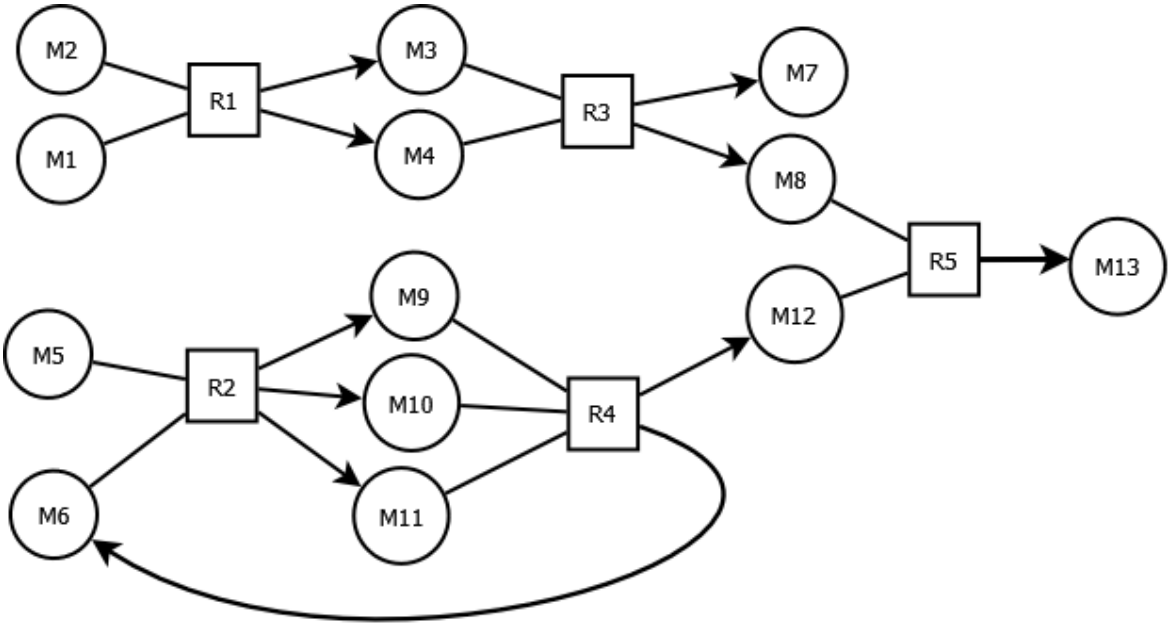

Figure 2.1: Network $\mathcal{M}$

Note that the tail of R_1 is composed of metabolites M_1 and M_2, and the head is correspondingly composed of metabolites M_3 and M_4 as indicated by the arrows.

Cut Sets

A cut set is a set of hyperedges defined by two disjoint vertex subsets. Thus, given a directed hypergraph $\mathcal{H} = (V, E)$, defining two subsets $V_H \subset V$, and $V_T \subset V$, we have defined a cut set by

$$cutset(\mathcal{H}, V_H, V_T) = \{e_i \in E \mid head(e_i) \cap V_H \neq \emptyset \wedge\ tail(e_i) \subseteq V_T\}$$

where $head(e_i)$ and $tail(e_i)$ indicate the set of vertices that respectively conform the head and tail of e_i. Note that any edge e_i for which $head(e_i) \cap V_T \neq \emptyset$ can still be

included in the cutset, however if $tail(e_i) \cap V_H \neq \emptyset$ then e_i is not included in the cutset.

Suppose $V_H := \{M_7, M_8, M_{12}, M_{13}\}$ and $V_T := V - V_H$, $cutset(\mathcal{M}, V_H, V_T)$ then becomes $\{R_3, R_4\}$, as in the following graph.

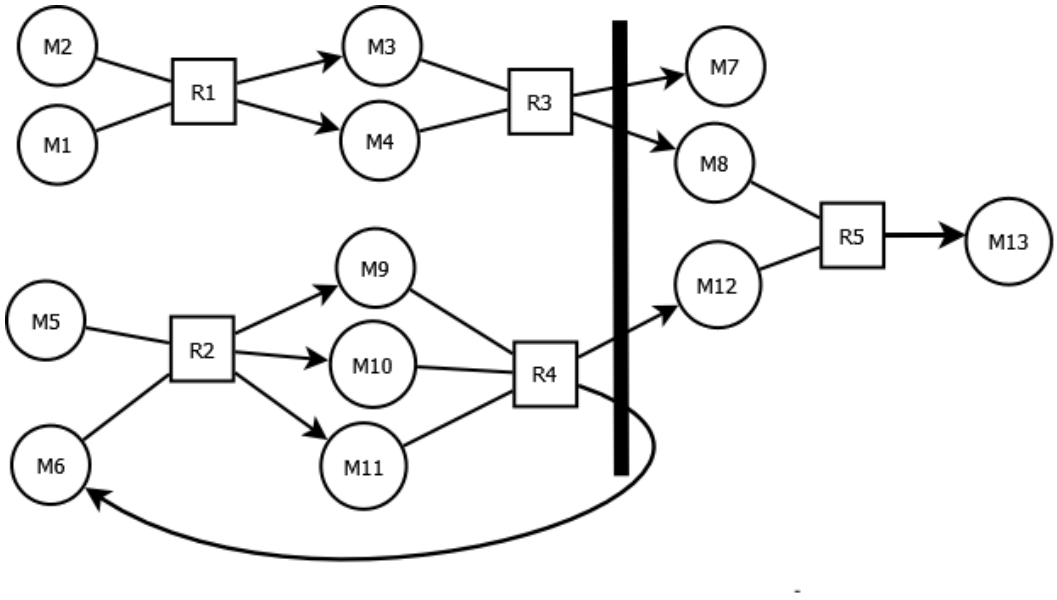

Figure 2.2: Cut 1

A helpful thing to notice in this example is that all metabolites in V_H are to the right of the black bar whereas all the metabolites in V_T are to the left. Thus, the reader can visually ascertain that neither the tail of R_3 nor the tail of R_4 intersect with V_H and that even though the head of R_4 is not completely within V_H it is still included in the cutset.

Now suppose $V'_H := V_H + M_3$ and $V'_T := V - V'_H$ as illustrated in the following graph:

This leads to the exclusion of R_3 and the inclusion of R_1 in the new cutset, which can be formally defined as $cutset(\mathcal{M}, V'_H, V'_T) := \{R_1, R_4\}$.

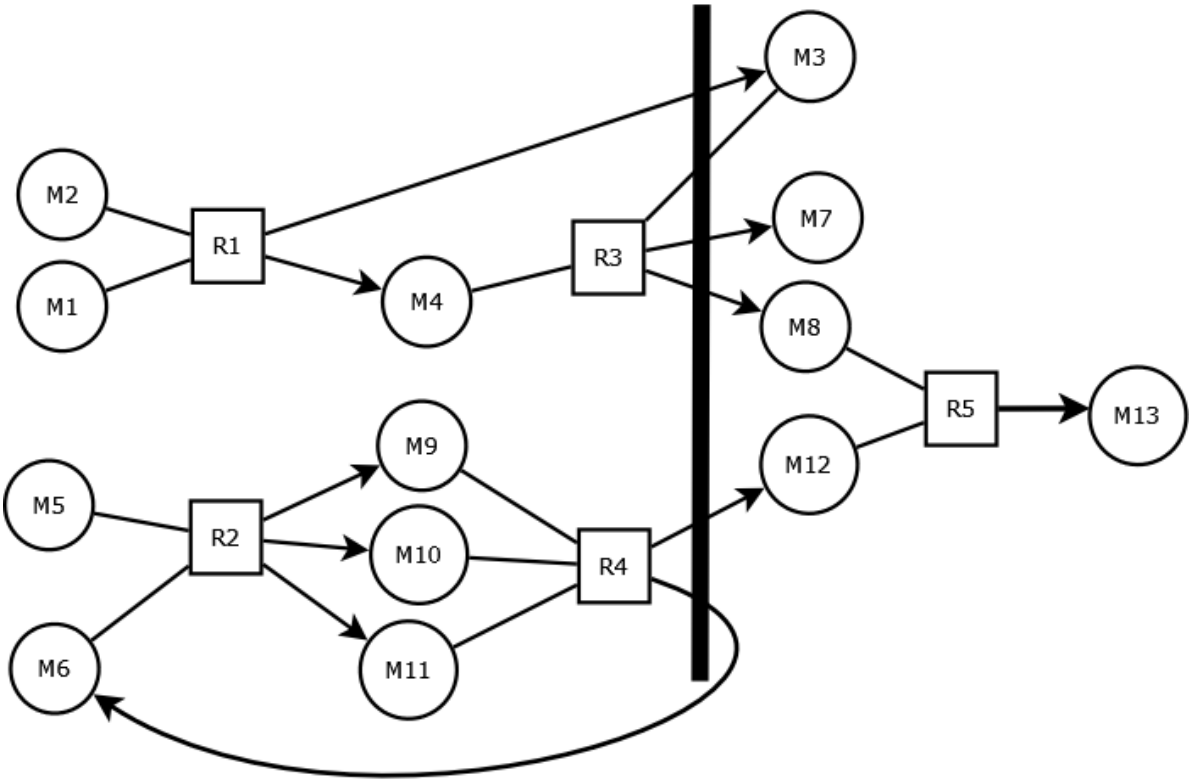

Figure 2.3: Cut 2

Cutsets define a special dependency in the context of hypergraphs derived form metabolic networks. For the applications in this thesis V_H is chosen to include metabolites whose production is undesirable. A cutset derived from this set of undesired metabolites becomes the set of reactions which must be removed so that the metabolites in V'_H are not produced.

In this context the rules which define a cutset are easier to understand: the edges whose head intersect with V_H become the reactions that produce any of the metabolites in V_H. Also, those reactions whose tail is in V_T are now those reactions which are not dependent on the metabolites in V_H as reactants. Should they be

dependent on any metabolite in V_H, the removal of V_H should render the reactions impossible. Thus, the reactions in a cutset are the minimum set of reactions that must be removed to ensure V_H is not produced.

Linear and Quadratic Programming

Linear programs are problems of the form

minimize

$$c^T x$$

subject to

$$Ax = b$$

$$Px \leq d$$

where x is a vector representing the values of a set of variables, and c represents different weights corresponding to each variable. A and P represent matrices which express constraints on the values of x along with the b and d vectors. The function being minimized is referred to as the *objective* function.

Continuing with our sample network $\mathcal{M}$, let the vector x indicate the flux through the reactions in R, where the ith element x represents the flux of the ith reaction in R. Also, let vector c represent the cost of each reaction in R, where the ith element in c represents the cost for the ith reaction in R. The total cost of a flux profile is, as indicated above, $c^T x$. Solving the linear program above would yield the flux profile of least cost. Note that, as per the definition beginning this section, we are minimizing the function $cost(x) := c^T x$; in general, we can maximize any $c^T x$ by minimizing $-c^T x$.

Several algorithms have been developed to solve linear programs. Linear programs can be solved in polynomial time, though there are caveats to each algorithm used to

solve them [10]. Their polynomial complexity makes them an ideal framework with which to find the low cost flux profiles that cells have adapted to and efficient profiles for cells to be used in mass metabolite production. For a more thorough introduction to linear programming, we refer the reader to [11].

Quadratic programs are more general than linear programs, because they allow the minimization of an objective function with quadratic components. This imposes some constraints on the problems that can be solved in polynomial time. Quadratic programs are defined by

minimize

$$x^T \frac{1}{2} Q x + c^T x$$

subject to

$$Ax = b$$

$$Px \leq d$$

where Q is derived from the coefficients which weight the quadratic components of the objective function. That is, $q_{ij} := c_{x_i x_j}$, where $c_{x_i x_j}$ is the coefficient for the term $x_i x_j$ of the objective function. To illustrate, given the objective function,

$$ax_1^2 + bx_2^2 + cx_2x_3$$

the corresponding Q matrix can be defined as,

$$Q := 2 \begin{vmatrix} a & 0 & 0 \\ 0 & b & c \\ 0 & 0 & 0 \end{vmatrix}$$

where the scalar multiplication by two is included to offset the $\frac{1}{2}$ coefficient in the formal cost function definition. Note that though c is at q_{23} it may equivalently be at q_{32}.

In the simplest case, FBA can be done by defining a linear program

minimize

$$c^T x$$

subject to

$$Sx = 0$$

$$Bx \leq b$$

where x is a vector representing the flux for each reaction in R, c is the set of coefficients for each flux in the objective function, S is the stoichiometric matrix derived from R, and B sets inequality constraints on the possible values for each element in x. Additional constraints may be added by expanding S or B matrices.

Network $\mathcal{M}$ is then expressed as a stoichiometric matrix S by designating each row in S for the stoichiometric coefficients of a specific metabolite M_i, and a column for each reaction R_j in R. The stoichiometric coefficient for M_i in the reaction column R_j will be negative if M_i is one of reactants of R_j.

The stoichiometric matrix S for $\mathcal{M}$ can be seen in the following page. Rows 1 to 13 are respectively designated for $M_1, M_2, ...M_{13}$, and columns 1 to 5 for $R_1, R_2, ...R_5$ respectively.

Diligent readers will have begun to notice what the constraint $Sx = 0$, denoted the *stoichiometric constraint*, accomplishes; namely, it ensures that for each metabolite M_i, the net quantity produced by the network under the fluxes in x is exactly zero.

$$S := \begin{vmatrix} -8 & 0 & 0 & 0 & 0 \\ -14 & 0 & 0 & 0 & 0 \\ 6 & 0 & -4 & 0 & 0 \\ 9 & 0 & -7 & 0 & 0 \\ 0 & -21 & 0 & 0 & 0 \\ 0 & -25 & 0 & 9 & 0 \\ 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 78 & 0 & -3 \\ 0 & 9 & 0 & -9 & 0 \\ 0 & 6 & 0 & -3 & 0 \\ 0 & 4 & 0 & -1 & 0 \\ 0 & 0 & 0 & 2 & -9 \\ 0 & 0 & 0 & 0 & 3 \end{vmatrix}$$

Note that M_{13} is not consumed by any reaction in R. Therefore, for the the stoichiometric constraint to hold, x must have a flux of zero for all reactions that produce M_{13}. Since the flux of M_{13} will be zero, metabolites M_8 and M_{12} will not be consumed, and so on throughout $\mathcal{M}$. In fact, with the given R the only solution is for all fluxes of x to be set to zero.

To deal with external metabolites[2] their rows may be removed from S, or they may be made internal by adding reactions to R. These reactions would remove or provide the excess metabolite as appropriate. To internalize M_{13} we add R_x to R, defined as:

$$M_{13} \rightarrow \emptyset$$

[2]An *internal* metabolite is one that is both produced and consumed by reactions in R; external metabolites therefore are either not produced or not consumed by any reaction in R.

where the empty right hand side simply means that R_x consumes one unit of M_{13} but produces nothing. Note that to deal with all external metabolites, reactions should be added to produce M_1, M_2, M_5, and to consume M_7.

Introducing these reactions allows for the rate at which metabolites leave or enter the network to be explicitly controlled by constraints applied to their fluxes. These reactions, however, should only be applied to deal with external metabolites, since the internal metabolites must be balanced by the equations representing actual reactions. Otherwise, the solution space becomes all possible fluxes and pathways cannot be found. Thus, S finally becomes

$$S := \begin{vmatrix} -8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -14 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 6 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & -7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -21 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & -25 & 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 78 & 0 & -3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & -9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & -1 \end{vmatrix}$$

Where R_x is in column 10.

Now that we have derived S, all we need is an objective function to begin flux balance analysis. Suppose we wanted to know the maximum amount of M_{13} network $\mathcal{M}$ could produce, given a limited supply of M_1 and M_2. We define the objective as

the amount M_{13} that the network produces as a function of fluxes,

$$3x_5$$

and, since the linear program will minimize the objective, by changing the sign of the coefficients

$$-3x_5$$

becomes the appropriate equivalent to maximizing the amount of M_{13} produced. Given the above function we can define the cost vector c as

$$c := [0\,0\,0\,0\ -3\,0\,0\,0\,0\,0]$$

Finally, we must set the bounds on the fluxes of each reaction. If all reactions in R are irreversible, then fluxes from zero to infinity are possible. However, there are limits to the activities of enzymes as well as how fast metabolites may enter or leave the system. For this example, only limited supplies of M_1 and M_2 are specified. Therefore, the fluxes of R_6 and R_7 will be set to be between zero and one. Finally, we will assume all reactions are irreversible. The fully specified linear program can be seen at the end of the chapter.

Elementary Modes

The stoichiometric constraint ensures that the solutions to the linear and quadratic programs represent steady states, i.e. no net production of metabolites, for the organism. The nullspace[3] of S, in which all solutions are contained, represents the collection of all fluxes profiles which achieve steady state. Though composed of an infinite number of flux profiles, the nullspace may be summarized by a finite number

[3] The nullspace of a matrix A is the set of all vectors x for which $Ax = 0$.

of flux profiles. This is because the nullspace of S has a special shape, the outline of which is the finite number of flux profiles we will use to describe it.

This shape is a *convex polyhedral cone*, and the flux profiles that conform its outline are its *elementary modes*. Formally, an elementary mode is defined as a flux profile in the nullspace whose support[4] is not a proper superset of the support of another flux profile also in the nullspace. This means no two flux profiles may share the exact same set of active reactions if they are elementary modes. Furthermore, if a flux profile is an elementary mode, its set of active reations cannot contain the set of active reactions of any other profile also in the nullspace. This gives elementary modes their name: they are elementary in that they cannot be broken down into more steady states. A significant consequence of this is that, in turn, every flux profile in the nullspace can be decomposed into a set of elementary modes. Any solution to our programs will simply be a linear combination of elementary modes.

Elementary modes have many interesting and useful properties. For a more thorough discussion on elementary modes, we refer the reader to [12].

[4] The support of a vector $x = [x_1, x_2...x_n]$ is the set of elements x_i for which $x_i \neq 0$. In the context of flux profiles, the support of a profile x is the set of reactions which have a non-zero flux, i.e. the set of active reactions in the profile.

Fully Specified Linear Program

minimize

$$[0, 0, 0, 0, -3, 0, 0, 0, 0, 0]^T x$$

subject to

$$\left|\begin{array}{cccccccccc} -8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -14 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 6 & 0 & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & -7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -21 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & -25 & 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 78 & 0 & -3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & -9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & -1 \end{array}\right| x = [0\,0\,0\,0\,0\,0\,0\,0\,0\,0]$$

$$\left|\begin{array}{cc} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{array}\right| x \leq [0\,0\,0\,0\,0\,1\,1\,0\,0\,0]$$

CUTSETS

Computing Cut Effects

Removing the cutset of a set of metabolites may inadvertently render impossible downstream reactions beyond the intended producers of undesired metabolites. Therefore, compiling a full list of the reactions affected by a cutset is necessary to completely evaluate the usefulness of removing a cutset.

To do so, let $\mathcal{M} := (M, R)$ be a metabolic network where again R the is the set of all reactions that participate in $\mathcal{M}$ and M the set of all metabolites that participate in any reaction $R_i \in R$. Also, let $X \subset R$ be the set of reactions that will be cut from $\mathcal{M}$. Recall that $reactants(R_i \in R)$ denotes the set reactants of R_i and $producers(M_i \in M)$ is the set of reactions which produce M_i.

Algorithm 3.1 is pseudocode which will compile a list of the metabolites and reactions eliminated by cutting the reactions in X from $\mathcal{M}$. This list is compiled by iteratively growing the list of eliminated network elements to include those reactions whose reactants have been eliminated along with those metabolites whose producers have all been eliminated.

The complexity of this algorithm is polynomial, $n^3 log(n)$, as will be demonstrated in the following paragraphs.

Lines 2, 3 and 7 can be done in constant time.

The complexity of $reactants(R_i)$ and $producers(M_j)$ is K, given a hash table $\mathcal{H}$ with a hash function of complexity K, where the keys are reactions and metabolites and the values are the appropriate list. Formally, $\mathcal{H}[M_j] := reactants(M_j)$ and $\mathcal{H}[R_i] := reactants(R_i)$. Deriving $\mathcal{H}$ from $\mathcal{M}$ has a complexity of n^2, where $n := max\{||M||, ||R||\}$.

```
1.  Procedure ELIMINATED(M, X)
2.      previousReactions ← ∅, previousMetabolites ← ∅
3.      newReactions ← X, newMetabolites ← ∅
4.    repeat
5.        previousReactions ← previousReactions ∪ newReactions
6.        previousMetabolites ← previousMetabolites ∪ newMetabolites
7.        newReactions ← ∅, newMetabolites ← ∅
8.      for M_i ∈ M
9.        if producers(M_i) ⊆ previousReactions
10.           newMetabolites ← newMetabolites + M_i
11.       endif
12.     endfor
13.     for R_i ∈ R
14.       if reactants(R_i) ∩ previousMetabolites ≠ ∅
15.           newReactions ← newReactions + R_i
16.       endif
17.     endfor
18.   until newMetabolites - previousMetabolites = ∅ and
19.         newReactions - previousReactions = ∅
20. EndProcedure
```

Algorithm 3.1: Algorithm ELIMINATED

Lines 5 and 6 can vary according to the algorithm and data structures being used. In this case the lists being combined are sets, and the elements of the sets are unique strings and therefore hashable. Two lists can therefore can be merged at complexity $2kn$ if the lists are sorted. Sorting the lists initially can be done in $nlog(n)$ (in line 2), and elements will be added so that the list remains sorted.

Lines 10 and 15 can be done in $log(n)$, as they are the addition of a single element to a sorted list.

Subtracting the sets in 18 and 19 can also be done in $2kn$ time, since both sets are sorted lists.

Lines 9 and 14 are also set operations which can be done in $2kn$ time.

The loop at line 4 will repeat at most $2n$ times. This is because of the relationship between $previousMetabolites$ and $newMetabolites$ along with the relationship between $previousReactions$ and $newReactions$.

To show this let

$$oldElements = previousMetabolites \cup previousReactions$$

$$newElements = newMetabolites \cup newReactions$$

Any iteration of the loop will either add one or more new elements from $M \cup R$ to $newElements$ or it will not. Due to the conditions in lines 18 and 19, if $newElements$ does not receive at least one new element the loop will finalize. Note that this will happen because $newElements$ is empty or because all the elements in $newElements$ are already in $oldElements$. Since elements are never removed from $oldElements$, it holds all the elements that were in $newElements$ until the current iteration. Given a finite $M \cup R$, line 4 will iterate at most $||M \cup R|| + 1$, or $2n$, times. That is, by iteration $||M \cup R|| + 1$ the set $oldElements$ will hold all the elements in $||M \cup R||$, causing $newElements$ to be empty and thus fulfilling the stopping condition.

The Improvement Heuristic

Given a set of undesired metobolites Z, we have implicitly defined a cutset X by partitioning the set V into two subsets as follows

$$X := cutset(\, \mathcal{M},\, Z,\, V - Z)$$

This is not, however, the only cutset which will eliminate Z from the network. In fact, any cutset for which $Z \subseteq V_H$ and $V_T = V - V_H$ will eliminate Z from the network.

Finding the V_H which will eliminate Z and yield the smallest cutset is an NP-hard problem for hypergraphs [13]. However, given an initial V_H, improving it via

the addition of the metabolites can be tackled applying all sorts of polynomial time heuristics. A review of the heuristics is beyond the scope of this thesis, but, as an example, consider algorithm 3.2. Let the set $C \subset M$ be defined as the tail of the reactions in X. Explicitly, C is defined as

$$C := \{M_i \in M \mid M_i \in reactants(R_i), \forall R_i \in X\}$$

Recall one can disqualify any reaction X_i from X by making $tail(X_i)$ intersect with V_H. Since metabolites in C compose the tail of the reactions in X, adding a metabolite from C to V_H will remove at least one reaction from X. This, of course, results in the possible inclusion of reactions not already in X, so we only add those metabolites from C which will result in a net reduction of the size of X. Algorithm 3.2 improves V_H by this method.

```
1.  Procedure IMPROVE(M, V_H, V_T)
2.   reducing ← ∅
3.   X ← cutset(M, V_H, V_T)
4.   C ← {M_i ∈ M | M_i ∈ reactants(R_i), ∀R_i ∈ X}
5.  forall M_i ∈ mets(C)
6.      added ← size( producers(M_i) − X )
7.      subtracted ← size( {R_j ∈ X | M_i ∈ reactants(R_j)} )
8.     if size(subtracted) > size(added)
9.         reducing ← reducing + M_i
10.    endif
11. endfor
12.  V_H ← V_H + reducing
13. EndProcedure
```

Algorithm 3.2: Algorithm IMPROVE

From this we can see that what the above pseudocode does is cull the metabolites for which the number of reactions which would be removed exceeds the number of the ones which would be added with its inclusion in V_H. Then, one of those cut-reducing metabolites is added to C. Though algorithm 3.2 deals strictly with addition of

metabolites, removing metabolites from V_H can also be used to reduce the size of the cut, by a similar method. Expanding this improvement method into an effecive heuristic is a matter of creativity, e.g. it may be used to influence an ant's path in ant colony optimization [14] where the ant's path is the set of metabolites to add, or similarly guide the generation of candidate solutions in other generative algorithms, as well as finding the local minima some search algorithms employ.

Generating Cuts

Generating cuts using the only the improvement heuristic, however, neglects a principle which would reduce the initial search space from the $2^{|M|}$ possible V_H to only the relevant cuts. That is, since a cut of four or more reactions is infeasible, searching only among feasible cuts may reduce the search space considerably. Cuts may be of one, two, or three reactions, so size of the search space now becomes

$$\binom{|R|}{3} + \binom{|R|}{2} + \binom{|R|}{1}$$

which is usually smaller than $2^{|M|}$ even though, since some metabolites are used in multiple pathways, there are generally more reactions than metabolites. For perspective on the reduction of the search space, even a small metabolic network composed of 35 metabolites and 62 irreversible reactions, as found in [8], will have a search space ratio of

$$r = \frac{2^{35-size(Z)}}{\binom{62}{3} + \binom{62}{2} + \binom{62}{1}}$$

where $size(Z)$ is the size of the set of metabolites whose production we wish to avoid. If we wish to avoid a single metabolite, the ratio between the size of the spaces is on the order of half a million. Even if ten metabolites (almost one third of the total) are to be avoided, the ratio is on the order of one thousand.

Designating all subsets of R of size less than three, referred to as $subsets_3(R)$, as candidate cuts is naive in that is is unkown a priori whether a candidate cut X will eliminate Z from the network. This is especially wasteful if the cuts are to be optimized using FBA to measure how much of the objective they can produce, since all evaluations of ineffective cuts will be wasted computation.

Fortunately, algorithm 3.1 can be used to fully qualify the effects of cutting X in polynomial time. However, applying it to all cuts in $subsets_3(R)$ is also neglecting an important property of cuts. Cuts are monotonic with respect to the metabolites and reactions they eliminate. More formally,

$$X \subset X' \rightarrow Eliminated(\mathcal{M}, X) \subseteq Eliminated(\mathcal{M}, X')$$

where the arrow denotes implication. Intuitively, this is because once we have removed X from $\mathcal{M}$, removing yet another reaction R_x from $\mathcal{M}$ adds R_x to the eliminated reactions (if it was not already there) along with any metabolites R_x was the sole producer of, thus only incrementing both sets.

Formally, that *Eliminated* is monotonic with respect to X will be proven as follows. First, let $S \subset S'$ and let $loop_i$ denote the loop in *Eliminated* that begins at line i. By $loop_8$, $newMetabolites$ can be defined as all metabolites whose producers are a subset of $previousReactions$. We may therefore write $newMetabolites$ as a function,

$$newMetabolites(previousReactions) := \sum_{M_i \in M^*} M_i$$

where $M^* := \{M_i \in M \mid producers(M_i) \subseteq previousReactions\}$ If $producers(M_i)$ is a subset of $previousReactions$, it must be a subset of all sets which contain $previousReactions$. Thus, $newMetabolites(S')$ will contain all metabolites in $newMetabolites(S)$.

Similarly, by $loop_{13}$, $newReactions$ may be written as a function

$$newReactions(previousMetabolites) := \sum_{R_i \in R^*} R_i$$

where $R^* := \{R_i \in R \mid reactants(R_i) \cap previousMetabolites \neq \emptyset\}$. Note that if $reactants(R_i)$ intersects with $previousMetabolites$, it must intersect with any set that contains $previousMetabolites$. Thus $newReactions(S')$ will contain all reactions in $newReactions(S)$.

From this we can see that for two cuts $X \subset X'$, $previousReactions$ for $Eliminated(\mathcal{M}, X)$ will be a subset of $previousReactions$ for $Eliminated(\mathcal{M}, X')$ at line 2. Thus, by the first iteration of $loop_4$ we know that $newMetabolites$ for $Eliminated(\mathcal{M}, X)$ is a subset of (or equal to) $newMetabolites$ for $Eliminated(\mathcal{M}, X')$. By the arguments above, this ensures that $Eliminated(\mathcal{M}, X) \subseteq Eliminated(\mathcal{M}, X')$.

Algorithm 3.3 generates a set of candidate cuts given a set Z of undesired metabolites and an integer $MaxSize$ indicating the maximum size of a feasible cut. All of the candidate cuts returned will be effective. Algorithm 3.3 exploits monotonicity by testing all cuts of size one or two before it evaluates cuts of size three. When a successful cut is found of size one is found, all its supersets are immediately added to the set of cuts to be returned. The same is done for successful cuts of size two.

To ensure that none of the supersets computed and added are redundant, a canonical ordering is imposed on R. Since R is typically a set of reaction names, i.e. a list of alphanumeric strings, this ordering can be assumed to be lexicographic. Thus, whenever a successful cut X is found, $supersets(X)$ will compute a list of supersets whose elements are greater than the last element in X. As an example, suppose $R := \{A, B, C, D\}$ and a succesful cut $X := \{B\}$ has been found. All the sets $supersets(X)$ will be added to the set of successful cuts, i.e. sets $\{B, C\}$, $\{B, C, D\}$, and $\{B, D\}$ will be added. Because the main function in algorithm 3.3,

GenerateCuts, performs a depth-first search, set $\{A, B\}$ has already been added. In fact, it is because of the depth-first search along the canonical ordering that we can say that all supersets of X have been tested which include elements less in ordering than the last element of X.

```
1.  Procedure GenerateCuts(M, Z, MaxSize)
2.  R ← canonical(R), goodCuts ← ∅, subset ← ∅
3.  goodCuts ← GetCuts(M, Z, MaxSize,goodCuts, subset)
4.  EndProcedure
5.
6.  Procedure GetCuts(M, Z, MaxSize,goodCuts,subset)
7.  if Z ⊆ Eliminated(M, subset)
8.      goodCuts ← goodCuts + subset
9.      goodCuts ← goodCuts ∪ supersets(R,subset)
10. elsif MaxSize > 0
11.     forall super ∈ supersets(R,subset)
12.         GetCuts(M, Z, MaxSize − 1,goodCuts,super)
13.     endfor
14. endif
15. EndProcedure
16.
17. Procedure Supersets(R,subset)
18. supersets ← ∅
19. i ← index(R, max(subset))
20. while i < size(R)
21.     newSuperset ← subset + R[i]
22.     supersets ← supersets + newSuperset
23. i ← i + 1
24. endwhile
25. EndProcedure
```

Algorithm 3.3: Algorithm GenerateCuts

Generating Cutsets from a Gene Set

We will now relax an implicit assumption that has been made thoughout the chapter. Recall that the method by which a reaction is removed from the network is

by disabling a gene. Up to now we have assumed that by disabling a single gene in the organism's genetic code, exactly one reaction will be removed from the network. This assumption is valid under the condition that the gene contains all the genetic material capable of catalyzing the reaction.

This condition may be violated by a variety of circumstances. Some reactions, possibly quite essential to the organism, may be protected against catastrohpic replication errors (those that would render the enzyme encoded useless) by having multiple copies encoding the enzyme across multiple genes. It may also be that several complexes encoded across multiple genes are ultimately combined into our target enzyme. There may also be multiple reactions catalyzed by the same enzyme, or whose catalysts are encoded in the same gene. Finally, a reaction may be catalyzed by mutliple enzymes.

To address these concerns, we will formally define the generalized problem. A network $\mathcal{N}$ is defined as

$$\mathcal{N} := (M, R, G, f)$$

where M is the set of metabolites which participate in the set of reactions R, and G is a set of genes which f maps to R. We may then generate our list of potential cuts by finding all subsets of G of size less than three, and using f to figure out what reactions will be in X.

COST FUNCTIONS

As discussed, an organism's phenotype adapts to express the pathways, and therefore reactions, most appropriate to its current situation. Knowing the relative costs of reactions can serve as an indication of which modes will be applied in a nutrient-rich medium versus one that is more frugal. Depending on what cost functions reflect, however, applying cost functions to reactions allows for much more than just finding the pathways that cost an organism less to metabolize nutrients with.

Problem Context

The linear and quadratic programs in Chapter 2 can be generalized as

minimize

$$cost(x)$$

subject to

$$Sx = 0$$

$$Bx \leq b$$

where $cost(x)$ is the cost of a flux profile $x := \{x_1, x_2, ...x_n\}$ and is defined by

$$cost(x) = \sum_{j=1}^{n} c_j(x_j)$$

In the case of a linear program

$$cost(x) = \sum_{j=1}^{n} c_j(x_j) = \sum_{j=1}^{n} k_j x_j = k^T x$$

where k_j is a constant weight assigned to reaction x_j, and $k = \{k_1, k_2, ...k_n\}$ is the vector of weights. In the case of a quadratic program

$$cost(x) = \sum_{j=1}^{n} c_j(x_j) = \sum_{j=1}^{n} q_j x_j^2 + k_j x_j = x^T Q x + k^T x$$

where Q is as explained in Chapter 2. When paired with proper lower bounds, the above generalization can be used to balance multiple objectives.

For example, suppose we wished to maximize the production of a metabolite m while guaranteeing an appropriate growth rate for the cell. Our approach would be to solve two linear programs. First, to find the maximum flux possible for the growth reaction we set the cost function of the first program to

$$cost_1(x) = g^T x$$

where g is composed of all zeros except for g_i corresponding to the growth reaction, which is set to a negative coefficient. The solution to this problem contains the maximum flux possible x_i^* through the growth reaction. The second problem is similar, with cost

$$cost_2(x) = h^T x$$

where h is composed of all zeros except for h_j corresponding to the reactions which produce or consume m, which are set to the stoichiometric coefficient of m in those reactions, with opposite sign. To ensure that the growth is not affected significantly by the optimization, we set the lower bound for the flux on the growth reaction to a fraction of the maximum growth flux computed in the first problem corresponding to an acceptable loss in growth. As an example, the lower bound could be set to 90% of the maximum growth.

We may cultivate any number of behaviors by sequentially solving programs and refining constraints. Multiple objectives may be cultivated alternatively by including coefficients for both objectives into the same cost functions and optimizing the combined objective cost function. The potential disadvantage of this approach is that performance is not guaranteed for either objective, i.e. the final solution may be

optimized only for one objective since the payoff for optimizing the single objective was greater than the payoff for optimizing both.

Finding solutions which satisfy several optimizations can be used to discriminate among cuts which eliminate the set of undesired metabolites. Recall from Chapter 3 that every cut set has a set of reaction and a set of metabolites that it will eliminate from the network. However, it may be difficult to know a priori whether eliminating one set from the network will yield better performance than another. Flux profiles which satisfy multiple optimizations can be used to find the reactions necessary for performance on multiple objectives. Thus, cuts which preserve as many of the reactions active in these flux profiles are more desirable than those which remove them. These flux profiles can be decomposed into elementary modes, which can also be compared to the list of eliminated elements to see which elementary modes were preserved versus those that were removed.

Optimizations are limited to the forms of $cost(x)$ that can be solved efficiently. This chapter discusses optimizing functions in general and derives an approximation on to use with a special kind of non-linear function.

General Problem

Let $\{c_j : \mathbb{R} \Rightarrow \mathbb{R}\}$, $j \leq n$ be a set of functions, where $c_j(x)$ denotes the cost of running reaction j at flux x. The cost of a flux profile $x = \{x_1, x_2, ..., x_n\}$ is given by

$$C(x) = \sum_{j=1}^{n} c_j(x_j)$$

Each cost function $c_j(x_j)$ can be approximated by a corresponding polynomial function $p_j(x_j)$. The values for x_j which yield the lowest $p_j(x_j)$ can be deduced from its derivative $p'_j(x_j)$.

More formally, if we denote by $\mathfrak{M}_j$ the set of points for which $p_j(v_j)$ reaches a *local* minimum, we have that for all $m_i \in \mathfrak{M}_j$:

$$p_j'(m_i) = 0 \tag{4.1}$$

$$p_j'(m_i - \epsilon) \leq 0 \tag{4.2}$$

$$p_j'(m_i + \epsilon) \geq 0 \tag{4.3}$$

where ϵ approaches, but is greater than, zero. Note that $\mathfrak{M}_j$ may be empty, as in the case where p_j is monotonic, or infinite, as in the case where p_j is periodic.

For any $\mathfrak{M}_j \neq \emptyset$ there exists a subset $\mathfrak{M}_j^* \subseteq \mathfrak{M}_j$ which yield the *global* minima of p_j. More formally, $m_i^* \in \mathfrak{M}_j^*$ implies that:

$$p_j(m_i^*) \leq p_j(m_k),\ \forall m_k \in \mathfrak{M}_j \tag{4.4}$$

Given $\mathfrak{M}_j^*$ for all j, the flux profiles x which yield the global minima of $C(x)$ can be found by taking the cross product of all sets $\mathfrak{M}_j^*$ for all j.

This approach is sufficient for a problem which satisfies certain conditions. The first is that a precise approximation requires polynomials with a small number of roots. Also, the global minima must be easily distinguishable from local minima. Finally, it must be easy to ascertain whether the constraints on the solutions to the problem are satisified by a certain point. Note that even if all three conditions are fulfilled, none of the minima may be inside the feasible region, which makes the fourth condition: at least one global minimum must be within the feasible region.

The problem central to this thesis, optimizing metabolic networks, in general does not satisfy the fourth condition, which is why linear and quadratic programming is required. Also, the accuracy of linear and quadratic approximations of step functions decays as the size of the step, which may cause a violation of the first condition.

The accuracy and practicality of the approximations derived in this chapter will be explored in Chapter 5.

Step Functions Approximation

We introduce a linear step function

$$s(i) := \begin{cases} 0 & : i = 0 \\ mi + b & : i \geq 0 \end{cases} \tag{4.5}$$

where m and b are respectively the slope and the y-intercept of the linear component of s. Also, s is undefined for i less than zero.

Given the nonlinear nature of step functions[1], if any cost function is a step function, we are unable to use linear or quadratic programs to optimize the cost objective. However, we may use both linear and quadratic programs to approximate the objective if we approximate each cost function with a linear or quadratic function.

There are many linear and quadratic approximations applicable to a step function s, but there are two properties that an approximation a should have. First, a should be a lower bound for s, that is, for the continuous range of points $\mathbf{I}$ we are considering,

$$a(i) \leq s(i), \forall i \in \mathbf{I}$$

This guarantee will allow us to bound the value of the optimal solution.

Also, the approximation a should be as precise as possible. That is, given a set of parameters for the approximation, these parameters should minimize the error of the approximation. More formally, let the approximation parameters be a vector $\mathbf{P} := \{\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}\}$, the parameters should have values $\mathbf{P}^*$ such that

$$error(s, a, \mathbf{P}^*) \leq error(s, a, \mathbf{P}), \forall \mathbf{P}$$

[1]It should be noted that if $b = 0$, s is indeed linear.

where $error(s, a, \mathbf{P})$ measures the error of an approximation function a with parameters $\mathbf{P}$. As an example, a linear function has two paramters, the slope and the y-intercept.

Optimizing without Approximation

Optimization problems whose objective is the sum of linear step functions can be solved directly by solving a linear program with special binary variables. Linear programs with binary variables in general, however, have been shown to model several NP-Complete problems [15] and, as such, there exists no polynomial time algorithm to solve the general problem. As implied by the name, binary variables are restricted to have values of zero or one.

The program formulation is as follows. Given a flux profile $x = [x_1, x_2, ..., x_n]$, let

$$s_i(x_i) := \begin{cases} 0 & : x_i = 0 \\ m_i x_i + b_i & : x_i \geq 0 \end{cases} \tag{4.6}$$

describe the cost s_i of running reaction R_i at flux x_i. Let $x^b = [x_1^b, x_2^b, ..., x_n^b]$ be the set of binary variables, one for each flux in x. Finally, let $m = [m_1, m_2, ..., m_n]$, where $m_j \in m$ is the slope coefficient for the cost s_j of flux x_j, and let $b = [b_1, b_2, ..., b_n]$ be the same for the y-intercepts. Given these variables, the program becomes

minimize

$$m^T x + b^T x^b$$

subject to

$$Sx^T = 0$$

$$x_i \leq x_i^b, \ \forall x_i \in x$$

Linear Approximation

In the following sections we will derive both a linear and a quadratic approximation to step functions. Each cost function s_i can be approximated by a function a_i. Thus, the cost of a flux profile, defined as

$$\sum_{i=1}^{n} s_i(x_i) \tag{4.7}$$

will approximated by

$$\sum_{i=1}^{n} a_i(x_i) \tag{4.8}$$

Because the same method will be applied to approximate all cost functions, we will limit our discussion to deriving a single approximation a for a single cost function s.

Since a is a lower bound for s, the error can be defined in general as

$$error(s, a) = \int_{\mathbf{I}_0}^{\mathbf{I}_f} s(x) - a(x) \tag{4.9}$$

Throughout the rest of the calculations, we will assume that fluxes are bound to be between zero and one. That is,

$$\mathbf{I} = [\mathbf{I}_0, \mathbf{I}_f] = [0, 1]$$

This assumption is used without loss of generality since it represents a normalization of the flux vector. Also, we will assume $b_s > 0$, since $b_s = 0$ needs no approximation, and $b_s < 0$ implies a negative cost for at least one $i \in \mathbf{I}$, something not permitted in our model. We assume $m_s > 0$ since our model posits that the cost of a reaction is based on enzyme production, and an increase in flux requires more frequent enzyme production.

Thus, the error for a linear approximation $a(x) = m_a x + b_a$ of a step function with linear component $m_s x + b_s$ approaches

$$\epsilon = \int_0^1 (m_s x + b_s) - (m_a x + b_a)$$

Note that the true error is not exactly ϵ, since $s(0) = 0$ and not b_s, as the integral implies. The true error is therefore no greater than ϵ, given that $a(0)$ is at most zero, since the difference between $a(0)$ and $s(0)$ will be less than the difference between $a(0)$ and b_s. Note also that, because a is a lower bound for s, $a(0) \leq 0$ and therefore $b_a \leq 0$. Once we integrate, ϵ becomes

$$\epsilon = \left| \frac{1}{2}(m_s - m_a)x^2 + (b_s - b_a)x \right|_0^1$$

and finally

$$\epsilon = \frac{1}{2}(m_s - m_a) + (b_s - b_a) \tag{4.10}$$

Recall that we have two parameters $\mathbf{P} := \{m_a, b_a\}$ for which ϵ must be minimized. We see that ϵ will be minimized as m_a and b_a increase. Since $b_a \leq 0$, the maximum value for b_a is zero. Because $m_s x + b_s$ and $m_a x + b_a$ are lines, we may infer that they intersect at most once in $\mathbf{I}$. If $m_a > m_s$, once the lines intersect at x^*, $a(x)$ will be greater than $s(x)$ for all values of x greater than x^*. Thus, if $m_a > m_s$, a and s must intersect at a single point in $\mathbf{I}$ and, specifically, this point must be at $\mathbf{I}_f = 1$ to ensure no $a(x) > s(x)$. Setting $s(1) = a(1)$ and solving for m_a yields a maximum m_a^* of

$$m_a^* = m_s + b_s$$

Quadratic Approximation

Parameters which minimize error for a quadratic approximation can be derived by a similar analysis. First, we calculate the error ϵ of a quadratic approximation $a(x) = q_a x^2 + m_a x + b_a$ by integrating

$$\epsilon = \int_0^1 (m_s x + b_s) - (q_a x^2 + m_a x + b_a)$$

which yields

$$\epsilon = \left| -\frac{1}{3} q_a x^3 + \frac{1}{2}(m_s - m_a)x^2 + (b_s - b_a)x \right|_0^1$$

and finally

$$\epsilon = -\frac{1}{3}q_a + \frac{1}{2}(m_s - m_a) + (b_s - b_a) \tag{4.11}$$

Minimizing ϵ we may begin by setting b_a to its maximum value of zero. We may be tempted to increment q_a and m_a indiscriminately, but m_a and q_a cannot be set independently if the lower bound requirement is to be kept, as will be discussed.

We know that a is a parabola, and therefore it will intersect $m_s x + b_s$ at most two times. However, they will intersect at most once in **I**. To show this, recall that at $x = 0$, a is zero and thus less than $m_s x + b_s$. Therefore, until $a(x) = m_s x + b_s$, we may assume the derivative $a'(x)$ is positive and greater than m_s, at least for x immediately preceding the meeting point x^*. For $q_a > 0$, an "u"-shaped parabola, once a' becomes positive its value continues increasing. Therefore, if a and $m_s x + b_s$ meet it must be at $x^* = 1$ to keep the lower bound requirement. For $q_a < 0$, a "n"-shaped parabola, once a' is positive it continually decreases. If a and the linear component of s meet it may be at $a'(x^*) = m_s$, where the linear component of s is tangent to a, or otherwise. If $a'(x^*) > m_s$ then x^* must be 1 to prevent a from becoming greater than $m_s x + b_s$ in **I**. If $a'(x^*) < m_s$ we know the lower bound requirement is not kept since the values $a(x)$ preceding x^* are greater than $m_s x + b_s$. We know this because, going backwards from x^*, $m_s x + b_s$ would decrease more rapidly than $a(x)$ until the two derivatives become equal.

In the previous paragraph, we have shown that a and the linear component of s will intersect at most once in **I**, and that if a' is not equal to the linear component of s at their intersection, then they must intersect at $x^* = 1$. We will show that for a and s and intersection point x^* such that

$$a'(x^*) = 2q_a x^* + m_a = m_s \tag{4.12}$$

$$a(x^*) = q_a (x^*)^2 + m_a x^* = s(x^*) = m_s x^* + b_s \tag{4.13}$$

the value for x^* which will minimize ϵ is 1. To show this we will rewrite the error function in terms of x^*. First, will write m_a in terms of q_a using equation 4.12

$$m_a = m_s - 2q_a x^* \tag{4.14}$$

Substituting into equation 4.13 we get

$$q_a(x^*)^2 + (m_s - 2q_a x^*)x^* = m_s x^* + b_s$$

which simplifies to

$$-q_a(x^*)^2 = b_s$$

from which we can derive

$$q_a = -\frac{b_s}{(x^*)^2} \tag{4.15}$$

Finally, we substitute q_a and m_a into the error function (eq. 4.11), as follows

$$\epsilon = \frac{b_s}{3(x^*)^2} + \frac{1}{2}(m_s - (m_s - 2q_a x^*)) + b_s$$

which simplifies to

$$\epsilon = \frac{b_s}{3(x^*)^2} + q_a x^* + b_s$$

Applying equation 4.15 again we get

$$\epsilon = \frac{b_s}{3(x^*)^2} - \frac{b_s}{(x^*)^2}x^* + b_s$$

which finally simplifies to

$$\epsilon = (\frac{1}{3} - x^*)\frac{b_s}{(x^*)^2} + b_s \tag{4.16}$$

From equation 4.16 it is clear that the higher the value for x^*, the smaller the error becomes. In other words, x^* must be 1 for all parabolas. To summarize, from equations 4.15 and 4.14 we can derive the optimal parameter values

$$q_a^* = -b_s$$

$$m_a^* = m_s + 2b_s$$

for $a' = m_s$ at x^*.

Whether or not to use a concave ("n"-shaped) or convex parabola can be easily explained graphically. As shown in figure 4.1, the error for a concave parabola will be greater than that of a linear approximation. The error for a convex parabola will be even less.

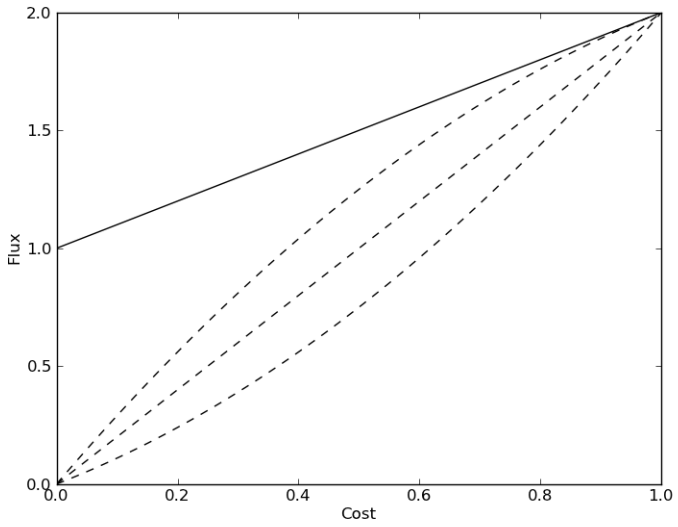

Figure 4.1: Linear, convex quadratic, and concave quadratic approximations

The solid line represents the linear function $s(x) = x + 1$, and the dashed lines represent the three approximations discussed in this chapter.

Formally, the benefit of a concave function can also be shown from 4.11 if we rearrange the terms and set b_a to zero, as follows

$$\epsilon = (\frac{1}{2}m_s + b_s) - \frac{1}{3}q_a - \frac{1}{2}m_a$$

Knowing that $x^* = 1$ we get

$$m_s + b_s = m_a + q_a$$

and since m_s, b_s are constants, it becomes clear that as m_a increases, q_a must decrease in exactly the same proportion, and vice versa. The error ϵ can therefore be minimized by maximizing m_a since it reduces ϵ by a factor of $\frac{1}{2}$, while incrementing q_a only decreases ϵ by a factor of $\frac{1}{3}$. Thus, ϵ for $q_a \leq 0$ will always be less than ϵ for $q_a > 0$, which forms a convex parabola.

EXPERIMENTS

This chapter will ascertain the problem conditions for which the approaches to optimizing metabolic networks discussed in this thesis are practical. First, the parameters which qualify a metabolic network are discussed. Published metabolic networks are then surveyed to establish a set of appropriate network parameters for testing. Finally, the tests on metabolic networks generated according to these parameters will be described along with their results.

Metabolic Network Parameters

Metabolic networks can be described, when abstracted to a hypergraph, by describing the topology of the appropriate hypergraph.

Recall that a hypergraph is defined by

$$\mathcal{H} = (V, E)$$

where $e_i \in E$ is a pair of subsets of V: one for the head and one for the tail. The most basic parameters a hypergraph are therefore the number of vertices and the number of edges which conform the graph. However, graphs that have the same number of vertices and hyperedges can differ significantly in their structure.

To further describe $\mathcal{H}$, we may refer to how well the vertices are distributed. A graph may have vertices distributed evenly amongst hyperedges, or it may not. If we were to express "evenly distributed" in terms of sets, we may say that the vertices are evenly distributed if

$$\forall e_i, e_j \in E,\ \frac{|head(e_i) \cup tail(e_i)|}{|head(e_j) \cup tail(e_j)|} \leq \rho$$

where ρ is the bound for the size difference between the vertices of two hyperedges, and $|head(e_i) \cup tail(e_i)| \geq |head(e_j) \cup tail(e_j)|$. Determining an appropriate value

for ρ may be done according to various criteria. For this thesis, ρ will be determined according to the observed values in published stoichiometric matrices.

An even distribution of vertices should not be confused with modularity. Suppose that, for all $e_i, e_j \in E$, $tail(e_i) = tail(e_j)$ and $head(e_i) = head(e_j)$. This is a distribution of vertices as even as can be made, since ρ will always equal one, its lowest value possible. However, this network is the least modular network possible. Modularity is intuitive when thought about in the context of metabolic networks. In a modular network, most metabolites participate in a relatively small number of reactions. This is because metabolic networks often have *pathways*. Pathways are sequences of reactions which have a distinct functionality, e.g. the production of ATP or the breakdown of lactose[1]. This has been used as an approach to organize metabolic networks, and is especially useful for their visualization and interpretation [16, 17]. For the theory behind metabolic pathways we refer the reader to [8, 18]. Formally, a modular network is one for which

$$\forall e_i \in E, \frac{|\{e_j \in E \mid (head(e_i) \cup tail(e_i)) \cap (head(e_j) \cup tail(e_j)) \neq \emptyset\}|}{|E|} \leq \beta$$

where β represents the maximum fraction of the edges a single edge may share a vertex with. This also bound the number of reactions a metabolite may participate in. Again, setting β will be done according to values derived from published networks.

[1] All sequences of reactions break down precursors and synthesize products, making a specific pathway structure for a network arguably arbitrary. Criteria for pathway definition are beyond the scope of this thesis, however it should be noted that pathways have two types of functionality, one defined by what metabolites they produce/consume, and the other by what those metabolites are used for in the organism. Pathways are usually understood bearing the latter criteria in mind.

Parameters of Published Metabolic Networks

There are several online respositories for metabolic networks to be published. For convenience, we have chosen only to survey those which have been compiled into the online[2] repository at MetaNetX [19]. MetaNetX incorporates models from BiGG [20], BioCyc [21], UniPathway [22], and the semi-automated SEED tool [23, 24].

Number of Metabolites and Reactions

MetaNetX has a "Pick from Repository" page which summarizes the properties of all the networks in the repository. The following histograms summarize the sizes of the reaction and metabolite sets for the models. A total of 505 networks were examined.

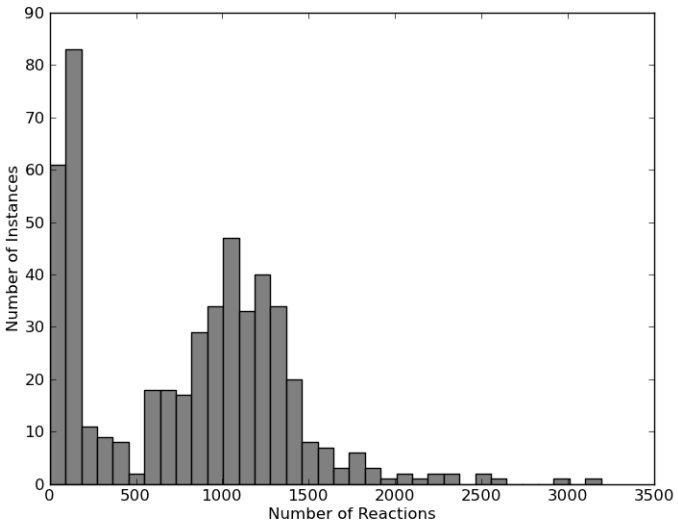

Figure 5.1: Networks Summary: number of reactions

As can be observed, there are two peaks for the number of reactions; one in the 100 to 200 range, and the other at the 1000 to 1300 reactions range. Similarly, the

[2]http://metanetx.org

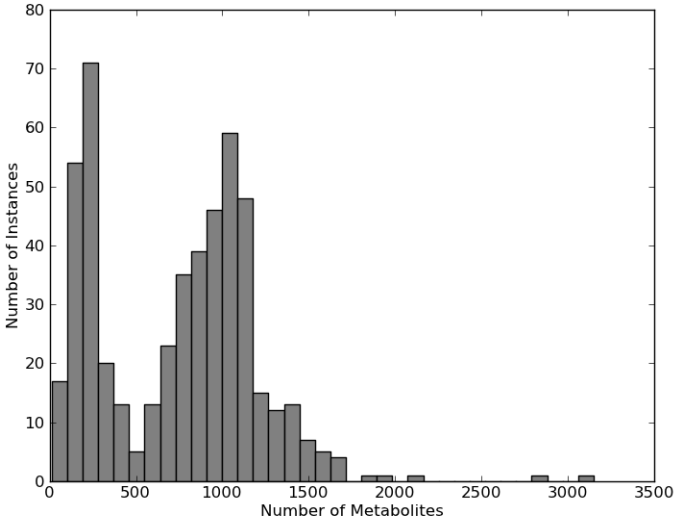

Figure 5.2: Networks Summary: number of metabolites

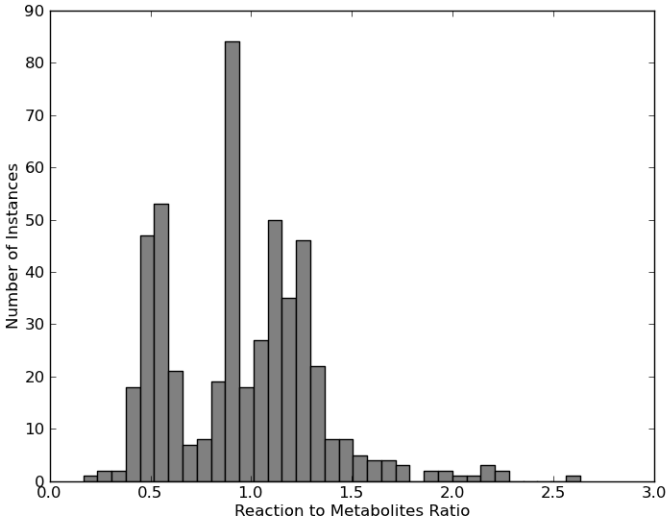

Figure 5.3: Networks Summary: Ratio of reactions to metabolites

number of metabolites has two peaks, one at 100 to 300 and the next at 900 to 1200 metabolites. The ratio of reactions to metabolites is mostly between 0.5 and 1.5, but it is unclear whether the range 1.0 to 1.5 represents two peaks or just one.

Vertex Distribution

While difficult to quantify due to the high variability of the networks available, there is a trend in the distribution of metabolites as figures 5.4, and 5.6 exemplify. Each figure shows the number of reactions whose metabolite set was of size n, for a single network. Even in large networks, the vast majority of metabolite sets that participate in a reaction are not larger than ten times the smallest metabolite set. There is, however, a very small set of reactions whose metabolite set is much larger than ten times the smallest set of metabolites in a reaction. It is likely this is the biomass reaction. Since they model an organism's growth, these reactions use as reactants the products of several pathways to produce the *biomass* metabolite. This trend is not present in networks which lack a biomass reaction, like the one in 5.5. Due to our application, we will focus on networks which do have biomass reactions.

Modularity

As anticipated, networks retained a high modularity. However, as shown by the figures derived from exemplary networks, even though the networks were modular for the most part, there were outliers just as for distribution. Quite possibly these are metabolites like ATP, whose functionality can be applied throughout multiple pathways, and even in several transformations within the same pathway. Figures 5.7, 5.8, and 5.9 all demonstrate the modularity exhibited by a single network by measuring the frequencies at which a metabolite participates in n reactions.

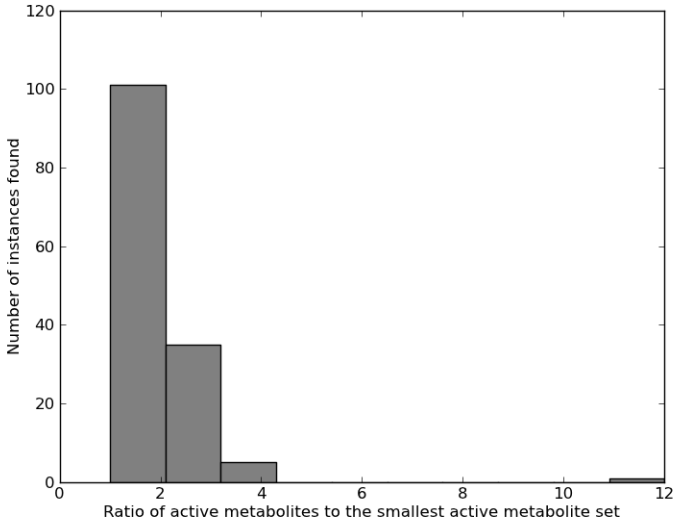

Figure 5.4: Metabolite distribution of sample network 1

Setting Parameters

Since the tests will be used to determine the viability of the approaches presented, there is no bound on the size of the networks tested. It is clear, however, that the performance of the alrogithms on networks with a number of reactions between 100 and 1200 metabolites is key. Also, the ratio of reactions to metabolites should be between 0.5 and 1.5.

Note that these numbers are all applied to networks with reversible reactions. Following the transformation of these networks into networks with all irreversible reactions should skew the numbers since the conversion requires that reversible reactions be "split" into two irreversible ones. Therefore, a network with 100 reversible reactions is actually a network with 200 irreverible ones. Thus, our range of relevant sizes can be expanded to include networks with up to 2400 reactions. This is also true for the ratios, which can be expanded to include ratios of up to 3.0.

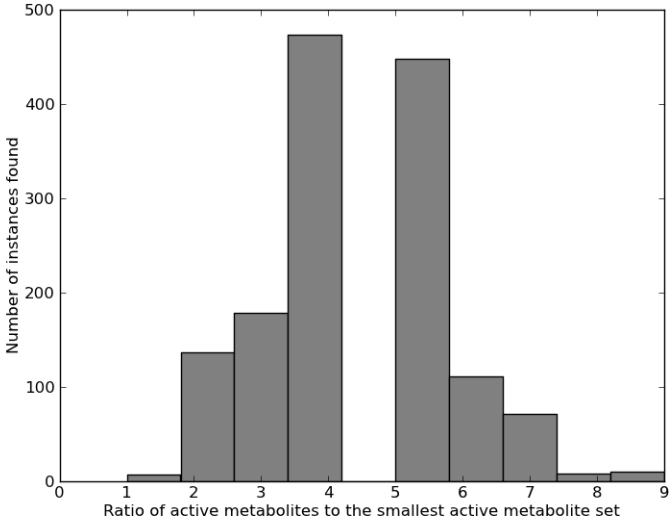

Figure 5.5: Metabolite distribution of sample network 2

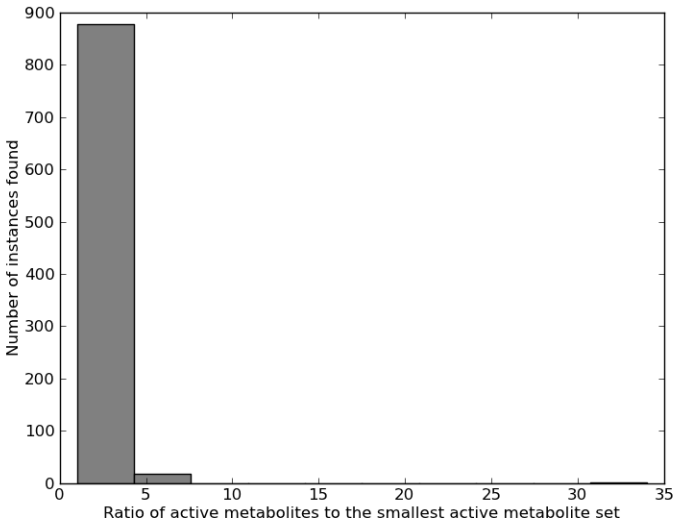

Figure 5.6: Metabolite distribution of sample network 3

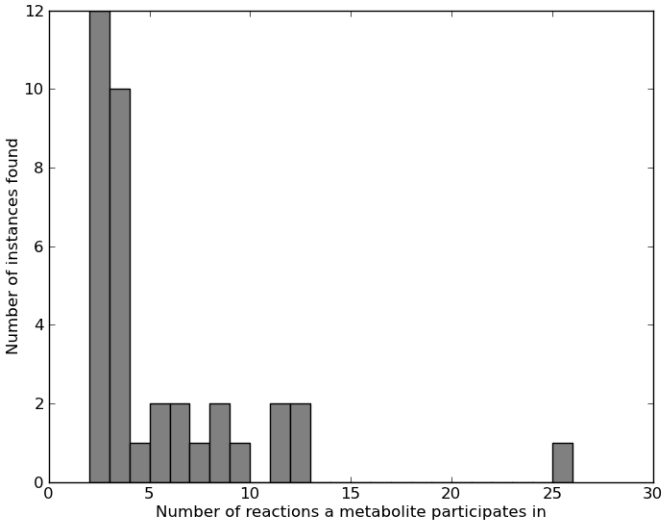

Figure 5.7: Modularity of sample network 1

Setting values for ρ and β can be done by averaging the values over all networks. However, from the graphs for modularity and distribution it is clear that a more precise description can be achieved by complementing these values with a probability function. That is, if ρ and β serve as bounds for the possible values of their respective functions, we may assign each value within the bounds a probability. We begin by assigning the number n of metabolites a reaction may have a weight $w(n)$:

$$w(n) = \begin{cases} 0 & : n > \rho \\ \frac{1}{cn^k} & : 0 < n \leq \rho \\ 0 & : n = 0 \end{cases} \tag{5.1}$$

where c and k are tunable parameters, and $w(n)$ is the weight given to n. The probability that a reaction will have n metabolites involved is proportional to $w(n)$. In the above equation, n may not exceed ρ since ρ is a bound on the size ratio between any two metabolite sets, and the smallest set is assumed to be of size one. This is a valid assumption given the reactions that produce and consume the external

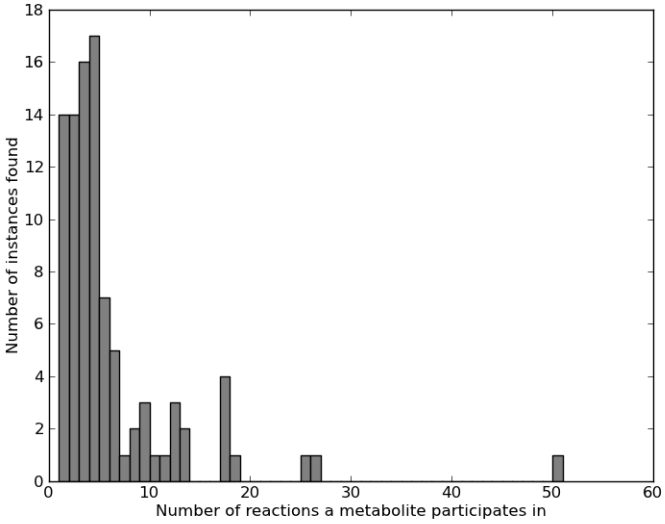

Figure 5.8: Modularity of sample network 2

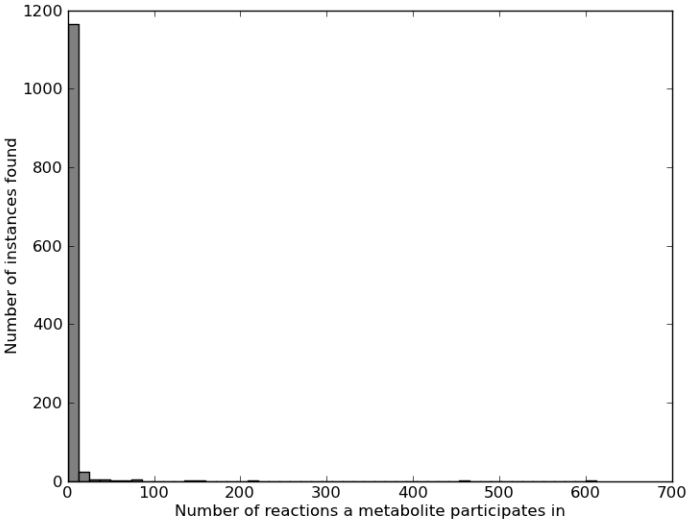

Figure 5.9: Modularity of sample network 3

metabolites, which involve only that external metabolite. The value for ρ varies with the size of each network, but averages $0.42m$, where m is the number of metabolites in the network. Parameters c and k were both set to one for all networks since it was the simplest heuristic and tuning c and k yielded no immediate benefits.

Setting the value for β was done with a similar function. For the networks examined, β averaged $0.46r$, where r is the number of reactions in the network. The greatest number of reactions a metabolite participates in that is not an outlier averages ten. Thus, the probability function becomes

$$w(n) = \begin{cases} 0: & n > \beta \\ \frac{1}{c(n-1)^k}: & 1 < n \leq \beta \\ 0: & n \leq 1 \end{cases} \tag{5.2}$$

where the probabiliy of a metabolite participating in n reactions is proportional to its weight $w(n)$, and c and k are again tunable parameters set to one. No metabolite participates in only one reaction: internal metabolites are both consumed and produced by reactions in the network, and reactions involving external metabolites are complemented by adding consuming or producing reactions where needed. Therefore n must be greater than one to have a positive probability.

The weight function is a measure of relative importance, e.g. $w(n_1) = 2 \times w(n_2)$ implies that n_1 is a value twice as likely to be present in the network than n_2, but the probability of n_1 and n_2 can only be deterimined by taking into account all n in the acceptable range. Deriving $p(n)$ from $w(n)$ can be done as follows,

$$p_d(n) = \frac{w(n)}{\sum_{i=1}^{\rho} w(i)} \tag{5.3}$$

where p_d describes the probability of that a reaction involves n metabolites, and

$$p_m(n) = \frac{w(n)}{\sum_{n=2}^{\beta} w(i)} \tag{5.4}$$

where p_m describes the probability that a metabolite will participate in n reactions.

Testing Eliminated

Generating Networks

Testing ELIMINATED was done on metabolic networks generated according to the parameters discussed, rather than on actual networks from the repository. This was done to ensure a variety of cut effects would be measured. That is, since cuts are restricted to one, two, or three reactions, the vast majority of cuts will eliminate only small portions of the networks. This, in turn, yields a small number of iterations of the main loop in ELIMINATED. A variety of cases (including the worst case where all elements of the network will be eliminated) were tested for all networks by generating metabolic networks from digraphs[3] with a special structure.

All digraphs were generated by iteratively adding children to the set of network ends. This set will contain the nodes in the digraph that do not have children. The set of network ends, from now on referred to as E, begins with a single father node. This node then fathers three children. These children are added to E, while the father node is removed from E. The main iteration then begins: a random node from the E is selected, and, according to a probability[4], it fathers from zero to five nodes. If it fathers a positive number of nodes, it is removed E, and its children are added to E. If it fathers zero nodes, then it "adopts" one of the children already in E. Since the node now has a child, it is from removed from E, where its adopted child remains. Each father-child relationship will represent a reaction in the final metabolic network. Once the desired number of reactions is reached, the digraph will be turned into a hypergraph. The digraph is several interconnected paths, with three beginnings and $|E|$ ends.

[3] A "normal" graph; one for which both the head and tail of all edges consist of a single vertex.

[4] During testing, the probability of having n children was given a weight $\frac{1}{n}$.

Transforming the digraph into a hypergraph with the parameters discussed above is simply a matter of assigning a set of vertices to each node in the digraph. That is, if there exists in the digraph an edge $e_d = \{n_1, n_2\}$, then the corresponding edge in the network hypergraph will be $e_h = \{v(n_1), v(n_2)\}$, where $v(n_i)$ returns the subset of the hypergraph vertices V assigned to n_i. Vertices assigned to nodes will not overlap, i.e. $v(n_i) \cap v(n_j) = \emptyset$ for all n_i and n_j in the digraph. Therefore assigning vertices to nodes becomes a matter of computing how many to assign, which can be done via equation 5.3.

This design gives the hypergraphs generated desirable properties. First, by cutting the first three reactions (the ones with the father node), we can test the worst case. Second, the modularity of the network is controlled by the number of edges in which a node in the digraph participates. A high modularity ensures that metabolites are produced and consumed by a small number of reactions. Setting a limit on the number of children a node may have sets an initial upper bound of five. However, the number of parents a node may have is not strictly limited, since it depends on how many nodes "adopt" it before the node itself fathers children. The expected value for the children a node will have is 2.19, while 0.91 is the expected value for the number of times a node will be adopted. This means the number of parents most nodes will have is two, making them more vulnerable to cuts. This modularity is similar to that of the sample networks, where metabolites have a an average reaction participation of four.

Test Results

Tests on ELIMINATED were run on a Dell Precision T5400 workstation with eight Intel Xeon E5420 processors (2.50 GHz) and 16GB of RAM. The code itself, however, was not parallelized. Subsequent tests run on an Intel i7 quad-core Dell Latitude

E6420 (2.70 GHz) therefore showed no performance loss. The algorithm was implemented in Python, and the time was measured using the standard *time* module.

Figures 5.10, 5.11, and 5.12 summarize the concrete effects of network size and the number of iterations on evaluating the network elements eliminated by a single cut. Figure 5.13 records the time required to compile, from a stoichiometric matrix, the hash table required for the *reactants* and *producers* functions. Testing was done on networks up to six times larger than the previously specified range to denote the trend more clearly.

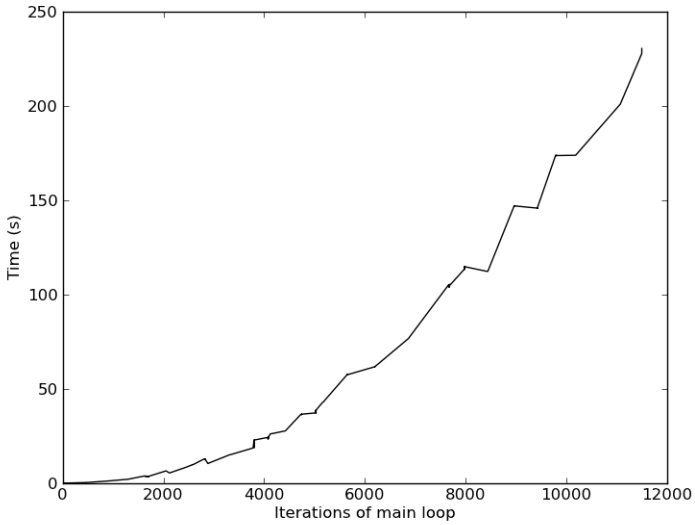

Figure 5.10: Relationship of time to main loop iterations for ELIMINATED

Results Discussion

The practicality of using ELIMINATED as a subroutine in GENERATECUTS is discussed in 5. The tests tend to agree with the complexity analysis in 3. If we treat the time it took to compile the *reactants* and *producers* directory as a function of the network size, we can regress a polynomial curve to estimate the complexity. Fitting

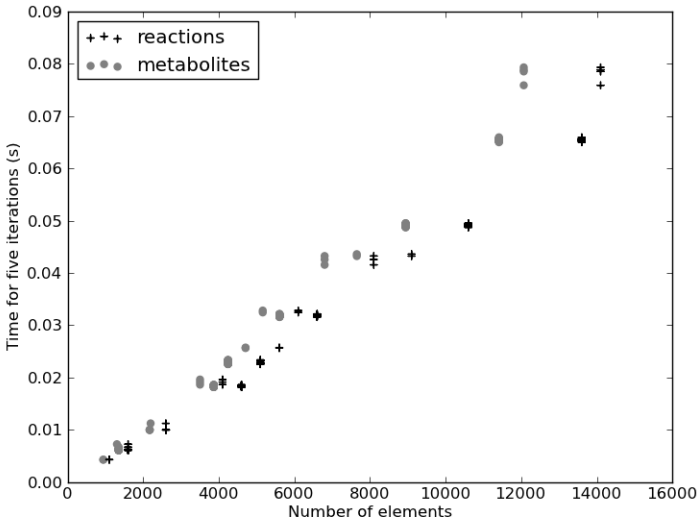

Figure 5.11: Relationship of time to the size of the network

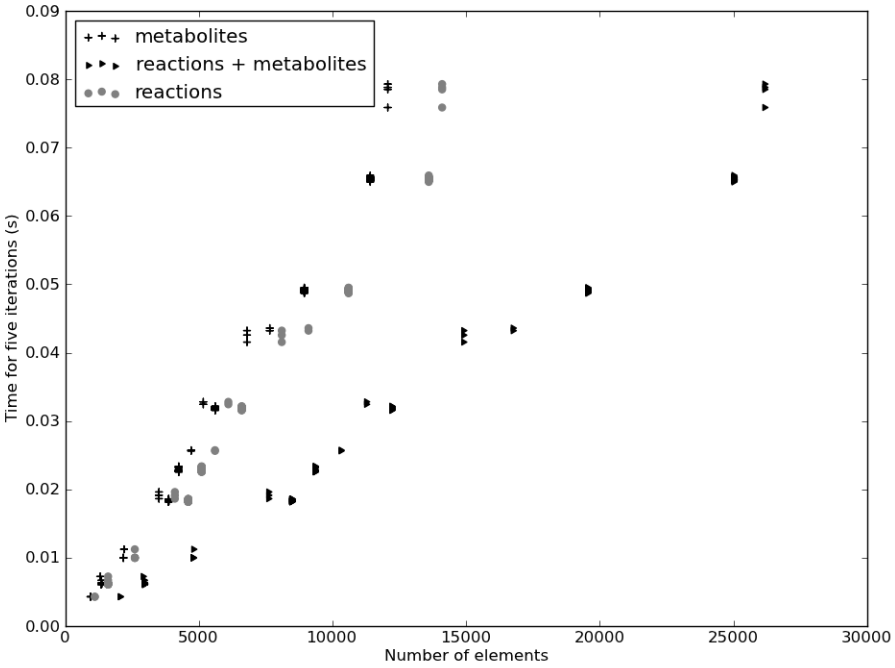

Figure 5.12: Relationship of time to the size of the network

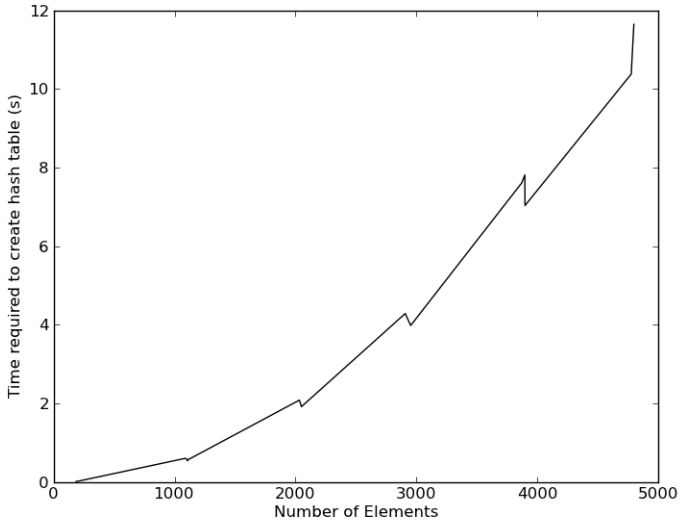

Figure 5.13: Time required to build the *reactants* and *producers* directory

a polynomial curve of degree two yields a residual of 50.19, vastly better than the 570.68 residual the curve of degree one yields. Increasing the degree to three only minimally improves the residuals to 49.28, that is, about two percent, suggesting the n^2 complexity predicted.

The practical complexity of computing the eliminated elements was also approximated. Each iteration of the main loop was predicted to have approximately n^2 complexity. Regressing a polynomial curve to the computation time of ELIMINATED as a function of the number of iterations of the main loop the residual went down from 62.61 to 5.19 when changing the degree of the polynomial from one to two. It improved to 4.11 when incrementing the degree to three, about twenty percent. This significant similarity to n^3 can be attributed to an increase in the network size along with the increased iterations. The residual only improved three percent when incrementing the degree to four.

The time for computing the eliminated elements as a function of the network size showed a linear relationship, as in figures 5.11 and 5.12. The respective residuals of the linear and the quadratic approximations, 2.89 and 2.87, differed by less than one percent. This is less than the predicted with n^2 complexity. In practice, the $n^3 log(n)$ complexity is not encountered in the worst case due to the networks' topology. The worst case complexity is in^2, where i is the maximum number of iterations the main loop may have. This was proven to be up to $2n$, but only when all elements are eliminated and a single network element is discovered per iteration. Because the networks are sequential, even when the top three reactions are cut to eliminate all network elements, multiple reactions and metabolites are added per iteration. Also, the sizes of $previousReactions$ and $previousMetabolites$ are rarely size n. If they ever are, it is on the main loop's last iteration. The $log(n)$ term was not included in the regressions since the sorting occured at most once, and the merging of lists was done by Python internals and is thus very fast compared to the other operations.

Testing GenerateCuts

Networks and Undesired Metabolites Used

Testing GenerateCuts was done on two networks from MetaNetX, called "bigg_textbook" [25], which models the core *E. coli* metabolism, and "bigg_iJR904", which "expands" [26] bigg_textbook. These networks were chosen because this thesis is primarily intended for use in *E. coli* applications. They also have the sizes which are most common to metabolic networks according to our survey. Network bigg_textbook has 95 reactions (142 irreversible) with 94 metabolites, while bigg_iJR904 has 1066 reactions (1320 irreversible) with 906 metabolites.

For testing on bigg_iJR904 the undesired metabolite was chosen from a set known to be eliminated by cutting three reactions. This allows for data on successful cuts,

and has no bearing on the solutions to be tested or how quickly they are evaluated. For bigg_textbook all metabolites were tested one by one as the undesired metabolite, since it was viable to do so within time constraints. Finally, to generate more data on successful cuts, if a cut of one or two reactions was successful, the test code still tested its supersets to see which elements were eliminated.

Test Results

The following tables summarize the results of the GenerateCuts experiments.

Network	Cuts Evaluated	Time (s)	Avg. Time Per Cut	Successful Cuts
bigg_textbook	477,333	54.25	1.13×10^{-4}	10152
bigg_iJR904	8,500,00	18071.00	2.23×10^{-3}	38876

Table 5.1: GenerateCuts: Total time and cuts evaluated

Tables 5.2 and 5.3 summarize the effects of cuts eliminated the chosen undesired metabolite from the network verus those that did not. Statistics on table 5.2 are from cuts for a specific undesired metabolite in bigg_iJR904, while 5.3 has statistics from tests on multiple undesired metabolites.

Effect on bigg_iJR904 (Average)	Successful Cut	Unsuccessful Cut
Metabolites Eliminated	9.78	1.68
Reactions Eliminated	13.12	4.64
Eliminated Iterations	13.42	3.57
Eliminated Time (s)	1.17×10^{-2}	3.01×10^{-3}

Table 5.2: Effects of Successful and Unsuccessful Cuts in bigg_iJR904

Effect on bigg_textbook (Average)	Successful Cut	Unsuccessful Cut
Metabolites Eliminated	1.22	0.33
Reactions Eliminated	5.15	3.54
GetEliminated Iterations	3.04	1.60
GetEliminated Time (s)	2.11×10^{-4}	1.10×10^{-4}

Table 5.3: Effects of Successful and Unsuccessful Cuts in bigg_textbook

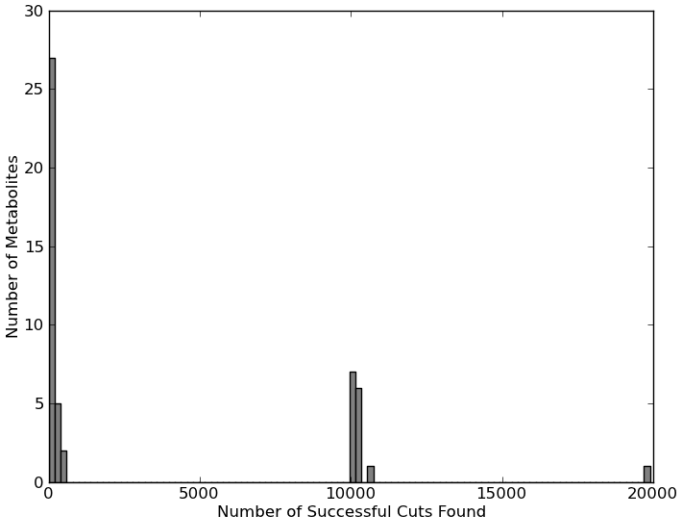

Figure 5.14: Total successful cuts found per metabolite in bigg_textbook

Tests were also run to accurately measure how many of the successful cuts had subsets which were also successful. Tables 5.4 and 5.5 summarize this statistic for the cuts which eliminated the a specific undesired metabolite from each network.

Cut Size	Total	Subset also Succesful	Percentage
1	3	N/A	N/A
2	3955	3954	99.97
3	34918	34909	99.97

Table 5.4: Distribution of Successful Cut Size Evaluated in bigg_iJR904

Cut Size	Total	Subset also Successful	Percentage
1	1	N/A	N/A
2	142	141	99.30
3	10009	9870	98.61

Table 5.5: Distribution of Successful Cut Size Evaluated in bigg_textbook

Figure 5.14 summarizes the number of successful cuts found for each undesired metabolite tested in bigg_textbook. Note that this figure includes only those metabolites for which at least one successful cut was found: a slight majority at 49.

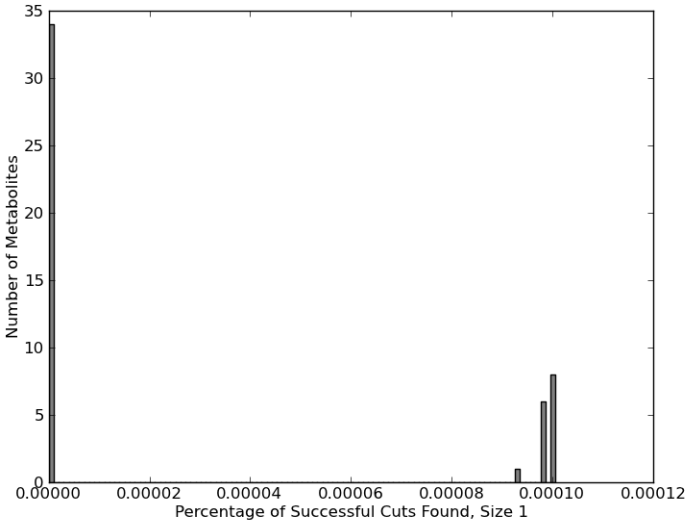

Figure 5.15: Distribution of percentages of successful single-reaction cuts

Figure 5.15 shows what percentage of the successful cuts found consisted of a single reaction, for each of the metabolites tested in bigg_textbook. Figures 5.16 and 5.17 respectively show the percentages for cuts of two and three reactions. As figure 5.17 suggests, some metabolites can only be eliminated by removing three reactions. Figures 5.18 and 5.19 show what percentage of successful cuts had successful subsets, for each metabolite tested.

Results Discussion

Using GenerateCuts to find the set of cuts which will eliminate a set of undesired metabolites Z from the network is clearly practical for smaller networks. Note that *all* possible 477,333 combinations of reactions were examined in under a minute. Running the 94 iterations of GenerateCuts necessary to test each of the metabolites in bigg_textbook as Z took less than two hours. More importantly, GenerateCuts need only be run once to relate each cut to a set of eliminated elements.

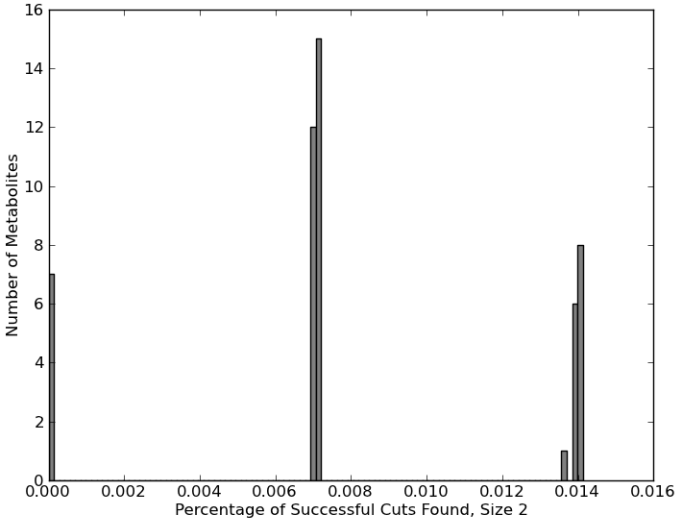

Figure 5.16: Distribution of percentages of successful double-reaction cuts

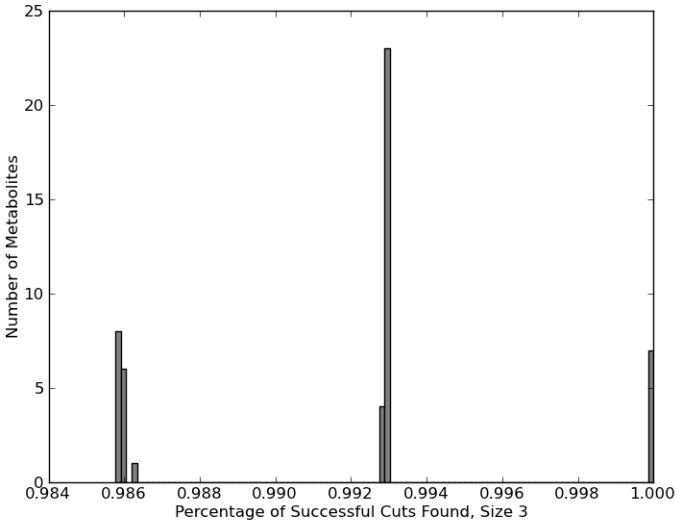

Figure 5.17: Distribution of percentages of successful triple-reaction cuts

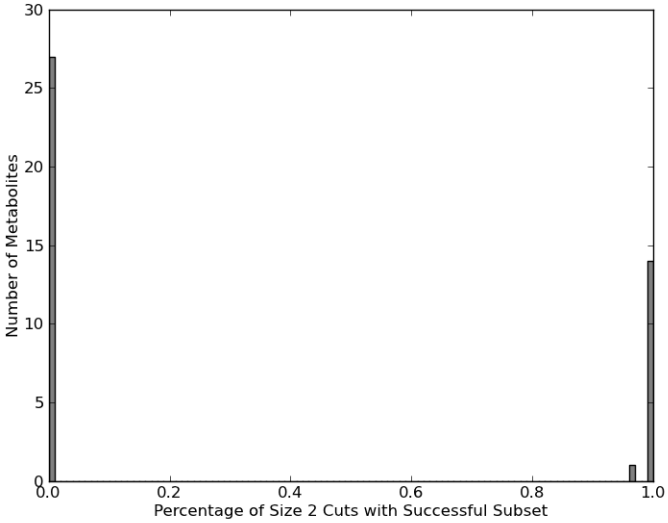

Figure 5.18: Successful Two-Reaction Cuts that are Supersets of a Successful Cut

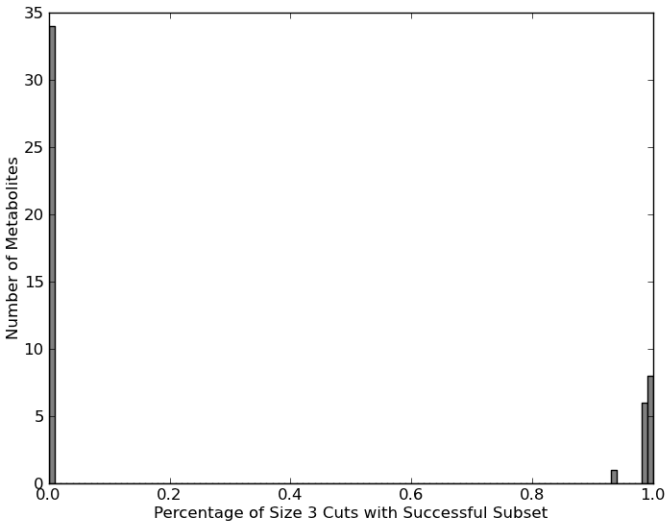

Figure 5.19: Successful Three-Reaction Cuts that are Supersets of a Succesful Cut

If the algorithm is modified to create an index system from cuts to elements and vice versa, all subsequent queries would be as quick as using the index.

In fact, this becomes the only practical way to use GenerateCuts on larger networks. The performance of GenerateCuts is influenced by the time it takes to create the directory and the time it takes to evaluate each cut. However, because even for small networks Eliminated iterates several hundred thousand times, it is the execution time of Eliminated that becomes the determining factor in the performance of GenerateCuts. Recall from the complexity analysis in 3 and the tests in 5 that the execution time of a single iteration of Eliminated increases linearly with the size of the network. If we compound this with the increase in the size of the search space, we can begin to assess the limitations of GenerateCuts. From table 5.1 we can see that the average time taken to evaluate a cut in bigg_iJR904 is 2.13 $\times 10^{-3}$ seconds, 19 times greater than the time for a cut in bigg_textbook, a network 9 times smaller. The search space has incremented by a factor of 803, from 477,333 to 383,329,100 cut combinations. When compounding these the time necessary to evaluate all possible cuts in bigg_iJR904 comes to 9.02 days. This is certainly too long for a query, but may be acceptable as a one time investment for all future queries, especially if the code is parallelized.

The benefit of optimizing using the monotonicity of Eliminated will now be discussed. First, recall that if cut X removes a set Z of network elements, the elements removed by a superset of X will be a superset of Z. To know the full set of elements eliminated by a superset of X, we would have to run Eliminated on that superset. Monotonicity should not be used when building an index, and therefore we will discuss its benefits only in the context of retrieving successful cuts.

The conditions under which this optimization is beneficial are well known, but our tests show that the exact performance improvement is hard to know a priori. Using

monotonicity allows us to ascertain whether a cut will be successful or not *without* having to run ELIMINATED on it, but only if the cut is a superset of an already successful cut. Therefore, optimizing using monotonicity will improve performance only in cases successful cuts exist for the undesired metabolites. Specifically, the optimization will improve performance only when the successful cuts found are single or double-reaction cuts.

We will extrapolate from bigg_textbook for our discussion. Testing showed that 49 of the 94 metabolites (52.11%) had at least one successful cut. Of these metabolites, 34 could not be eliminated by cutting a single reaction, as shown in figure 5.15. This is 69.38% of the 49 that could be eliminated and and 36.17% of the total. Six could not be eliminated by removing two reactions, 12.24% of those which could be eliminated and 6.38% of the total. The total number of cases for which the optimization is useless comes to 51: 45 metabolites which could not be eliminated plus the six cases which have no successful cuts of size two. This is 54.25% of the 94 metabolites studied. Finally, we assess the concrete improvement in the best case, where successful cuts of size one and two exist for the chosen undesired metabolite. Multiplying the average time ELIMINATED takes to process successful cut (see 5.3 and table 5.5) by the number of cuts which came from successful subsets we get an improvement of 2.11 seconds. This is 3.91% of the total running time.

Testing the Quadratic Approximation

Networks and Cost Functions Tested

Testing the accuracy of the quadratic approximation was done on the newtork described in [27]. Even though is a relatively small network, at 62 irreversible reactions and 35 metabolites, its behavior has been studied using FBA and has been used to successfully design *E. coli* strains of efficient growth [27] as well as predicting stress

response [28]. It has also been used as a basis for larger metabolic networks [29]. It was used for this test because its graph [27, 30] is simple enough to help interpret the data results.

The costs for all reactions were assigned randomly. Reaction costs may be based on anything from the energy required to produce its catalyst, to the energy required to power the reaction itself, to the number of molecules which compose the catalyst, or other factors biologically relevant. It is not, however, the focus of this thesis to measure how a model may be improved by the use of cost functions in the objective. The intent is to measure the extent to which these functions, which we have assumed to be step functions, may be accurately described by a quadratic approximation. Specifically, we would like to know whether the error induced by the approximation will lead to a significant difference in the approximate least cost flux profile and the actual least cost profile.

Though assigned randomly, the cost functions used had some restrictions. Even when choosing an approximation's parameters to minimize error, its final accuracy depends on the parameters of the cost function[5]. Cost functions, as linear step functions, are subject to four parameters. Two of these describe the linear component of the step function, i.e. the slope and the y-intercept, from now on referred to as m_s and b_s. The remaining two are the constant value the step function has before becoming linear, set to zero according to our model, and the value at which the step occurs, also zero according to our model. Testing was designed to measure the impact of these parameters on the accuracy of the approximation.

All values for m_s and b_s were set to be between zero and one. This was done to ensure that the trends in accuracy would not be skewed by a reaction cost that

[5]Equation 4.11

dominated the rest. For example, given a reaction whose cost function has $b_s >> 1$, no matter how small its flux is, it will still dominate the cost of the whole profile. The same can be said for $m_s >> 1$. By restricting m_s and b_s, even reactions with the highest possible cost, $m_s = b_s = 1$, running at the highest flux will have a moderate cost of two, which can be offset if the rest of the reactions run at a low enough cost, especially since there are 62 reactions even in this small model. The approximation is thus able to diverge more freely in its choice of which fluxes to keep low.

To test for the impact of values for m_s and b_s sixteen test cases were devised. The values for m_s and b_s were binned into four categories: low (0-0.25), low medium (0.25-0.50), high medium (0.50-0.75), and high (0.75-1). Then, tests were run on each of the combinations of categories, e.g. one for low m_s and b_s values, one for low m_s and low medium b_s values, and so on. The parameter values for each reaction were chosen randomly according to the test being run. As an example, all low values for m_s were picked from the range 0-0.25 according to a uniform random distribution. New random values were picked for every test.

Finally, the objective function was set to be the sum of all cost functions. To avoid the all zero flux solution, the flux of the biomass reaction was constrained to be greater than to 1.0 $\times 10^{-3}$, about 80% of its maximum possible flux of 1.2 $\times 10^{-3}$. This value was chosen to provide some flexibility in the reactions that could form part of the solution, yet still guarantee a high growth rate.

Test Results

Table 5.6 summarizes the cost for the flux profiles derived by finding the optimal solution and compares them to the cost of the profiles derived from minimizing the approximation. Note that the cost given for both profiles is the *actual* cost, not the approximate cost.

m_s	b_s	Optimal Cost	Approx. Cost	Approx./Opt.
Low	Low	2.094	2.158	1.030
Low	Low Medium	4.665	5.266	1.129
Low	High Medium	6.469	7.643	1.181
Low	High	9.205	11.103	1.206
Low Medium	Low	4.344	4.355	1.003
Low Medium	Low Medium	6.956	7.880	1.133
Low Medium	High Medium	9.372	9.377	1.001
Low Medium	High	11.182	11.204	1.002
High Medium	Low	6.946	6.946	1.000
High Medium	Low Medium	9.325	9.336	1.001
High Medium	High Medium	11.214	11.230	1.001
High Medium	High	13.671	13.672	1.000
High	Low	8.559	8.571	1.001
High	Low Medium	11.439	11.450	1.001
High	High Medium	13.551	13.561	1.001
High	High	15.804	15.818	1.001

Table 5.6: Cost of the Optimal and Approximate Solutions, for all Test Cases

Figures 5.20, 5.21, and 5.22 compare the optimal flux profile to the profile derived from mimizing the quadratic approximation, for a sample of the tests. Each flux has two parts, the optimal (left) and the approximation (right).

Results Discussion

Testing revealed a major flaw in our approach. We intended to use interior point methods to solve the approximation's quadratic program. Interior point methods find the optimal solution with polynomial complexity, but impose certain constraints on the quadratic objective function [31]. Among other constraints, the quadratic coefficients, (q_a in Chapter 4) must be zero or positive. Because this would decrease the accuracy of our approximation, we opted to use the active set method solver in the QUADPROG optimizer for MATLAB. While it does not guarantee optimality, a higher number of iterations yields higher optimality. The tests were run in case active

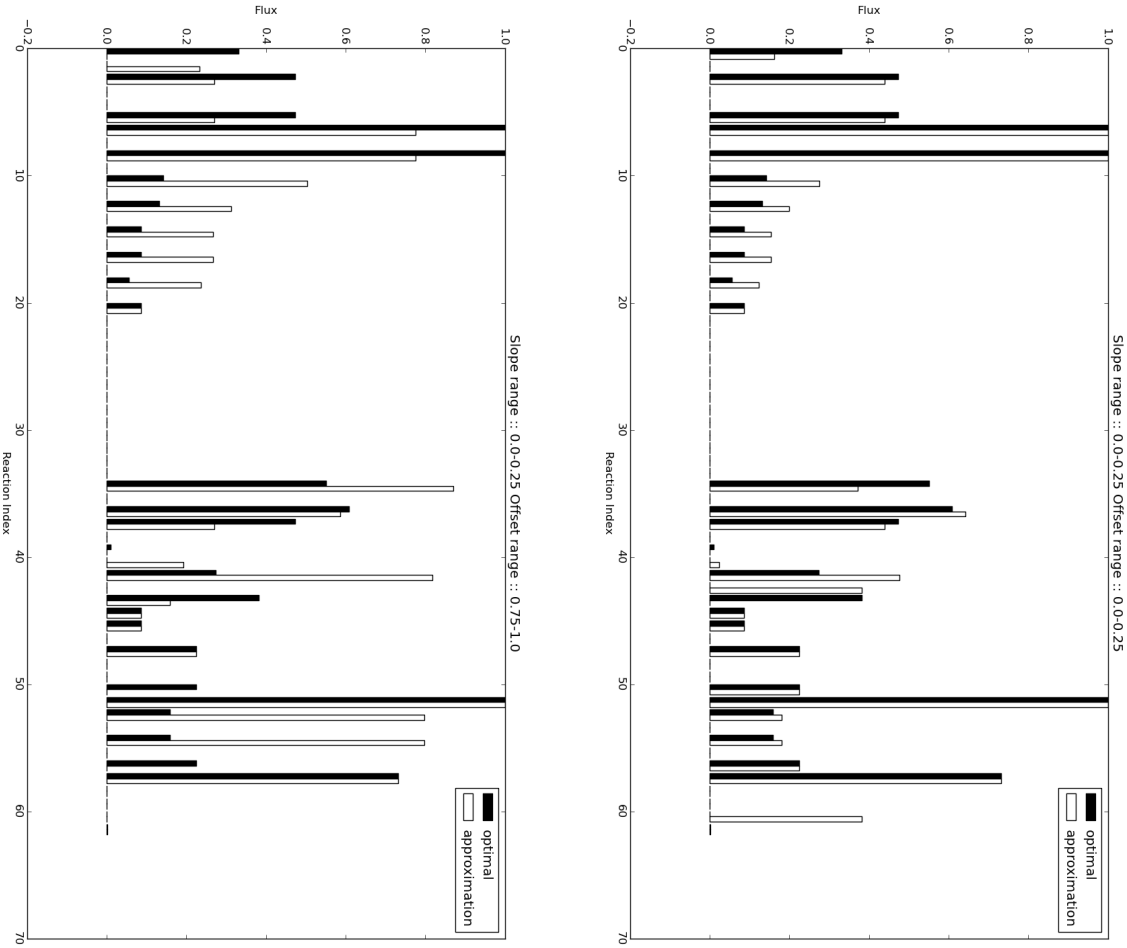

Figure 5.20: Optimal vs. Approx. Fluxes, Low m_s-Low b_s and Low m_s-High b_s

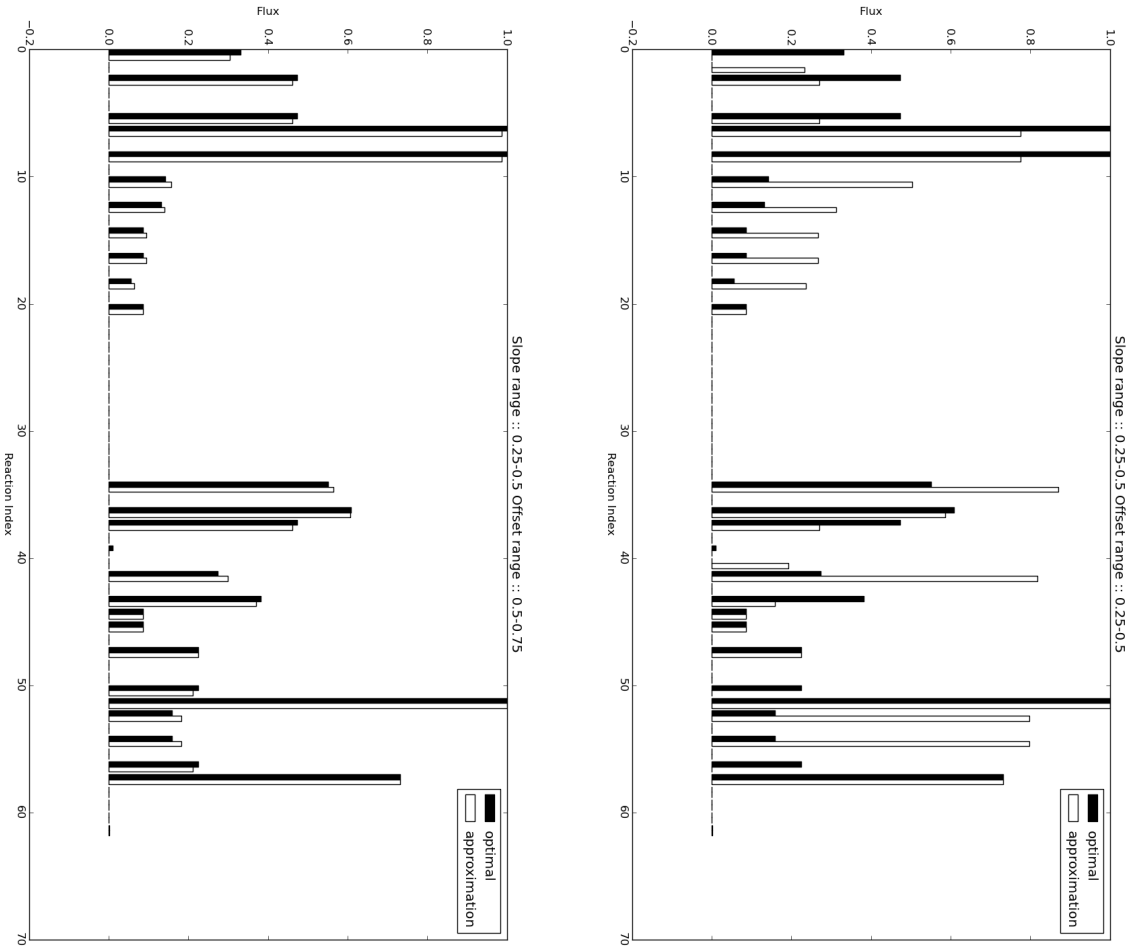

Figure 5.21: Optimal vs. Approx. Fluxes, Low Medium m_s-Low Medium b_s and Low Medium m_s-High Medium b_s

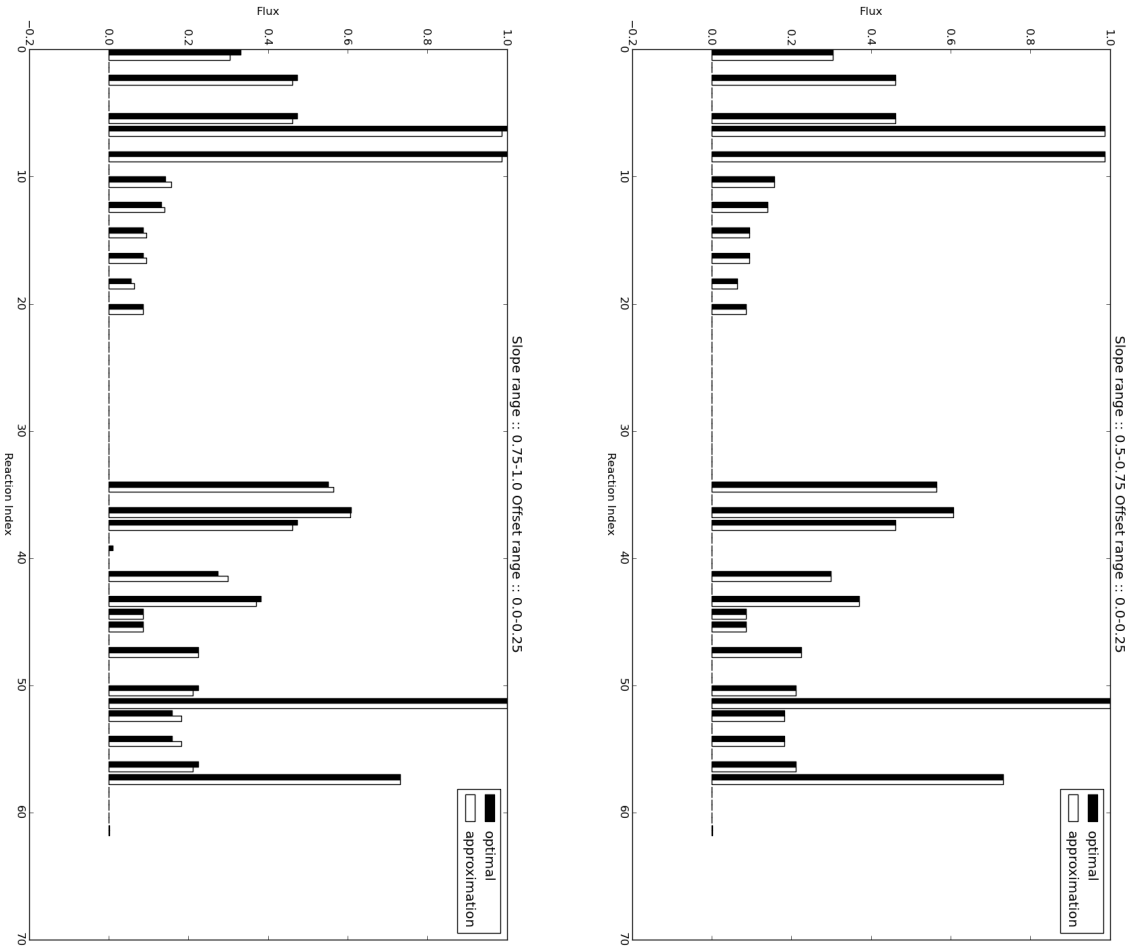

Figure 5.22: Optimal vs. Approx. Fluxes, High Medium m_s-Low b_s and High m_s-Low b_s

set methods could achieve acceptable accuracy in much less time. The binary linear program was also solved using MATLAB's optimizing toolbox.

On average, both the binary linear program and the quadratic approximation were solved in less than one second. However, as the results show, especially figure 5.20, the difference in accuracy can be significant. Therefore, using the approximation is not justified. To fully differentiate the difference in performance, a stress test was performed using one of the largest *E. coli* networks available: seed_Opt83333_1 with 1639 metabolites and 1769 (2972 irreversible) reactions. Solving the binary linear program took an hour with four parallel processes, much shorter than the four hours it took the quadratic program solver to stop. When it did stop, it did so because the maximum number of iterations allowed had been exceeded. Clearly, it is detrimental to both accuracy and time to use a quadratic approximation such as we devised.

Some curious trends in error can be seen in, table 5.6, the details of which can be observed in figures 5.20, 5.21, and 5.22. Namely, the accuracy of the approximation improves with a higher m_s, and the optimal solution may be very nearly found using a quadratic approximation even when $b_s > 0$. The approximation accuracy will decay, however, if m_s becomes too high, even with a low b_s.

Due to the random reassignment of m_s and b_s values, the exact values for the optimal flux profile change between tests. However, the average difference between optimal flux profiles is only 4.96×10^{-3} per flux, that is, 2.61% of the average value of non-zero fluxes of 1.90×10^{-1}. The largest difference in an optimal flux between tests was 2.73×10^{-2} on an average flux size of 3.29×10^{-1}: a change of 8.28%. What this implies is that for the given network and parameter space, the optimal solution is unique. That is, for all tests the optimal soution will give preference to the same set of reactions. Given a unique optimal solution, we may consider discrepancies between the optimal flux profile f_o and the approximation flux profile f_a as errors, not valid

alernatives. Furthermore, if f_a differs from f_o in the same way across two tests, we may infer that the same error was made.

Figures 5.20, 5.21, and 5.22 illustrate the trends in error. Discrepancies between f_a and and f_o, for low m_s are salient to the point of running the forward version of a reaction in f_o and the reverse version in f_a (reaction indices 1 and 2), to f_a having a flux of almost 0.5 on a reaction with flux zero in f_o (index 61), as in fig. 5.20. However, as m_s increases the fluxes in both profiles become very similar, but are never quite equal. Unfortunately it is unclear whether this is due to stopping criteria of the solver or the error from the approximation.

CONCLUSIONS

This chapter will discuss the usefulness of the algorithms presented in the thesis, in light of the results in Chapter 5.

Chapter 5 shows that, for our application, directly solving the binary linear program is preferable to using a quadratic approximation. Even though in larger networks this may take hours to solve, the active set solver by no means provided any benefit to performance. Quadratic approximation in general should be discarded as an approach, since negative q_a values, the only ones which provide a better accuracy than a linear approximation, make the problem non-convex. Research has shown this type of problem to be NP-Hard even for simple cases [32], even though interior point methods have been used as subroutines for non-convex problems [33].

On the other hand, Chapter 5 proves that both Eliminated and GenerateCuts perform well for small metabolic networks. The performance of GenerateCuts, however, decays rapidly with the size of the network since the number of iterations of Eliminated increases combinatorially. This is aggravated by an increase in running time per iteration of Eliminated proportional to the increase in the size of the network. Even though processing a network with 142 reactions takes just under a minute, extrapolating from the results in Chapter 5, networks with 300 reactions and 300 metabolites would take fifty minutes to process and those with 500 reactions and 500 metabolites would take four and a half hours.

The performance enhancements given by the monotonicity optimization are doubtful. Since the optimization only reduces the running time when a small successful cut is found, its overall impact is very limited in most cases. Perhaps the best use of the monotonicity optimization would be to use it to come up with a set of successful cut alternatives quickly. Trying all cuts of one or two reactions first would take much

less time than the depth first approach in GenerateCuts and, assuming that the metabolites in bigg_textbook are not more vulnerable to cuts than those in other networks, it may be that the metabolite we wish to eliminate is one of nearly half of the metabolites in the network that can be eliminated by two or less reactions. Extrapolating, all cuts of size one or two for a network of 500 reactions and 500 metabolites may be evaluated in around two minutes. However, this approach can only be guaranteed to find a subset of the successful reactions, and in networks as large as bigg_iJR904, trying out all cuts of size one and two would still take around a half hour.

Even with its $n^3 log(n)$ complexity, Eliminated is quick enough to completely enumerate the effects of any cut in less than one minute, even if we include building the hash table required as part of the algorithm's cost. Chapter 5 introduced the notion of modifying GenerateCuts to build an index mapping cuts to effects as an investment for all future queries. However, to properly assess whether the investment is worth making, a discussion of the alternative approaches to generating cut sets is necessary. Hädicke and Klamt, in [34], distinguish three approaches to generating cut sets, including their own.

The main idea behind their approach is to eliminate a set of undesired reactions from the network by disabling all the elementary modes in which these reactions occur. Their approach has three steps. First, compute the set of elementary modes. Then, make an undirected hypergraph $\mathcal{H}$ whose edges are the set of elementary modes which contain the reactions we wish to eliminate, and whose vertices are the set of reactions in those modes. An edge e_i will contain a vertex r_j if r_j has a non-zero flux in the elementary mode e_i. Finally, compute the transversal of $\mathcal{H}$. The transversal of $\mathcal{H}$, or $tr(\mathcal{H})$, is a hypergraph that has the same vertices as $\mathcal{H}$, but the edges

are the minimal vertex covers[1] of $\mathcal{H}$. Therefore, selecting an edge from $tr(\mathcal{H})$ and removing all the vertices in that edge from $\mathcal{H}$ is tantamount to removing a minimal set of reactions that form part of all elementary modes that contain the undesired reactions.

Comparing this approach, henceforth TRANSVERSAL, to GENERATECUTS it should be noted that they do not solve exactly the same problem. TRANSVERSAL finds the set of reactions which will eliminate a set of undesired elementary modes. Though removing a target reaction (especially target reactions like "biomass" which cannot be cut directly) was the algorithm's original purpose [13], it is really only a means of choosing which elementary modes are undesirable. Elementary modes might be considered undesirable for other reasons [34]. On the other hand, modifying GENERATECUTS to judge whether a cut is successful or not based on whether it eliminates a set of elementary modes can be done since ELIMINATED computes a list of the reactions that a cut removes, which can then be shown to intersect with all the undesired modes, or not. However, since GENERATECUTS was developed to remove metabolites from the network, the data we have is pertinent for this application only. Therefore, our comparison is done exclusively in this context.

Comparing the complexity of TRANSVERSAL and GENERATECUTS is difficult. In [35], Acuña, et al. mention that the complexity of computing the set of elementary modes is unknown even though it is known that there may not be more than $\binom{|R|}{|M|+1}$ of them. Computing the transversal of a hypergraph is also of unknown complexity; even the complexity for the most popular approach, Berge's algorithm, is unknown [13]. Thus, any comparisons in performance must be done by running time alone.

[1] A vertex cover for a graph $G = (V, E)$ is a set $v_c \subseteq V$ such that for all $e_i \in E$, $e_i \cap v_c \neq \emptyset$. A minimal vertex cover v_c^* is one for which no set $v_s \subset v_c^*$ is also a vertex cover.

GenerateCuts takes less time on average.. Performance tests for Transversal are found in [13], where the tests were run on a Sun Fire V90 with 32GB of RAM and 16 processors (1200MHz). The tests were as follows: the set of elementary modes to be eliminated was given to Transversal as binary array with rows for each mode and columns for each reaction. The minimal cut sets were then derived by computing the transversal of the input modes after some preprocessing steps. The metabolic network used for testing was the *E. coli* central metabolism, consisting of of 110 reactions and 89 metabolites. To assess the vulnerability of the system, the biomass reaction was chosen as the reaction to be eliminated. Four runs were made: each one for modes active under different substrates, e.g. one included only the modes active while the organism grew exclusively on acetate, another for the modes active under glucose growth, etc. Three of the four runs lasted more than ten minutes, the remaining one ran for five seconds.

We may compare the running time for both algorithms because GenerateCuts is able to solve the same problem, even though, unlike Transversal, it requires the stoichiometric matrix of the original metabolic network to do so. Suppose that, along with the matrix, GenerateCuts is given the matrix of active modes that Transversal is given. From this list we may derive the subset of reactions in the original metabolic network we will consider and use that as the input stoichiometric matrix for GenerateCuts. Also, we replace the biomass reaction with the biomass metabolite as our target. The run of Transversal which was solved in five seconds dealt with 104 original reactions. However, the runs which lasted longer than ten minutes also dealt with 104 or 105 reactions. The average running time for Transversal was four hours. The last two runs, which dealt with several thousand modes rather than the several hundred for the first two tests, caused this steep dis-

crepancy. Interestingly, even in the worst case, the Berge algorithm never ran more than 12 minutes, meaning most of the computational time was spent in preprocessing.

Though GenerateCuts outperforms Transversal in the context of finding feasible cuts for eliminating metabolites from a network, it should be recognized that Transversal will find the minimal cut sets, i.e. it will find a set of reactions that remove the metabolites even if they are infeasible by modern technology. Granted, this is considered a useless result for our application, but it may yet be useful to the researcher as technology progresses. The performance of GenerateCuts should be expected to decay quite rapidly with an increase of the number of feasible cuts, since $\binom{|R|}{4} >> \binom{|R|}{3}$ for our range of values. We will defer the comparison with the other two approaches, OptKnock [36] and Minimal Metabolic Functionality [37]. These approaches also generate minimum cut sets, but the criteria for generating cuts is that they must maximize biomass production. As such, comparing them to GenerateCuts is not only more difficult, but also less sensible. While part of the motivation behind GenerateCuts is to improve the efficiency of the organism, this does not inform the generation of cuts directly.

In conclusion, the contribution of this thesis is two-fold. GenerateCuts is shown to be a feasible alternative for the derivation of feasible cuts for the removal of elements from small networks. It is also shown to be worth the investment of computing power to study larger networks still within the 1300 reaction range, especially since the code has high parallelizing potential, and the efficiency of alternative approaches is dependent on the number of elementary modes in these larger networks, which can reach half a million even for networks of size 110 reactions by 89 metabolites [38]. These performance enhancements are perhaps a product of the limitations on the solution space imposed by GenerateCuts.

FUTURE WORK

As pointed out in previous chapters, there are several straightforward improvements that should be applied to GenerateCuts. Perhaps the most useful one is to parallelize the code to allow the user to take full advantage of the computer power he or she may have. A second improvement would be to modify the iterative structure of GenerateCuts. Recursive algorithms usually consume a lot of memory, and there is no requirement for GenerateCuts to iterate through the cuts in any order. In fact, it may be better to iterate through all the smaller cuts first (breadth-first search) since this may yield more varied solutions earlier in the execution. These two improvements may complement each other; we may choose to parallelize the code by separating the cuts to be evaluated into queues and assigning to each queue a process which evaluates Eliminated on each cut in the queue and stores the result. The smaller cuts would be the first to be distributed evenly among the queues. In fact, the traversal itself of the search space is ripe for improvement. With the queue structure, all that is needed is a special process that computes the next cut for any of the processes that request one to evaluate. This can serve as a testing ground for any number of heuristics that we may wish to try, e.g. enumerate the combinations of reactions nearest to the undesired network elements first, and then gradually include the farther ones.

An extension of GenerateCuts to improve subsequent queries is to build an indexing system to map cuts to eliminated elements. Ideally this would allow the user to query both ways. That is, via the indexing system a user would be able to input a set of elements to be removed, and the server would return the appropriate set of cuts to try out, along with the complete list of elements removed by those cuts: the functionality of GenerateCuts. The reverse would be the functionality of

ELIMINATED where the query is a cut and the response would be the set of eliminated elements.

ELIMINATED may also be put to use outside of GENERATECUTS. *Haus, et al.* ran performance tests in [13] on TRANSVERSAL because they were testing it against another algorithm they had developed. This algorithm takes the same input as TRANSVERSAL, but unlike TRANSVERSAL, it does have a complexity bound, and it is, in their words, "surprisingly good". The bound is $m^{o(log(m))}$ where m is the combined size of the elementary modes fed as input and the transversal to be output. This is a good bound considering the possibly combinatorial expansion of the alternatives. This algorithm requires an oracle which will decide whether a given cut is a valid cut set. Haus, et al. used an LP which tried to maximize the flux of the undesired reactions as their oracle. If the flux was non-zero, then the cut was not a valid cut set. ELIMINATED has the possibility of being much faster and therefore improving the performance considerably.

The functionality of GENERATECUTS and ELIMINATED should also be complemented by a module designed to rank the cuts returned. All other approaches to generating cuts seem to take the biomass objective (or some other objective besides minimizing the flux of undesirable reactions) into consideration as a way of ranking their cuts. A ranking system may be put in place after GENERATECUTS has generated the list of candidates to help the researcher distinguish among candidates, the simplest of these ranking systems being the biomass flux. Interestingly, in [34] the cuts returned are required to preserve a certain number of desired modes. A similar policy of preservation is straighforward to implement in GENERATECUTS since it is a matter of confirming that certain reactions are *not* members of the set of eliminated elements.

REFERENCES CITED

[1] Pei-Qi Liu, Edmond M. Chan, Gregory J. Cost, Lin Zhang, Jianbin Wang, Jeffrey C. Miller, Dmitry Y. Guschin, Andreas Reik, Michael C. Holmes, John E. Mott, Trevor N. Collingwood, Philip D. Gregory. Generation of a triple-gene knockout mammalian cell line using engineered zinc-finger nucleases. *Biotechnology and Bioengineering*, 106(1):97–105, 2010. doi:10.1002/bit.22654.

[2] ClaudiaE. Vickers, Daniel Klein-Marcuschamer, JensO. Krmer. Examining the feasibility of bulk commodity production in escherichia coli. *Biotechnology Letters*, 34:585–596, 2012. doi:10.1007/s10529-011-0821-3.

[3] Jennifer L. Reed, Bernhard . Palsson. Thirteen years of building constraint-based in silico models of escherichia coli. *Journal of Bacteriology*, 185(9):2692–2699, 2003. arXiv:http://jb.asm.org/content/185/9/2692.full.pdf+html, doi:10.1128/JB.185.9.2692-2699.2003.

[4] B. Ras, C. Chassagnole, J.-P. Mazat. Control of threonine pathway in e. coli. application to biotechnologies. *Acta Biotheoretica*, 43:285–297, 1995. doi:10.1007/BF00713554.

[5] L.P. Yomano, S.W. York, S. Zhou, K.T. Shanmugam, L.O. Ingram. Re-engineering escherichia coli for ethanol production. *Biotechnology Letters*, 30:2097–2103, 2008. doi:10.1007/s10529-008-9821-3.

[6] GeorgeN. Bennett, Ka-Yiu San. Engineering e. coli central metabolism for enhanced primary metabolite production. SangYup Lee, editor, *Systems Biology and Biotechnology of Escherichia coli*, pages 351–376. Springer Netherlands, 2009. doi:10.1007/978-1-4020-9394-4_17.

[7] Jone-Long Ko, ML Harter. Plasmid-directed synthesis of genuine adenovirus 2 early-region 1a and 1b proteins in escherichia coli. *Molecular and cellular biology*, 4(8):1427–1439, 1984.

[8] Stefan Schuster, David A Fell, Thomas Dandekar, i in. A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. *Nature biotechnology*, 18(3):326–332, 2000.

[9] Giorgio Gallo, Justino Longo, Steffano Pallotino. Directed hypergraphs and applications. *Discreet Applied Mathematics*, 42(2-3):177–201, 1993.

[10] Bernhard Korte, Jens Vygen. Linear programming algorithms. *Combinatorial Optimization*, Volume 21 series *Algorithms and Combinatorics*, pages 73–99. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-24488-9_4.

[11] W.D. Wallis. Linear programming. *A Beginner's Guide to Finite Mathematics*, pages 281–357. Birkhuser Boston, 2012. doi:10.1007/978-0-8176-8319-1_6.

[12] Stefan Schuster, Claus Hilgetag. On elementary flux modes in biochemical reaction systems at steady state. *Journal of Biological Systems*, 2(02):165–182, 1994.

[13] Utz uwe Haus, Steffen Klamt, Tamon Stephen. Computing knock-out strategies in metabolic networks. *Journal of Comp. Biology*, pages 259–268, 2008.

[14] Marco Dorigo, Mauro Birattari, Thomas Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[15] George L Nemhauser, Laurence A Wolsey. *Integer and combinatorial optimization*, Volume 18. Wiley New York, 1988.

[16] Markus Rohrschneider, Christian Heine, André Reichenbach, Andreas Kerren, Gerik Scheuermann. A novel grid-based visualization approach for metabolic networks with advanced focus&context view. *Graph Drawing*, pages 268–279. Springer, 2010.

[17] Minoru Kanehisa, Susumu Goto. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000.

[18] Christophe H Schilling, David Letscher, Bernhard O Palsson, i in. Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. *Journal of theoretical biology*, 203(3):229, 2000.

[19] Mathias Ganter, Thomas Bernard, Sébastien Moretti, Joerg Stelling, Marco Pagni. Metanetx. org: a website and repository for accessing, analysing and manipulating metabolic networks. *Bioinformatics*, 29(6):815–816, 2013.

[20] Jan Schellenberger, Junyoung O Park, Tom M Conrad, Bernhard Ø Palsson. Bigg: a biochemical genetic and genomic knowledgebase of large scale metabolic reconstructions. *Bmc Bioinformatics*, 11(1):213, 2010.

[21] Peter D Karp, Christos A Ouzounis, Caroline Moore-Kochlacs, Leon Goldovsky, Pallavi Kaipa, Dag Ahrén, Sophia Tsoka, Nikos Darzentas, Victor Kunin, Núria López-Bigas. Expansion of the biocyc collection of pathway/genome databases to 160 genomes. *Nucleic acids research*, 33(19):6083–6089, 2005.

[22] Anne Morgat, Eric Coissac, Elisabeth Coudert, Kristian B Axelsen, Guillaume Keller, Amos Bairoch, Alan Bridge, Lydie Bougueleret, Ioannis Xenarios, Alain Viari. Unipathway: a resource for the exploration and annotation of metabolic pathways. *Nucleic acids research*, 40(D1):D761–D769, 2012.

[23] Matthew DeJongh, Kevin Formsma, Paul Boillot, John Gould, Matthew Rycenga, Aaron Best. Toward the automated generation of genome-scale metabolic networks in the seed. *BMC bioinformatics*, 8(1):139, 2007.

[24] Christopher S Henry, Matthew DeJongh, Aaron A Best, Paul M Frybarger, Ben Linsay, Rick L Stevens. High-throughput generation, optimization and analysis of genome-scale metabolic models. *Nature biotechnology*, 28(9):977–982, 2010.

[25] Jeffrey D Orth, R Fleming, Bernhard Palsson. Reconstruction and use of microbial metabolic networks: the core escherichia coli metabolic model as an educational guide. http://dx.doi.org/10.1128/ecosal.10.2.1.

[26] Jennifer L Reed, Thuy D Vo, Christophe H Schilling, Bernhard O Palsson, i in. An expanded genome-scale model of escherichia coli k-12 (ijr904 gsm/gpr). *Genome Biol*, 4(9):R54, 2003.

[27] Ross Carlson, Friedrich Srienc. Fundamental escherichia coli biochemical pathways for biomass and energy production: identification of reactions. *Biotechnology and bioengineering*, 85(1):1–19, 2004.

[28] Nathan D Price, Jennifer L Reed, Bernhard Ø Palsson. Genome-scale models of microbial cells: evaluating the consequences of constraints. *Nature Reviews Microbiology*, 2(11):886–897, 2004.

[29] Cong T Trinh, Pornkamol Unrean, Friedrich Srienc. Minimal escherichia coli cell for the most efficient production of ethanol from hexoses and pentoses. *Applied and Environmental Microbiology*, 74(12):3634–3643, 2008.

[30] Ross Carlson, Reed Taffs. Metabolic systems analysis workshop. Department of Chemical and Biological Engineering Workshop, Montana State University, 2010.

[31] Lieven Vandenberghe, Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996.

[32] Panos M Pardalos, Stephen A Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[33] Yinyu Ye. On affine scaling algorithms for nonconvex quadratic programming. *Mathematical Programming*, 56(1-3):285–300, 1992.

[34] Oliver Hädicke, Steffen Klamt. Computing complex metabolic intervention strategies using constrained minimal cut sets. *Metabolic engineering*, 13(2):204–213, 2011.

[35] Vicente Acuña, Flavio Chierichetti, Vincent Lacroix, Alberto Marchetti-Spaccamela, Marie-France Sagot, Leen Stougie, i in. Modes and cuts in metabolic networks: Complexity and algorithms. *Biosystems*, 95(1):51, 2009.

[36] Anthony P Burgard, Priti Pharkya, Costas D Maranas. Optknock: a bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Biotechnology and bioengineering*, 84(6):647–657, 2003.

[37] Cong T Trinh, Ross Carlson, Aaron Wlaschin, Friedrich Srienc. Design, construction and performance of the most efficient biomass producing¡ i¿ e. coli¡/i¿ bacterium. *Metabolic engineering*, 8(6):628–638, 2006.

[38] Steffen Klamt, Jörg Stelling. Combinatorial complexity of pathway analysis in metabolic networks. *Molecular biology reports*, 29(1-2):233–236, 2002.