11-23-2008

Cycles in Metabolic Networks

Tim Nunamaker
Trinity University

Cycles in Metabolic Networks

Tim Nunamaker

Abstract

Modeling sophisticated biological systems in a way that makes them more apprehensible, without losing comprehension or robustness, is a challenging task. The flux balance analysis (FBA) model describes metabolic networks of single-celled organisms with the goal of simulating their steady-state behavior. FBA simulations yield reliable and biologically relevant growth rate values, but do not simulate intra-cellular behavior well.

Solutions to the FBA model are often degenerate and non-unique. Ideally, the model should simulate metabolic activity in exactly one way under any given circumstances–the way the real organism's metabolism behaves. The ultimate goal of this project is to better understand the FBA model in order to improve it, so that this ideal may be more achievable.

An overview of biological networks, metabolism, and FBA is given to familiarize the reader. Dominance, the amount that a reaction dominates consumption or production of its reactants and products over other reactions, is used to define a partition of the metabolism that groups reactions with similar properties. The term minimal environment is defined to describe a set of metabolic resources that are limiting for a certain growth rate, and an algorithm is developed for finding the minimal environment of a metabolic network.

Dominance is found to be an indicator for certain regions of the network that are cyclic, and thus problematic. An algorithm is developed for finding cycles in a graph, which is then applied to the metabolic network. The complexity of this algorithm when applied to E. coli makes it too computationally difficult to be used, but the algorithm is useful in other respects.

Acknowledgments

I wish to thank my parents and the rest of my family for always encouraging me and for having helped me to become the person I am, which of course includes being so supportive of my academic pursuits. I would also like to thank all of my friends for making my time at Trinity a blast and all of the work well worth it.

All of the computer science, mathematics, and engineering faculty I've taken classes from and have been involved with deserve my gratitude for being incredible educators. I also extend my thanks to Dr. Massingill for her Thesis template, which made typesetting this paper much easier.

I would like to thank Dr. Lewis for taking me on as an advisee, especially while on academic leave, and for making computer science classes both interesting and rewarding, a difficult task. I'd also like to thank Dr. Healy and Dr. Livingstone in Biology. They both played a critical role in elucidating the biology for me on many points of our research.

Finally, I would like to give Dr. Holder an enormous thank you for the years of teaching me, working with me, advising me, and making mathematics something I will always keep close. I am confident in saying that no one person has ever made such an impact on my education.

Cycles in Metabolic Networks

Tim Nunamaker

A departmental thesis submitted to the

Department of Computer Science at Trinity University

in partial fulfillment of the requirements for Graduation.

November 23, 2008

Thesis Advisor

Department Chair

Associate Vice President

for

Academic Affairs

Cycles in Metabolic Networks

Tim Nunamaker

Contents

List of Tables

List of Figures

Chapter 1

Metabolic Networks in Systems Biology

The natural science community has found itself at an advantageous position due to advances in computing capabilities. Large quantities of data to describe the microscopic details of cellular processes are available. Biology, particularly, is reaping the benefits, as demonstrated by the ongoing growth of computational biology.

Systems Biology is an interdisciplinary field that examines the behavior of organisms as systems. Specifically, systems biology investigates the mechanisms that underlie the functions of life. Biological entities are complex, and models that describe them can are generally large. Gaining insight from mathematical models of biological systems is possible as computing resources become more accessible and powerful. One such technique, Flux Balance Analysis (FBA), describes the metabolism of single-celled organisms and simulates cellular behavior. The model is normally assigned an objective of maximizing growth, which matches experimental values, although other objectives are sometimes considered. The FBA model is presently unable to provide exactly biological information. This is partially due to the model providing multiple simulated optimal solutions, instead of just one.

FBA aims to reveal information about the inner-workings of the cell that cannot be observed in biological environments, and if it can be improved to become a viable tool, would provide valuable information. Improving this model is the overarching goal of this project.

1.1 Networks

Networks are commonly used in applications of systems biology. We begin with a basic introduction to graph theory in support of our work. A *graph* is a pair of sets, a set of vertices V and set of edges E, where any $e \in E$ is associated with a pair of vertices, which "connects" them. *Directed* graphs are composed of edges that are ordered pairs so that $E \subseteq \{(a, b)|a, b \in V\}$. Graphs that are *undirected* are composed of edges that may be denoted by sets (unordered by definition) so that $E \subseteq \{\{a, b\}|a, b \in V\}$. Figure 1.1 shows an undirected graph and a directed graph. Notice that the edges in the directed graph have arrowheads to denote directionality.

For an undirected graph, the *degree* of a vertex is the number of edges that are associated with the vertex. For a directed graph, the *in-degree* and *out-degree* of a vertex are the number of edges directed into and out of it, respectively.

A *network* is a *weighted* directed graph, where a weighted graph is a graph that has a weight, or number, corresponding to each edge [4]. For example, weights are often used to indicate levels of flow. A graph whose vertices represent cities and edges represent roads between them might have each edge weight denote the average daily traffic on the road.

A *path* is a sequence of vertices in a graph so that each vertex shares an edge with the

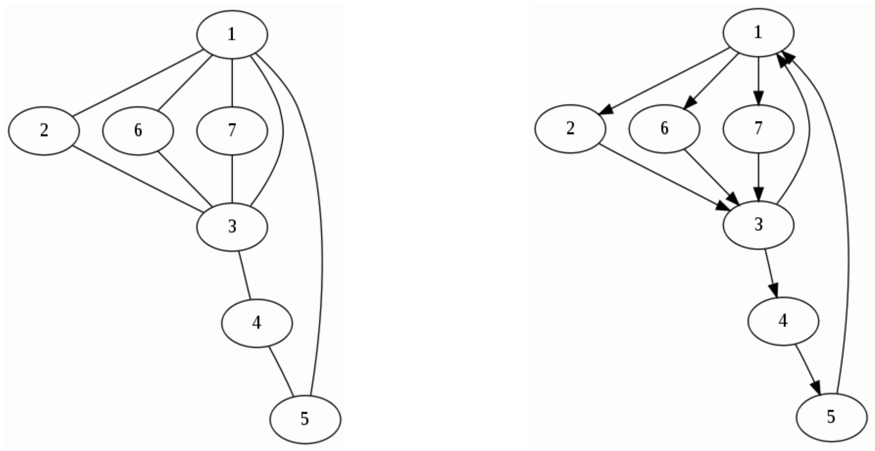

Figure 1.1: An undirected graph (left), and a directed graph (right).

subsequent vertex. A graph that is *complete* contains an edge for every pair of vertices (for directed graphs, the edge is bidirectional). A *cycle* is a path where the first vertex is also the last. Cycles are discussed at length in Chapter 2.

1.1.1 Biological Networks

Multiple layers of chemical activity across entire organisms such as Escherichia Coli (E. coli) can be captured in a network model. The three primary networks are the gene regulatory network, protein-protein interaction network, and metabolic network. Protein-protein interaction networks consist of vertices that represent intra-cellular proteins and edges that represent various types of interactions between proteins. These include interactions such as one protein modifying another or multiple proteins acting as a transducer for conveying signals from the exterior to the interior of the cell. Gene regulatory networks describe mRNA transcription by representing DNA and other components, such as useful proteins, as vertices, and representing various interactions as edges [8]. Metabolic networks consist of enzymes and metabolites, where one representation consists of representing the enzymes as vertices and metabolites as edges. Studying metabolic networks is the focus of this project.

1.2 The Metabolism

Biologist Christopher Mathews describes a *metabolism* to be "the totality of chemical reactions occurring within a cell" [6]. Reactions occur naturally within organisms and are catalyzed, or accelerated, by enzymes[1]. In general, each enzyme present within a cell is responsible for ensuring that an associated reaction takes place when the necessary chemicals are present. These reactants, as well as the products that the reaction produces, comprise the cell's *metabolites.* The set of metabolites includes molecules that act as raw materials, intermediary materials, and those that are excreted. Some metabolites are consumed to produce energy, such as Adenosine triphosphate (ATP), the main energy-storage molecule within cells. The process of interconverting metabolites is facilitated by the metabolism.

A chemical reaction is described by a stoichiometric equation, where each metabolite consumed and produced is weighted by a non-negative integer. For example, the stoichiometric equation

$$A + 2B \to C \tag{1.1}$$

describes a reaction where one unit of metabolite A reacts with two units of metabolite B to produce one unit of metabolite C. The *stoichiometric coefficient* of a reaction and a metabolite is the quantity that describes how many units of the metabolite are consumed or produced, which are the coefficients $-1, -2$ for the input metabolites and 1 for the output metabolite in Equation 1.1 (these coefficients can be signed to model input versus output metabolites). The *flux* of a reaction is the rate at which it occurs. It is essentially a multiplier that scales the stoichiometric coefficients to the quantities of metabolites that

[1]The terms "enzyme" and "reaction" will be used interchangeably with regard to the model.

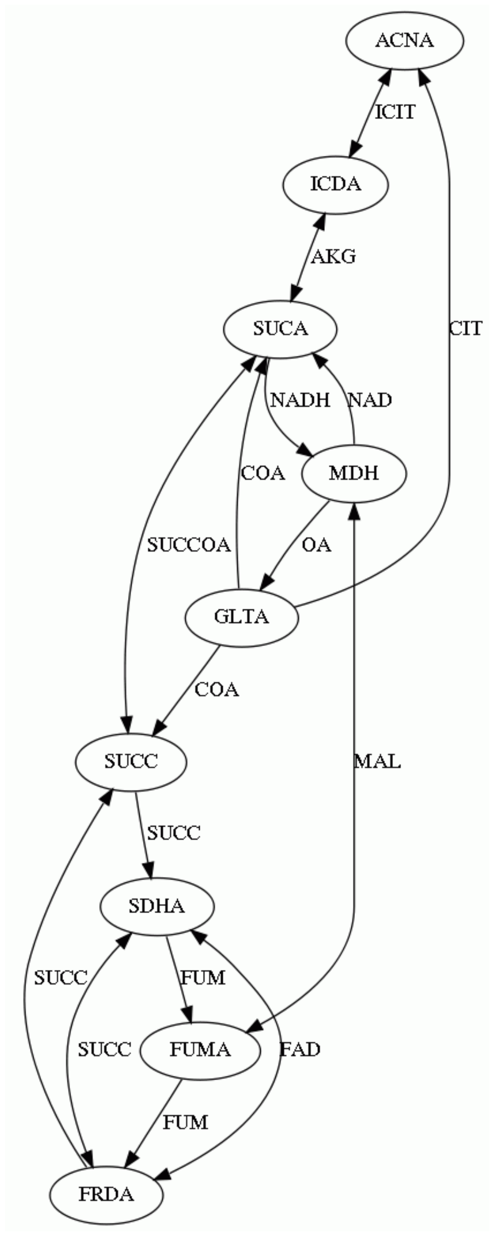

Figure 1.2: The TCA cycle as a network. Vertices are reactions, edges are shared metabolites.

are actually consumed and produced.

One representation of enzymes and metabolites is a network where vertices denote the reactions produced by each enzyme, and edges between vertices denote the metabolites that are consumed and produced by the reactions. Figure 1.2 illustrates a representation of the TCA cycle in E. coli as such a network.

Another representation has vertices denoted both reactions and metabolites, and each edge connects a single reaction and metabolite, with directionality indicating whether the metabolite is consumed by the reaction or produced by it. Figure 1.3 illustrates a bipartite representation of a subset of a metabolism.

Consider such a network where the weights assigned to each edge represent the volume of the metabolite that is produced by the first reaction and consumed by the second. This volume is the product of the flux of the reaction and the stoichiometric coefficient of the reaction-metabolite pair. Such a network is described by the LP discussed in Section 1.4, and in the object-oriented model in Section 2.3.1.

In practice, it is unnecessary to consider weights for metabolic networks since the model is used to determine the fluxes.

1.3 Linear Programming

A linear program (LP) is an optimization problem in which the function being optimized, called the *objective function*, is constrained linearly. The standard form of a linear program is

$$\min\{c^T x : Ax = b, x \geq 0\},$$

where c is an n-vector, b is a m-vector and A is an $m \times n$ matrix. The vector bound $x \geq 0$ is the same as $x_i \geq 0$ for each i.

For example, suppose an individual wishes to buy two foods in different quantities, and that he/she will buy A units of the first food and B units of the second. He/she requires at least C calories and R grams of protein and wants to fulfill these constraints by spending the least dollar amount possible. Let P_A denote the price of one unit of the first food and let P_B denote the price of one unit of the second food. Choosing the objective function as the total price $AP_A + BP_B$, he/she could solve the following LP to determine which foods to buy and in what quantities:

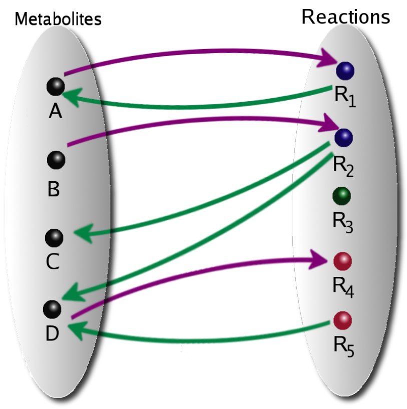

Figure 1.3: A bipartite graph of metabolite-reaction pairs. Purple edges indicate reactions that dominate consumption and green edges indicate reactions that dominate production. Blue reactions are in the HFN, red ones are in the IFN, and the green reaction is in the LFN. See Chapter 2 for definitions and a discussion of dominance and the HFN, IFN, and LFN.

$$\begin{aligned}
&\text{minimize:} && AP_A + BP_B \\
&\text{subject to:} && AR_A + BR_B \geq R \\
& && AC_A + BC_B \geq C \\
&\text{where} && A \geq 0 \text{ and } B \geq 0,
\end{aligned}$$

The terms R_i and C_i denote the grams of protein and calories, respectively, in food i. When solved, this LP returns the amounts of foods A and B that provide C calories, R grams of protein and the least cost.

We use an LP to construct an FBA model of a cellular metabolic network. This is discussed in section 1.4 and Chapter 3.

1.4 Flux Balance Analysis

Flux Balance Analysis is an approach to estimate cell behavior by constructing and solving an LP that encompasses the information of the metabolic network described in section 1.2.

The objective of the LP is usually assumed to be the maximization of growth, which is a natural interpretation of the behavior of single-celled organisms. Laboratory data has demonstrated that this assumption yields a strong correlation between simulated and actual growth values for different *environments* [7], where an environment is a collection of bounds that limit the rate at which metabolites can be transported into the cell through the cellular membrane and into the cytoplasm.

Environmental resources are bounded to within the ranges prescribed by the environment, which form some of the constraints in the LP. Realistic environments must have lower bounds of zero for resources, and upper bounds must be non-negative but may be finite or infinite. A resource with an infinite upper bound indicates growth is limited by the mechanics of the cell and not by resource availability.

An important property of the model is that it is treated as steady state, which is reflected by

$$\frac{\partial C_i}{\partial t} = 0, \tag{1.2}$$

where C_i is the concentration of a metabolite i. Equation 1.2 describes how the concentration of every metabolite is constant over time.

The FBA model solves for optimal growth and records the fluxes of all of the enzymes in the network.

1.4.1 Steady State

Suppose that metabolite M is used in exactly two reactions:

$$R_1 : \ A \to M$$

$$R_2 : \ 2M \to B$$

Suppose R_1 has flux ν_1 and R_2 has flux ν_2. Then the rate of change of the concentration C must be

$$\frac{\partial C}{\partial t} = \nu_1 - 2\nu_2.$$

Since the concentration is constant, we have that

$$\nu_1 - 2\nu_2 = 0.$$

This generalizes to

$$\frac{\partial C_i}{\partial t} = \sum_{R_j \in R} A_{i,j}\nu_j = 0,$$

where R is the set of the cell's reactions, $A_{i,j}$ is the stoichiometric coefficient of metabolite M_i in reaction R_j, and ν_j is the flux of reaction R_j. This prevents any metabolite from accumulating in the cell. Each metabolite must be converted into something that is consumed for energy or growth, or excreted from the cell..

The model contains a few reactions that are important to note, and are presented as specific to of E. coli. The first is called *VGRO* (virtual growth), which has a stoichiometric equation of

$$5 \times 10^{-5}\text{ALA} + 46\text{ATP} + \cdots + 0.4\text{VAL} \rightarrow \text{Biomass} + 46\text{ADP} + 46\text{PI} + 0.7\text{PPI}$$

This reaction defines the growth rate. In order to gain a unit of biomass, the metabolites that are inputs to this reaction must be present in the values denoted by the stoichiometric coefficients. ADP, PI, and PPI are by-products of the physical creation of more cell mass, so they are resources that must be accounted for.

Energy spent by the cell on constant maintenance is captured by the dummy reaction, ATPM, which has a stoichiometric equation of

$$\text{ATP} \rightarrow \text{ADP} + \text{PI}.$$

This reaction has a constant flux of 7.6, which represents the energy expenditure of non-metabolic processes. The ATP drain is of some significance when it is in cycles, discussed in Section 4.3.

Degeneracy An important aspect of using FBA is accounting for *degeneracy* in solutions. In general terms, degeneracy occurs in solving an LP when the solving algorithm cannot overcome some obstacle in working toward an optimal solution. In FBA, degenerate fluxes are those that can either be zero or non-zero for an optimal solution. Degeneracy contributes to the non-uniqueness of optimal solutions by allowing reactions to either be active or inactive, arbitrarily.

1.4.2 FBA Example

Consider the following reactions in the metabolic network for a hypothetical single-celled organism, with stoichiometric coefficients of $A_{3,1} = 2$ and $A_{i,j} = 1$ for $(i, j) \neq (3, 1)$, where $A_{i,j}$ is the coefficient for metabolite M_i and reaction R_j:

Import reactions:

R_1: $\rightarrow M_1$

R_2: $\rightarrow M_2$

Intracellular reactions:

R_3: $M_1 \rightarrow 2M_3$

R_4: $M_3 \rightarrow M_4 + M_5$

R_5: $M_2 + M_5 \rightarrow Biomass + M_3$

Export reactions:

R_6: $Biomass \rightarrow$

R_7: $M_2 \rightarrow$

R_8: $M_5 \rightarrow$

Let ν_j denote the flux of reaction R_j. Assign lower and upper bounds to the fluxes as follows:

$L_1 = 0 \leq \nu_1 \leq \infty = U_1$

$L_2 = 0 \leq \nu_2 \leq 10 = U_2$

$L_3 = 0 \leq \nu_3 \leq \infty = U_3$

$L_4 = -10 \leq \nu_4 \leq 10 = U_4$

$L_5 = L_6 = L_7 = L_8 = 0 \leq \nu_5, \nu_6, \nu_7, \nu_8 \leq \infty = U_5 = U_6 = U_7 = U_8$

The following LP can be used for simulation:

maximize: ν_6

subject to: $A_{i,j}\nu_j = 0$

where $L_j \leq \nu_j \leq U_j$

Maximizing the flux of R_6 requires maximizing the flux of R_5 to provide the most production of *Biomass*. Reaction R_5 can have a flux of up to 10 and is limited by reactions R_1 and R_4. Reaction R_1 has an unlimited flux, and R_4 has a maximal flux of 10. Reaction R_4 is limited by R_3's production of M_3, so by maximizing the amount of M_3 produced by R_3, maximal *Biomass* is available. Thus, an optimal solution is to have fluxes $\nu_1 = 5$, $\nu_2 = 10$, $\nu_3 = 5$, $\nu_4 = 10$, $\nu_5 = 10$, $\nu_6 = 10$, $\nu_7 = 0$, and $\nu_8 = 10$. For more than 10 units of M_3 produced by R_3, however, *Biomass* is unaffected. Reaction R_3 can produce M_3 in unlimited quantity, so the flux of R_3 influences the system the same way for any $\nu_3 \geq 5$, allowing infinitely many optimal solutions.

A systematic approach to finding an optimal solution involves describing information about the network in the matrix equation

$$A\nu = b,$$

where matrix A describes a system of equations that characterize the stoichiometric equations of enzymes in the network, vector b is zero because of the steady-state assumption, disallowing the accumulation of any metabolite, and ν is a column vector of fluxes for each of the reactions. In this case, optimal growth is estimated by using an LP to maximize ν_6. See Appendix A for AMPL model and data files for this example. The AMPL implementation is discussed in Section 3.1.

$$
\begin{array}{c}
 \\
\begin{array}{c|cccccccc}
 & R_1 & R_2 & R_3 & R_4 & R_5 & R_6 & R_7 & R_8 \\
\hline
M_1 & \mathbf{1} & 0 & \mathbf{-1} & 0 & 0 & 0 & 0 & 0 \\
M_2 & 0 & \mathbf{1} & 0 & 0 & \mathbf{-1} & 0 & \mathbf{-1} & 0 \\
M_3 & 0 & 0 & \mathbf{2} & \mathbf{-1} & \mathbf{1} & 0 & 0 & 0 \\
M_4 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
M_5 & 0 & 0 & 0 & \mathbf{1} & \mathbf{-1} & 0 & 0 & \mathbf{-1} \\
Biomass & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{-1} & 0 & 0
\end{array}
\end{array}
\times
\begin{array}{cc}
 & Flux \\
R_1 & \nu_{R_1} \\
R_2 & \nu_{R_2} \\
R_3 & \nu_{R_3} \\
R_4 & \nu_{R_4} \\
R_5 & \nu_{R_5} \\
R_6 & \nu_{R_6} \\
R_7 & \nu_{R_7} \\
R_8 & \nu_{R_8}
\end{array}
=
\begin{array}{cc}
 & b \\
M_1 & 0 \\
M_2 & 0 \\
M_3 & 0 \\
M_4 & 0 \\
M_5 & 0 \\
Biomass & 0
\end{array}
\tag{1.3}
$$

Equation 1.3 shows a description of the metabolic network from the example in a matrix form. Stoichiometric coefficients are listed as negative for metabolites on the left-hand side of the stoichiometric equation and positive for those on the right-hand side. Net production and consumption is zero for each metabolite due to the steady state assumption.

Chapter 2

Partitioning Enzymatic Reactions, Minimal Environments, and Cycles

Due to the diversity and large quantity of reactions, partitioning a set of reactions into smaller sets with common characteristics may make it easier to study metabolic pathways and to draw insights about similar reactions. In particular, identifying groups of reactions that are more heavily used than other reactions is generally valuable in identifying important pathways.

Investigating which resources are required for different levels of growth and in what ways they are required leads us to develop an environment for which every metabolite is limiting. Additionally, complications associated with partitioning the network direct us to investigate cycles within the network.

2.1 Partitioning Reactions with Dominance

For each collection of fluxes that achieve an optimal growth state for a particular environment, we have that each metabolite has one or more associated reactions that equally consume or produce more of it than any other reaction. Consider the set of all such reactions. These reactions are each *dominant* consumers or producers (of some metabolite). For one of these reactions, we say that the difference in flux between this reaction and the next greatest consumer (or producer) is the *dominance* of the reaction for this metabolite. We say that the maximum dominance is the *supremacy* of the reaction for this metabolite.

2.1.1 The HFB, Degeneracy and a Partition

Reactions that both dominate the consumption of a metabolite in this manner, as well as the production of the same or another metabolite, are in the *high flux backbone* (HFB) [2]. The HFB is defined for enzymes in a real-life organism. Thus, the HFB is not based on degenerate solutions because it is not defined in terms of a solution from the model, but in terms of the biological behavior that an actual organism performs. The HFB can be produced by the model for a hypothetical organism that operates exactly in the real-world as the model behaves for a particular solution. Such an HFB may not be meaningful, however.

Attempting to compute the actual HFB for an organism, which is biologically dependent on the environment, using FBA methods, gives a solution that is computationally dependent on the solution of the LP [1], and is thus not equal to the real HFB. The specific solution resulting from computing the HFB is dependent on the algorithm used by the solver; each solver identifies a single optimal solution from infinitely many. Which environmental resources are available to the cell greatly affects which pathways are possible. For example,

E. coli's metabolism includes aerobic pathways and anaerobic pathways, and depending on whether oxygen is an available resource, some significant reactions might not be used at all. Degeneracy issues call for an unambiguous partition for cellular reactions. We augment the concept given by the HFB to the following sets.

The *high flux network* (HFN) is similar to the HFB, but includes reactions that have the capacity of simultaneously dominating both consumption and production for any optimal setting. The *intermediate flux network* (IFN) is the set of reactions that can dominate consumption or production, but not both, in some optimal state, and the *low flux network* (LFN) is the set of reactions that are incapable of dominating production or consumption in any optimal state.

We say that a reaction is in the HFN, IFN, or LFN *independent of the environment* if it is in the set for every environment, otherwise it is in the set *dependent on the environment.*

2.2 Minimal Environment

For any organism, achieving the maximal growth rate requires using resources efficiently. Given resources in unbounded quantity, a single-celled organism can only transport them into the cell and process them at a rate limited by reaction fluxes, among other things. In such a scenario, processing resources more efficiently may require improving these physical limiting properties. If resources themselves are not limiting, using a greater or fewer number of resources may not directly affect the growth rate.

Since these organisms do not necessarily, and indeed most often do not, have access to resources in unlimited quantities, resource availability has a direct effect on the growth rate. This introduces the question, how little resources are necessary for optimal growth? That is, what environment can provide resources in as limiting a manner as possible and allow the cell to achieve a specified growth rate?

It is important to be precise about what is meant by “how few” in the context of maximizing growth with resources that are limiting, because to answer it with the aggregate quantity of different resources pressupposes the equality of one unit of a resource with one unit of any other resource. Is one unit of oxygen as vital as one unit of nitrogen? Is it as easily accessible?

One approach is to weight each resource and minimize the sum of total resources used by the cell. However, this solution requires an unambiguous weighting process, which may be difficult to develop in a way that yields meaningful results.

An alternative approach is to iteratively minimize a common upper bound on the import-fluxes of non-limiting resources, with the constraint that growth must be fixed at some desired value less than or equal to optimal growth. The solution to each minimization includes one or a group of the resources that limit the bound to be equal to the value of the common import-flux of these resources. When the reactions that remain are non-limiting, the algorithm is applied again without including the newly-limiting resources. It terminates when there are no non-limiting resources remaining. The set of import-fluxes for each resource is defined to be the *minimal environment.*

2.3 Cycle Analysis

Some reactions can dominate others in an unbounded quantity. Because there is no cost associated with catalyzing a reaction in the FBA model, reactions that are in cycles may attain unbounded fluxes, with the exception of cycles that link to the ATP drain, ATPM, the only reaction that has a fixed flux to account for energy spent by the cell on maintenance (a dummy reaction). Any reaction’s products can be reconverted to its reactants by restricting the export of products to reactions within the cycle. In other words, reactions can simply "trade off" metabolites amongst themselves. This is analogous to counting the number of cars that pass through a network of roads. A car that drives around the same block, in circles, could be mistakenly counted multiple times. Any optimal solution to the model could have arbitrary fluxes for these reactions. Not only is meaningful information about these fluxes lost, but infinitely many solutions are produceable with no clear indication about which of them most closely reflects reality. Consider an example consisting of the following reactions, as shown in Figure 2.1:

$$R1 : a \rightarrow b$$

$$R2 : b \rightarrow c$$

$$R3 : c \rightarrow a$$

Notice that, regardless of the quantity of metabolite a that reaction $R1$ has available, reactions $R1$, $R2$, and $R3$ can exchange metabolites a, b, and c arbitrarily many times, without affecting the actual quantity of c produced by $R3$.

Finding cycles in a metabolic network and investigating their properties may prove useful by providing a means of overcoming the problem they introduce in calculating supremacy. Unfortunately, certain questions pertaining to the set of all cycles of the network may not be answerable, as it turns out that it is computationally infeasible to to find all cycles for a network such as the metabolic network of E. coli. Instead, this analysis can be applied to subsets of the network. In particular, each dominance calculation that yields unbounded supremacy for some metabolite-reaction pair has an associated set of reactions whose fluxes are unbounded for that solution. A reaction with unbounded supremacy for some metabolite can have infinite dominance over all reactions not in that cycle. This is because it can have an arbitrarily large flux while reactions not in the cycle are unaffected.

Because the flux of the ATPM reaction is fixed, it cannot be part of an unbounded cycle. We can take the set of unbounded cycles that are individually generated by finding the supremacy of a metabolite-reaction pair and performing analysis on individual cycles as well as unions and intersections of multiple cycles. Each of these cycles is significant because its cyclic nature directly affects the information encoded in the solution.

A graph data structure is created for this project in Java to find cycles within a metabolic network. Before presenting our analysis, an object-oriented model is created as described in Section 2.3.1.

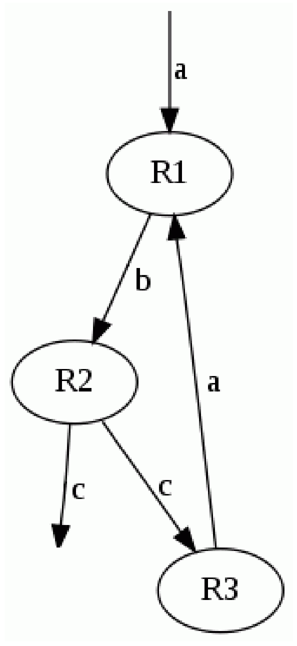

Figure 2.1: A three-reaction cycle. Reactions R1, R2, and R3 can trade off metabolites a, b, and c indefinitely.

2.3.1 Constructing the Network

Object orientation is constructed to serve as a flexible model for finding cycles. Three objects are used to describe the network, `Metabolite` objects, which represent metabolites used by reactions in the network, `Reaction` objects, which represent the reactions, and `Graph` objects, which are data structures that manage `Reaction` and `Metabolite` objects. Each `Reaction` object contains information about a reaction in the metabolism, including a list of metabolites associated with it, and each `Metabolite` object contains information about a metabolite, including a list of reactions by which it is consumed or produced.

Each `Graph` object provides the structure needed to model a metabolism, or a subset of one. The `Reactions` in the `Graph` are graphically represented by vertices, and associated

`Metabolites` are represented by edges. Edges are created between reactions when one reaction consumes a metabolite that another produces, with directionality toward the consuming reaction. The `Graph` object provides functionality for generating a graph-representation of the network as an image file. It also includes special functionality for analysis. This includes methods for finding unions and intersections of arbitrarily many `Graph`s and, most importantly, finding and retrieving all cycles within the network.

The `Reaction` object provides a way of abstracting information from a reaction by storing its name and the metabolites it consumes and produces. The reaction is also defined to have bounds on its flux, where the default bounds are $0 \leq \nu_i \leq \infty$ for the flux of reaction R_i, ν_i. Unless otherwise specified, a reaction can only run in the forward direction, and can do so with any flux. The `Metabolite` object stores the name of a metabolite and the reactions it is consumed and produced by.

Chapter 3

Computational Aspects

A description of computational methods for the project is given in this chapter. This includes the general layout and components of the AMPL model used to model metabolic networks. Data structures and algorithms that are used to partition the network, find the minimal environment, and find and investigate cycles are also discussed.

3.1 FBA Model

Constructing an FBA model requires information about a cell's metabolism, including each reaction in the metabolism and the associated metabolites. A Modeling Language for Mathematical Programming (AMPL) is used to construct the model. Two AMPL data files consist of a list of reactions and their stoichiometric coefficients for each metabolite, $A_{i,j}$ for metabolite M_i and reaction R_j, as well as bounds of the form L_j and U_j on the fluxes of reactions of the form ν_j, so that $L_j \leq \nu_j \leq U_j$, for reaction R_j. The model file interprets the data and defines the sets, parameters, constraints and objective function needed to describe the LP. This file is executed and solved in the AMPL environment, which returns the optimal growth value and associated fluxes.

The primary data file defines values used for the bounds on the fluxes of all reactions that do not import metabolites from the environment, as well as values used for the stoichiometric coefficients. A secondary data file defines values for the bounds of the reactions that import metabolites. Stoichiometric coefficients are integers that are signed to differentiate between metabolites that are consumed and those that are produced.

In the model file, the set of fluxes (reactions), FLUX, is defined and partitioned into three subsets: fluxes of reactions that transport resources into the cell (FLUXI), reactions that process metabolites within the cell (FLUXC), and reactions that export waste from the cell (FLUXO).

Metabolites are indexed by the set METS, where a constraint is added that forces each metabolite's net accumulation to be zero. Metabolite-reaction pairs given by the stoichiometric equations for each reaction form 2-element sets that are contained by sets PRSI, PRSC, PRSO. These three sets contain elements that are used for transporting metabolites into the cell, processing metabolites within the cell, and transporting reactions out of the cell, respectively. The stoichiometric coefficients of these pairs are defined in sets Ai, Ac, and Ao. Lower and upper bounds for reaction fluxes are defined by sets Ubd and Lbd, and are indexed by elements of FLUX. A constraint in the model requires the flux of a reaction to be within its lower and upper bounds. The objective function is the maximization of the sum of the fluxes of all of the reactions. The AMPL script, model file, and a data file for the example in Section 1.4.2 is shown in Appendix A.

3.2 The Partitioning Process

Partitioning the network into the HFN, LFN, and IFN is done in a 2-step process:

1. Execute an AMPL script (see Appendix B)
   (a) Calculate the supremacy of each metabolite-reaction pair
   (b) For pairs that have non-zero supremacy or tie for zero supremacy
      i. If supremacy is finite, flag the minimum growth as -1
      ii. If supremacy is unbounded, calculate the minimum possible growth with supremacy fixed as unbounded. Also record all reactions whose fluxes are unbounded
   (c) Catalog pertinent information
      i. If supremacy is non-zero or tied at zero, record as a candidate for the HFN or IFN
      ii. If supremacy is zero, not tied, record as an element of the LFN
2. Execute a PHP script (also in Appendix B)
   (a) Read LFN elements, write them to a file
   (b) Read HFN and IFN candidates
   (c) Find all instances of a reaction dominating consumption or production
      i. If the reaction dominates both consumption of at least one metabolite and production of at least one metabolite, record as an element of the HFN and write it to a file
      ii. If the reaction only dominates consumption or only dominates production, record as an element of the IFN and write it to a file
   (d) Write HFN and IFN reactions to a file

The script directs AMPL to read in the model and data files. This script runs in the AMPL environment, so AMPL commands are used to control solver options. Here we have the first step of constructing the HFN, which is to determine every possible metabolite-reaction pair for which the reaction can dominate the consumption and/or production of the metabolite under some environment (i.e., have a non-zero supremacy).

The AMPL script serves the purpose of identifying single cases of reactions dominating metabolite consumption or production, and the PHP script collects all such cases for a particular reaction and classifies the reaction accordingly.

Cycles from Unbounded Dominance Calculations During the partitioning process, whenever a solution is found in which the metabolite-reaction pair obtains unbounded supremacy/dominance, the pair is recorded, along with a list of reactions whose fluxes are unbounded for the solution. These lists can be used to construct cycles by constructing a graph for each list, where the graph contains reactions that are in the list. The object-oriented model program discussed in section 2.3.1 is used for this.

3.3 Finding the Minimal Environment

An AMPL script is used to compute the minimal environment for various growth rates by iterating over a set of discrete growth rate values. For each growth rate, k, the flux of the reaction "Growth" is fixed to k. Each import-reaction is added to the set of unfixed reactions, UNFIXED, which is the set of reactions whose fluxes are bounded by a common upper bound, B. The script enters a loop that iterates until the fluxes of reactions in UNFIXED can be minimized to zero, which is guaranteed to occur when the set is empty.

In each iteration, B is minimized under the constraint that the growth rate is attainable.

The resulting solution contains a subset of UNFIXED whose elements have fluxes that are equal to B. Most likely, all of these fluxes are equal to B because they are limiting at the value given by B and cannot be lowered, but it is possible that some of them could be lowered and merely attain a flux of B arbitrarily. Every flux is fixed to the value given in the solution. In order to identify reactions that are limiting, a loop is entered that iterates over each import-reaction that has attained flux B. The reaction's flux is unfixed and minimized, then fixed again. If it cannot be minimized below B, then it is limiting with flux B. It is removed from the set UNFIXED and added to the set FIXED and the flux is recorded.

After this is repeated for each reaction, the set UNFIXED is guaranteed to have lost elements, because some import-reaction must be limiting in order for B to be nonzero, and all such reactions are added to FIXED. Eventually, either UNFIXED is empty or B is zero, at which point the fluxes are recorded and represent the minimal environment for the network.

3.4 Finding all Cycles

A depth-first-search is used to find all cycles in the network. All possible paths through the network from each reaction are traversed. Paths that return to the starting reaction are cycles. The union of the sets of all cycles for all reactions forms the set of all cycles in the network.

3.4.1 Initial Algorithm

This algorithm traverses every path in the network that starts at the root and does not visit any vertex twice, with the exception of the terminating vertex. Paths that terminate at the root vertex are identified as cycles. Pseudo-code is shown in Listing 3.1.

Listing 3.1: Pseudo-code for the initial algorithm showing how the network is traversed.

```
getCycles(Graph graph) {
        while(graph.hasNextVertex()) {
                vertex = graph.getNextVertex();
                Graph startCycle = new Graph();
                getCyclesHelper(vertex, vertex, startCycle);
        }
}

getCyclesHelper(Node root, Node visiting, Graph cycle) {
        visiting.visited = false;
        cycle.add(visiting);
        while (visiting.hasNextChild()) {
                child = visiting.getNextChild();
                if(child == root) {
                        cycle.add(root)
                        print cycle;
                }
                else if(child.visited) {
                        getCyclesHelper(root, child, cycle);
                }
        }
        cycle.remove(visiting);
        visiting.visited = true;
}
```

3.4.2 Modified Algorithm

Because of problems introduced by the complexity of the initial algorithm (see Section 3.4.3), a modified version is developed. Given a root vertex, a traversal through the network occurs until the root vertex is reached, forming a cycle. The recursion of the algorithm is used for labeling a vertex as being part of a cycle with the root vertex if one of its children has such a label. This step of the algorithm isn't intended to find all cycles, but to determine which vertices are in cycles. A vertex only has to be identified as being part of at least one cycle to be included in the set of vertices considered by the second step of the algorithm. Because of this, vertices are only visited once (but may be referenced by vertices directed to them).

The second step of the modified algorithm is essentially the initial algorithm, but with the additional condition that child vertices are only visited if they are labeled as vertices that are in cycles with the root vertex. If the number of reactions in cycles is sufficiently smaller than the total number of reactions, performance improves because the second step of the algorithm is only applied to those reactions. This comes at the cost of preprocessing the reactions with a segment of the algorithm that is less complex (also discussed in Section 3.4.3). Pseudo-code is shown in Listing 3.2, and code used to implement it in Java is shown in Appendix C.

Listing 3.2: Pseudo-code for the modified algorithm showing how the network is traversed.

```
getCycles(Graph graph) {
        while(graph.hasNextVertex()) {
                vertex = graph.getNextVertex();
                recurseForCycles(vertex, vertex);
                buildCycles(vertex, vertex, new Graph);
        }
```

```
}

recurseForCycles(Node root, Node visiting) {
        visiting.visited = true;
        while(visiting.hasNextChild()) {
                child = visiting.getNextChild();
                if(child == root)
                then {
                        visiting.reaches(root) = true;
                } else if(child.visited == false) {
                        recurseForCycles(root, child);
                }

                if(child.reaches(root)) {
                        visiting.reaches(root) = true;
                }
        }
}

buildCycles(Node root, Node visiting, Graph cycle) {
        visiting.visited = false;
        cycle.add(visiting);
        while(visiting.hasNextChild()) {
                child = visiting.getNextChild();
                if(child.reaches(root)) {
                        if(child == root) {
                                cycle.add(root)
```

```
                                    print cycle;
                              }
                              else if(child.visited == true) {
                                    buildCycles(root, child, cycle);
                              }
                  }
            }
            cycle.remove(visiting);
            visiting.visited = true;
}
```

3.4.3 Complexity

Finding cycles for the entire metabolic network is intractable and computationally difficult. The model has 197 transport reactions that solely import and export metabolites into and out of the cell. Not including these reactions, E. coli has 631 reactions, with an average reaction out-degree of $\bar{c} = 35.09$ and median out-degree of $c_m = 11$. It is important to keep in mind that the out-degree of a reaction is not the number of different metabolites it produces, but the number of reactions that consume a metabolite that it produces. For small enough subsets of the network, however, enumerating all cycles is reasonable and may provide useful information.

The complexity of the algorithm is dependent on the network. The first case to consider is the worst case in which the network is complete and every vertex shares a bidirectional edge with every other vertex. Another case to consider is the one in which each vertex has the same out-degree, which may be much smaller than it is for a complete graph. The modified algorithm does the same thing but only considers vertices that are in cycles. The complexity of these cases is as follows, where we consider only the complexity for the first reaction as a root node. Each reaction must be individually considered as a root node to find every cycle.

1. Complete: $O((V-1)!)$, where V is the number of vertices,
2. Each vertex v_i with out-degree of c_i: $O(\prod_{i=1}^{V} c_i)$, or $O(c^V)$ for $c = c_i = c_j \ \forall i, j \leq V$
3. Considering only a subset of the vertices in cycles, V^*, and each vertex with outdegree c_i : $O(V+E) + O(\prod_{i \in I}^{V^*} c_i)$, where I indexes V^* and E is the number of edges

This problem is most computationally difficult when the network is complete and every vertex shares a bidirectional edge with every other vertex. For a vertex, the $V-1$ remaining vertices are visited. For these vertices, the remaining $V-2$ are visited, etc. Continuing in this fashion, the complexity of the algorithm is $O((V-1)!)$.

For a network in which each vertex v_i has out-degree c_i, the number of paths is multiplied by c_i at each vertex indexed by v_i, so to traverse every path in the network, the complexity for the algorithm is $O(\prod_{i=1}^{V} c_i)$. Estimating each c_i as the median, $\bar{c}$, we have that the algorithm is $O(11^{631})$. This is obviously impractical.

The first step of the modified algorithm is of complexity $O(V)$ because every vertex is visited exactly once. The first step finds all vertices that are in cycles, V^*, so the complexity to find all cycles among these is $O(\prod_{i \in I}^{V^*} c_i)$, where I indexes V^*, and the total complexity is $O(V) + O(\prod_{i \in I}^{V^*} c_i)$. We hope that V^* is small enough to make computation feasible.

Chapter 4

Results and Conclusions

This chapter includes a discussion of results and interpretations. All results are either for E. coli, or are not specific to any organism. See the appendices for tables of results when applicable. Results for FBA analyses and for cycle analysis are given, and future work is discussed. The model for E. coli is the iJE660a genome scale model [3], which can be used to reference reactions by acronyms in the FBA model. A newer model has been developed for E. coli, iJR904, with improved capabilities [5].

4.1 Network Partition

A partition of the network is given in Appendix D. The LFN makes up 93/631, or 14.7%, of the network, the IFN makes up 105/631, or 16.6%, and the HFN makes up 433/631, or 68.7%. What is surprising is that most reactions can, under some circumstances, simultaneously dominate production and consumption of metabolites (reactions in the HFN). Each metabolite is associated with an average of 2.96 reactions, and only one of them can dominate consumption and only one can dominate production of the metabolite for each solution. Because there are infinitely many solutions, we should expect that the dominant consumers and producers of metabolites should vary. It is important to keep in mind that different components of any biological organism are more or less critical and more or less used than others. In a metabolism, some pathways are very important and are used in nearly every steady-state condition. These include the TCA cycle, shown in Figure 1.2, which is included in the HFN in its entirety. This is a good indication that the HFN contains some of the more heavily used and maybe more important pathways.

A possible interpretation of the size of the HFN being quite large relative to the other parts of the partition is that E. coli contains many reactions and pathways that are specially suited for particular environments. More than two-thirds of the network can be dominant consumers and producers simultaneously. Many of these reactions may only achieve large fluxes, relative to other reactions, in the particular types of environments that they are suited for. Growth in an aerobic environment, for example, is known to require a very different set of active pathways than anaerobic growth. In oxygen-rich environments, many reactions that are specific to anaerobic growth are not used heavily, if at all. These reactions become critical when oxygen is limiting, however. Because E. coli can adapt to achieve maximal growth in a wide array of environments, it contains many reactions that are heavily used under specific circumstances.

4.2 Minimal Environment

The minimal environment for E. coli is shown in Figure 4.2 for different growth rates. Inorganic phosphate has the highest required import flux-per-unit of growth for nearly all growth rates, and a second group of 41 reactions all tie for the next highest required flux for nearly all growth rates.

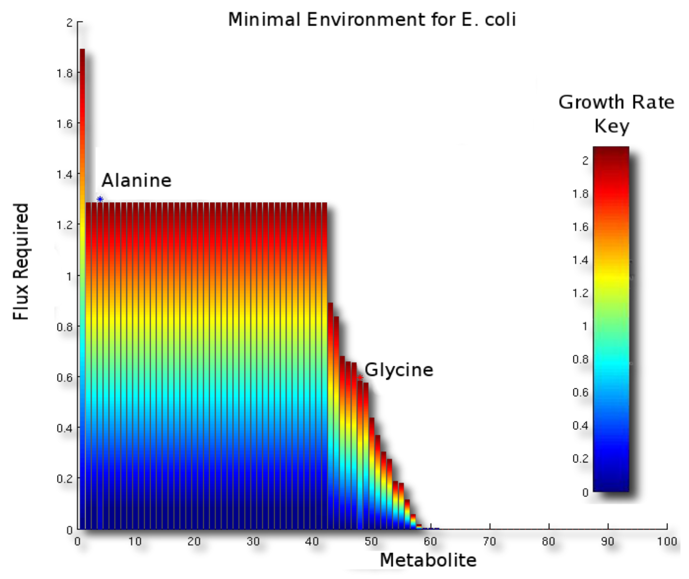

Figure 4.1: Minimal environment fluxes for E .coli. Red values correspond to high growth rates and blue values correspond to low growth rates. Flat regions identify metabolites that can be substitutes for one another.

The flux of every metabolite-import reaction correlates piecewise-linearly with the growth rate, with the exception of alanine and glycine. This correlation is linear in segments for these metabolites. There is one growth rate value at which the rate of flux-per-unit growth increases for glycine, and two such growth rates for alanine. This behavior is illustrated in Figure 4.2. For both alanine and glycine, there are growth rates before which growth can be achieved without using them, but after which they are required.

The large flat region in Figure 4.2 begs the question of whether the metabolites it includes are somehow related in a significant way. Each of these 41 metabolites have exactly the same minimal import-fluxes, with the exception of alanine, for growth rates that are less than 0.2. The answer to this question is that these metabolites are related in that they all provide raw materials required to make the same required resource. Growth can

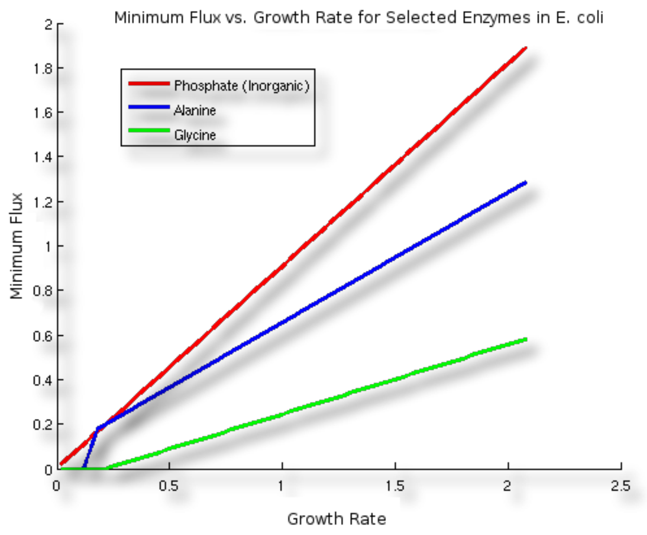

Figure 4.2: Minimal flux versus growth rate for three metabolites in E. coli. The relationship is non-linear for alanine and glycine.

be achieved for E. coli by either manufacturing certain metabolites from raw materials, or by importing the metabolites directly. Complex resources can also be broken down and converted into needed metabolites.

Many different metabolites can be used to provide the same function or to produce the same necessary metabolites. In this case, growth is not limited by any resource of the group in particular, but by the aggregate quantity of all resources in the group. For this reason, when the minimal environment is calculated, the upper bound for these metabolites is common among the group. If any metabolite in the group had a higher import-flux than another, a trade-off could be made where the import-flux of the first metabolite is decreased and the flux of the other is increased.

4.3 Cycles and Unbounded Dominance

Table 4.2: Intermediate Flux Network of E. coli.

Reactions					
ADHEx1	ALDA	DCUAx1	GALU	GPSAx1	PGIx1
ADK	ALR	FRDA	GLTP	NDK	PGIx2
ADKx1	DADX	GALF	GPSA	PGI	SDHA

Metabolites					
ADP	ATP	FADH	GL3P	NAD	SUCC
ALA	bDG6P	FUM	GTP	NADH	T3P2
AMP	DALA	G1P	HEXT	NADP	UDPG
ASP	F6P	G6P	LACAL	NADPH	UTP
ASPxt	FAD	GDP	LLAC	PPI	

All Cycles Unfortunately, 622 reactions are found to be in cycles (none of which are transport reactions). Only 8 reactions, not including transport reactions, are not found in cycles, and are shown in Table 4.1 . This means that the complexity of enumerating every cycle, estimated using the median reaction out-degree, $c_m = 11$, is $O(631) + O(11^{622})$. This is still too difficult to compute. We see that almost every reaction in E. coli's metabolic network's is in a cycle, so the intractability of the problem is unavoidable, and the cycles are not enumerated.

Table 4.1: Non-transport reactions not in cycles.

Reactions	
FADL	PNUC
FADLx1	PNUE
FADLx2	R0017
Growth	R0018

Dominance Cycles There are 52 metabolite-reaction pairs with unbounded supremacy for E. coli. Results are the same under both the M9-minimal environment, which is a commonly used medium in wetlab work, and unbounded environment. This is not surprising because unbounded supremacy appears to be driven by the existence of cycles, which can have arbitrary fluxes as long as the appropriate metabolites exist in non-zero quantities. These pairs contain 18 reactions and 29 metabolites, listed in Table 4.2 .

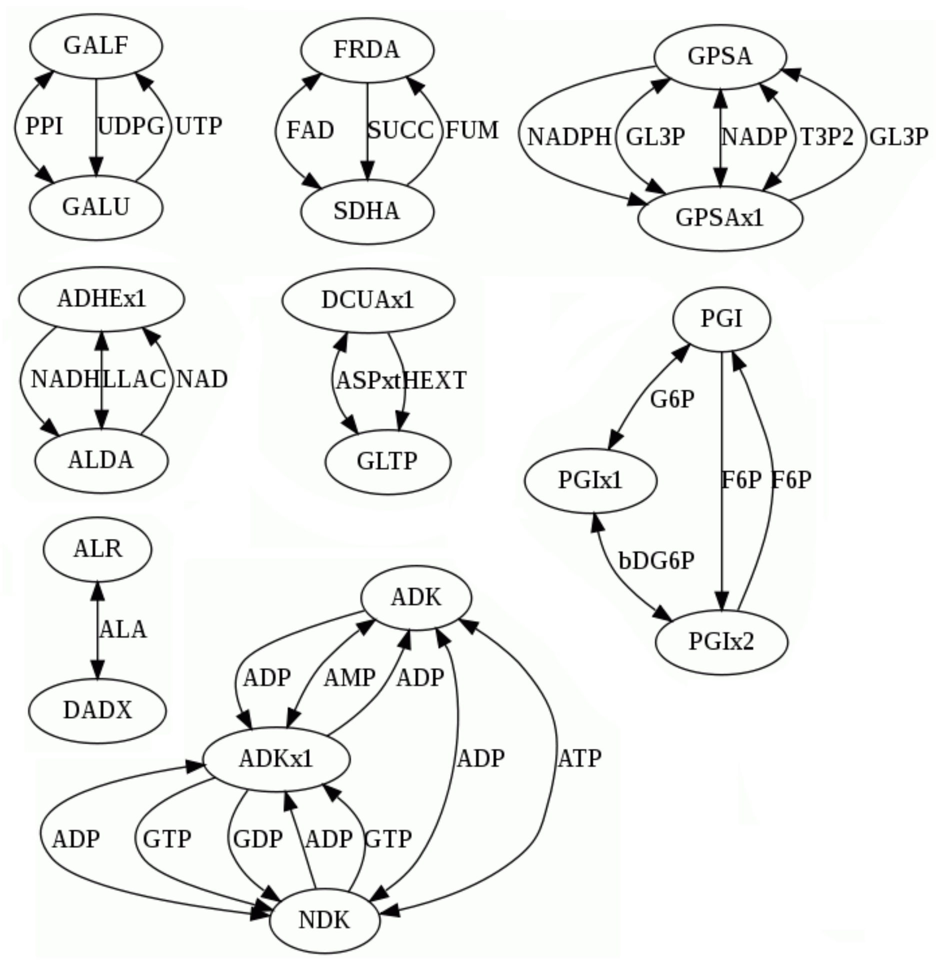

Figure 4.3: Cycles found by dominance calculations for E. coli with M9. These cycles are all disjoint.

The cycles characterized by these reactions are shown in Figure 4.3. The cycles corresponding to individually generated sets of reactions from dominance calculations are either equal or disjoint. They are equal when the dominating reactions that generate them are in the same cycle.

The ATP drain, ATPM, discussed in Section 1.4, limits the number of unbounded cycles. When its flux is not fixed, there are 698 metabolite-reaction pairs with unbounded supremacy, which include 182 reactions and 139 metabolites. The cycles that they generate are also disjoint, as expected. Additionally, the intersection of the set of cycles generated with the flux of ATPM unbounded and the set of cycles generated with the flux bounded is exactly the set of cycles generated with the flux bounded. That is, by removing the bounds on ATPM's flux, no new cycles are generated that do not include ATPM.

4.4 Future Work

Flux balance analysis is an attempt to simplify extremely complex biological systems to make them more manageable for the purpose of computational and mathematical study. There are still many unanswered questions about what the most appropriate methods for applying the model are, and about how the model should be modified to most closely emulate living organisms. In this project, we investigate aspects of the metabolic network as it behaves independent of the other networks. The protein interaction and gene regulatory networks, discussed in Section 1.1, are also important. Developing a model that accounts for all three networks and how they affect each other would probably be much more reliable and robust. Incorporating all three networks into a single model should be considered for future work.

The partition formed by the LFN, IFN, and HFN should be more closely inspected. Known biological pathways should be classified as being contained by one or more parts in order to determine what characteristics reactions and pathways generally have when they are partially or completely contained by each of the parts. This partition could potentially be used to develop a systematic method to classify regions of metabolic networks in one or more useful ways for any organism.

The partition formed by the LFN, IFN, and HFN is independent of any solution, but dominances are calculated for each solution. Consider the partition LFN*(s), IFN*(s), HFN*(s), the HFN, IFN, LFN where the solution space is a single solution. An interesting result would be to find the average distribution of reactions into LFN*(s), IFN*(s), HFN*(s), for all $s \in S$, where S is the entire solution space. We may define

$$\overline{\mathrm{LFN}^*} = \int_S \frac{|\mathrm{LFN}^*(s)|}{|M|},\ \overline{\mathrm{IFN}^*} = \int_S \frac{|\mathrm{IFN}^*(s)|}{|M|},\ \overline{\mathrm{HFN}^*} = \int_S \frac{|\mathrm{HFN}^*(s)|}{|M|}$$

where M is the set of all reactions in the metabolism. This information would give insight into how many reactions at a given time (for some solution) are maximum producers and consumers, one of the two, or neither of the two. Note that $|\mathrm{HFN}^*(s)| \leq |\mathrm{HFN}|$ and $|\mathrm{LFN}^*(s)| \geq |\mathrm{LFN}|$ for all s because the definition of the partition is such that a reaction is excluded from the LFN if it can dominate for any s. Considering more solutions decreases the number of reactions in the LFN and increases the number of reactions in the HFN and IFN. This could be considered in future work, along with other interesting or useful properties of the network associated with the partition.

The algorithms developed for finding cycles can be utilized for applications unrelated to metabolic networks. The object oriented program can be modified and generalized to serve as a toolbox for creating graphs, visualizing them, and finding cycles in them. With no great effort, this program could be made useful to individuals using graphs in a wide variety of contexts, not just cycle analysis of metabolic networks.

Bibliography

[1] E. Almaas. Optimal flux patterns in cellular metabolic networks. *Chaos*, 17(2), June 2007.

[2] E. Almaas, B. Kovacs, T. Vicsek, Z. N. Oltvai, and Barabasi. Global organization of metabolic fluxes in the bacterium, escherichia coli. *Nature*, 427(839), Februrary 2004.

[3] Palsson BO. Edwards JS. The escherichia coli mg1655 in silico metabolic genotype: its definition, characteristics, and capabilities. *Proc Natl Acad Sci USA*, 97(10), May 2000.

[4] A. Holder, editor. *Mathematical Programming Glossary.* INFORMS Computing Society, `http://glossary.computing.society.informs.org`, 2006–08. Originally authored by Harvey J. Greenberg, 1999-2006.

[5] Reed JL, Schilling Vo TD, and Palsson BO. CH. An expanded genome-scale model of escherichia coli k-12 (ijr904 gsm/gpr). *Genome Biology*, 4(9), August 2003.

[6] Christopher Mathews, Kensal van Holde, and Kevin Ahern. *Biochemistry.* Benjamin-Cummings, 1999.

[7] Daniel Segr, Dennis Vitkup, and George M. Church. Analysis of optimality in natural and perturbed metabolic networks. *Proc Natl Acad Sci USA*, 99(23), November 2002.

[8] C.S. Vollert and P. Uetz. *Encyclopedic Reference of Genomics and Proteomics in Molecular Medicine*. Springer Verlag, 2003.

Appendix A

FBA Code

. This appendix contains AMPL code for the FBA model. Files include a script used to execute the code, the model file, a data file for the network, and a data file for resources. The data used is from the example in Section 1.4.2.

Listing A.1: The AMPL script. This is used to set various solver options load the model and data files and solve the problem.

```
#!/usr/local/bin/ampl


model fba.mod;
data  iJE660a.dat;
data  iJE660a_env.dat;
let {j in FLUXI} Ubd[j] := Ubd2[j];
let {j in FLUXI} Lbd[j] := Lbd2[j];
option solver "/usr/local/ampl/cplexamp";
option cplex_options 'primalopt';
option presolve_eps 2.16e-05;
solve;
```

```
option display_1col '100000';
display flx > amplOutput.out;
display MGrwth;
```

Listing A.2: The AMPL model file.

```
set FLUXI;
set FLUXO;
set FLUXC;
set METS;

set FLUX := FLUXI union FLUXC union FLUXO;

set PRSI within {METS, FLUXI};
set PRSC within {METS, FLUXC};
set PRSO within {METS, FLUXO};

param Ai {PRSI} >= 0;
param Ac {PRSC};
param Ao {PRSO} <= 0;

param Ubd {FLUX};
param Lbd {FLUX};
param Ubd2 {FLUX};
param Lbd2 {FLUX};

var flx {j in FLUX} >= Lbd[j], <= Ubd[j];

maximize MGrwth: flx["Growth"];
```

```
subject to eqconstraints {i in METS}:
   sum {j in FLUXI : (i,j) in PRSI} Ai[i,j]*flx[j] +
   sum {j in FLUXC : (i,j) in PRSC} Ac[i,j]*flx[j] +
   sum {j in FLUXO : (i,j) in PRSO} Ao[i,j]*flx[j] = 0;
```

Listing A.3: The primary AMPL data file. Bounds are defined for intracellular and export reactions but not import reactions.

```
set FLUXI :=
R1 R2;

set FLUXC :=
R3 R4 R5;

set FLUXO :=
R6 R7 R8;

set METS :=
M1 M2 M3 M4 M5 Biomass;

set PRSI :=
M1       R1       1
M2       R2       1;

set PRSC :=
M1        R3       -1
M3        R3       2
M3        R4       -1
```

```
M4       R4     1
M5       R4     1
M2       R5     −1
M5       R5     −1
Biomass  R5     1;

set PRSO :=
Biomass      R6     −1
M2           R7     −1
M5           R8     −1;

param: Lbd :=
R3       0
R4       −10
R5       0
R6       0
R7       0;

param: Ubd :=
R3       10
R4       10
R5       INFINITY
R6       INFINITY
R7       INFINITY;
```

Listing A.4: The AMPL data file in which bounds are defined for import reactions to control resource availability.

```
param: Lbd2 :=
R1          0
R2          0;

param: Ubd2 :=
R1          INFINITY
R2          INFINITY;
```

Appendix B

Partitioning the Network

This appendix contains code for creating the network partition. Reactions go through a first cut and are classified as in the LFN, or in one of the HFN/IFN. The first cut is made by AMPL, and the second and final cut is made by PHP.

Listing B.1: Code for the AMPL file should go here

```
#!/usr/local/bin/ampl

# Read the model and dat files
model ecoli.mod;
data iJE660a_ATP_unbounded.dat;
data iJE660a_env.dat;

# make sure we get the correct environment variables
let {j in FLUXI} Ubd[j] := Ubd2[j];
let {j in FLUXI} Lbd[j] := Lbd2[j];

# Turn off that darn presolve and use original starting points
```

```
option presolve 0;
option reset_initial_guesses 1;
# solve the original problem
solve;
display flx["Growth"];
exit;

# add parameter to check that one of positive or negative flows is
    feasible
param FeasCheck;

# add paramter indicating positive flow, FlowDir = 1,
#     or negative flow, FlowDir = -1;
param FlowDir;
param Progress;
param Size;
# add the inf-norm bounding variable
var w;

# Add constraint to make sure sign of w is correct
subject to SignFlxBound: FlowDir * w >= 0;

# Add inf-norm type bounds
subject to FlowBound {i in TESTMETABOLITE, j in NOTTESTFLUX}:
   FlowDir * Aceq[i, j]*flx[j] <= FlowDir * w;

# make sure the metabolite is moving in the correct direction
subject to  SignedFlow {i in TESTMETABOLITE, j in TESTFLUX}:
    FlowDir * Aceq[i,j] * flx[j] >= 0;

# Fix the objective value
fix flx["Growth"];

# add new objectives for flows.
maximize MaxMetFlow:
    sum {i in TESTMETABOLITE, j in TESTFLUX}
        FlowDir * Aceq[i, j] * flx[j] - FlowDir * w;
objective MaxMetFlow;
let Size := card(METS);
#print "size of fluxc";
#print card(FLUXC);
# Remove the output and log file
shell 'rm -rf cutOneHFB';
shell 'rm -rf Log';
let Progress := 0;
for {i in METS} {
    let Progress := Progress+1;
    print "Progress:******************************";
    print 100*(Progress/Size);
    let TESTMETABOLITE := {i};
    # test metabolites into reactions (positive coefficients)
    for {j in FLUXC: (i,j) in PRSCEQ and Aceq[i, j] <> 0} {
        # get the reaction (flux) to be tested
        let TESTFLUX := {j};
        # get the reactions to compare with
        let NOTTESTFLUX :=
            {k in FLUXC: (i,k) in PRSCEQ and Aceq[i, k] <> 0} diff TESTFLUX
                ;
        # Set FeasCheck to 0 -i.e. fail
        let FeasCheck := 0;
        # Make sure metabolites are headed into reactions
        let FlowDir := 1;
        # solve the problem
        solve;
        # report connection
        # report connection
        if match(solve_message, "optimal solution") then {
            # Add the link if reaction dominates metabolite consumption
            if MaxMetFlow >= 0 then {
                printf "%s \t %s \t %4.4f \t %4.4f \t %4.4f \t %d \n",
                    i, j, MaxMetFlow, Aceq[i,j], flx[j], FlowDir >> cutOneHFB
                        ;
            };
            #Set FeasCheck to 1 -i.e. it passed
            let FeasCheck := 1;
        } else if match(solve_message, "unbounded problem") then {
            printf "%s \t %s \t Infinity \t %4.4f \t %4.4f \t %d \n",
                i, j, Aceq[i,j], flx[j], FlowDir >> cutOneHFB;
            #Set FeasCheck to 1 -i.e. it passed
            let FeasCheck := 1;
        };
        # Make sure metabolites are headed out of reactions
        let FlowDir := -1;
        # solve the problem
```

```
      solve;
      # report connection
      if match(solve_message, "optimal solution") then {
         # Add the link if reaction dominates metabolite consumption
         if MaxMetFlow >= 0 then {
            printf "%s \t %s \t %4.4f \t %4.4f \t %4.4f \t %d \n",
               i, j, MaxMetFlow, Aceq[i,j], flx[j], FlowDir >> cutOneHFB
                  ;
         };
         #Set FeasCheck to 1 -i.e. it passed
         let FeasCheck := 1;
      } else if match(solve_message, "unbounded problem") then {
         printf "%s \t %s \t Infinity \t %4.4f \t %4.4f \t %d \n",
            i, j, Aceq[i,j], flx[j], FlowDir >> cutOneHFB;
         #Set FeasCheck to 1 -i.e. it passed
         let FeasCheck := 1;
      };
      #If FeasCheck = 0, then both FlowDir are infeasible, which is
      #   is an error.  Record this in a log file
      if FeasCheck <> 1 then {
         printf "%s and %s was infeasibility in both flow directions.\n"
            ,
            i, j >> LogLFN
      };
   };
};
```

Listing B.2: Code for the PHP file should go here

```
#!/usr/bin/php/

<?php

$amplHFNIFN = explode("\n", file_get_contents($argv[1]));
$amplLFN = explode("\n", file_get_contents($argv[2]));
$rcnt = 0;
$LFN = array();
#This loop reads in the logged LFN reactions. Note that some of these
    might show up in the HFN or IFN for different metabolite-reaction
    pairs. This is taken care of after the HFN and IFN have been found.
for($k=0; $k<sizeof($amplLFN); $k++) {
        if(preg_match("/\S+\s+\S+\s+\S+\s+\S+\s+\S+/", trim($amplLFN[$k
            ]), $junk) > 0) {
                $line = preg_split("/\s+/", $amplLFN[$k]);
                array_push($LFN, $line[1]);
        }
}
#This loop reads in the logged HFN-IFN reactions.
for($k=0; $k<sizeof($amplHFNIFN); $k++) {
if(preg_match("/\S+\s+\S+\s+\S+\s+\S+\s+\S+\s+\S+/", trim($amplHFNIFN[$k
    ]), $junk) > 0) {
        $line = preg_split("/\s+/", $amplHFNIFN[$k]);
        $prod = floatval($line[3])*floatval($line[5]);
        if(preg_match("/\w+/", $line[1], $junk) > 0) {
                $reactions[$rcnt] = $line[1];
                $rcnt++;
        }
        $inpNum[$line[1]] = 0;
        $outNum[$line[1]] = 0;
}
}
$inpCnt = 0;
$outCnt = 0;
#This loop counts the number of metabolites that a reaction dominates
    consumption and production of.
for($i=0; $i<sizeof($amplHFNIFN); $i++) {
  if(preg_match("/\S+\s+\S+\s+\S+\s+\S+\s+\S+\s+\S+/", trim($amplHFNIFN[
      $i]), $junk) > 0) {
          $line = preg_split("/\s+/", $amplHFNIFN[$i]);
          $prod =       floatval($line[5]);
          if($prod < 0) {
                  $inputs[$inpCnt] = array($line[0], $line[1], $line[3],
                       $line[6], $line[2]);
                  $inpCnt++;
                  $inpNum[$line[1]]++;
          } else if($prod > 0) {
                  $outputs[$outCnt] = array($line[1], $line[0], $line
                      [3], $line[6], $line[2]);
                  $outCnt++;
                  $outNum[$line[1]]++;
          } else echo "There's a zero direction-something's wrong\n";
  }
}
```

```
#Remove duplicate reactions.
$reactionsUnique = array_values(array_unique($reactions));
$hfnCnt = 0;
$IFN = array();
#This loop checks the number of metabolites a reaction dominates in and
    out. If it can't do both, it must be in the IFN.
for($i=0; $i<sizeof($reactionsUnique); $i++) {
        if($inpNum[$reactionsUnique[$i]] > 0 &&
           $outNum[$reactionsUnique[$i]] > 0) {
                $HFN[$hfnCnt] = $reactionsUnique[$i];
                $hfnCnt++;
        } else {
                array_push($IFN, $reactionsUnique[$i]);
        }
}
sort($IFN);

#Any reactions in the HFN and IFN are removed from the initial set of
    LFN reactions.
$LFNUnique = array_values(array_unique(array_diff(array_diff($LFN, $HFN)
    , $IFN)));

#Write the partition to file.
file_put_contents($amplHFNIFNs["HFN"], implode("\n", $HFN));
file_put_contents($amplHFNIFNs["IFN"], implode("\n", $IFN));
file_put_contents($amplHFNIFNs["LFN"], implode("\n", $LFNUnique));
?>
```

Appendix C

Cycle Analysis Code

This appendix shows highlights of the Java implementation of the cycle finding algorithm and associated data structure.

Listing C.1: Code for the modified algorithm should go here

```
public void getCycles(String directory) {
        hashCodes = new ArrayList<Integer>(1);
        for (int i = 0; i < cycReactions.size(); i++) {
                recurseForCycles(cycReactions.get(i), cycReactions.get(i
                    ), directory);
                buildCycles(cycReactions.get(i), cycReactions.get(i),
                    new Graph(), directory);
                for (int j = 0; j < cycReactions.size(); j++) {
                        cycReactions.get(j).visited = false;
                        cycReactions.get(j).reachesRoot = false;
                        cycReactions.get(j).listOfOutgoingCycles = new
                            ArrayList<Reaction>(1);
                }
        }
        return;
}

private void recurseForCycles(Reaction root, Reaction vertex, String
    directory) {
        ArrayList<Reaction> nextRxns = vertex.getNextReactions(this);
        int numNextRxns = nextRxns.size();
        vertex.visited = true;
        for (int i = 0; i < numNextRxns; i++) {
                if (nextRxns.get(i).equals(root)) {
                        vertex.reachesRoot = true;
                        vertex.listOfOutgoingCycles.add(root);
                } else if (!nextRxns.get(i).visited) {
                        recurseForCycles(root, nextRxns.get(i),
                            directory);
                }
                if (nextRxns.get(i).reachesRoot) {
                        vertex.reachesRoot = true;
                        vertex.listOfOutgoingCycles.add(nextRxns.get(i))
                            ;
                }
        }
}

private void buildCycles(Reaction root, Reaction vertex, Graph cycle,
    String directory) {
        vertex.visited = false;
        cycle.addReaction(vertex);
        ArrayList<Reaction> outgoingCycles = vertex.listOfOutgoingCycles
            ;
        for (int i = 0; i < outgoingCycles.size(); i++) {
                if (outgoingCycles.get(i).equals(root)) {
                        cycle.addReaction(root);
                        // Write to File//////////////////////////////
                        int hash = cycle.getSortedHash();
                        if (!hashCodes.contains(hash)) {
                                cycle.writeFile(directory + "/cycles/
                                    cycle" + cycleCounter);
                                cycleCounter++;
                                hashCodes.add(hash);
                        }
                        cycle.removeForwardEdge(vertex, root);
                } else if (outgoingCycles.get(i).visited) {
                        cycle.addForwardEdge(vertex, outgoingCycles.get(
                            i));
                        buildCycles(root, outgoingCycles.get(i), cycle,
                            directory);
                        cycle.removeForwardEdge(vertex, outgoingCycles.
                            get(i));
                }
        }
        cycle.cycReactions.remove(vertex);
        vertex.visited = true;
}
```

Appendix D

Partition Results

The following tables list reactions found to be in the LFN, IFN, and HFN for E. coli.

Table D.1: Low Flux Network of E. coli.

Low Flux Network of E. coli									
ACPS	CYOE	EPD	HEMA	HEMX	MENDx1	PDXA	RHAD	THIM	UBIG
ADKx2	CYSG	FUCA	HEMB	HYAA	MENE	PDXB	RHAT	THIN	UBIH
ATOB	CYSGx1	FUCI	HEMC	ISPA	MENF	R0008	SAPA	THRCx1	YAES
BIOA	DEOAx1	FUCK	HEMD	ISPAx1	MENG	R0010	SERCx1	TNAAx1	
BIOB	DPPA	FUCO	HEME	ISPB	NAGA	R0011	THIB	UBIA	
BIOD	ENTA	FUCP	HEMF	MALX	NAGE	R0012	THIC	UBIB	
BIOF	ENTB	GALM	HEMG	MENA	NANA	R0019	THID	UBIC	
BRNQ	ENTD	GOR	HEMH	MENB	NANT	R0020	THIG	UBID	
CADA	ENTE	GSHA	HEML	MENC	NHAA	RHAA	THIK	UBIE	
CPSG	ENTF	GSHB	HEMM	MEND	OPPA	RHAB	THIL	UBIF	

Table D.2: Intermediate Flux Network of E. coli.

Intermediate Flux Network of E. coli									
ADD	CDD	EDD	GLNA	GUAC	NAGB	PNTA	R0016	TDKx1	USHAx8

Continued on next page

Table D.2 – continued from previous page

Intermediate Flux Network of E. coli									
ADHEx1	CDH	FABB	GLTB	KDPA	NUPGx5	PNTB	R0021	TNAA	USHAx9
ADKx4	CLS	FABBx1	GLTJ	LIG	PCKA	PPC	R0022	UDKx1	YBAS
AGP	COAE	FABBx2	GLTK	LIVJx1	PDXK	PROA	R0025	UPP	YGJG
AMN	CODA	FABBx3	GLTP	LPXA	PDXKx1	PROV	R0028	USHA	YICP
ANSA	CYCAx1	FABF	GPT	MAEA	PDXKx2	PURA	RIBA	USHAx1	YNBA
APT	CYCAx5	FADB	GPTx1	MAEB	PFKA	PUTAx1	RIBB	USHAx10	
ARAF	CYSK	FBP	GPTx2	METK	PGL	PYRG	SDAA	USHAx11	
ARGEx1	DADX	FOLE	GSK	MURA	PGSA	R0006	SERA	USHAx5	
ARGT	DUT	GALF	GSKx1	MURI	PNCB	R0009	SERB	USHAx6	
ASPA	EDA	GLK	GUAB	MUTT	PNCC	R0015	TDK	USHAx7	

Table D.3: High Flux Network of E. coli.

High Flux Network of E. coli									
ACCA	AVTA	DSDA	GLGP	ILVCx1	MTLD	NUPGx8	PTA	R0033	TPIA
ACEA	BGLX	ENO	GLMM	ILVD	MURB	NUPGx9	PTSG	R0034	TREA
ACEB	CARA	FABBx4	GLMS	ILVDx1	MURC	PABA	PURB	R0035	TREC
ACKA	CDDx1	FABD	GLMU	ILVE	MURD	PABC	PURBx1	R0036	TRKA
ACNA	CDSA	FABH	GLNH	ILVEx1	MURE	PANB	PURBx2	R0037	TRPA
ACS	CMK	FADA	GLPD	ILVEx2	MURF	PANC	PURC	RBSA	TRPC
ADDx1	CMKx1	FADD	GLPF	ILVN	MURG	PAND	PURD	RBSK	TRPCx1
ADHC	CMKx2	FADE	GLPK	KBL	MUTTx1	PANE	PURE	RFAL	TRPD
ADHE	COAA	FADL	GLPT	KDSA	MUTTx2	PANF	PURF	RIBD	TRPDx1
ADK	CODB	FADLx1	GLTA	KDSB	NADA	PAT	PURH	RIBDx1	TRXB
ADKx1	CYCA	FADLx2	GLTPx1	KDTA	NADB	PDXH	PURHx1	RIBE	TYNA
ADKx3	CYCAx2	FBAA	GLYA	KDTAx1	NADC	PDXHx1	PURK	RIBF	TYRA
ALAB	CYCAx3	FDHF	GLYAx1	KDTB	NADD	PDXHx2	PURL	RIBFx1	TYRB
ALDA	CYCAx4	FDNG	GLYAx2	KGTP	NADDx1	PDXHx3	PURM	RIBH	TYRBx1
ALDH	CYDA	FOCA	GMK	LACY	NADE	PFKB	PURN	RPE	TYRP
ALR	CYOA	FOLA	GMKx1	LEUA	NADF	PFLA	PURT	RPIA	UDK
AMTB	CYSC	FOLB	GND	LEUB	NDH	PFS	PURU	SAD	UDP
ARAA	CYSD	FOLC	GNTS	LEUC	NDK	PGI	PUTA	SCR	URAA

Continued on next page

Table D.3 – continued from previous page

High Flux Network of E. coli									
ARAB	CYSE	FOLD	GNTV	LIVH	NDKx1	PGIx1	PUTP	SDAC	USHAx2
ARAD	CYSH	FOLDx1	GPMA	LIVJ	NDKx2	PGIx2	PYKA	SDHA	USHAx3
ARAE	CYSI	FOLK	GPSA	LIVJx2	NDKx3	PGK	PYRB	SDHB	USHAx4
ARGA	CYSP	FOLP	GPSAx1	LLDP	NDKx4	PGM	PYRC	SERC	VGRO
ARGB	DADA	FRDA	GPTx3	LPCA	NDKx5	PGPA	PYRD	SPEA	XAPA
ARGC	DAPA	FRUA	GPTx4	LPDA	NDKx6	PHEA	PYRE	SPEB	XAPAx1
ARGD	DAPB	FRUK	GUAA	LPXB	NDKx7	PHEAx1	PYRF	SPEC	XAPAx2
ARGE	DAPC	FUMA	GUFP	LPXC	NRDA	PITA	PYRH	SPED	XAPAx3
ARGF	DAPD	GABD	Growth	LPXD	NRDAx1	PLSC	R0001	SPEE	XAPAx4
ARGG	DAPE	GABP	HISA	LPXK	NRDAx2	PNCA	R0002	SRLA1	XAPAx5
ARGH	DAPF	GABT	HISB	LYSA	NRDB	PNUC	R0003	SRLD	XAPAx6
AROA	DCD	GADA	HISBx1	LYSP	NRDD	PNUCx1	R0004	SUCA	XAPB
AROB	DCTA	GALE	HISC	MANA	NRDDx1	PNUE	R0005	SUCC	XYLA
AROC	DCUA	GALK	HISD	MANX	NRDDx2	POTA	R0007	TALB	XYLAx1
AROD	DCUAx1	GALP	HISE	MDH	NRDDx3	POTE	R0013	TDCC	XYLB
AROE	DCUB	GALPx1	HISF	MELA	NTPA	POXB	R0014	TDH	XYLE
AROF	DDLA	GALT	HISG	MELB	NUOA	PPA	R0017	TDHx1	XYLF
AROK	DEOA	GALU	HISI	META	NUPG	PPSA	R0018	THRA	YRBH
AROP	DEOB	GAPA	HISJ	METB	NUPGx1	PROB	R0023	THRAx1	ZWF
ARTP	DEOBx1	GATA	HISM	METC	NUPGx10	PROC	R0024	THRB	
ASD	DEOC	GATD	HPT	METD	NUPGx11	PROW	R0026	THRC	
ASNA	DFP	GATY	HTRB	METF	NUPGx2	PRR	R0027	THYA	
ASNB	DFPx1	GCVH	ICDA	METH	NUPGx3	PRSA	R0029	TKTA	
ASPC	DGKA	GDHA	ILVA	MGLA	NUPGx4	PSD	R0030	TKTAx1	
ATPA	DLD	GLGA	ILVB	MRAY	NUPGx6	PSSA	R0031	TMK	
ATPM	DLDx1	GLGC	ILVC	MTLA	NUPGx7	PSTA	R0032	TNAB	