Pathway Analysis of Metabolic Networks using Graph Theoretical Approaches

A dissertation

submitted by

Ehsan Ullah, B.Sc., M.Sc.

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

August 2014

Advisor: Prof. Kyongbum Lee, and Prof. Soha Hassoun

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

For my father and grandfather

Acknowledgments

The work described in this dissertation is the result of advice, guidance, support, and collaborations of many colleagues and friends.

I am grateful to my advisor, Prof. Soha Hassoun, for her patience and guidance. She is a very kind person but a strict professor and advisor with high expectations. She has played a vital role in my research and provided opportunities for my professional growth. I am thankful to my co-advisor, Prof. Kyongbum Lee, for his support and guidance. He has always been a source of inspiration for me. I am thankful to Prof. Shuchin Aeron for his guidance. I would like to thank Prof. Anselm Blumer and Prof. Cong T. Trinh for serving on my dissertation committee.

It was a great experience working with Prof. Alva Couch, Prof. Bruce Molay and Graduate Institute for Teaching (GIFT). You have taught me how to teach. I have also learned a lot from Prof. Diane Souvaine, Prof. Lenore Cowen, Prof. Benjamin Hescott, Prof. Samuel Guyer and Prof. Chris J. Myers.

I am thankful to my colleagues and friends Gautham Sridharan and Mona Yousefshahi for their collaboration and support.

I am thankful to Tufts University Computer Science Department, especially George Preble, Donna Cirelli, Jeannine Vangelist, Gail Fitzgerald, Michael Bauer, Erik Patton, and Jon Frederick for their support and help. I will miss you all.

I am thankful to my friends Zafar Imran, Nauman H. Khan, Mashhood Ishhaque, Bilal Ahmed, Salman Asif, Saeed Majidi and Ali Mirvakili who have been very kind and supportive during difficult times of my Ph.D.

Finally, I would not have reached this point without love and support of my parents, Aman Ullah and Rashda Aman, my wife, Mayra Ehsan, my brother,

Zeeshan Ullah and my son, Abdullah-bin-Ehsan. I love you all.

Ehsan Ullah

TUFTS UNIVERSITY

August 2014

Pathway Analysis of Metabolic Networks using Graph Theoretical Approaches

Ehsan Ullah

Advisor: Prof. Kyongbum Lee, and Prof. Soha Hassoun

Cellular pathways defining biochemical transformational routes are often utilized as engineering targets to achieve industrial-scale production of commercially useful biomolecules including polyesters, building blocks for polymers, biofuels, and therapeutics derived from isoprenoids, polyketides, and non-ribosomal peptides. Identifying target pathways can be expedited using computational tools, leading to reduced development cost, time, and effort, and enabling new discoveries with potential positive impact on human health and the environment.

This thesis addresses three cellular pathway identification problems within metabolic networks. In the first problem, we identify all stoichiometrically balanced, thermodynamically feasible and genetically independent pathways, known as Elementary Flux Modes (EFMs), that can be used to express flux distributions and characterize cellular function. We develop an algorithm, *gEFM*, that incorporates structural information of the underlying network to enumerate all EFMs. The results show that *gEFM* is competitive with state-of-the art EFM computation techniques for several test cases, but less so for networks with larger number of EFMs. In the second and third problems, we identify individual target pathways with pre-specified characteristics. We develop an algorithm, *PreProPath*, for identifying a target pathway for up-regulation such that the path is predictable in behavior, exhibiting small flux ranges, and profitable, containing the least restrictive flux-limiting reaction

in the network. The results show that *PreProPath* can successfully identify high ethanol production pathways across multiple growth rates, and for succinate production in *Escherichia coli* (*E. coli*) as published in the literature. We also develop an algorithm, *Dominant-Edge Pathway*, that identifies thermodynamically-favored reactions along a pathway within the network from a given source metabolite to the desired destination. The algorithm is used to identify thermodynamically-limiting pathways in *Zymomonas mobilis* (*Z. mobilis*), *E. coli* and rat liver cell.

The novelty of this thesis is in utilizing graph-based methods to enumerate EFMs and to efficiently explore the pathway design space. Overall, the thesis advances the state-of-the-art techniques for metabolic pathway analysis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Engineering of living cells has shown promise in the production of commercially useful biomolecules, including polyesters [3], building blocks for industrial polymers [4], biofuels [5], and therapeutic natural products derived from isoprenoids [6, 7, 8, 9], polyketides [10, 11], and non-ribosomal peptides [9]. Advancing the engineering of biological organisms will lead to reduced development cost, time, and effort, which in turn will enable new discoveries that have a positive impact on human health and the environment.

Current cell engineering approaches broadly fall into one of three categories. The first approach is to embed non-native reactions into a host organism to enable a synthesis route. For example, production of butanol [12, 13] and isopropanol [14], two potential biofuels, was enabled in *E. coli* by importing different genes from *Clostridium acetobutylicum*. The second approach is to eliminate pathways that compete for cellular resources [2] or otherwise inhibit product synthesis. In a recent example, Yomano and co-workers deactivated the methyl glyoxal pathway to reduce catabolite repression and thereby accelerate co-metabolism of hexose and pentose

sugars into ethanol [15]. The third approach is to tune the activities of existing pathways, for example by altering enzyme concentrations through gene expression changes. It should be noted that the above categorization is far from strict. Indeed, combinations of the various approaches are increasingly used to simultaneously enable new synthesis routes and optimize the yield. Keasling and co-workers have recently reported on engineered strains of *E. coli* capable of producing a variety of fatty esters (biodiesel), fatty alcohols, and waxes directly from simple sugars [5]. Fatty acid overproduction was achieved by over-expressing native thioesterases and acyl-CoA ligases while eliminating β-oxidation. To produce branched chain alcohols which are non-native to *E. coli*, a biosynthetic operon for branched chain amino acids (*thrABC*) was over-expressed, genes encoding competing pathways were deleted, and additional genes encoding the missing synthesis steps were imported from *Salmonella typhimurium* and *Corynebacterium glutamicum* [16].

A common thread in these approaches is that the engineered interventions targeted pathways, as opposed to individual reactions, as the functional units of cellular biosynthesis. While experimental approaches have often achieved significant success, identifying intervention targets and verifying result optimality remain open questions due to the complexity of biological systems. In this regard, computational methods can serve as useful guides to efficiently explore the pathway design space.

We explore in this thesis three pathway identification problems. In the first problem, we revisit the problem of EFM analysis, where we seek to enumerate all stoichiometrically balanced, thermodynamically feasible and genetically independent pathways. We utilize a graph-based approach to solve this problem. In the second and third problems, we shift to focus on identifying individual pathways with

pre-specified characteristics. Our solutions to these two problems are also based on graph-based algorithms, and offer computational efficiency with runtimes that are polynomial in the size of the network graph. Using such a guided-search approach provides a computationally efficient alternative to enumeration-based approaches.

1.1 Computational Tools to Guide Strain Engineering

Computational tools can play a critical role in expediting the design process by exploring design options and validating design choices using simulations before performing lab experiments [17]. Stoichiometric, mass balance, thermodynamic and regulatory constraints play a critical role in defining and characterizing the flux space of metabolic networks. Constraints have driven two types of analyses. One type is based on mathematical optimization frameworks such as linear, quadratic, and non-linear programming, and are exemplified by techniques such as Flux Balance Analysis (FBA) [18] and Flux Variability Analysis (FVA) [19]. Such approaches are used frequently not only to optimize strains [20, 21, 22, 23] but to also validate models [24, 25, 26].

Another type of analysis is based on pathway identification, where the focus is on characterizing individual or groups of pathways and their contributions instead of the overall network behavior. Such analysis has mostly focused on EFM analysis [27], and there has been little computational effort in efficiently identifying a single reaction or pathway of interest.

1.1.1 EFM Analysis

This thesis addresses problems of identifying network pathways of interest for strain engineering. One particular powerful pathway analysis technique for analyzing cellular metabolism is EFM analysis. A "flux mode" represents a steady-state flux pattern where the proportions of fluxes are fixed while their absolute magnitudes are indeterminate [27]. EFM analysis decomposes a metabolic network into routes that have three properties: thermodynamic feasibility, quasi steady-state operation and independence of other pathways [27]. Thermodynamic feasibility imposes that each irreversible reaction proceeds to have a non-negative flux (turnover) rate. Quasi steady-state operation ensures that metabolites internal to the network are neither accumulated nor depleted. Mutual independence of other pathways, together with the other two properties, guarantees that the EFM decomposition is unique. EFM analysis has been used for increasing product yield [28, 2], validating metabolic model construction [29], analyzing and understanding metabolic networks including robustness, redundancy, reaction correlations, and cellular regulation [30, 31, 32, 33, 34, 35, 36, 37, 35], analyzing competitive microbial strategies [38], identifying substrate cycles [39, 40], and assessing plant fitness and agricultural productivity [41].

Computing EFMs has been shown equivalent to computing the extreme rays of a convex pointed cone [1]. More precisely, once each reversible reaction is split into a forward and a reverse reaction, the steady-state operation and irreversibility constraints define a pointed convex cone that lies in the positive quadrant of the space defined by the network reactions. Any steady-state flux vector for the network lies within this convex cone and can be expressed as a non-negative linear

combination of the extreme rays (edges) of the cone. Algorithms for computing the generating vectors of a convex polyhedral cone can be utilized to compute the elementary modes. To compute EFMs, Schuster and Hilgetag [27] applied one such algorithm [42] where rows of the transposed stoichiometric matrix augmented by the identity matrix are combined pairwise to generate the elementary modes. This method was later elaborated by adding a dependency test criterion to eliminate redundant modes [43]. The earlier Schuster and Hilgetag approach [27] is referred to as the 'Canonical Basis' approach [1]. Wagner [44] and Urbanczik and Wagner [45] proposed to first derive the basis vectors of the null space of all steady-state conditions, and then calculate the elementary modes by linearly combining the basis vectors. The algorithm is referred to as the "Null Space" approach, and provides significant speed up over the Canonical Basis approach [45]. Gagneur and Klamt [1] showed that the two approaches, the Canonical Basis and the Null Space, are variants of the double-description method, used to enumerate the extreme rays of a convex cone (see [46, 47, 48, 49] for a description of this method). The two most widely used EFM tools, Metatool [50] and EFMTool [51], are based on the Null Space approach.

We revisit in this thesis the Canonical Basis approach as described by [43], but from a graph-traversal perspective. The basic underlying idea is from the works of Mavrovouniotis *et al.* on the synthesis of metabolic pathways from a given substrate(s) to a given product(s) [52, 53, 54]. Conceptually, the Mavrovouniotis approach iteratively incorporates the set of stoichiometric constraints associated with each metabolite, and transforms an initial set of reactions (one-step pathways) into a final set of pathways that satisfy all the constraints. Schuster *et al.* [43] suggested

that the row elimination within the Canonical Basis approach is equivalent to the Mavrovouniotis approach; however, it was suggested that using matrix formalism instead of graph theory is more elegant when tackling structural analysis of metabolic networks.

We formulate in this thesis an algorithm, *gEFM*, that relies on graph traversal to compute the EFMs. The algorithm combines the Mavrovouniotis pathway synthesis approach [52] with the dependency test identified by Schuster and Hilgetag [43]. The main contribution of this work is showing that graph-based approaches are viable for computing EFMs. Naturally, this contribution extends to enumerating rays of a convex cone. Because it retains the network structure, *gEFM* is accessible and intuitive. Our results show that the runtime of *gEFM* compares well or improves on that of comparable tools, namely Metatool [50] and EFMTool [51], for a number of networks. We also examine the impact of network compression and constraint ordering, and show that there is a constraint ordering based on the analysis of the underlying network structure that benefits *gEFM*.

1.1.2 Pathway Identification

Large-scale stoichiometric models of cellular metabolism nearly always have large degrees of freedom. Since the models are severely underdetermined, it is not possible to specify an operating point, i.e. unique flux distribution; rather, the model circumscribes an operating cone, i.e. flux ranges bounded by physicochemical, regulatory, and measurement-derived constraints. The flux ranges provide a quantitative basis to evaluate the profitability of an engineering intervention, as some interventions will only produce marginal improvements in flux that are subsumed by the uncertainty

in the model. We present in this thesis an algorithm for identifying a target pathway for up-regulation such that the path is predictable in behavior, exhibiting small flux ranges, and profitable, containing the least restrictive flux-limiting reaction in the network. Our *PreProPath* algorithm identifies a path from a starting substrate to a desired product that most likely contains one or more flux-limiting reactions, where the likelihood is determined by considering the degrees of freedom in the network. We evaluate the algorithm through two case studies and comparisons with other pathway engineering strategies and examples discussed in the literature.

The activity or flux (rate of turnover of molecules) distribution through a cellular reaction network is highly uneven, and it is unlikely that every possible route leads to an equally valid target with the same capacity. A more plausible scenario is that the pathways' degrees of engagement vary with the cell's operating environment (e.g. temperature, pH and nutrient concentration) and regulatory state. In this context, finding a favorable reaction route with the highest degree of engagement is an important next step for biochemical pathway analysis, especially for the purpose of engineering a synthetic pathway. We present in this thesis a pathway search algorithm based on thermodynamic weights. We utilize the Gibbs free energy change (ΔG), a metric whose sign predicts if the reaction favors the formation of the reactants (positive sign) or products (negative sign). A ΔG close to zero indicates that a reaction is near equilibrium. Among parallel reactions, our algorithm selects the energetically favored or dominant reaction based on the sign and magnitude of the ΔG. Importantly, this algorithm identifies the thermodynamically-limiting reactions in the network from a given source metabolite to the desired destination. We demonstrate the utility of our algorithm by identifying thermodynamically-limiting

pathways in *Z. mobilis*, *E. coli* and a rat liver cell.

1.2 Thesis Contributions

This thesis provides three graph-based algorithms to solve pathway identification problems in metabolic networks. The key contributions are:

- Developing a graph-based algorithm, *gEFM*, for computing EFMs.
- Identifying robust constraint ordering for the Canonical-Basis approach by incorporating structural information of the underlying network.
- Demonstrating that graph-based approaches are as viable for implementing the double-description method for enumerating EFMs as matrix-based approaches.
- Showing that *gEFM* is competitive with state-of-the-art EFM computational techniques for several test cases, but less so for networks with a larger number of EFMs.
- Creating an algorithm,*PreProPath*, for identifying a path that is predictable in behavior, exhibiting small flux ranges, and profitable, containing the least restrictive flux-limiting reaction in the network.
- Using *PreProPath* to identify high ethanol production pathways across multiple growth rates, and high succinate production pathways in *E. coli*.
- Adapting the bottleneck edge algorithm [55] to identify bottleneck reactions in metabolic networks.

- Developing a generalized algorithm,*Dominant-Edge Pathway*, for identifying a pathway with thermodynamically favored reactions.

- Using *Dominant-Edge Pathway* to identify thermodynamically-limiting pathways in *Zymomonas mobilis* (*Z. mobilis*), *E. coli* and a rat liver cell.

1.3 Thesis Organization

This thesis consists of 6 chapters. Chapter 2 provides an overview of metabolic pathway analysis with a focus on EFM analysis and its important applications.

Chapter 3 describes the *gEFM* algorithm, a graph-based algorithm used to enumerate all genetically independent, thermodynamically feasible pathways, EFMs, that can operate at steady state. The algorithm is proved correct and compared to existing approaches for computing EFMs.

Chapter 4 describes an algorithm, Predictable Profitable Path (*PreProPath*), a graph-based algorithm used to identify a pathway containing potential engineering targets for over production of a desired product from a given source. The algorithm uses flux measurements and uncertainties associated with the measurements to identify engineering targets.

Chapter 5 describes *Dominant-Edge Pathway* algorithm, a graph-based algorithm used to identify bottleneck reactions and an associated pathway from a specified source metabolite to a desired target metabolite. *Dominant-Edge Pathway* is used to identify thermodynamic bottleneck reaction in the metabolic networks.

Chapter 6 summarizes the thesis and outlines directions for future research.

Chapter 2

Background

Pathway analysis is a powerful approach to enable the rational design of biochemical networks for optimizing metabolic engineering and synthetic biology objectives such as production of desired chemicals or biomolecules from specific nutrients. This chapter provides a historical background on cellular pathway analysis, a review of the theoretical aspects of EFM analysis, an overview of computational applications where the use of EFM analysis is imperative, and a review of approximate alternative EFM decomposition techniques.

2.1 Metabolic Pathway Analysis: A Historical Perspective

Foundations for stoichiometric pathway analysis were laid by Clarke in 1980 while focusing on network stability [56]. Clarke considered all the reversible reactions as irreversible reaction pairs, and the feasible steady-state flux space formed a convex pointed cone. His work provided the basis for convex analysis for metabolic

networks operating at steady state. Seressiotis and Bailey developed a software system for metabolic pathway synthesis (MPS) to identify independent pathways in a metabolic network [57]. Their algorithm identified all independent pathways from a given source metabolite to the desired destination metabolite. Mavrovouniotis et al. developed a computer-aided system for synthesis of pathways in a metabolic network [52]. Their algorithm iteratively satisfied steady-state constraints to build all possible pathways in a network. Schuster et al. further extended Mavrovouniotis algorithm by using the concept of genetic independence to generate elementary flux modes [27]. Later, Schilling et al. introduced the concept of Extreme Pathways (EP), a minimal set of pathways in a metabolic network that satisfies all EFM conditions (see next section) along with the following condition: an extreme pathway cannot be represented as a non-negative linear combinations of other extreme pathways [58]. It is important to compute the EFMs and not just the EPs because many biochemically important pathways such as glycolysis are not found using EP analysis [59].

2.2 Elementary Flux Mode Analysis

A flux mode is a steady-state flux pattern in which flux proportions are fixed while their absolute magnitudes are indeterminate [36]. Given an $m \times n$ stoichiometric matrix, $\mathbf{S}$, representing m internal metabolites and n reactions, and a n-vector $\mathbf{v}$ of reaction fluxes, three conditions must be met to label $\mathbf{v}$ as an "elementary flux mode", or "elementary mode", or "elementary pathway":

1. **EFM Condition 1 (C1):** The network reactions proceed in a direction dictated by thermodynamic feasibility. The flux in a reaction is greater than

or equal to zero if the reaction is irreversible. This condition can be expressed as $v_i \geq 0$ for all irreversible reactions.

2. **EFM Condition 2 (C2):** The network is in quasi steady-state condition with no accumulation or depletion of internal metabolites in the network. Mathematically, this condition can be expressed as $\mathbf{S}\ \mathbf{v} = \mathbf{0}$, where the rows of $\mathbf{S}$ include only metabolites internal to the network.

3. **EFM Condition 3 (C3):** Each elementary mode $\mathbf{v}$ must be genetically independent from any other elementary mode in the network. In other words, there is no other vector $\mathbf{y}$ ($\mathbf{y} \neq \mathbf{v}$ and $\mathbf{y} \neq \mathbf{0}$ and $\mathbf{y}$ fulfills **C1** and **C2**) such that the set of reactions participating in $\mathbf{v}$ is strictly a proper subset of the reactions in $\mathbf{y}$.

EFMs can be used to characterize the flux space of a network operating at steady-state. The feasible flux space at steady-state comprises all possible operating states and is captured via a pointed convex polyhedron if all the reactions in the network are irreversible, as shown in Figure 2.1. Mathematically, the feasible steady-state flux space of a network with m internal metabolites and n reactions can be represented as:

$$\mathbf{P} = \{\mathbf{v} \in \mathbb{R}^n : \mathbf{S}\ \mathbf{v} = \mathbf{0} \text{ and } v \geq 0\} \tag{2.1}$$

where $\mathbf{S}$ is stoichiometric matrix of the network and $\mathbf{v}$ represents a steady-state flux. Networks with reversible reactions can be reconfigured by splitting reversible reactions into irreversible reaction pairs. The extreme rays (edges) of the polyhedral correspond to the EFMs and an extreme ray cannot be expressed as a linear

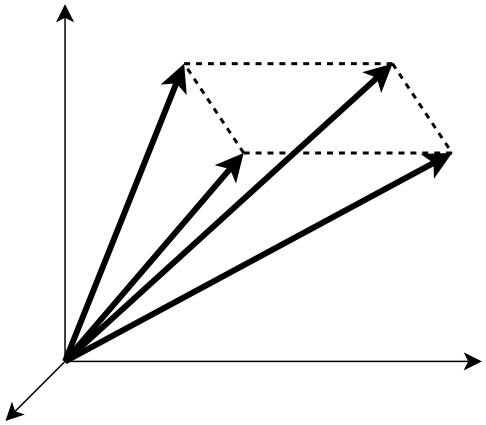


Figure 2.1: A pathway (mode) is geometrically represented as a ray. The dark lines are the extreme rays of a convex cone drawn in 3-dimensional space. The extreme rays correspond to EFMs.

combination of two others. Any steady-state flux distribution in the network can be represented as a non-negative, perhaps non-unique, linear combination of EFMs.

There are two known techniques for computing the extreme rays or vertices of a polyhedron. The first is based on the double-description method, a convex analysis technique used to compute the extreme rays of convex polyhedron. The second is based on pivotal methods, similar to the simplex algorithm. While the double-description method has been directly utilized in computing EFMs, pivotal methods have not been directly utilized to implement a tool for computing EFMs. Furthermore, the performance of pivotal techniques relative to those based on the double-description method is an open question. Here, an overview of existing EFM computational techniques based on the double-description method is presented.

2.2.1 The Double-Description Method

The double-description method provides a mechanism for converting between two equivalent descriptions of a convex cone [49]. The first description captures the

constraints that form the convex cone, while the second description enumerates the extreme rays (or EFMs). The double-description method has been rediscovered under different names such as Motzkin elimination [46], Chernokova's algorithm [47], and beneath-and-beyond methods [60, 61].

The double description method provides an alternative representation of the flux space by transforming the representative matrix of the network $\mathbf{A}$ to an equivalent representative matrix $\mathbf{R}$. Given a representative matrix $\mathbf{A}$ of the network the steady-state flux space can be described as

$$\mathbf{P} = \{\mathbf{v} = \mathbf{R}\boldsymbol{\lambda} \; \forall \; \lambda \geq 0\} \tag{2.2}$$

where $\mathbf{R}$ represents the set of extreme rays.

The general steps of the double-description method [1] are shown in Figure 2.2. First, matrices $\mathbf{A}$ and $\mathbf{R}$ are initialized. Next, constraints in $\mathbf{A}$ are iteratively processed. Each row of matrix $\mathbf{A}$ represents a constraint. The constraint is satisfied by combining adjacent rays (rays represented as columns) in $\mathbf{R}$ to generate new rays, a step referred to as Gaussian Combination [1]. The double-description method then identifies adjacent rays that lie on the same cone face intersected by the current constraint. Matrix $\mathbf{R}$ is augmented with additional columns representing newly generated rays. When all constraints are processed, a post-processing step removes futile two-cycles that result from splitting reversible reactions, and split reactions are combined to restore the original network configuration. The final set of EFMs is in $\mathbf{R}$.

Although the treatment of reversible reactions provides a subtle difference

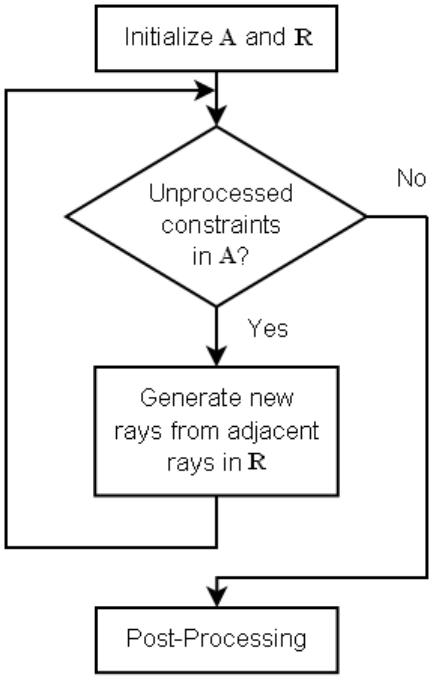


Figure 2.2: Flowchart of the double-description method [1].

between EFMs and extreme pathways (EPs) [58, 62], EFM analysis can also be applied by treating reversible reactions in the same way as that of EP analysis. For the presentation that follows, it is assumed that reversible reactions in **S** are split into forward and reverse independent reactions. Earlier algorithms such as Schuster's algorithm [27] used non-split reversible reactions by processing reversible reactions differently compared to irreversible reactions.

The double-description method has been adapted for EFM computation, and two techniques, the Canonical Basis [27] and Null Space [44] approaches, have emerged. The two approaches differ in the initial representative matrix **A**. The Canonical Basis approach utilizes the constraints within **S** matrix, thus representing constraints on metabolite balancing. The Null Space approach utilizes the null space of the **S** matrix, which provides a set of constraints that describe the feasible

Table 2.1: Outline of Canonical Basis and Null Space techniques, modified from [1].

Steps	**Canonical basis approach**	**Null Space approach**
Reconfiguration	$\mathbf{S} \leftarrow [\mathbf{S} \ \ -\mathbf{S}_{rev}]$	$\mathbf{S} \leftarrow [\mathbf{S} \ \ -\mathbf{S}_{rev}]$
Initialization	$\mathbf{A} \leftarrow \mathbf{S}$ $\mathbf{R} \leftarrow \mathbf{I_n}$	$\mathbf{A} \leftarrow null(\mathbf{S})$ $\mathbf{R} \leftarrow \mathbf{A}$
Constraint processing	for each unprocessed row A_i of $\mathbf{A}$ $J^+ \leftarrow \{j \in J : A_i \cdot \mathbf{r}^j > 0\}$ $J^- \leftarrow \{j \in J : A_i \cdot \mathbf{r}^j < 0\}$ $J^0 \leftarrow \{j \in J : A_i \cdot \mathbf{r}^j = 0\}$ $\mathbf{R}' \leftarrow \{\mathbf{r}^j : j \in J^0\}$ for$(j^+, j^-) \in J^+ \times J^-$	for each unprocessed row A_i of $\mathbf{A}$ $J^+ \leftarrow \{j \in J : r_i^j > 0\}$ $J^- \leftarrow \{j \in J : r_i^j < 0\}$ $J^0 \leftarrow \{j \in J : r_i^j = 0\}$ $\mathbf{R}' \leftarrow \{\mathbf{r}^j : j \in J^0 \cup J^+\}$ for$(j^+, j^-) \in J^+ \times J^-$
Adjacency test	if $\mathbf{r}^{j^+}$ and $\mathbf{r}^{j^-}$ adjacent in $\mathbf{R}$	if $\mathbf{r}^{j^+}$ and $\mathbf{r}^{j^-}$ adjacent in $\mathbf{R}$
Gaussian combinations	$\mathbf{R}' \leftarrow \mathbf{R}' \cup \{(A_i \cdot \mathbf{r}^{j^+})\mathbf{r}^{j^-} - (A_i \cdot \mathbf{r}^{j^-})\mathbf{r}^{j^+}\}$	$\mathbf{R}' \leftarrow \mathbf{R}' \cup \{r_i^{j^+}\mathbf{r}^{j^-} - r_i^{j^-}\mathbf{r}^{j^+}\}$
Update	$\mathbf{R} \leftarrow \mathbf{R}'$	$\mathbf{R} \leftarrow \mathbf{R}'$
Post-processing	$\mathbf{R} \leftarrow \mathbf{R} \setminus \{\text{futile two-cycles}\}$	$\mathbf{R} \leftarrow \mathbf{R} \setminus \{\text{futile two-cycles}\}$
Reconfiguration	back-configuration of $\mathbf{R}$	back-configuration of $\mathbf{R}$

flux space in which all metabolites internal to the network are balanced. Conceptually, during the execution of the double-description method, the Null Space approach enforces only thermodynamic feasibility constraints (ensuring every flux mode coefficient is positive for irreversible reactions), while the Canonical Basis approach enforces both thermodynamic feasibility and metabolite balancing constraints. The differences between the Canonical Basis and Null Space implementations of the double-description method are summarized in Table 2.1. A unifying description of the Canonical Basis and Null Space approaches were presented by Gagneur and Klamt [1].

2.2.2 The Canonical Basis Approach

The double-description method was first used by Schuster et al. to compute EFMs [27]. The algorithm was an adaptation of Mavrovonoitis' approach for synthesizing balanced pathways in metabolic networks [52]. Later, Schuster's algorithm was re-

labeled as the Canonical Basis approach. Schuster's algorithm provided a matrix-based implementation of the double-description method, and Schuster et al. claimed that graph-based implementations are not viable for computing EFMs.

In the Canonical Basis approach, **S** is used as the network representative matrix **A**. Each row in **A** corresponds to an internal metabolite and represents a mass-balance constraint. The corresponding matrix **R** is initialized with the identity matrix of size n, where each column r in **R** represents a flux vector (a ray in the flux space) that is initialized to a network reaction. When processing constraint A_i, the net production or consumption of ith metabolite in r^j is represented by scalar product $A_i r^j$. A positive value indicates net production, a negative value represents a net consumption, and zero value represents zero net consumption or production of the metabolite (metabolite is balanced). In each iteration of the double-description method, the flux vectors are divided into three groups: J^+ (flux vectors producing the metabolite), J^- (flux vectors consuming the metabolite), and J^0 (flux vectors for which the metabolite is balanced). In the Gaussian Combination step, two rays are combined such that metabolite i is balanced. The adjacency test ensures that only adjacent rays with respect to constraint A_i are combined. There are three possible implementations of adjacency testing. In the first, the adjacency of two rays in J^+ and J^- is decided using a rank test. In the second, two rays are combined, and then tested for dependency on all newly generated rays within **R**. In the third, two rays are combined, and then tested for dependency on any existing ray r^+ or r^- or r^0 within **R**. The performance of the rank test is dependent on the size of the input matrix. The performance of the dependency testing is dependent on the number of generated rays either in the current step (second test), or in the prior

step (third test) of the algorithm. In the second test, the number of comparisons for each generated ray is equal to the product of the number of rays in J^+ and J^-. In the third test, the number of comparisons for each generated ray is equal to the sum of the number of rays in J^+, J^- and J^0. The third test often outperforms the second test.

2.2.3 The Null Space Approach

The Null Space approach was proposed by Wagner [44]. Both the network matrix $\mathbf{A}$ and equivalent representative matrix $\mathbf{R}$ are initialized using the null space of S,

$$\mathbf{A} = \mathbf{R} = \text{null}(\mathbf{S}) = \{\mathbf{v} \in \mathbb{R}^n : \mathbf{S}\ \mathbf{v} = \mathbf{0}\} \tag{2.3}$$

The double-description method processes each row of $\mathbf{A}$, which represents constrains on the a constraint that enforces the thermodynamic feasibility of the associated reaction. The constraint associated with each row is thermodynamic feasibility of the reaction corresponding to the row. Relative activity of reaction i in flux vector j is represented by r_i^j. A positive value indicates the reaction operating in the forward direction (thermodynamically feasible reaction), a negative value indicates the reaction operating in the reverse direction (thermodynamically infeasible reaction), and zero value represents the reaction is not active meaning the reaction is not present in the pathway represented by the flux vector. In each iteration of the algorithm, the flux vectors are grouped into three groups: J^+ (flux vectors with the thermodynamic feasible reaction), J^- (flux vectors with the thermodynamic infeasible reaction), and J^0 (flux vectors for which the reaction is not active). The thermodynamic feasibility of the reaction is satisfied by combining flux vectors r^{j^+}

and r^{j^-} (Gaussian combination). The Gaussian combination step combines two balanced flux vectors resulting in a balanced flux vector. Only adjacent flux vectors are combined to generate independent flux vectors.

2.3 Imperative Uses for EFM Analysis

While EFM analysis has been utilized in many applications, here we review some applications where the use of EFM analysis is imperative as it provides fundamental computational advantages that cannot be captured by other computational methods.

2.3.1 EFM to Characterize the Flux Space

As each flux distribution, $\mathbf{v}$, can be expressed as a non-negative linear combination of EFMs, it is desirable to compute the contribution of each EFM. This problem can be mathematically expressed as:

$$\mathbf{v} = \mathbf{P}\alpha \tag{2.4}$$

where α_i in α describes the contribution of a particular EFM in the set of EFMs, $\mathbf{P}$. However, such a contribution may not be unique as the system is typically underdetermined, and several approaches have been proposed to identify either a single representative α or a range of relevant values.

In the α-spectrum method [63], the contribution of each EFM to the flux distribution can be characterized by finding the range of each α_i. This can be

obtained by solving the following linear optimization problems:

$$\begin{aligned} &\text{Minimize (or Maximize)} \quad \alpha_i \\ &\qquad\qquad \text{Subject to} \\ &\qquad\qquad\qquad \mathbf{v} = \mathbf{P}\alpha \\ &\qquad\qquad\qquad 0 \leq \alpha_i \leq 1 \text{ for } i = 1, 2, 3, \ldots n \end{aligned} \tag{2.5}$$

The α-weighting of one EFM is dependent on α-weightings of other EFMs. Schwartz and Kanehisa selected the unique weightings that minimize distance between EFMs and the given flux distribution using quadratic programming [64] to solve the following objective in the optimization problem above:

$$\text{Minimize} \quad \Sigma\ \alpha_i^2 \tag{2.6}$$

Hard constraints such as network structure and thermodynamic feasibility [65], as well as soft constraints such as regulatory and environmental constraints can further limit the feasible flux space and provide tighter bounds for the α-weightings. For example, Covert et al. were able to eliminate 67.5% of extreme pathways by adding regulatory constraints to an example network representing core metabolism.

2.3.2 EFM Count as a Representative Metric for Metabolic Function

The number of EFMs can be used as a direct measure of metabolic function. EFM analysis was used to design an E. coli cell minimal in function, only capable of sustaining cell growth and producing ethanol from pentoses and hexoses [2]. The

minimal cell functionality was identified by reducing the number of EFMs by knocking out the reactions participating in a large number of EFMs having lower ethanol and biomass yield. This was achieved through the successive application of the following steps:

1. Each network reaction was scored based on the number of EFMs remaining after knocking out the reaction. The score signifies the participation of reactions in cell functionality.

2. The maximum ethanol and biomass yield was computed in each EFM in order to filter EFMs with lower ethanol and biomass yield.

3. Reactions with the least score and maximum ethanol and biomass yield were kept in the cell whereas other reactions were considered for knockouts.

A total of 15000 EFMs were computed for the wild type cell, of which 1000 EFMs were capable of converting sugars to biomass and ethanol. The designed minimal cell was achieved using six gene knockouts and contained only eight pathways capable of converting sugars to biomass and ethanol. The engineered cell was experimentally validated. Flux-balance based optimization frameworks (e.g. OptKnock [21] and MOMA [66]) identify cell modifications to optimize both growth and metabolite production. Such approaches are not suited to minimize cell function defined in terms of EFM count as in the Trinh approach.

2.3.3 EFM Count as a Representative Metric for Redundancy

The number of EFMs or EPs can be used as a metric to represent redundancy in a biochemical network. Papin et al. defined redundancy in the context of an external

state, defined as a vector of exchange fluxes of the network [31]. Redundancy was defined as the number of EPs that have same external state. A high number of EPs corresponding to an external state indicates redundancy in the network. Cyclical internal EPs do not contribute to redundancy analysis. Using this technique, Papin et al. observed a high degree of redundancy in *Haemophilus influenzae* (*H. influenza*) network [31]. An average of 49 EPs corresponded to a unique external state associated with the production of non-essential amino acids without production of succinate. It was also observed that the distribution of redundancy across unique external states is not uniform indicating that some external states have more redundant pathways compared to others. Using the same technique, Price et al. calculated redundancy for *Helicobacter pylori* (*H. pylori*) network [67]. An average of 46 extreme pathways per unique external state was identified for *H. pylori*, indicating a more rigid network compared to *H. influenzae*.

2.3.4 Using EFMs to Define Network Robustness

Stelling et al. used the number of EFMs as a qualitative indicator of network robustness [30]. Stelling et al. defined network robustness as insensitivity of network function to perturbations such as mutations. The results of random gene deletions in the central metabolism of *E. coli* show that the network is robust as the maximal biomass yield of mutants is comparable to that of the wild type strains. Additionally, Stelling et al. related the number of EFMs to the functions performed by a network. The total number of EFMs provides a quantitative measure of degree of freedom for cell to perform a cell function. For example, it was shown that *E. coli* could consume glucose in 45 times more different ways compared to acetate. The

number of EFMs provides a network robustness metric but does not correlate with mutant viability.

2.3.5 Cyclical EFMS are Equivalent to Substrate cycles

A substrate cycle refers to a set of metabolic reactions arranged in a loop resulting in zero net consumption or production of the metabolites. The cycle operates by transforming a cofactor, e.g. oxidizing a reducing equivalent. Substrate cycles have been found experimentally in many parts of metabolism, and have more recently been ascribed physiological functions, for example, thermogenesis [68]. Futile or substrate cycles are not to be confused with infeasible loops, which refer to thermodynamically infeasible cycles. Schuster et al. suggested that substrate cycles can be identified from cyclic EFMs with zero net metabolite production or consumption [59].

Teusink et al. identified ATP-based substrate cycles in Lactobacillus plantarum WCFS1 network using EFM analysis [39]. ATP, ADP, phosphate, water, and protons were considered as external metabolites. EFM analysis was performed on the modified network and the resulting cyclical EFMs were labeled substrate cycles. To avoid enumerating all EFMs for larger networks, Gebauer et al. [69] used two different approaches, mixed integer linear programming and EFMEvolver [70] to identify ATP-based substrate cycles. An artificial reaction for ADP phosphorylation was added to the network and all the EFMs containing this reaction were considered substrate cycles. To identify non-ATP-based substrate cycles and to also tackle the lack of scalability of EFM analysis, Sridharan identified substrate cycles in human liver metabolism model (HepatoNet1) in the context of modularity

[71]. Sridharan et al. used ShReD partitioning metric [72] which favors conservation of cycles within modules. The network was partitioned and exchange reactions for all the metabolites except cofactors were removed from modules. EFM analysis was then performed on each module to identify within module substrate cycles. In this approach, cyclical EFMs based on multiple cofactors are identified compared to previous approaches [69, 39] where only ATP based substrate cycles were identified.

2.4 Alternative Network Decomposition Methods

While there are several fundamental uses for EFMs, their identification in genome-scale networks is computationally intractable [73]. Alternate computational techniques can be classified into two categories: sampling-based; or pathways within a sub-network or under specific constraints.

Barrett et al. [74] used Monte Carlo sampling and principal component analysis to obtain reactions that account for all range of flux states in the metabolic network. Their method comprises five steps:

1. Finding the active reactions for a given growth environment
2. Generating large number of random allowable flux states using Monte Carlo sampling
3. Computing bases using principal component analysis
4. Rotating bases to find biochemically meaningful interpretation of eigenvectors
5. Identifying the reaction sets based on rotated bases

The sampling algorithm generates physiologically unrealistic inefficient flux

distributions corresponding to high substrate uptake rates and very low growth rates. To overcome this problem, biasing in the sampling was introduced to generate flux distributions with growth rates of at least 90% of the maximum achievable growth rate. The analysis was applied to a reconstructed integrated transcriptional regulatory and metabolic network of *E. coli* [75], revealing that the top seven principle components are representative of the regulation of gene product activity by post-translational mechanisms.

As an alternative to finding EFMs in a large network, Kaleta et al. proposed a method to identify balanced pathways in sub-networks in the context of the entire network [76]. Kaleta et al. introduced the concept of flux patterns that define balanced pathways in a sub-network of a bigger network operating at steady-state. The flux values of the reactions in the sub-network are considered to be non-negative while considering the flux values of rest of the reactions in the network to be zero. Specifically, a flux pattern s in a sub-network of reactions $1 \leq i \leq k$ satisfies the following conditions:

$$v \geq 0 \tag{2.7}$$

$$\mathbf{S}\ \mathbf{v} = \mathbf{0} \tag{2.8}$$

$$\forall i \in \mathbf{s} : v_i > 0 \tag{2.9}$$

$$\forall j \in \{1 \ldots k\} \setminus \mathbf{S} : v_j = 0 \tag{2.10}$$

Elementary flux patterns, EFPs, are identified by iteratively solving a mixed-integer linear program (MILP). Constraints of the MILP are updated in each iteration such that no flux pattern is identified that is a combination of already found

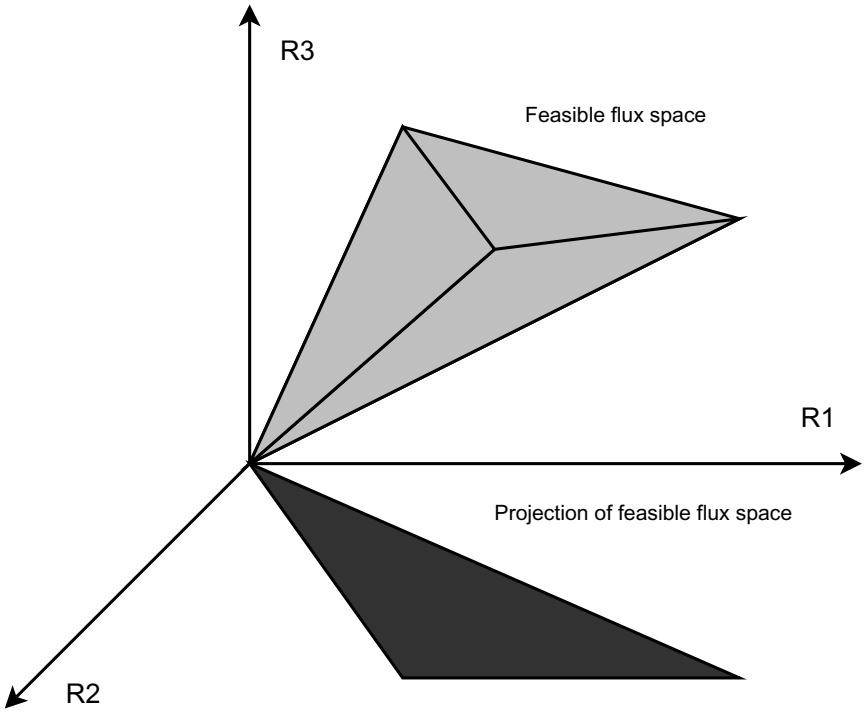


Figure 2.3: Projection of flux space onto a lower dimension subspace.

flux patterns. The computational complexity of the algorithm is polynomial in the size of the entire system and exponential in the size of the sub-network. Kaleta et al. showed that each elementary flux pattern can correspond to at least one EFM in the complete network. Elementary flux patterns can be used to determine the composition of minimal media required for the production of a compound of interest, to compute minimal cut sets [77], to determine the robustness of metabolic networks, and to analyze host-pythogen interactions. Although each EFP can be mapped to at least one EFM, EFPs does not necessarily cover all EFMs. Additionally, different EFMs can be represented using the same EFPs as the relative flux contributions along an EFP are not considered.

Marashi et al. addressed the shortcomings of EFPs by introducing the concept of Projected Cone Elementary Modes, ProCEMs, projections of EFMs onto a

lower dimensional subspace defined by the reactions of the sub-network. Figure 2.3 shows an example in which the flux cone of network comprising three reactions R1, R2 and R3 is projected onto the subspace defined by reactions R1 and R2. ProCEMs are computed by projecting the flux cone using the block elimination method [78] and then using the double-description method on the projected cone to compute extreme rays, or the ProCEMs.

Kaleta et al. proposed using genetic algorithms to explore the EFM search space [70]. During exploration, the genetic algorithm generated constraints of an optimization problem, which was used for the identification of an EFM satisfying the constraints. MILP can also be used as an alternate approach to find redundant alternate pathways in a metabolic network. Lee et al. formulated a MILP that can recursively identify alternate pathways for a given pathway [79].

Chapter 3

Enumerating Elementary Flux Modes

Elementary flux mode analysis is a powerful computational technique that has been used for modeling, analysis, and design optimization for designing many industrially relevant micro-organisms [80] and developing synthetic biology applications [81]. In this chapter, an EFM enumeration algorithm, termed graphical EFM or *gEFM*, is developed. The algorithm is based on graph traversal, an approach previously assumed unsuitable for implementing the double-description method. The approach is derived from a pathway synthesis method proposed by Mavrovouniotis et al. [52]. The algorithm is described and proved correct. *gEFM* is applied to several test cases with various sizes and runtimes are reported in comparison with other EFM computation tools. Unlike other EFM computational techniques, *gEFM* is robust to constraint ordering as it retains the structural information of the underlying network. *gEFM* is shown competitive with state-of-the-art EFM computational techniques for several test cases, but less so for networks with a larger number of

EFMs.

3.1 Methods

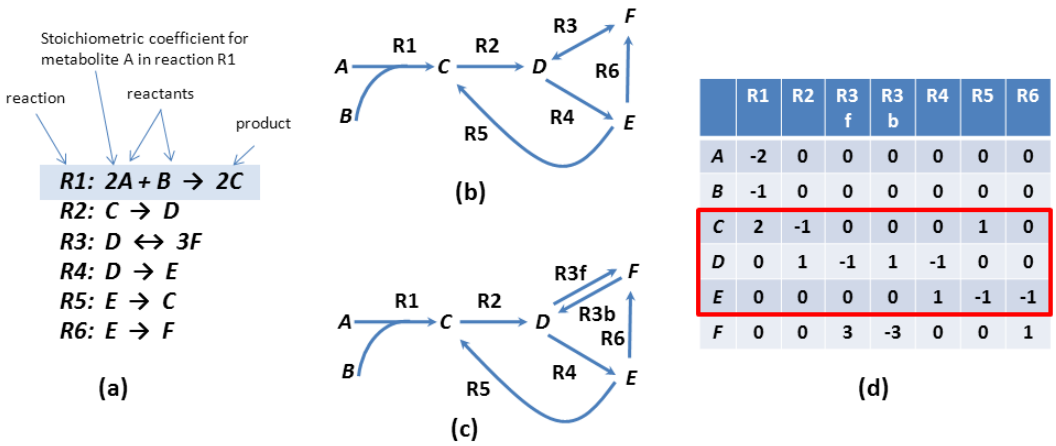
3.1.1 Definitions

Before presenting the details of the *gEFM* algorithm, we provide some definitions to clarify the exposition. Boldface capital letters denote matrices. Boldface lower case letters denote vectors. The i^{th} entry of a vector, $\mathbf{p}$, is referred to using the notation, $\mathbf{p}[i]$. Unless otherwise stated, vectors are column vectors of the appropriate dimensions.

Definition 1 *The structure of a biochemical network is represented by an $m \times n$ stoichiometric matrix $\mathbf{S}$, where m is the number of metabolites internal to the network and n is the number of reactions.*

The network reactions include exchange reactions that connect the network to a set of external metabolites not captured in the stoichiometric matrix. The i, j-th entry of the matrix $\mathbf{S}$, denoted by $\mathbf{S}_{ij}$, is negative (positive) if metabolite m_i is a reactant (product) participating in reaction r_j. A zero entry $\mathbf{S}_{ij}$ indicates that metabolite m_i does not participate in reaction r_j. Each reversible reaction is split into a forward and a reverse reaction, which we will refer to as a *reversible pair*.

Metabolites in the network are classified as either external or internal [82]. External metabolites, sometimes referred to as pool metabolites or sources/sinks, can accumulate. Internal metabolites however do not accumulate under steady-state conditions.



	R1	R2	R3 f	R3 b	R4	R5	R6
A	-2	0	0	0	0	0	0
B	-1	0	0	0	0	0	0
C	2	-1	0	0	0	1	0
D	0	1	-1	1	-1	0	0
E	0	0	0	0	1	-1	-1
F	0	0	3	-3	0	0	1

(d)

Figure 3.1: Example network. Metabolites A, B and F are considered as external metabolites. (a) Network reactions. (b) Original network. (c) Network after splitting reaction R3 into R3f and R3b. (d) Augmented Stoichiometric matrix, with the **S** matrix inside the delineated box.

Fig. 3.1(a) presents the set of reactions for an example metabolic network. The network is specified to have three external metabolites A, B, and F, while all others are specified as internals. The network can be represented as a hypergraph, as shown in Fig. 3.1(b). Fig. 3.1(c) shows the network after splitting R3 into a reversible pair, R3f and R3b. Fig. 3.1(d) illustrates an augmented matrix that includes both internal and external metabolites. The highlighted section shows the matrix **S** associated only with internal metabolites.

Definition 2 *A biochemical network is at steady state if the net production rate equals the net consumption rate for each metabolites internal to the network.*

The graph representation of the network provides topological insight into the underlying network. **S** represents the incidence matrix for a hypergraph G specified by a set of vertices and a set of edges. A vertex corresponds to a metabolite, and an edge corresponds to a reaction. The number of vertices equals $m + e$, where e represents number of external metabolites, and the number of edges equals n. The

terms network, graph, and hypergraph are used interchangeably, and so are the terms reaction, edge, and hyperedge.

An edge in the network may be a hyperedge (e.g. reaction R1 in Fig. 3.1), with potentially multiple sources and multiple sinks, allowing for multiple reactants and products. A path, or pathway, in the network is therefore loosely defined as a sequence of reactions such that products of a reaction are reactants of the next reaction(s) in the sequence.

A pathway can be represented in two possible ways. A vector of *reaction coefficients*, $\mathbf{p} \in \mathbb{R}^n$, represents the relative turnover rate of each reaction along the pathway. A pathway may also be represented using a pair of vectors: One vector of *binary values*, $\mathbf{b} \in \{0,1\}^n$, indicates reaction participation or lack thereof. A '1' value indicates that the reaction is active in the pathway, while a '0' value indicates that the reaction is inactive. A second vector of *metabolite coefficients*, $\mathbf{c} \in \mathbb{R}^m$, represents the net metabolite balance. A positive $\mathbf{c}[i]$ coefficient indicates a net production of metabolite i; a negative value indicates a net consumption. A zero value indicates a net balance in production and consumption, and the relevant metabolite is referred to as a balanced metabolite.

The values of $\mathbf{b}$ are derived from the reaction coefficients as follows:

$$\mathbf{b}[i] = \begin{cases} 1 & \text{if } \mathbf{p}[i] > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Metabolite and reaction coefficients are related as follows:

$$\mathbf{c} = \mathbf{S}\ \mathbf{p} \tag{3.2}$$

For example, a pathway involving R1, R2, and R3 in Fig. 3.1 operating at steady-state can be represented as:

$$\mathbf{p} = \begin{bmatrix} 1 & 2 & 2 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{b} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

In the *gEFM* algorithm, we utilize metabolite coefficients to identify pathways producing or consuming a particular metabolite. Reaction coefficients needed to specify the EFMs are computed from the binary coefficients as described in section 3.1.2.8.

Definition 3 *A balanced pathway* $\mathbf{p}$ *induces a steady-state condition on a network* $\mathbf{S}$ *iff*

$$\mathbf{S}\ \mathbf{p} = \mathbf{0} \tag{3.3}$$

Therefore, all internal metabolite coefficients along a balanced pathway must be zero.

Definition 4 *A pathway is non-decomposable or independent if it cannot be represented as a non-negative linear combination of other pathways.*

The independence of two pathways can be readily derived from their binary representation using *bitwise and* operation [49] [1].

Lemma 1 *Given two pathways* $\mathbf{p}$ *and* $\mathbf{p}'$*, with binary representations* $\mathbf{b}$ *and* $\mathbf{b}'$

respectively, $\mathbf{p}'$ *is dependent on* $\mathbf{p}$ *iff:*

$$\mathbf{b}\ AND\ \mathbf{b}' = \mathbf{b} \tag{3.4}$$

For example, the independence of pathway $\mathbf{p_a}$ consisting of R1, R2, and R4, and pathway $\mathbf{p_b}$ consisting of R1, R2, R4, and R5 in Figure 3.1 can be verified by comparing their binary representation using Lemma 1. Here, the active reactions in $\mathbf{p_a}$ are a subset of the active reactions in $\mathbf{p_b}$, making $\mathbf{p_b}$ dependent on $\mathbf{p_a}$, and $\mathbf{p_a}$ independent of $\mathbf{p_b}$.

An *elementary flux mode*, or *flux mode*, or *elementary mode* is a steady-state flux pattern in which flux proportions are fixed while their absolute magnitudes are indeterminate [59]. A formal definition of a flux mode is provided below.

Definition 5 *Given an* $m \times n$ *stoichiometric matrix,* $\mathbf{S}$*, three conditions must be met to label a pathway* $\mathbf{p}$ *as an* elementary flux mode*:*

C1: The network reactions proceed in a direction dictated by thermodynamic feasibility. Each reaction coefficient in $\mathbf{p}$ must be non-negative.

C2: The network is in quasi steady-state condition with no accumulation of internal metabolites in the network. Mathematically,

$$\mathbf{S\ p} = \mathbf{0}$$

C3: Each elementary mode must be independent from any other elementary mode in the network.

A vector $\mathbf{p}$ is thus an EFM if and only if $\mathbf{p}$ is thermodynamically feasible, satisfies quasi-steady state conditions, and there is no other non-null flux vector (up to a scaling) that satisfies both **C1** and **C2** and involves a proper subset of the reactions participating in $\mathbf{p}$.

The elementary flux modes for the example in Fig. 3.1 are illustrated in Fig. 3.2. The reaction coefficients for each are:

$$\mathbf{EFM}_1 = \begin{bmatrix} 1 & 2 & 2 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{EFM}_2 = \begin{bmatrix} 1 & 2 & 0 & 0 & 2 & 0 & 2 \end{bmatrix}^T$$
$$\mathbf{EFM}_3 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}^T$$
$$\mathbf{EFM}_4 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}^T$$

Each pathway is an EFM because all reaction directions are consistent with thermodynamic feasibility as specified in the original network. Additionally, each pathway is balanced, where each metabolite can be produced and consumed without net accumulation under the stoichiometric constraints specified by the original set of reactions in 3.1(a). Finally, each of the EFMs is independent of all others, as specified by the test in Lemma 1.

The benefit of the EFM decomposition is that any steady-state flux distribution in the network can be represented as a non-negative linear combination of EFMs. For example, a flux distribution of $\mathbf{p} = \begin{bmatrix} 5 & 15 & 4 & 0 & 11 & 5 & 6 \end{bmatrix}^T$ can be written as the linear combination of EFM1, EFM2, and EFM3, weighted by 2, 3, and 5, respectively.

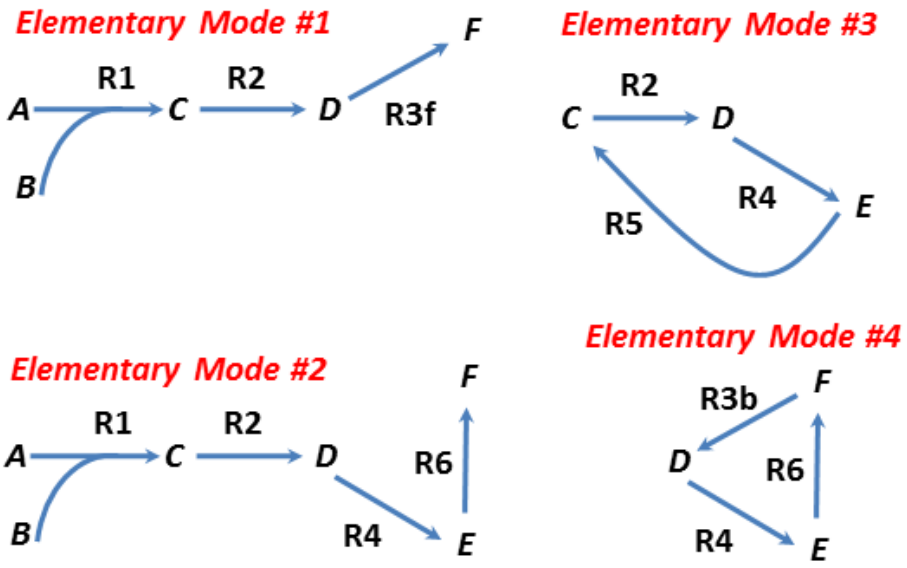


Figure 3.2: Elementary flux modes for the network in Fig. 3.1(b).

3.1.2 The *gEFM* Algorithm

The *gEFM* algorithm is an iterative algorithm that processes one internal metabolite at a time to construct partially balanced pathways. Each partially balanced pathway, referred to as a *partial pathway*, is balanced (metabolite coefficients of zero) with respect to processed metabolites, but not necessarily balanced with respect to unprocessed internal metabolites. Initially, *gEFM* treats each reaction in the network as a partial pathway. When a metabolite i is processed, new partial pathways are constructed by combining each partial pathway that produces i with each partial pathway that consumes i. Partial pathways containing a reversible pair (edges associated with a split reversible reaction) and dependent pathways are identified and discarded. The process repeats until all internal metabolites are processed. The remaining pathways are all EFMs. The pseudo code of the algorithm is presented in **Algorithm 1**, and implementation details are presented in Appendix A. We

Algorithm 1: *gEFM* Pseudocode

```
1  allPathways^(0) ← All reactions in the network
2  for k = 1 to m do
3      v^(k) ← Unbalanced internal metabolite at index k
4      inputs^(k) ← All pathways in allPathways^(k−1) for which v^(k) is a
       reactant
5      outputs^(k) ← All pathways in allPathways^(k−1) for which v^(k) is
       a product
6      allPathways^(k) ← allPathways^(k−1) \ (inputs^(k) ∪ outputs^(k))
7      candPathways^(k) ← inputs^(k) × outputs^(k)
8      Remove dependant pathways from candPathways^(k)
9      allPathways^(k) ← allPathways^(k) ∪ candPathways^(k)
10 Compute reaction coefficients for each pathway in allPathways^(m)
```

describe the key steps.

3.1.2.1 Initialization (line 1)

Initially, $\texttt{allPathways}^{(0)}$, the set representing viable partial pathways, is initialized with all network reactions. Each partial pathway is stored as $(\mathbf{b}_j,\ \mathbf{c}_j)$ where $\mathbf{b}_j$ represents bit vector for reactions, $\mathbf{c}_j$ represents metabolite coefficients, and $j = 1, 2, ..., |\texttt{allPathways}^{(0)}|$ is an index of a pathway in $\texttt{allPathways}^{(0)}$.

3.1.2.2 Metabolite Selection (line 3)

The metabolite $\texttt{v}^{(\texttt{k})}$ that generates the minimal number of input-output combinations is selected during each of the $m = 1 \ldots k$ iterations of the algorithm[1].

At the beginning of each iteration, the number of combinations is calculated as the product of the number of input and output partial pathways for each unprocessed metabolite. A partial pathway in $\texttt{allPathways}^{(\texttt{k}-1)}$ that produces metabo-

[1] k is the index of the internal metabolite selected at step k. Without loss of generality one can order the metabolites as they appear in the algorithm.

lite $\mathtt{v}^{(\mathtt{k})}$ is treated as an input pathway, and a partial pathway that consumes $\mathtt{v}^{(\mathtt{k})}$ is treated as an output pathway. For example, in the first iteration of the algorithm when used for the example network in Fig. 3.1(b), metabolite C has such a product (equals to 2) and it is processed first. The sign of the metabolite coefficient associated with $\mathtt{v}^{(\mathtt{k})}$ determines if a partial pathway produces or consumes metabolite $\mathtt{v}^{(\mathtt{k})}$. This metabolite selection scheme is the same that was suggested by the Mavrovouniotis pathway synthesis approach [52].

3.1.2.3 Identifying Input, Output, and Non-Participating Partial Pathways (lines 4-6)

All input partial pathways are stored in $\mathtt{inputs}^{(\mathtt{k})}$ and all output partial pathways are stored in $\mathtt{outputs}^{(\mathtt{k})}$; they are subsequently removed from $\mathtt{allPathways}^{(\mathtt{k}-1)}$. Thus, only partial pathways that do not participate in producing or consuming $\mathtt{v}^{(\mathtt{k})}$ are contained in $\mathtt{allPathways}^{(\mathtt{k})}$ (line 6).

3.1.2.4 Constructing Candidate Partial Pathways (line 7)

Consider an input partial pathway $\mathtt{i} \in \mathtt{inputs}^{(\mathtt{k})}$ and an output partial pathway $\mathtt{j} \in \mathtt{outputs}^{(\mathtt{k})}$ of metabolite $\mathtt{v}^{(\mathtt{k})}$. For each pair of partial pathways, $(\mathtt{i}, \mathtt{j})$, a new partial pathway is generated by computing *bitwise or* of the binary reaction coefficients of the input and output partial pathways:

$$\mathbf{b}_{\mathtt{i},\mathtt{j}} = \mathbf{b}_{\mathtt{i}} \ OR \ \mathbf{b}_{\mathtt{j}} \tag{3.5}$$

The metabolite coefficients for the newly generated pathways are updated as follows:

$$\mathtt{scale}_{\mathtt{i,j}} = -\mathbf{c}_{\mathtt{i}}[\mathtt{k}]/\mathbf{c}_{\mathtt{j}}[\mathtt{k}] \tag{3.6}$$

$$\mathbf{c}_{\mathtt{i,j}} = \mathbf{c}_{\mathtt{i}} + \mathtt{scale}_{\mathtt{i,j}} \cdot \mathbf{c}_{\mathtt{j}} \tag{3.7}$$

In a newly generated partial pathway, the coefficient for $\mathtt{v}^{(\mathtt{k})}$ is zero because the output pathway, $\mathbf{c}_{\mathtt{j}}$, is scaled such that the input pathway, $\mathbf{c}_{\mathtt{i}}$ completely consumes $\mathtt{v}^{(\mathtt{k})}$. That is, $\mathtt{v}^{(\mathtt{k})}$ is balanced along the newly generated pathway. By design, the scaling factor $\mathtt{scale}_{\mathtt{i,j}}$ is always positive and ensures that reactions in the resulting pathway operate in the forward direction. A newly generated partial pathway thus satisfies C1.

For the network in Fig. 3.1(b), the following is a list of all bit vectors and metabolite coefficients corresponding to the network reactions. Note that the pathway indices for $\mathbf{p}_4$, $\mathbf{p}_5$, ... $\mathbf{p}_7$ are off by 1 from reaction indices in the figure because reaction 3 is split into forward and reverse reactions. The ordering of the metabolites

in the **c** vectors is alphabetical.

$$\mathbf{b}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_1 = \begin{bmatrix} -2 & -1 & 2 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{b}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_2 = \begin{bmatrix} 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{b}_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_3 = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 3 \end{bmatrix}^T$$
$$\mathbf{b}_4 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_4 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & -3 \end{bmatrix}^T$$
$$\mathbf{b}_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_5 = \begin{bmatrix} 0 & 0 & 0 & -1 & 1 & 0 \end{bmatrix}^T$$
$$\mathbf{b}_6 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_6 = \begin{bmatrix} 0 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}^T$$
$$\mathbf{b}_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}^T$$

When metabolite C is selected in the first iteration of *gEFM*, metabolite C has two input partial pathways $\mathbf{p}_1$ and $\mathbf{p}_6$ and one output partial pathway $\mathbf{p}_2$. Partial pathways $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_5$, and $\mathbf{p}_7$ do not participate in producing or consuming metabolite C. The following two candidate partial pathways are generated by

processing metabolite C:

$$\mathbf{b}_{1,2} = (\mathbf{b}_1 \ OR \ \mathbf{b}_2) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

$$\texttt{scale}_{1,2} = 2$$

$$\mathbf{c}_{1,2} = \begin{bmatrix} -2 & -1 & 0 & 2 & 0 & 0 \end{bmatrix}^T$$

$$\mathbf{b}_{6,2} = (\mathbf{b}_6 \ OR \ \mathbf{b}_2) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T$$

$$\texttt{scale}_{6,2} = 1$$

$$\mathbf{c}_{6,2} = \begin{bmatrix} 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}^T$$

After processing metabolite C, the network is shown in Fig. 3.3. The follow-

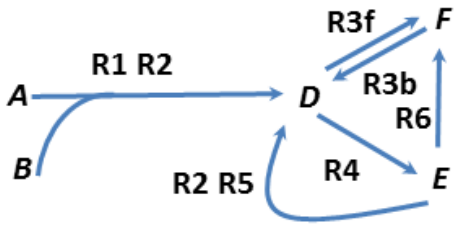


Figure 3.3: The network in Figure 3.1 after processing metabolite C.

ing is the list of partial pathways after processing metabolite C (line 7):

$$\mathbf{b}_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_3 = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 3 \end{bmatrix}^T$$
$$\mathbf{b}_4 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_4 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & -3 \end{bmatrix}^T$$
$$\mathbf{b}_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_5 = \begin{bmatrix} 0 & 0 & 0 & -1 & 1 & 0 \end{bmatrix}^T$$
$$\mathbf{b}_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}^T$$
$$\mathbf{b}_{1,2} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_{1,2} = \begin{bmatrix} -2 & -1 & 0 & 2 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{b}_{6,2} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T$$
$$\mathbf{c}_{6,2} = \begin{bmatrix} 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}^T$$

Partial pathways containing a reversible pair are excluded from the candidate pathways. We use a novel data structure, *reversible reaction tree* to accomplish this task. Details of reversible reaction trees are available in section 3.1.4.

During each iteration of *gEFM*, a reversible reaction tree is created for the set of input pathways and another for the set of output pathways (lines 4 and 5). The reversible trees are used when creating new candidate pathways (line 7). The two subtrees are recursively combined to generate new pathways. When an input subtree

is combined with an output subtree, their labels are combined using a bitwise or operation to generate a new label (*combo label*). If a reversible reaction pair is found to be active in *combo label*, the subtrees are not combined because every pathway generated will have both reactions active, and a reversible reaction can only operate in one direction for a particular steady state. Checking the subtree labels against each other thus avoids generating a large number of invalid pathways.

3.1.2.5 Dependency checking (line 8)

All dependent partial pathways in $\texttt{candPathways}^{(\texttt{k})}$ are identified and removed. A partial pathway $\mathbf{p}_c^{(k)} \in \texttt{candPathways}^{(\texttt{k})}$ is identified as dependent if it is dependent on any other pathway in $\texttt{candPathways}^{(\texttt{k})}$ or if it is dependent on a pathway in $\texttt{allPathways}^{(\texttt{k})}$. We use the bit pattern trees [73] data structure to implement pathway dependency checking.

After processing metabolite C in the network in Fig. 3.1(b), two new partial pathways $\mathbf{p}_{1,2}$ and $\mathbf{p}_{6,2}$ are generated. These partial pathways are not discarded because they are independent of each other and of non-participating pathways $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_5$, and $\mathbf{p}_7$.

3.1.2.6 Updating the List of Independent Partial Pathways (line 9)

After removing dependent pathways from $\texttt{candPathways}^{(\texttt{k})}$, the set is combined with all non-participating reactions stored in $\texttt{candPathways}^{(\texttt{k})}$ to generate a listing of all EFMs with respect to the network induced by the first k metabolites.

3.1.2.7 Algorithm Termination

The algorithm iterates until all metabolites are processed. At any stage k, the partial pathways in $\texttt{allPathways}^{(\texttt{k})}$ are balanced with respect to the first k metabolites. Since dependent pathways are removed, all pathways in $\texttt{allPathways}^{(\texttt{k})}$ are EFMs with respect to the first k metabolites. All of these facts along with combining all possible input/output partial pathways at each iteration shows that *gEFM* algorithm is correct and will generate all EFMs for the original network.

3.1.2.8 Computing reaction coefficients (line 12)

After finding all EFMs, their numerical reaction coefficients are calculated from the binary vectors [1]. Consider a pathway having bit vector $\mathbf{b}$ and corresponding reaction coefficients $\mathbf{p}$. From the binary representation, numerical reaction coefficients for the non-participating reactions are zero.

By solving the following homogenous linear equation, unknown values for $\mathbf{p}$ can be obtained:

$$\mathbf{S}\ \mathbf{p} = \mathbf{0} \tag{3.8}$$

The reaction coefficients can be normalized to the pathway's first participating reaction.

Consider an EFM of the network in Fig. 3.1(b) having the following bit vector:

$$\mathbf{b} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

The corresponding reaction coefficients for the bit vector would be:

$$\mathbf{p} = \begin{bmatrix} p_1 & p_2 & p_3 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

Since $\mathbf{p}$ can be normalized, assuming $p_1 = 1$

$$\mathbf{p} = \begin{bmatrix} 1 & p_2 & p_3 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

By solving Equation 3.8, coefficients p_2 and p_3 are found to be:

$$\mathbf{p} = \begin{bmatrix} 1 & 2 & 2 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

3.1.3 Correctness of the *gEFM* algorithm

New definitions are introduced, and several Lemmas are presented to prove the correctness of *gEFM* in constructing pathways that meet the three conditions in Definition 5.

Definition 6 *A stoichiometric matrix* $\mathbf{S}^{(k)}$ *is an* $k \times n$ *submatrix of* $\mathbf{S}$ *that includes the first* k *rows (metabolites) of* $\mathbf{S}$.

Metabolites that are not represented in $\mathbf{S}^{(k)}$ are considered external with respect to $\mathbf{S}^{(k)}$. The matrix $\mathbf{S}^{(\mathbf{0})}$ is a zero-row matrix representing the network without any internal metabolites.

The *gEFM* algorithm constructs partially balanced pathways with respect to $\mathbf{S}^{(k)}$. That is, each metabolite along a pathway $\mathbf{p}$ will have a zero metabolite coefficient. More formally,

Definition 7 *A partially balanced pathway (or partial pathway)* $\mathbf{p}^{(k)}$ *is balanced with respect to the first* k *metabolites in* $\mathbf{S}$.

Following Lemmas argue the correctness of *gEFM* with respect to the construction incremental in the number of metabolites.

Lemma 2 *Partial pathways generated in every iteration of* gEFM *satisfy conditions C1 in Definition 5.*

Proof Initially, prior to the first iteration of the algorithm, all reactions operate in their specified direction. By construction, a partial pathway producing a metabolite is combined with a partial pathway that consumes the metabolite. The direction of all the reactions along the new partial pathways are consistent with earlier construction steps. Therefore, all partial pathways constructed at each step of the algorithm, and overall, satisfy condition C1 in Definition 5.

Lemma 3 *Partial pathways generated in iteration* k *of* gEFM *satisfy condition C2 in Definition 5 for the network specified by* $\mathbf{S}^{(k)}$.

Proof For $\mathbf{S^{(0)}}$, all the network reactions represent partial pathways because no metabolite is considered internal and each reaction stoichiometry is independently balanced. A new partial pathway, $\mathbf{p}^{(k)}$, is constructed by *gEFM* by combining two balanced partial pathways in $\mathbf{S}^{(k-1)}$ (i.e. they are balanced with respect to the first $k-1$ metabolites) and balancing a new metabolite, $\mathtt{v^{(k)}}$. During step k of the algorithm, the scaling operations (Equations 3.6 and 3.7) do not affect the metabolite balance for the first $k-1$ metabolites as their coefficients are already

zero. The following thus holds:

$$\mathbf{S}^{(k)}\ \mathbf{p}^{(k)} = \mathbf{0}$$

Therefore, condition C2 of Definition 5 for $\mathbf{S}^{(k)}$ is satisfied.

Lemma 4 *The set of partial pathways generated after every iteration of* gEFM *contains only independent partial pathways that satisfy C3 in Definition 5 for the network specified by* $\mathbf{S}^{(k)}$.

Proof At each iteration, each generated pathway is tested for independence using the combinational test (Lemma 1). The resulting set of pathways therefore satisfy condition C3 of Definition 5 for $\mathbf{S}^{(k)}$.

Lemma 5 gEFM *produces all EFMs for the network defined by* $\mathbf{S}^{(k)}$.

Proof At each iteration of the *gEFM* algorithm, all possible input/output partial pathway combinations are explored, ensuring that all possible ways of balancing a metabolite $\mathtt{v}^{(\mathtt{k})}$ are considered. Combined with Lemmas 2, 3, and 4, all EFMs for $\mathbf{S}^{(k)}$ are generated.

Theorem 1 *The* gEFM *algorithm generates all EFMs.*

Proof In every iteration of *gEFM* an internal metabolite is balanced. When all the internal metabolites are balanced, *gEFM* terminates and the following holds:

$$\mathbf{S} \equiv \mathbf{S}^{(m)} \tag{3.9}$$

All partial pathways generated after the last iteration satisfy C1 and C2 based on Lemmas 2 and a 3, respectively. Since the set of generated pathways contain all the EFMs (lemma 5) and all the dependent pathways are removed (Lemma 4), C3 is satisfied. Therefore, the set $\texttt{allPathways}^{(\texttt{m})}$ only contains EFMs.

3.1.4 Implementation Details

Reversible Reaction Trees

Due to splitting reversible reactions into reversible pairs, the *gEFM* algorithm may identify a pathway that contains a forward reaction and its corresponding reverse reaction. Such pathways are rejected during dependency checking because they will be dependent on the EFM consisting of the cyclic pathway that contains the reversible pair. EFMs consisting of reversible reaction pairs must be removed from the final EFM listing. Dependency testing for pathways containing reversible pairs is costly in terms of computational time. Therefore, the algorithm is modified to reject such pathways before adding them to the candidate pathway set on line 7 using a *reversible reaction tree*.

A reversible reaction tree is a binary tree constructed by recursively splitting the set of pathways based on a reaction belonging to a reversible pair. All pathways for which the reaction is active are stored in one subtree and all pathways for which the reaction is inactive are stored in the other. The reaction that splits the pathways more evenly is selected as the splitting reaction. The process is applied recursively until each tree node cannot be split any further. For each subtree, a bit vector *label* is computed that keeps track of reversible reaction pairs. A '1' value in the label indicates if a reaction is active (e.g. part of a pathway) within the pathways of the

the subtree, and a '0' value indicates otherwise.

During each iteration of *gEFM*, a reversible reaction tree is created for the set of input pathways and another for the set of output pathways (lines 4 and 5). The reversible trees are used when creating new candidate pathways (line 7). The two subtrees are recursively combined to generate new pathways. When an input subtree is combined with an output subtree, their labels are combined using a bitwise or operation to generate a new label (*combo label*). If a reversible reaction pair is found to be active in *combo label*, the subtrees are not combined because every pathway generated will have both reactions active, and a reversible reaction can only operate in one direction for a particular steady state. Checking the subtree labels against each other thus avoids generating a large number of invalid pathways.

Dependency Checking

Dependency checking is a fundamental and computationally expensive operation when generating EFMs. To ensure independence, all partial pathways generated by the algorithm must be tested for dependency against each other and against non-participating pathways. However, the implementation can be made more efficient by discarding any partial pathway with length larger than $\mathbf{rank}\ \mathbf{S}+1$ [83]. Another method of speeding the implementation is to compare the newly generated partial pathway against the generating input and output partial pathways instead of other generated partial pathways. The details of the comparison follow Lemma 7.

Lemma 6 *Pathway dependency is transitive. If* $\mathbf{p}_1$ *is dependent on* $\mathbf{p}_2$ *and* $\mathbf{p}_2$ *is dependent on* $\mathbf{p}_3$*, then* $\mathbf{p}_1$ *is dependent on* $\mathbf{p}_3$*.*

Proof Consider three pathways $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$ with binary representations $\mathbf{b}_1$, $\mathbf{b}_2$

and $\mathbf{b}_3$. Let $\mathbf{p}_1$ be dependent on $\mathbf{p}_2$, and $\mathbf{p}_2$ is dependent on $\mathbf{p}_3$. Using Lemma 1,

$$\mathbf{b}_1 \; AND \; \mathbf{b}_2 \; = \; \mathbf{b}_2$$

$$\mathbf{b}_2 \; AND \; \mathbf{b}_3 \; = \; \mathbf{b}_3$$

Consider the dependency test of $\mathbf{p}_1$ and $\mathbf{p}_3$:

$$\begin{aligned} \mathbf{b}_1 \; AND \; \mathbf{b}_3 &= \mathbf{b}_1 \; AND \; (\mathbf{b}_2 \; AND \; \mathbf{b}_3) \\ &= (\mathbf{b}_1 \; AND \; \mathbf{b}_2) \; AND \; \mathbf{b}_3 \\ &= \mathbf{b}_2 \; AND \; \mathbf{b}_3 \\ &= \mathbf{b}_3 \end{aligned}$$

The above derivation shows that $\mathbf{p}_1$ is dependent on pathway $\mathbf{p}_3$.

Lemma 7 *If a pathway* $\mathbf{p}$ *is dependent on a pathway* $\mathbf{p}_{combo}$*, then pathway* $\mathbf{p}$ *is also dependent on the pathways* $\mathbf{p}_{in}$ *and* $\mathbf{p}_{out}$ *generating* $\mathbf{p}_{combo}$*.*

Proof A pathway $\mathbf{p}_{combo}$ generated by combining two pathways $\mathbf{p}_{in}$ and $\mathbf{p}_{out}$ is naturally dependent on $\mathbf{p}_{in}$ and $\mathbf{p}_{out}$. Since pathway dependency is transitive (lemma 6), a pathway $\mathbf{p}$ is dependent on a pathway $\mathbf{p}_{combo}$, the pathway $\mathbf{p}$ is also dependent on the pathway $\mathbf{p}_{in}$ and on pathway $\mathbf{p}_{out}$.

Consider the set of pathways $\texttt{candPathways}^{(\texttt{k})}$ is generated by combining the set of input pathways $\texttt{inputs}^{(\texttt{k})}$ and the set of output pathways $\texttt{outputs}^{(\texttt{k})}$ in an iteration of *gEFM*. The dependency test is performed for each pathway $\mathbf{p} \in \texttt{candPathways}^{(\texttt{k})}$, generated by $\mathbf{p}_{in} \in \texttt{inputs}^{(\texttt{k})}$ and $\mathbf{p}_{out} \in \texttt{outputs}^{(\texttt{k})}$, and all pathways in $\texttt{candPathways}^{(\texttt{k})} \backslash \texttt{P}$. Using lemma 7, the same dependency analysis results are obtained by comparing pathway $\mathbf{p}$ with each pathway $\mathbf{p}'_{in} \in \texttt{inputs}^{(\texttt{k})} \backslash \mathbf{p}_{\texttt{in}}$ and

$\mathbf{p}'_{out} \in \texttt{outputs}^{(\texttt{k})} \backslash \mathbf{p}_{\texttt{out}}$. Within *gEFM*, the bit pattern trees [73] data structure is used to implement pathway dependency checking.

3.2 Results

3.2.1 Test Cases

To assess the performance of *gEFM* and compare with other tools, several biochemical networks with a varying number of reactions and metabolites are selected. The first network represents adipocyte central carbon metabolism, and was used for flux profiling and modularity analysis [84]. The second network is a reduced model capturing central carbon metabolism of the Chinese Hamster Ovarian (CHO) cell [85]. The next three networks are variations of an *E. coli* cell model, which was utilized when engineering a minimal *E. coli* cell for the efficient production of ethanol from hexoses and pentoses [2]. In *E. coli*(irrev), all reactions in the network are made irreversible by forcing reversible reactions to operate only in the forward direction. In *E. coli*(gluc), glucose is considered as the only carbon source for the production of ethanol. In the next network, *E. coli*(xbio), *E. coli* is allowed to grow on all sugars without any biomass production. In *Helicobacter pylori* (*H. pylori*) [86], a human gastric pathogen, only one compartment (cytoplasm) is considered with cofactors removed. In yeast, *Saccharomyces cerevisiae* (*S. cerevisiae*) iND750 model [87], cofactors are removed and only one compartment (cytoplasm) is considered. The last model represents primary metabolism in *Chlamydomonas reinhardtii* (*C. reinhardtii*) [88], a single celled green alga. Reactions in mitochondria are considered with phosphate and water removed.

Table 3.1: Statistics for the test case networks.

Test Case	Uncompressed		Compressed		EFMs
	Reactions[a]	Metabolites[b]	Reactions[a]	Metabolites[b]	
Adipocyte	34 (0)	34 (26)	15 (0)	7 (7)	78
CHO	34 (10)	40 (26)	30 (9)	12 (12)	1,431
E. coli (irrev)	70 (0)	68 (52)	26 (0)	12 (12)	840
E. coli (xbio)	70 (19)	68 (52)	47 (8)	21 (21)	40,693
E. coli (gluc)	70 (19)	68 (52)	51 (12)	26 (26)	33,220
H. pylori	413 (0)	287 (287)	140 (0)	23 (23)	753,664
S. cerevisiae (iND750)	179 (19)	147 (147)	81 (19)	29 (29)	4,535,802
C. reinhardtii	121 (0)	39 (39)	110 (0)	28 (28)	4,152,658

[a]Number of reversible reactions are in parenthesis
[b]Number of internal metabolites are in parenthesis

Several compression methods provided by EFMTool [89] were utilized to minimize the size of the test cases. The dead-end metabolite removal method eliminates internal metabolites that are either only produced or only consumed. Reactions associated with such metabolites are also eliminated. The coupled-zero compression method removes all reactions that always carry zero flux at steady state. The unique-flows compression method removes metabolites that are produced by only one reaction and consumed by only one other reaction by combining the producing and consuming reactions. The coupled-combine compression method removes all flux-coupled reactions, ones for which their relative flux is always constant, except one reaction. The flux values for the removed reactions can be computed based on the retained reaction. Similarly, the coupled-contradicting compression method removes negatively coupled reactions.

All test case statistics (total number of reactions, number of reversible reactions, total number of metabolites, and number of internal metabolites) are listed in Table 3.1. The table also lists the statistics for the compressed models, where significant reductions in the number of reactions and metabolites are attained. The table lists the number of EFMs for each case.

3.2.2 Computing Platform

gEFM has been benchmarked against Metatool [50] and EFMTool [51]. The MATLAB implementation of MetaTool 5.1 is used with MATLAB 2013. The Java implementation of EFMTool is used with Java runtime environment 1.6. Because *gEFM* does not currently have a multi-threaded implementation and to provide a fair comparison, multi-threading was disabled for all tools. All experiments were performed on a 2.83 GHz Intel Xeon E5440 CPU with 6 MB cache running Red Hat Linux.

3.2.3 Runtime Analysis

The runtimes (in seconds) for the uncompressed and compressed models are reported in Table 3.2. The first column lists the model names. The next three columns list the runtime in seconds for all three tools for the uncompressed model, and the following three columns report the runtimes for the compressed model. When run times were less than 0.01 seconds, they were reported as < 0.01. Several entries are labeled as TO (timeout), where the computation did not complete within a given time frame. A different number of seconds was used for the timeouts depending on network size, consistently allowing for a timeout window of at least $2\times$ that of the fastest running tool.

Following observations are made regarding the results. Consistently, Metatool 5.1 computes the EFMs for the smaller examples for both the compressed and uncompressed models, but the larger examples do not complete as Metatool 5.1 crashes without reporting any errors. EFMTool and *gEFM* compute EFMs for all compressed and uncompressed test cases. The network size of *H. pylori* is larger than the last two test cases (*S. cerevisiae* and *C. reinhardtii*), but *H. pylori* has a

Table 3.2: Runtime comparison for Metatool, EFMTool, and gEFM for uncompressed and compressed models. Runtime is in seconds. TO means timeout, where the computation did not complete in a time frame of twice the maximum time for other tools.

	Uncompressed			Compressed		
	Metatool	EFMTool	*gEFM*	Metatool	EFMTool	*gEFM*
Adipocyte	0.11	0.07	< 0.01	0.09	0.05	< 0.01
CHO(small)	3.50	0.50	0.04	3.17	0.38	0.04
E. coli(irrev)	0.81	0.57	< 0.01	0.73	0.19	< 0.01
E. coli(gluc)	982.05	39.63	2.70	660.60	2.23	1.82
E. coli(xbio)	3,082.09	145.60	0.81	2,468.42	3.78	0.42
H. pylori(iIT341)	TO	5,138.54	1,717.17	TO	4,813.18	646.14
S. cerevisiae(iND750)	TO	27,029.99	534,375.00	TO	1,094.89	22,553.10
C. reinhardtii	TO	153,132.05	471,316.00	TO	104,169.10	150,017.00

significantly smaller number of EFMs.

3.2.4 Comparisons Performed

To better understand how EFMtool and *gEFM* differ in runtimes, the cumulative amount of work performed by each tool is quantified. In Table 3.3, the number of iterations, the cumulative number of combinations generated, and the cumulative number of comparisons performed by each tool for the compressed and uncompressed models are listed. The last three columns list the numbers relative to *gEFM*. For *gEFM*, the number of iterations corresponds to the number of internal metabolites, m, that must be balanced. For EFMTool, the number of iterations corresponds to the number of constraints on the steady-state operation derived after computing the null space kernel, and is equal to $n - k$ [1], where n is the number of reactions, and k is the size of the null space kernel matrix (which is bounded by m).

Combinations correspond to pathways generated by balancing an internal metabolite in *gEFM* whereas they correspond to rays generated after processing a constraint in EFMTool. In each iteration of *gEFM*, combinations are compared to the set of (input, output and non-participating) pathways; therefore, the number

Table 3.3: The number of iterations, cumulative number of generated combinations, cumulative number of comparisons for *gEFM*, EFMTool, for (a) uncompressed, and (b) uncompressed models.

(a) Uncompressed

	gEFM			EFMTool			EFMTool to *gEFM* ratios		
	Iterations	Combinations	Comparisons	Iterations	Combinations	Comparisons	Iterations	Combinations	Comparisons
Adipocyte	26	9.36×10^{02}	4.88×10^{04}	26	4.43×10^{02}	1.84×10^{04}	1.00	0.47	0.38
CHO(small)	26	3.09×10^{05}	4.14×10^{08}	26	8.78×10^{04}	6.21×10^{07}	1.00	0.28	0.15
E. coli(irrev)	52	2.31×10^{03}	4.10×10^{05}	47	2.29×10^{05}	1.93×10^{08}	0.90	99.17	470.57
E. coli(gluc)	52	2.58×10^{07}	3.07×10^{11}	42	5.40×10^{07}	1.27×10^{12}	0.81	2.10	4.14
E. coli(xbio)	52	7.75×10^{06}	6.89×10^{10}	47	2.11×10^{08}	7.42×10^{12}	0.90	27.27	107.66
H. pylori(iIT341)	287	4.76×10^{09}	1.83×10^{15}	281	4.29×10^{09}	1.15×10^{15}	0.98	0.90	0.63
S. cerevisiae(iND750)	147	4.98×10^{11}	9.08×10^{17}	143	2.13×10^{10}	4.37×10^{16}	0.97	0.04	0.05
C. reinhardtii	39	8.58×10^{11}	2.12×10^{18}	38	1.62×10^{11}	1.75×10^{17}	0.97	0.19	0.08

(b) Compressed

	gEFM			EFMTool			EFMTool to *gEFM* ratios		
	Iterations	Combinations	Comparisons	Iterations	Combinations	Comparisons	Iterations	Combinations	Comparisons
Adipocyte	7	3.62×10^{02}	1.55×10^{04}	7	1.20×10^{02}	4.60×10^{03}	1.00	0.33	0.30
CHO(small)	12	7.11×10^{04}	9.14×10^{07}	12	8.14×10^{03}	9.92×10^{06}	1.00	0.11	0.11
E. coli(irrev)	12	1.86×10^{03}	3.13×10^{05}	9	4.21×10^{03}	2.07×10^{06}	0.75	2.26	6.60
E. coli(gluc)	26	1.72×10^{07}	2.12×10^{11}	21	1.43×10^{06}	2.40×10^{10}	0.81	0.08	0.11
E. coli(xbio)	21	2.62×10^{06}	1.98×10^{10}	17	2.98×10^{06}	6.65×10^{10}	0.81	1.14	3.36
H. pylori(iIT341)	23	3.04×10^{09}	1.17×10^{15}	21	9.98×10^{08}	5.26×10^{14}	0.91	0.33	0.45
S. cerevisiae(iND750)	29	2.94×10^{10}	3.15×10^{16}	26	7.00×10^{08}	1.16×10^{15}	0.90	0.02	0.04
C. reinhardtii	28	7.76×10^{10}	1.88×10^{17}	27	8.84×10^{10}	3.09×10^{17}	0.96	1.14	1.64

of comparison is equal to product of the number of combinations and the number of pathways. Similarly in each iteration of EFMTool, the number of comparisons performed in each iteration is equal to product of the number of combinations and the number of rays (positive, negative, and zero rays). The number of iterations and the number of combinations generated in each iteration was recorded and the number of comparisons was computed for each test case.

For both compressed and uncompressed models, *gEFM* has a higher number of iterations than EFMtool, as typically metabolic networks are underdetermined (fewer constraints than metabolites). For the uncompressed models, *gEFM* provides significantly (two orders of magnitude) fewer combinations and comparisons than EFMTool for *E. coli*(irrev), *E. coli*(gluc) and *E. coli*(xbio), while EFMTool provides significantly fewer combinations and comparisons for *S. cerevisiae* and *C. reinhardtii*. The runtimes for *gEFM* are smaller than for EFMTool for examples with similar number of iterations and slightly smaller number of combinations and comparisons.

3.2.5 Impact of Compression

Comparing the number of comparisons and constraints for the uncompressed and compressed models, it is clear that EFMTool benefits extensively from compression, while *gEFM* does not. *gEFM* benefits from compression in two ways. First, some compression methods reduce the number of reactions in the network, which may reduce the size of the bit vectors used for storing the reactions and in turn reduce the runtime associated with processing the bit vectors. Each bit vector is a 32-bit integer array, large enough to represent each reaction with a bit. The reduction in the number of reactions will be beneficial only if the number of integers needed

to represent the bit vectors is reduced. Second, compression reduces the number of metabolites in the network, which reduces the number of iterations within the *gEFM* algorithm. All compression methods can reduce the number of reactions in the network whereas only some compression methods (i.e. dead-end metabolites, unique flows, and coupled-combine methods) can reduce the number of metabolites. Metabolites removed by compression have a very small number of reactions associated with them, and would have been balanced in the early iterations of *gEFM* when applied to the uncompressed network. The reduction in the number of comparisons for these metabolites is therefore small. For the *E. coli*(gluc) network, compression reduces the number of comparisons by 23% whereas for the *C. reinhardtii* network, the reduction in the number of comparisons 91%. EFMTool also benefits from compression techniques in two ways. First, the size of bit vectors used for storing the reactions is reduced because of reduced number of reactions. Second, the number of constraints is reduced because of the reduced number of reactions in the network, which produces substantial savings in runtime. EFMtool does not directly benefit from reducing the number of metabolites.

3.2.6 Impact of Metabolite/Constraint Ordering

The impact of metabolite/constraint ordering on runtime performance of *gEFM* and EFMTool was investigated by comparing the runtimes under heuristic ordering with the best of 10 random orderings. For *gEFM*, the metabolite to be balanced was chosen at random. For EFMTool, the rows of the null space kernel matrix were randomly ordered before applying the double-description method. For each test case, Table 3.4 lists the best and average runtimes for the 10 runs normalized to the

Table 3.4: The best of and average runtimes of 10 runs where metabolites (constraints) are randomly ordered. The runtimes are normalized to their respective runtimes using the default heuristics. The '-' indicates that the runtimes were all less than < 0.01 seconds.

	gEFM		EFMTool	
	Best	Average	Best	Average
Adipocyte	-	-	0.76	0.91
CHO (small)	1.50	1.53	0.78	0.91
E. coli(irrev)	-	-	0.82	1.00
E. coli(gluc)	4.14	57.84	0.34	0.77
E. coli(xbio)	1.66	43.04	0.70	1.65

runtimes reported in Table 3.2. For *gEFM*, the fastest runtime among the 10 runs is always larger than the original runtime of *gEFM*. On average, random metabolite ordering significantly ($> 10\times$) increases the runtime as showing for *E. coli*(gluc) and *E. coli*(xbio). For EFMTool, the best runtime among the 10 random runs always decreases the runtime by 18% to 66%. The average of the 10 random runs is at most $1.65\times$ the runtime of the original heuristic. Both tools are thus sensitive to metabolite/constraint ordering. However, the ordering heuristic for *gEFM* provides the smallest runtime, whereas the EFMTool ordering heuristic does not.

3.3 Conclusion

In this chapter, the algorithm originally proposed in 1990 by Mavrovouniotis et al [52] for pathway synthesis is adopted to compute EFMs. While earlier work [43] adapted this algorithm for EFM computation by identifying a test to remove dependent pathways, it was suggested that a matrix-based implementation is a more appropriate realization of the algorithm than graph traversal. In the present study, the proposed algorithm, *gEFM*, utilizes graph traversal in constructing the EFMs. For the first time, it is shown that graph traversal provides a viable approach for

computing EFMs. The *gEFM* implementation is shown to be competitive with state-of-the-art EFM computational techniques for several test cases, but less so for networks with a larger number of EFMs.

EFMs correspond to the extreme rays of a pointed polyhedral cone. *gEFM* is rooted in the double-description method, which establishes two equivalent characterization of a pointed convex cone: one based on the constraints that describe the hyperplanes forming the convex cone, and another based on the rays spanning the cone. *gEFM* implements the double-description method (see [46, 47, 48, 49] for a description of this method). In each iteration of *gEFM*, a constraint on balancing an internal metabolite is processed, and new extreme rays are identified. The *gEFM* algorithm combines the input and output pathways of the metabolite to generate intermediate pathways that lie on the hyperplane associated with the constraint. Removal of dependent pathways in *gEFM* (as well as in prior implementations) identifies the extreme rays. The large number of intermediate pathways and the resulting demanding dependency checking are currently the major bottlenecks in the *gEFM* (and other) implementation.

All methods based on the double-description have been reported sensitive to the ordering of the constraints, and that dynamic ordering methods are not necessarily superior [49]. Urbanczik and Wagner observe that it is difficult to remove such dependency [45]. The results demonstrate this sensitivity for both *gEFM* and EFMTool. The variations in the runtime are significantly more pronounced for *gEFM* than for EFMTool, showing that *gEFM* (and thus the canonical approach) is likely more sensitive to constraint ordering than EFMTool (Null Space approach). However, it is shown that *there is* a clear profitable ordering for *gEFM*. At each

step, the metabolite selection is based on a tangible choice: producing the smallest number of new partial pathways. Knowledge of the network structure allows for a more robust ordering heuristic. In contrast, network topology information is lost when computing the null space for techniques such as EFMTool and Metatool.

The runtime of *gEFM* directly correlates first and foremost with the overall number of generated rays (combinations) and comparisons needed for dependency checking. Specifically, the runtimes in Table 3.2 correlate with the the number of cumulative comparisons in Table 3.3. For the same number of comparisons, *gEFM* benefits when employing a smaller number of iterations. For example, the number of comparisons is comparable for the compressed and uncompressed models for *H. pylori* while the number of iterations is smaller for the compressed model, and the runtime for *H. pylori* is significantly reduced for the compressed model. The compression techniques utilized were specifically developed for EFMTool, and they enabled some runtime savings, with an average runtime savings of 55% for EFMTool and 17% for *gEFM* for the set of test cases.

Computing elementary modes is a fundamentally computationally intractable problem. This task is no harder than enumerating the vertices of a bounded polyhedron, whose complexity is a long-standing open problem [90]. It was proven that it is not possible to enumerate all elementary modes in polynomial time in the number of reactions unless P=NP [91]. Thus, as the network size scales in the number of reactions, runtime and memory usage simply increases in proportion greater than any polynomial function in the number of reactions. The runtime of any algorithm associated with computing the EFMs will therefore not be computationally efficient (i.e. polynomial) in the number of reactions or metabolites in the network. Analysis

of larger metabolic networks will thus require alternate pathway analysis methods, as demonstrated in the coming chapters.

Chapter 4

Finding Predictable Profitable Path

Identification of potential engineering targets in metabolic networks to improve yield of desired products is a computationally challenging problem. In this chapter, *PreProPath* (Predictably Profitable Path) is developed to identify target pathways best suited for engineering modifications. The algorithm utilizes uncertainties about the metabolic networks operating state inherent in the underdetermined linear equations representing the stoichiometric model. Flux Variability Analysis is used to determine the operational flux range. *PreProPath* identifies a path that is predictable in behavior, exhibiting small flux ranges, and profitable, containing the least restrictive flux-limiting reaction in the network. The algorithm is computationally efficient because it does not require enumeration of pathways. The results of case studies show that *PreProPath* can efficiently analyze variances in metabolic states and model uncertainties to suggest pathway engineering strategies that have been previously supported by experimental data.

4.1 Methods

4.1.1 Flux Variability Analysis

Given a metabolic network, a flux (network flow), v_i, is associated with each reaction i. It is possible to identify the maximum (or minimum) flux within a network by repeatedly applying Flux Balance Analysis (FBA) [18]. This procedure, known as Flux variability analysis (FVA) [19], identifies flux ranges by maximizing and minimizing each network flux subject to stoichiometric, physicochemical (e.g. thermodynamic irreversibility), cell growth, and measurement constraints [92, 93]. Mathematically, FVA can be expressed as:

$$Min/Max \quad v_i \tag{4.1}$$

$$Subject\ to$$

$$\mathbf{S\ v = 0} \tag{4.2}$$

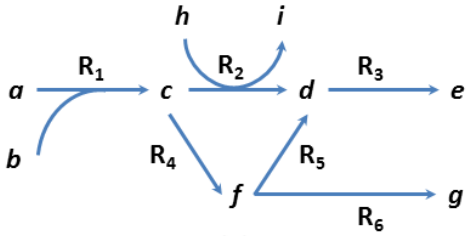
$$v_i^{lb} \leq v_i \leq v_i^{ub} \tag{4.3}$$

where $\mathbf{S\ v = 0}$ implies that the network is operating at quasi-steady state, with no net production or consumption of metabolites. For exchange reactions, flux bounds, v_i^{lb} and v_i^{ub}, represent the maximum and minimum nutrient uptake or secretion rates. Flux bounds for the rest of the reactions correspond to network constraints relevant to particular operating conditions such as reaction directions. The maximum (minimum) flux value identified by FVA for a reaction i is denoted by v_i^{max} (v_i^{min}).

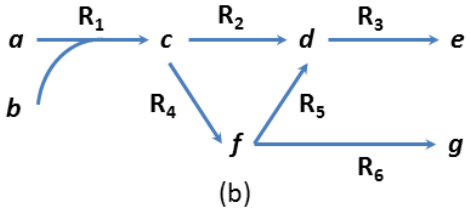
Figure 4.1(a) illustrates a small hypothetical network. Reaction R_2 has co-

factors h and i. Figure 4.1(b) shows the network after eliminating the cofactors. Cofactors and currency metabolites were not included in the graph after calculating flux ranges because paths consisting of these vertices do not reflect carbon flux [94, 95]. Figure 4.1(c) shows the stoichiometric matrix corresponding to the network in Figure 4.1(b). Metabolites a, b, e, and g are not included in the **S** matrix as they are external to the network; however, exchange reactions R_1, R_3, and R_6 are included in the matrix.

Applying FVA to the **S** matrix in Figure 4.1(c) with the following flux bounds, $v_1 = 10$ and $v_6 \geq 2$, the resulting flux ranges, (v^{min}, v^{max}), for reactions R_1 through R_6 are: (10, 10), (0, 8), (0, 8), (0, 8), (2, 10), (2, 10).



(a)



(b)

	R_1	R_2	R_3	R_4	R_5	R_6
c	1	-1	0	-1	0	0
d	0	1	-1	0	1	0
f	0	0	0	1	-1	-1

(c)

Figure 4.1: (a) Example network. (b) Example network without co-factors h and i. (c) S matrix for network in (b).

4.1.2 Edge Weighting During Graph-Based Analysis

Flux values can be utilized as edge weightings, w_e, during graph-based analysis. It is possible to identify a path between a source, s, and a destination, d, utilizing one of the following edge weightings:

- To identify a path capable of carrying maximal flux, edge weights are assigned maximum flux values (v_i^{max}). This approach is optimistic as it assumes that the path is capable of operating under the most favorable conditions, which may not be attainable in practice.

- To guarantee a minimal flux flowing through a path, edge weights are assigned minimum flux values (v_i^{min}). This approach is conservative in identifying flux capabilities. Operationally, an edge along the path may carry a flux higher than the minimal flux v_i^{min}.

- To identify the path with the least flux variability (i.e. operationally providing the most predictable fluxes under specified operating conditions), each edge weight is assigned the flux range, the difference between the maximum and minimum reaction fluxes obtained using FVA. An edge with a low weight here indicates a more predictable operating condition when compared to an edge with a higher weight. Identifying a path from s to d utilizing these edges weights results in the most predictable path as it operates in the tightest flux ranges.

4.1.3 Definitions

A path is an alternating sequence of vertices and edges, $v_0, e_0, v_1, e_1, v_2, \ldots, e_{n-1}, v_n$, beginning and ending with vertices. A path between a source and a destination vertex may contain hyperedges such that some vertices associated with the hyperedges may not be part of such path. When analyzing a graph, a particular weight of interest is the $bottleneckWeight_i$, which limits the maximum amount of flux flowing from s to d along any single path, p_i. Within a graph, and among all paths p_i between s and d, the maximum value among all $bottleneckWeight_i$ is referred to as the *bottleneckWeight* [55]. The edge associated with *bottleneckWeight* is a bottleneck edge. More formally,

$$bottleneckWeight = max\ _{\forall p_i}\ min\ _{e \in p_i}\ w_e$$

Any path between s and d capable of carrying a flux equal to or greater than the *bottleneckWeight* is referred to as a profitable path, as it contains the least restrictive flux-limiting reaction in the network and can be an engineering target that can yield profitable increase in yield.

When utilizing flux ranges as weights, one edge e_1 is less variable than an edge e_2 if $w(e_1)$ is less than $w(e_2)$. A path p_1 is less variable than a path p_2, if the maximum edge weight along p_1 is smaller than the maximum edge weight along p_2. If the maximum edge weights for p_1 and p_2 are equal, then successively smaller maximum weights along each path are compared instead. A path that is least variable is also the most predictable.

4.1.4 Predictable Profitable Path Conditions

For a given graph, G, a source vertex, s, destination vertex, d, and a flux range associated with each edge (v_i^{min} and v_i^{max}), the *PreProPath* algorithm finds the least variable path that contains the reactions capable of carrying the maximum flux from s to d. More specifically, *PreProPath* identifies a path p as a *predictably profitable path* if it meets the following two conditions:

Condition 1. Path p is profitable: all reactions along p are guaranteed to have a flux carrying capacity equal to or greater than the *bottleneckWeight*.

Condition 2. Path p is predictable: p is the least variable path among all profitable paths p_j.

To illustrate these conditions, consider paths $P_1 - P_4$ from s to d in a hypothetical network graph. Each path consists of three edges, with v_i^{max} edge weights representing the maximum possible fluxes obtained using FVA:

$$P_1 = (30, 50, 100)$$

$$P_2 = (30, 70, 120)$$

$$P_3 = (30, 100, 110)$$

$$P_4 = (20, 80, 130)$$

Paths $P_1 - P_3$ have the same largest (among all paths) smallest (within path) weight of 30, which is the *bottleneckWeight*. P_1, P_2, and P_3 are equivalent in terms of flux capacity limits as the largest flux through each of these paths will be at most 30. Paths P_1, P_2, and P_3 therefore satisfy Condition 1; however, P_4 does not.

Now consider weights assigned to P_1, P_2, and P_3 based on flux ranges (v_i^{max}

- v_i^{min}) found using FVA:

$$P_1 = (2, 3, 8)$$

$$P_2 = (3, 6, 8)$$

$$P_3 = (1, 7, 11)$$

Examining the largest range within each of the three pathways, both P_1 and P_2 have the smallest (among paths) maximum (within path) range of 8. Between P_1 and P_2, P_1 is less variable than P_2, as P_1 has the next smallest maximum range (value 3), thus satisfying Condition 2. Among all four paths, P_1 is profitable (capable of carrying flux above the *bottleneckWeight*) and the most predictable because it exhibits the least variability when compared to other profitable paths.

When analyzing a network graph without explicit path enumeration, identifying the least variable path, one with the smallest (among paths) maximum (within path) weight, is not straightforward. A naïve approach is to successively select the lowest edge weight until a path is found from source to destination. Consider for example network in Figure 4.2, with source s and destination d. Such a naïve approach will result in considering the edges in the following weight order: 3, 4, 5, 6, 7, 8, and then 10. At that point, there is a path from s to d, but it encompasses multiple paths, and in this case, the entire network is selected. The algorithm, *PreProPath*, selects the edges of the profitable path successively, first selecting the edge with the largest-weight edge necessary to complete the path from s to d, and then selecting the edge with the next largest weight, and so on.

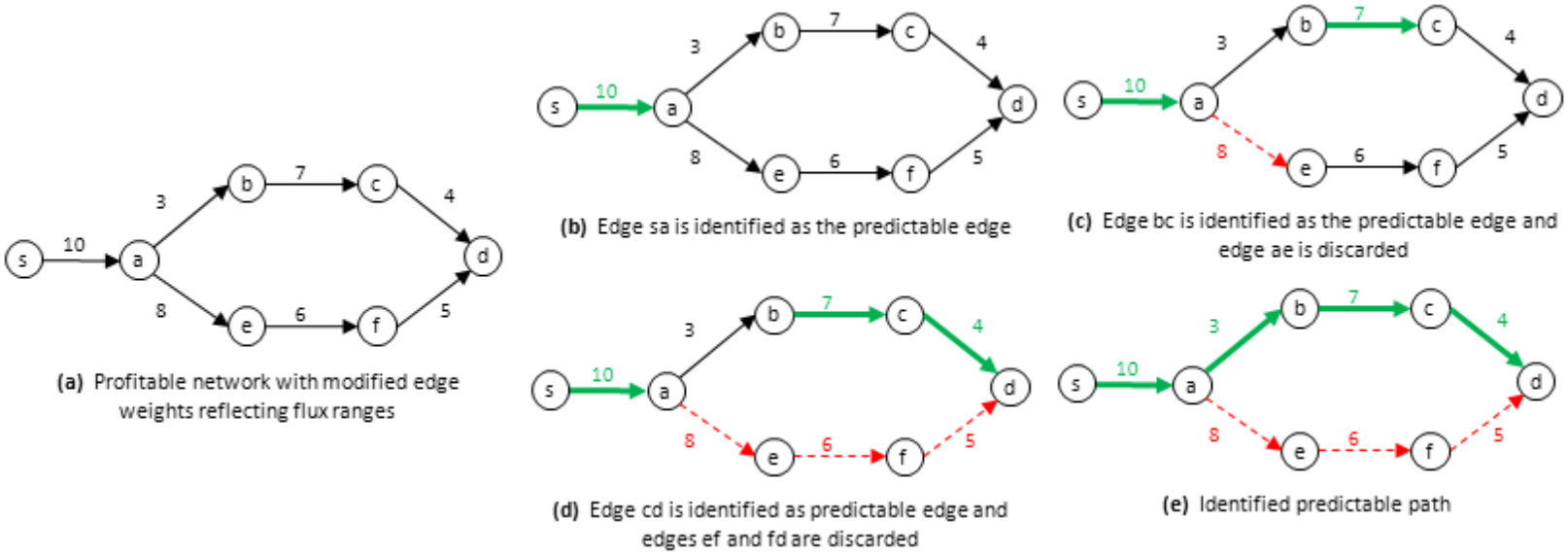


Figure 4.2: Example network to illustrate selection of predictable path.

4.1.5 *PreProPath* Algorithm

The *PreProPath* algorithm identifies a predictably profitable path from source to destination by executing two consecutive searches on the network graph. The first search identifies a profitable graph, a subset of the original graph G in which every reaction can operate at or above the flux limit, *bottleneckWeight*. The profitable graph can be found by removing from G all edges having weight less than *bottleneckWeight*. Every path from s to d in the profitable graph thus meets **Condition 1**.

In the second search, the objective is to identify the least variable path in the profitable graph. Edge weights in the profitable graph are set to flux ranges as calculated using FVA. Edges are selected successively (through multiple passes) to build the predictably profitable path. During each pass and in order of increasing edge weights, edges in profitable graph are selected for building a path from s to d. The passes stop when a path from s to d can be established using the selected edges. A post-processing step allows the identification of the path that meets both **Condition 1** and **Condition 2**.

The pseudo code for the *PreProPath* algorithm is presented in Fig. 4.3. On line 1, a weighted graph G is created from the **S** matrix. The edge weights are set to either v_{max} or v_{min} depending on the metabolic engineering application and the appropriateness of utilizing an optimistic or conservative approach. On line 2, the *bottleneckWeight* is found in G using the single source-single destination bottleneck algorithm (e.g. [55]).

The first search spans lines 3 through 7. Starting with an empty graph *profitableGraph*, edges with weight equal to or greater than *bottleneckWeight* are

***PreProPath* Algorithm(S, s, d, v_{min}, v_{max})**

1. Create a graph G from **S** using either v_{max} or v_{min} as edge weights
2. Identify *bottleneckWeight* for paths from s to d
3. Create an empty graph *profitableGraph*
4. for each edge e in G
5. if(weight(e) $\geq$ *bottleneckWeight*)
6. add edge e to *profitableGraph*
7. for each edge e, set edge weights in *profitableGraph* to be $(v_{max}[e] - v_{min}[e])$
8. Create an empty graph *predictableProfitableGraph*
9. while there does not exist a path from s to d in *predictableProfitableGraph*
10. *predictable* $\leftarrow$ maximum limiting edge in *profitableGraph* from s to d
11. remove *predictable* and all edges with weight greater than weight of *predictable* from *profitableGraph*
12. add *predictable* to *predictableProfitableGraph*
13. return path from s to d in *predictableProfitableGraph*

Figure 4.3: Pseudo code for *PreProPath* Algorithm.

added to *profitableGraph*. Each edge in *profitableGraph* is then assigned the flux range as a weight in preparation for the second search.

The second search spans lines 8 through 13. An empty graph *predictableProfitableGraph* is created. The maximum limiting edge, predictable, in *profitableGraph* is iteratively selected and removed from *profitableGraph*. This maximum limiting edge is found by first sorting all edge weights in *profitableGraph*, and then successively adding edges in increasing weight to an initially empty graph until a path from s to d is found. The last added edge is the maximum limiting edge. All edges with weights larger than the maximum limiting weight are removed from *profitableGraph*. The edge predictable is then added to *predictableProfitableGraph*. The process is repeated until a path from s to d is found in *predictableProfitableGraph*. The iterative process successively builds the predictable profitable path in *predictableProfitableGraph*, one edge at a time in order of decreasing variability. The returned path on line 13 is the *predictableProfitablePath* from s to d.

Fig. 4.2(b-e) illustrates the execution of the second search in the *PreProPath*

Algorithm (lines 8-13) on the graph in Fig. 4.2(a). First, the edge with weight 10 is identified as the one with the maximum limiting weight. It is removed from *profitableGraph* and added to the *predcitableProfitableGraph* (colored green), as shown in Fig. 4.2(b). Next, the edge with weight 7 is identified as the limiting edge. It is removed from *profitableGraph* and added to the *predcitableProfitableGraph* (colored green), as shown in Fig. 4.2(c). Additionally, the edge with weight 8 is removed from *profitableGraph* (colored dotted red). This process is repeated in Fig. 4.2(d) and Fig. 4.2(e). After the second search terminates, there is only one path (colored green) in Fig. 4.2(e) from s to d. This is the predictable profitable path.

4.2 Results

The *PreProPath* algorithm is applied to two test cases involving a small- and large-scale metabolic model with 48 and 2382 reactions, respectively.

4.2.1 Ethanol Production in *E. coli*

The first test case examined the production of ethanol from glucose in *E. coli*. As a first-generation biofuel, ethanol garnered significant attention from the metabolic engineering research community [96]. The network model used in the case study was constructed to represent *E. coli* growing on a minimal medium with glucose as the sole carbon and energy source [97]. The model comprised the following metabolic pathways: glycolysis, pentose phosphate pathway, TCA cycle, anapleurotic reactions, redox-associated reactions, oxidative phosphorylation/maintenance reactions, membrane transport reactions, and biomass synthesis. This model was used to calculate the metabolic flux distributions in *E. coli* at several steady states. For each

reaction in the model, a lower and upper bound was calculated by minimizing and maximizing the flux subject to stoichiometric and measurement constraints. Experimental data for glucose uptake and growth rates were taken from published studies involving chemostat cultures [98], and are summarized in Appendix B. The growth rates corresponded to the following approximate doubling times: 200, 100, 80, 60, 50, 40 and 30 min. Overall, the relative magnitudes of flux ranges with respect to glucose uptake were similar across different growth rates. The reactions with higher maximal flux values were found in glycolysis and pathways producing formate, ethanol and acetate as shown in Fig. 4.4. At all growth rates, the largest flux ranges were calculated for conversion of phosphoenolpyruvic acid (PEP) to pyruvate (PYR), followed by fructose-6-phosphate (F6P) to fructose-1,6-biphosphate (F16P), and malate (MAL) to oxaloacetate (OAA).

The flux data corresponding to different growth rates were at first separately analyzed. The maximum reaction flux (upper bound) is used as the edge weight to determine the flux-limiting reaction. Reactions with weights smaller than the flux-limiting step were pruned to isolate the profitable network. For every growth

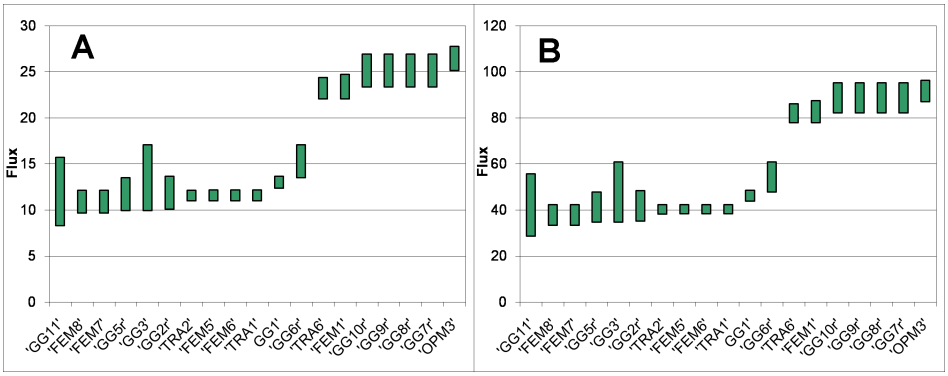


Figure 4.4: Flux distributions (mmol/gDCW/hr) in *E. coli* network for different growth rates. Flux distributions are plotted for growth rates of (A) 200 and (B) 50 minutes. Reactions are arranged in ascending order of flux lower bound.

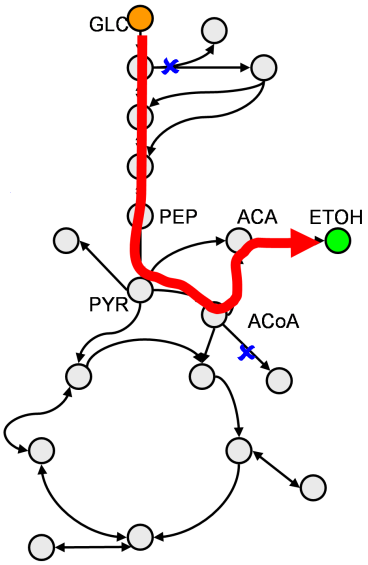


Figure 4.5: *E. coli* network for ethanol production. Upper bounds of flux range are used as weights for the identification of profitable network. Red lines highlight pathways in the profitable network. The competing pathways deleted by Trinh et al. [2] are marked with X.

rate, the profitable network comprised glycolysis reactions (Fig. 4.5). As there were no competing pathways in the profitable network, the subsequent search for the predictable pathway led to the trivial result.

A pessimistic approach of using the minimal reaction flux (lower bound) as the edge weight is also investigated to identify the profitable network. For every growth rate, the resulting profitable network comprised glycolysis and the pentose phosphate pathways (Fig. 4.6). The subsequent search for the predictable pathway based on flux ranges eliminated the pentose phosphate pathway, which contained reactions with larger flux ranges compared to glycolysis.

To determine if there was a predictably profitable consensus pathway across different growth rates, the above analysis was repeated using pooled data. For each reaction, the lower and upper flux bounds were set to the minimum of the lower bounds and maximum of the upper flux bounds respectively, irrespective of the

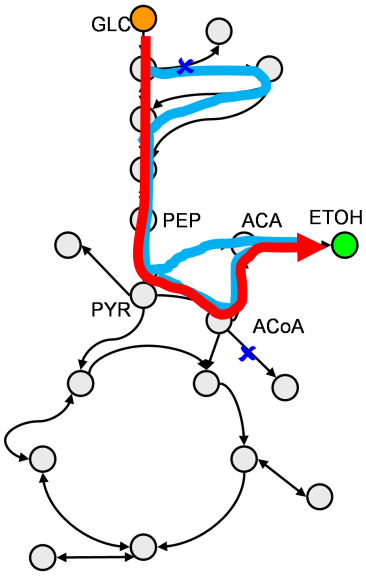


Figure 4.6: *E. coli* network for ethanol production. Lower bounds of flux range are used as weights for the identification of profitable network. Red and blue lines highlight competing pathways in the profitable network for the production of ethanol from glucose. The red pathway is more predictable when compared to blue. The competing pathways deleted by Trinh et al. [2] are marked with X.

growth rate. As was the case for each of the different growth rates, glycolysis was more predictably profitable compared to the pentose phosphate pathway.

4.2.2 Succinate Production in *E. coli*

In the second test case, succinate production from glucose is analyzed using a genome-scale model of *E. coli* metabolism (iAF1260) [99]. Succinate is a commercially valuable chemical used as a precursor for numerous industrial products, including pharmaceuticals and biodegradable polymers [100].

The upper and lower bounds of the reaction fluxes in the model were calculated by constraining a subset of internal and external fluxes using previously reported measurements for the MG1655 strain of *E. coli* assuming an error range of ±5% on the measured fluxes [101]. The upper bounds of ethanol transport reac-

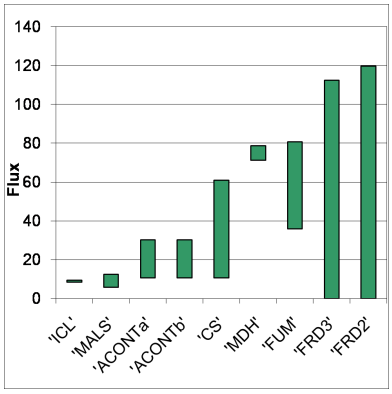


Figure 4.7: Flux distributions (mmol/gDCW/hr) of reactions in TCA cycle of *E. coli* network. Reactions are arranged in ascending order of flux upper bound.

tions were reduced, similar to a previous study [102]. Based on these flux ranges, the profitable network comprised the reactions of glycolysis and the TCA cycle. In this network, the flux ranges of reactions in the reductive arm of the TCA cycle, involving the conversion of oxaloacetate (OAA) to malate, fumarate, and eventually to succinate, were smaller than the flux ranges of the remaining reactions in the TCA cycle (Fig. 4.7). Consequently, the most predictably profitable synthesis route consisted of 14 reactions spanning the reactions of glycolysis and the reductive arm of the TCA cycle (Fig. 4.8).

4.2.3 Increasing the Flux through the Profitable Pathway

To evaluate the predictably profitable pathway identified by the *PreProPath* algorithm as a target for succinate overproduction, the impact of over-expressing one or more enzymes in the pathway is investigated. Similar to a previous study [103], the smallest level of guaranteed succinate production was calculated by solving a linear program whose objective is to minimize the succinate flux. Over-expression of an enzyme was modeled by raising the lower bound of the corresponding reaction flux. Flux ranges were computed using FVA. Glucose uptake was set to a nominal value

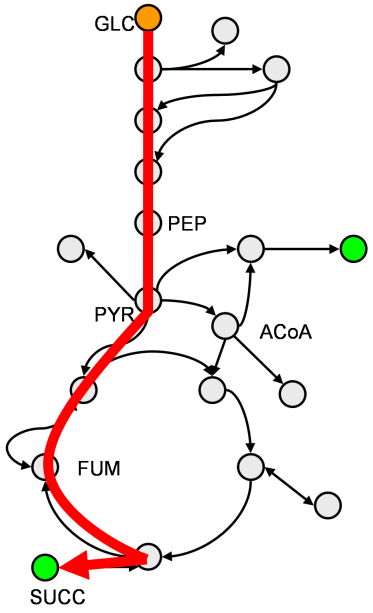


Figure 4.8: *E. coli* network for succinate production. Red lines highlight pathways for the production of succinate from glucose.

of 100 mmol/gDCW/hr and allowed to vary ±5%. All flux ranges computed by FVA are provided in Appendix C. Two sets of experiments were performed where the lower bound of the biomass flux was set to 1% and to 5% of the wild-type (iAF1260) value (3 mmol/gDCW/hr).

First the impact of over-expressing a single enzyme is investigated, i.e. increasing the lower bound of a reaction flux. An increase in the minimal succinate flux was found for three enzymes in the profitable pathway; these were enolase (ENO), fumarate reductase (FRD3), and phosphoenol pyruvate carboxylase (PPC) (Table 4.1). The magnitude of the minimal succinate flux depended on the enzyme and varied with the amount of increase in the lower bound (Fig. 4.9 and 4.10). However, the lower bound could not be increased without limit. For all three enzymes, a threshold was found beyond which the linear program became infeasible. The widest range of feasible solutions was found for FRD3 (Table 4.1). Over-expressing

Table 4.1: Flux Ranges of Enzymes Increases Minimum Succinate Yield. Units are mmol/gDCW/hr.

Enzyme	Wild-type flux range	At least 1% biomass production				At least 5% biomass production			
		Max succinate yield	Flux range for at least x% of max. theoretical yield of succinate			Max succinate yield	Flux range for at least x% of max. theoretical yield of succinate		
			x = 33	x = 50	x = 75		x = 33	x = 50	x = 75
ENO	172−205.7	89	251 − 260	254 − 260	258 − 260	85	229 − 258	250 − 258	252 − 258
FRD3	0 − 112.3	99	94 − 210	124 − 210	168 − 210	96	96 − 209	126 − 209	117 − 209
PPC	81.5 − 88.2	48	335 − 356	–	–	44	327 − 344	–	–

FRD3 also afforded the highest minimal succinate flux. Near the upper limit of the lower bound for FRD3, the minimal succinate flux reached 99% and 96% of the theoretical maximum for 1% and 5% of the wild-type biomass flux, respectively. Over-expressing PPC resulted in the lowest minimal succinate flux. However, even intervention yielded significant increases in succinate flux, above 48% and 44% of the theoretical maximum, when the lower bounds for the biomass flux were set to 1% and 5% of the wild-type flux, respectively.

Next it is investigated whether over-expressing an enzyme, one at a time, outside of the profitable pathway could also lead to an increase in the minimal succinate flux. The only enzyme over-expressions able to produce succinate at a level similar to that obtained by over-expressing enzymes in the predictably profitable pathway were oxidative phosphorylation reactions acting as cellular transport reactions.

Finally, the impact of over-expressing pairs of enzymes is characterized (Fig. 4.11 and 4.12). The calculations were performed exhaustively, since the predictably profitable pathway consisted of only 10 reactions, excluding exchange reactions. Approximately half of the 45 unique combinations (55%) resulted in a non-zero minimal flux of succinate. The best combinations (supporting a minimal succinate flux exceeding 75% of the theoretical maximum) involved at least one of the three enzymes identified from the single over-expression analysis (ENO, FRD3, and PPC). In addition, fumarase (FUM) was also identified as an attractive engineering target, specifically in combination with FRD3 or PPC for 1% biomass production. In these cases, the main contribution of FUM was to enlarge the effective over-expression range of the other enzyme. For example, when the lower bound of FUM flux is placed within the range of the wild type, only a narrow range is available

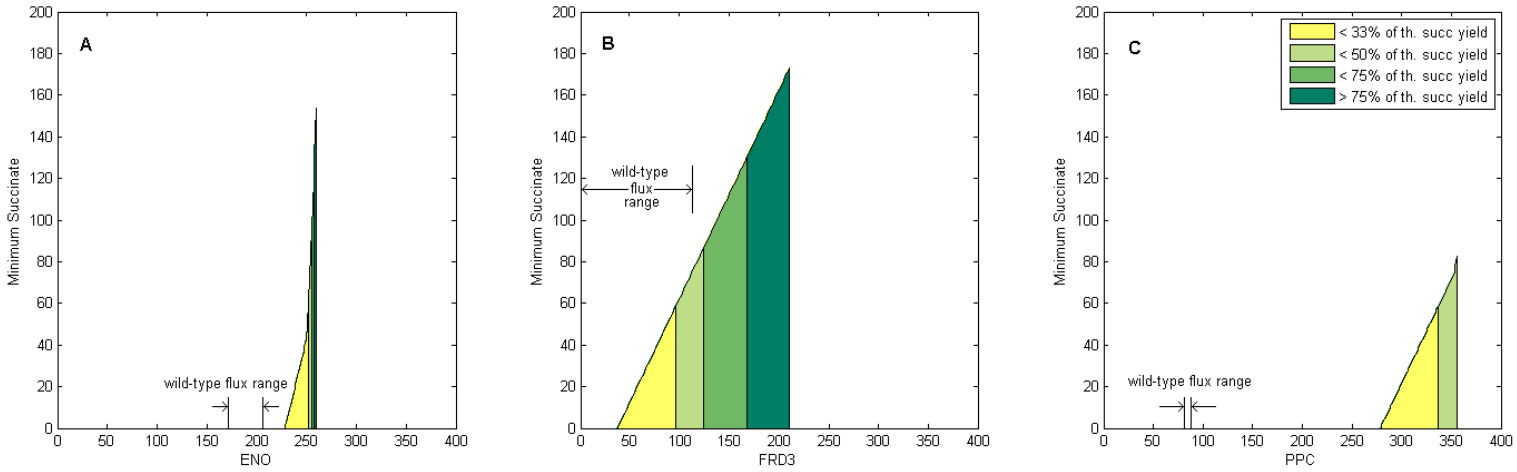


Figure 4.9: Minimum guaranteed succinate flux for single intervention for at least 1% biomass production. Succinate flux (mmol/gDCW/hr) is plotted against lower bound of reaction flux (mmol/gDCW/hr).

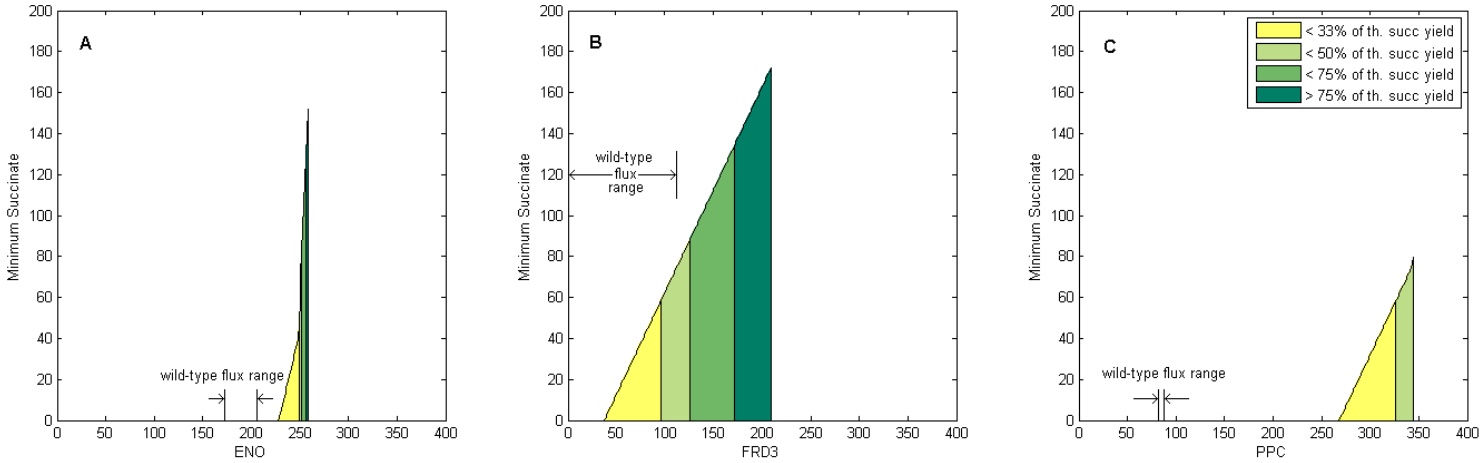


Figure 4.10: Minimum guaranteed succinate flux for single intervention for at least 5% biomass production. Succinate flux (mmol/gDCW/hr) is plotted against lower bound of reaction flux (mmol/gDCW/hr).

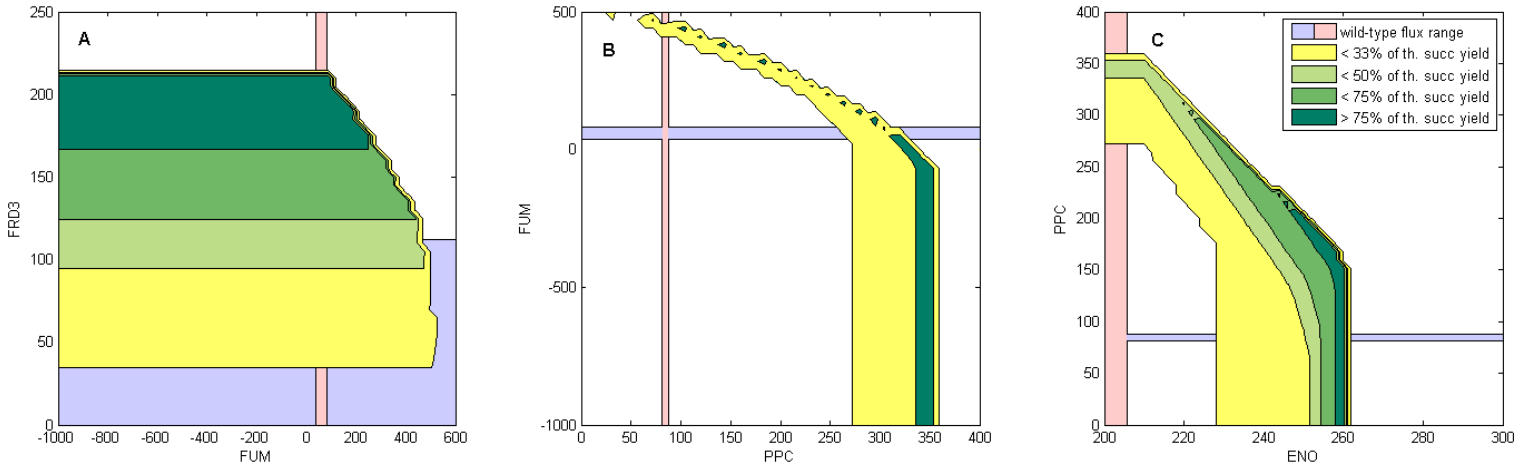


Figure 4.11: Contour plot of minimum guaranteed succinate yield for at least 1% biomass production. Minimum succinate yield is plotted for lower bounds of reaction flux (mmol/gDCW/hr) of the two intervened enzymes. The operating range of reaction fluxes in wild-type characterized strain is also shown.

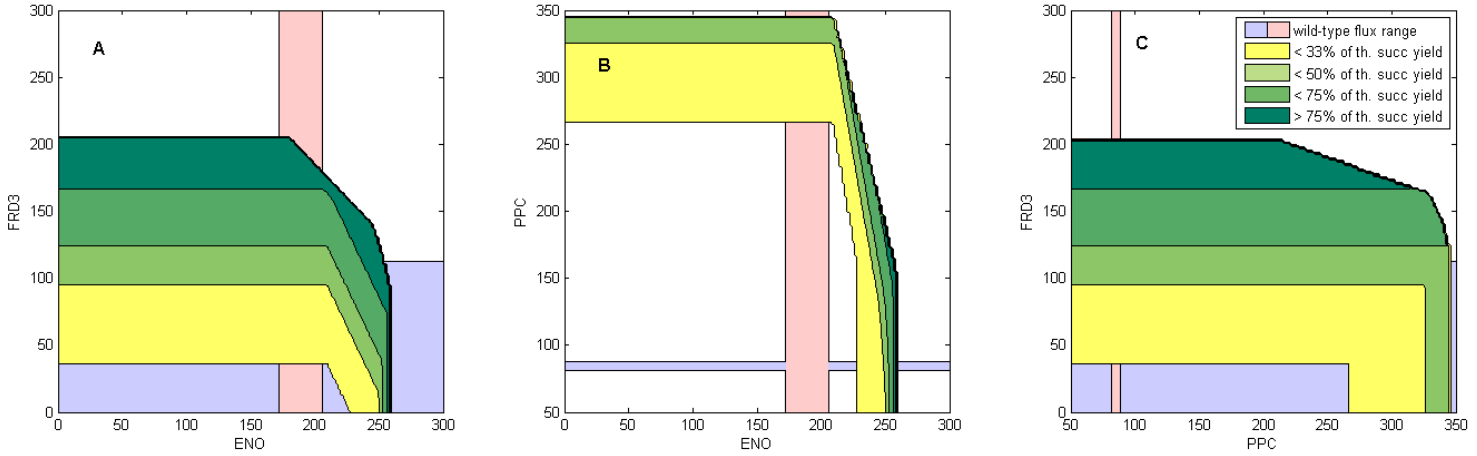


Figure 4.12: Contour plot of minimum guaranteed succinate yield for at least 5% biomass production. Minimum succinate yield is plotted for lower bounds of reaction flux (mmol/gDCW/hr) of the two intervened enzymes. The operating range of reaction fluxes in wild-type characterized strain is also shown.

for PPC over-expression to achieve a higher minimal succinate flux (exceeding 75% of the theoretical maximum). In contrast, reducing FUMs lower bound to below 20 mmol/gDCW/hr widens the PPC over-expression range four-fold (Fig. 4.11b). Similarly, the lower bound of PPC flux when reduced below 150 mmol/gDCW/hr, widens the ENO over-expression range five-fold (Fig. 4.11c). In all cases supporting a minimal succinate flux greater than 75% of the theoretical maximum, the increase in the minimal succinate flux positively correlated with an increase in the lower bound of ENO, FRD3 or PPC. In this regard, the double over-expressions did not identify any new enzymes for flux increase. Therefore, additional combinations involving triple or quadruple over-expressions were not further investigated.

The engineering targets identified by the *PreProPath* algorithm varied with different biomass production rates due to the different requirements for biomass precursors. For example, FUM was not identified as a potential enzyme for intervention when the lower bound for biomass flux was set to 5% of the wild-type flux. The over-expression ranges also depended on the growth rates. For higher growth rate, the over-expression level of the enzymes was lower compared to lower growth rate, reflecting a lower yield of succinate for a faster growing cell. For example, the flux range of PPC was found to be 335-356 mmol/gDCW/hr for 1% biomass production compared to 327-344 mmol/gDCW/hr for 5% biomass production.

4.3 Conclusion

In this chapter, an efficient computational method is developed to identify engineering targets for increased production of compounds in metabolic networks. The method is "uncertainty-aware" as it considers degrees of freedom in the model and

multiple metabolic states arising because of different uptake rates. The algorithm is based on guided search and avoids exhaustive exploration of all pathways in the network. The effectiveness of the method was demonstrated by applying it to two test cases. In the first test case, the algorithm identified pathways for the maximum production of target compounds across different steady-state flux distributions reflecting different growth rates. In the second test case, the over-expression of enzymes along the succinate-producing pathway in *E. coli* is characterized, which was identified by the algorithm as the predictable profitable path. An important feature of the *PreProPath* algorithm is that it can take into account different flux states, as determined by measured rates of metabolite exchange with the medium, when searching for pathways with particular attributes. In the first test case, *PreProPath* identified the same pathway, glycolysis, across different growth rates, underscoring the singular importance of this pathway in ethanol production.

PreProPath is effective in analyzing biochemical pathways without direct enumeration of all possible pathways. The algorithm is an alternative to explicitly enumerating all elementary pathways followed by search for a pathway with a specific property (predictability and profitability, in this case). To increase ethanol production in *E. coli* as in the first test case, Trinh et al. identified and then analyzed over 15,000 EFMs to determine gene knockout targets [2]. To narrow down the candidate pathways, EFMs that do not contribute to ethanol production were eliminated. The remaining ethanol-producing EFMs were then grouped into six "families" based on the type of sugar substrate. Using eight gene knockouts, pathways competing with ethanol producing pathways were removed. The identified predictably profitable pathway is the same one identified by Trinh et al.

In the case studies, bottleneck reactions that refer to flux-limiting reactions are considered. A reaction can be "flux-limiting" due to various reasons such as reaction kinetics or regulatory effects. The method simply identifies such a reaction within the context of a specific network at a particular flux distribution. A distinction between flux-limiting and rate-limiting is drawn. A flux-limiting reaction does not necessarily correspond to a "rate-limiting" reaction, which was believed to be the slowest step in a series of reactions, and was often associated with the first committed step of a pathway. Metabolic Control Analysis [103], applicable only in the context of small network perturbations, shows that such flux-limiting reactions exist if the first reaction step is completely insensitive to its product, which is not typically the case.

The results of the case studies suggest that the *PreProPath* algorithm can efficiently guide the search for pathway engineering targets. While the results were promising, they also pointed to limitations of the present analysis. First, the analysis does not distinguish between degrees of freedom in a model arising from insufficient equality constraints and variances associated with measurements. These two different sources of uncertainty can both lead to flux variability, which forms the basis of the *PreProPath* algorithm. However, the relative magnitudes of these uncertainties directly influence the results. For example, the second case study showed that it is possible to obtain a different predictably profitable pathway depending on the metabolic state. Clearly, metabolic states can only be distinguished meaningfully, if the uncertainties in the measurements are sufficiently small. One way to discriminate between the variances arising from the two different sources of uncertainty is through sensitivity analysis, for example based on Monte Carlo simulations, which

systematically assess the impact of measurement errors. Second, the analysis assumed that a reaction with a small value for its flux range is more profitable to genetically boost than another with a higher range value. The algorithm presented here can be adapted to utilize edge weights that correspond to a different desired objective.

Chapter 5

Finding Bottleneck Reactions

Increasing the yield of a desired product from a given source metabolite in a microorganism can improve the efficiency of production. One approach to solve this problem is to alleviate bottleneck in the pathways from the source metabolite to the desired product. Due to complexity of the metabolic networks, identification of all the pathways from the given source to the desired product is a computationally challenging task. In this chapter, the problem of identifying a bottleneck reaction in metabolic networks is addressed. The algorithm developed in the chapter is based on Dijkstra's shortest-path algorithm that explores the network without enumerating all pathways. The algorithm identifies the best bottleneck reaction, Dominant-Edge, from the source metabolite to the desired product and identifies a path, Dominant Path, containing the Dominant-Edge. Later, the dominant path is augmented to identify a pathway from the source metabolite to the desired product. The algorithm has polynomial runtime in terms of number of metabolites and reactions in the network. The chapter focuses on application of the algorithm to identify thermodynamic bottlenecks in metabolic networks.

5.1 Methods

5.1.1 Free Energy

Gibbs free energy is most useful for chemical processes at a given temperature and pressure (isothermal and isobaric) and often used in biology [104]. In this thesis, the standard Gibbs free energy change is used to estimate the expected likelihood of the corresponding reaction. The Gibbs free energy change (ΔG) of a reaction is a thermodynamic quantity whose sign in principle indicates whether a reaction is likely to occur (negative) or not occur (positive) spontaneously. Very recently, a related thermodynamic quantity, entropy (ΔS), of an EFM has been shown to significantly correlate with its flux [105]. Here, group contribution theory [106] is used to estimate the standard ΔG (ΔG$^\circ$) values of metabolic reactions, which in turn serves as a first-order approximation of the "true" ΔG values under a well-defined and idealized condition (1 M concentrations of all reactants and products, 25 °C and neutral pH). The algorithm uses ΔG to weight the edges to conceptually simplify the formulation.

5.1.2 Representation of Metabolic Networks

A metabolic network is represented as a graph $G_m = (V, E)$, where V and E are sets of metabolites and reactions, respectively. When an edge represents part of a reaction the edge is referred as a sibling edge. The reactants or products of a single reaction are referred to as sibling vertices. A value g_e is associated with each edge, representing the ΔG of the corresponding reaction. A path $s \to t$ is defined as a sequence of vertices and edges starting from vertex s and ending at vertex t.

A transpose of the network graph is obtained by reversing the direction of

every edge, maintaining sibling relationships. The union operator $\cup$ is used between a subgraph N and a set of edges or nodes to create an equal or larger subgraph in terms of number of nodes and/or edges.

5.1.3 Dominant Paths

The algorithmic objective is to identify a network path that is energetically favored or "dominates" in the production of a particular metabolite. The limiting step (bottleneck) in production along a path is the reaction (edge) that has the smallest g_e (i.e. least negative ΔG). Among several parallel paths, a dominant-edge path will have the largest limiting step. The bottleneck shortest path is a well-known problem [107]. Therefore, terms similar to those in the Bottleneck Shortest Path Problem are used, whose goal is to determine the limiting capacity of any path between two specified vertices in a given network [108].

The *bottleneck energy* b_p of a path p from s to t is defined as:

$$b_p = max\ _{e \in p}\ g_e \tag{5.1}$$

The edge along p responsible for setting the bottleneck energy for the path is referred to as the *bottleneck edge* for path p.

The bottleneck of a vertex t is defined as:

$$b_v = min\ _{p:p \text{ is a } s \to t \text{ path}}\ b_p \tag{5.2}$$

5.1.4 Stoichiometrically Balanced Pathways

Isolating a *path* in a graph sense is desirable and meaningful in many conventional applications (e.g., traveling salesman problem, network flow algorithms). However, in the context of biological applications, it is more meaningful to identify a *pathway*. A *pathway* from s to v is a subgraph in the network that contains a path $s \rightarrow v$, and an augmenting set of connected edges and nodes. These augmenting components are needed to ensure overall stoichiometric balance. That is, a stoichiometrically balanced pathway will not have any dangling internal nodes. The augmenting components can also be thought of as paths that complete the conversion of any remaining intermediates (unused by the main path) to the target metabolite.

5.1.5 Problem

The following problem is solved: Given a metabolic network graph $G_m = (V, E)$, and starting and ending vertices s and t, find the dominant-edge pathway from s to t. In this thesis, a dominant-edge pathway is defined based on ΔG. However, other measures such as flux data can also be used, if available, to determine the dominant-edge pathway.

Two sub-problems are identified. The first involves finding the dominant-edge path $s \rightarrow t$, and the second consists of augmenting the dominant-edge path to create a stoichiometrically balanced pathway. The first problem resembles the Bottleneck Shortest Path Problem. However, as explained shortly, when the first problem is solved, not only a path from $s \rightarrow t$ is found, but additional sibling edges and sibling nodes are also found that are integral parts of the dominant-edge path-

way. The second problem therefore involves graph traversals to identify the relevant augmenting components to produce a stoichiometrically balanced pathway. Therefore an algorithm, *Dominant-Edge Pathway*, is provided which first finds a partially dominant pathway, PDP, and then augments it to produce a stoichiometrically balanced pathway, SBP.

5.1.6 *Dominant-Edge Pathway* Algorithm

The details of the algorithm are given in Figure 5.1. Algorithm *Dominant-Edge Pathway* begins by finding a set of edges R that contains all edges responsible for setting the bottleneck energy for all vertices in G_m. Next, based on R, the function EXTRACT-DOMINANT-PATH determines the dominant-edge path from $s \rightarrow t$, along with all sibling edges and vertices associated with this path. That path is referred to as PDP. Then, to ensure stoichiometric balance, the augmentation technique must be implemented iteratively in both the forward and backward directions, because sibling edges and vertices can occur in either the forward or reverse direction. Therefore, first AUGMENT-PATHWAY based on PDP is called, and then AUGMENT-PATHWAY based on the transpose of PDP is called. The process repeats until SBP does not grow.

To find the dominant-edge path, modified Dijkstra's algorithm [109] is used, which identifies the single-source shortest path. In Dijkstra's algorithm, all distances are initialized to infinity with the exception of the source vertex distance, which is initialized to zero. Each vertexs predecessor is set to NIL. Dijkstra's algorithm utilizes relaxation. Relaxing an edge (u, v) checks if the shortest distance to v found so far can be improved by going through u, and if so, the shortest distance to v is

DOMINANT-EDGE-PATHWAY(G_m, s, t)
1. $R \leftarrow$ FIND-BOTTLENECK-ENERGIES (G_m, s)
2. $PDP \leftarrow$ EXTRACT-DOMINANT-PATH (s, t, R)
3. $SBP \leftarrow PDP$
4. while SBP is growing
5. $SBP \leftarrow$ AUGMENT-PATHWAY (s, t, SBP)
6. $SBP \leftarrow SPB \cup$ transpose(AUGMENT-PATHWAY($t, s,$ transpose (SBP)))
7. return SBP

FIND-BOTTLENECK-ENERGIES(G_m, s)
1. INITIALIZE-DOMINANT-PATH (G_m, s)
2. $Q \leftarrow$ the set of all nodes in G_m except s
3. $S \leftarrow s$; $R \leftarrow \{\}$
4. while Q is not empty
5. $x \leftarrow$ extract lowest energy vertex in Q
6. $r \leftarrow \{\}$ if $previous[x]$ is undefined OR $r \leftarrow$ edge($previous[x], x$)
7. $U \leftarrow$ set of products of r in $Q \cup \{x\}$
8. $S \leftarrow S \cup U$; $R \leftarrow R \cup \{r\}$
9. for each vertex v in U
10. RELAX(v)
11. remove v from Q
12. return SBP

INITIALIZE-DOMINANT-PATH(G_m, s)
1. for each vertex v in G_m
2. $energy[v] \leftarrow \infty$
3. $previous[v] \leftarrow$ undefined
4. $reaction[v] \leftarrow$ undefined
5. for each neighbor v of s
6. $energy[v] \leftarrow g_e$(edge(s, v))

RELAX(u)
1. for each neighbor v of u
2. $alt \leftarrow \max(energy[u], g_e(\text{edge}(u, v)))$
3. if $alt < energy[v]$
4. $energy[v] \leftarrow alt$
5. $reaction[v] \leftarrow$ edge(u, v)
6. $previous[v] \leftarrow u$

EXTRACT-DOMINANT-PATH(s, t, R)
1. $PDP \leftarrow \{t\}$
2. $u \leftarrow t$
3. while u is not equal to s
4. $PDP \leftarrow \{previous[u], reaction[u]\} \cup PDP$
5. if edge e is a sibling edge, then
6. $PDP \leftarrow PDP \cup \{$sibling edges($u$)$\} \cup \{$sibling vertices($u$)$\}$
7. $u \leftarrow$ source(e)

AUGMENT-PATHWAY(s, t, PDP)
1. $augmentMore \leftarrow$ TRUE
2. while $augmentMore$
3. $augmentMore \leftarrow$ FALSE
4. for each vertex v in PDP
5. if outdegree(v) = 0 and v is not t
6. $R' \leftarrow$ FIND-BOTTLENECK-ENERGIES(v)
7. $PDP \leftarrow PDP \cup$ EXTRACT-DOMINANT-PATH(v, t, R')
8. $augmentMore \leftarrow$ TRUE
9. return PDP

Figure 5.1: Pseudo code of *Dominant-Edge Pathway* algorithm.

updated. The predecessor to v responsible for this new shortest path value is also updated. Dijkstra's algorithm maintains a set S of vertices whose shortest-path

weights from the source have already been determined. The algorithm repeatedly selects the vertex u, not in S, with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u. A min-priority queue keyed by the distance values of the vertices is used to efficiently extract u.

The algorithm, FIND-BOTTLENECK-ENERGIES, differs from Dijkstra's shortest path algorithm as follows. Each vertex is associated with three variables: *energy*, *reaction* and *previous*. $energy[v]$ refers to the bottleneck energy of path from source to the vertex v that will be assigned to v. $reaction[v]$ refers to the edge from a vertex u to v responsible for setting $energy[v]$. $previous[v]$ refers to a vertex u connected to v through an edge (u, v) where u is responsible for setting $reaction[v]$. The initialization step sets the energies to ∞, except for the source vertex from which the search begins. Note that an edge in the graph may have more than one source, and thus both reaction and previous variables are needed for implementing the algorithm. Another difference is in the relaxation step. The energy assigned to a vertex v is the maximum energy of g_e, the ΔG associated with edge e leading from u to v, and the energy of vertex u. A min-priority queue is used, Q, keyed by energy of the vertices stored in Q. The set S stores all the vertices whose bottleneck energies have already been determined.

The algorithm FIND-BOTTLENECK-ENERGIES works as follows. While visiting vertices, this algorithm models the effect of selecting favored reactions by including minimum energy vertex (metabolite) into its frontier, S. All variables are initialized as shown in INITIALIZE-DOMINANT-PATH. Q is initialized with all nodes in G_m. The algorithm repeatedly extracts a vertex x with the minimum energy, and process it as follows. First, the set of all sibling vertices associated with

vertex x are found and stored into a set U (steps 6 & 7). In step 8, U is added to S, as the bottleneck energy of all vertices in U are now determined. This step ensures that once a reaction was used to set the bottleneck energy of a vertex, the energy of all sibling vertices are set and cannot be changed by further processing of the vertices. Similarly, R is augmented to include an edge r responsible for placing a vertex x in S. Each outgoing edge of the sibling vertices is then relaxed. The extraction continues until all vertices in Q have been processed.

Once the bottleneck energy from source vertex s to every node in the graph and each $reaction[v]$ values are found, all edges in R are removed that do not belong to the dominant-edge path. Function EXTRACT-DOMINANT-PATH executes a traversal from the target to the source, adding vertices and edges to PDP, including sibling vertices and edges. The traversal includes sibling edges and sibling vertices and thus results in a PDP (as opposed to a path).

The function AUGMENT-PATHWAY finds a dangling node d (no outgoing edges) in PDP (line 5), and finds a partial dominant pathway, PDP, from d to t. This operation occurs by computing bottleneck energies starting with d using FIND-BOTTLENECK-ENERGIES, and then adding vertices and edges found using EXTRACT-DOMINANT-PATH between d and t. This process applies to all dangling nodes originally in PDP as well as nodes found during finding partial dominant pathways from d.

The runtime of FIND-BOTTLENECK-ENERGIES is similar to Dijkstra's algorithm. It depends on the implementation of the priority queue. For a binary max-heap implementation when all vertices are reachable from the source, the run time is $O(|E|ln|V|)$. The run time of EXTRACT-DOMINANT-PATH is $O(|V|)$.

The run time for AUGMENT-PATHWAY is dominated by FIND-BOTTLENECK-ENERGIES, which is executed multiple times, but less than $|V|$. Based on the empirical results, the number of times FIND-BOTTLENECK-ENERGIES is called is typically small, and can be treated as a constant.

5.1.6.1 Example 2

The goal is to find the dominant-edge pathway from vertex s to vertex t in Figure 5.2. The energy of each reaction edge is marked along the edges. Consider four parallel paths from s to t: $\{s, e, f, d, t\}$, $\{s, e, d, c, t\}$, $\{s, f, d, t\}$, and $\{s, f, t\}$. The bottleneck energy along each of these paths is -5, -9, -15, and -15. In this example, the first two paths are thus not dominant paths. However, the last two paths contain sibling vertices d and t. The dominant pathway cannot have one of the paths and not the other to ensure stoichiometric balance. If, for example, vertices $\{s, f, t\}$ are chosen, then the reaction with Gibbs energy -20 will produce metabolite d. Therefore will must include *both* of these paths to produce a dominant-edge pathway including vertices $\{s, f, t, d\}$, and edges $\{(s, f), (f, t), (f, d), (d, t)\}$. The bottleneck energies associated with applying FIND-BOTTLENECK-ENERGIES are denoted next to each vertex. The dashed edges are found using EXTRACT-DOMINANT-PATH. The dotted edge (d, t) is found using AUGMENT-PATHWAY.

5.2 Results

The *Dominant-Edge Pathway* algorithm was tested on three examples with varying numbers of metabolites and reactions. The results were compared to those found using the EFM analysis tool efmtool [110]. The examples are culled from the literature

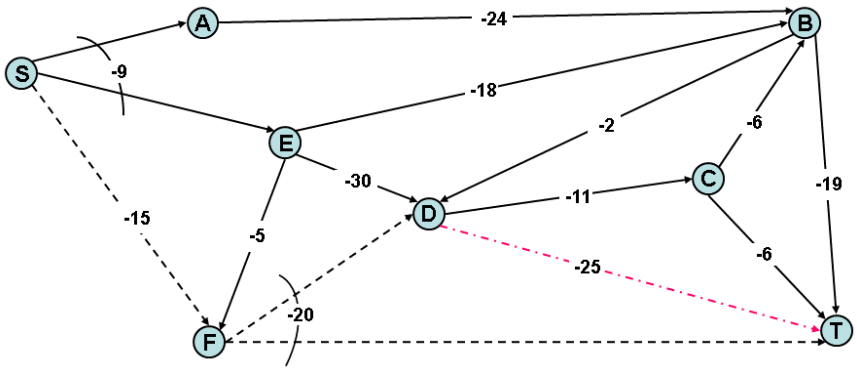


Figure 5.2: Example metabolic network. Numbers along edges indicate the Gibbs Free Energy Change. Numbers above each vertex denote bottleneck energies associated with each vertex. The dashed and dotted lines are the edges associated with the dominant pathway.

as there currently are no benchmark suites available to evaluate the algorithm. The ΔG for each reaction was computed using an available online tool [111]. The first test case consists of 21 metabolites and 20 reactions and includes pathways comprising the central carbon network of *Z. mobilis* expressing heterologous enzymes for xylose utilization [112]. The number of EFMs for this model was 2. The second test case is based on a recently published model of an ethanol producing strain of *E. coli* [2]. This network consists of 47 metabolites and 60 reactions, with three metabolites inputs: fructose, glucose, and xylose. The number of EFMs for this model was 33,000. The second test case was modified by removing a reaction responsible for biomass production (cell growth). This modified model is referred as case 2A. The number of edges of the graph for 2A was significantly reduced because the reaction removed was associated with several sibling edges. The third test case is a model of the rat liver cell [113]. This model consists of 38 metabolites and 60 reactions. While this model is of the same scale as the *E. coli* test models, it supports a larger number of reversible input-output pairings. A more detailed liver model having 110

metabolites and 119 reactions is also considered as test case. This detailed model is referred as 3A.

The relationship between the EFMs and the pathways found using *Dominant-Edge Pathway* is highlighted, before presenting the results of each case study. One way to use EFM analysis to find the dominant-edge pathway is to analyze all elementary modes connecting the source and target metabolites. This subset of elementary modes can then be rank-ordered based on the least negative reaction ΔG in each mode to identify the pathway containing the lowest thermodynamic barrier. The pathway(s) found using this method may coincide, be part of, or partially overlap with the dominant-edge pathway. EFM analysis does not necessarily find the same pathway identified using *Dominant-Edge Pathway* as EFM finds all possible pathways that can cover the source and destination metabolites.

The overlap possibilities between EFM and *Dominant-Edge Pathway* results are summarized in Table 5.1. The columns in the table indicate the following: test case number, source metabolite, target metabolite, number of metabolites and reactants along the dominant-edge pathway found using the *Dominant-Edge Pathway* algorithm, and the number of relevant modes found by EFM.

In the first test case, the EFM and *Dominant-Edge Pathway* analyses produce identical pathways. In the second example (test case 2), the three dominant-edge pathways, each corresponding to a different input metabolite, are proper subsets of 169, 156, and 725 elementary modes. In test case 2A, two of the dominant-edge pathways are identical to those found using EFM, and the third pathway was having partial overlap with EFM. In the third example, there was only partial overlap between the dominant-edge pathways and the modes found using EFM

analysis. The number reported in the last column in the table indicates the number of elementary modes that contained at least 50% of the reactions found in the

Table 5.1: Results of DOMINANT PATH Algorithm compared to paths (modes) found using EFM analysis.

Test Case	Inputs	Output	Metabolites	Reactions	EFM Modes
1	Glucose	Ethanol	13	12	1
1	Xylose	Ethanol	20	19	1
2	Fructose	Ethanol	12	11	169
2	Glucose	Ethanol	13	12	156
2	Xylose	Ethanol	21	19	725
2A	Fructose	Ethanol	12	11	1
2A	Glucose	Ethanol	13	12	1
2A	Xylose	Ethanol	21	19	1
3	Alanine	Glucose	10	10	14
3	Alanine	Urea	8	8	3
3	Cysteine	Glucose	10	10	14
3	Cysteine	Urea	8	8	3
3	Glycine	Alanine	2	3	1
3	Glycine	Glucose	11	11	20
3	Glycine	Cysteine	2	3	1
3	Glycine	Urea	9	9	9
3	Tyrosine	Glucose	12	1	18
3	Tyrosine	Urea	8	7	5
3	Acetyl-CoA	Glucose	15	14	30
3	Acetyl-CoA	Urea	11	10	9
3	Serine	Glucose	10	10	15
3	Serine	Urea	8	8	5
3A	Alanine	Glucose	33	33	-
3A	Alanine	Urea	13	15	-
3A	Cysteine	Glucose	54	62	-
3A	Cysteine	Urea	17	17	-
3A	Glycine	Alanine	47	52	-
3A	Glycine	Glucose	56	63	-
3A	Glycine	Cysteine	6	4	-
3A	Glycine	Urea	20	19	-
3A	Tyrosine	Glucose	42	46	-
3A	Tyrosine	Urea	38	42	-
3A	Acetyl-CoA	Glucose	33	33	-
3A	Acetyl-CoA	Urea	21	26	-
3A	Serine	Glucose	54	61	-
3A	Serine	Urea	21	20	-

corresponding dominant-edge pathway. For the test case 3A EFM analysis was not completed (over 1.5 million modes were reported by the tool before it crashed after 3 days of execution) so only number of metabolites and number of reactions present in dominant-edge pathway are reported.

The significance of the *Dominant-Edge Pathway* algorithm thus lies in its ability to efficiently identify thermodynamically favored reaction routes without costly enumeration-based path analysis. This is evident in the runtime and memory requirements needed to perform the analysis. The run time for all test cases was < 1 second using a single 3 GHz quad-core Pentium computer with 4 GB of RAM. The efmtool run time was 1 second for the first and third examples, and 10 seconds for the second example. When the *Dominant-Edge Pathway* algorithm was applied to the detailed model, the resulting pathways had similarity with the pathways found using reduced model while maintaining a run time of less than 1 second.

5.3 Conclusion

This thesis presents a novel algorithm for pathway analysis of biochemical networks. The *Dominant-Edge Pathway* algorithm departs from prior efforts on exhaustive enumeration in the following ways. Given a desired pathway feature, e.g. thermodynamic favorability, the algorithm merges the weight assignment and the path identification steps. The algorithm provides an efficient search process compared to enumeration-based approaches such as EFM and extreme pathway analysis. Stoichiometric balancing is applied at the end of the search process, after the main trunk of the path has been generated, again saving run-time. The main limitation of the algorithm deals with the uncertainty of the Gibbs free energy estimates used

to characterize the thermodynamic favorability of the reactions. On the other hand, the algorithm is general with respect to the type of the reaction (edge) weight, and could be expanded to use measurement derived steady-state flux weights. In conclusion, the results of the analysis indicate that the algorithm presented in this thesis provides an efficient alternative to the enumeration based approaches, especially for applications where the input and output metabolites are *a priori* defined.

Chapter 6

Conclusions and Future Directions

This thesis presented three graph-based algorithms for the analysis of metabolic networks. The algorithms developed here include *gEFM*, a graph-based algorithm used to enumerate elementary flux modes in metabolic networks; *PreProPath* algorithm, used to identify predictable profitable pathways; and *Dominant-Edge Pathway* algorithm, used to identify bottleneck reactions. The results demonstrate that graph-based approaches are not only powerful but are also viable for metabolic network analysis.

6.1 Thesis Summary

The thesis provides some key contributions to pathway analysis of metabolic networks. First, an algorithm, *gEFM*, is developed for computing the elementary flux modes within a metabolic network. The algorithm is iterative, processing one

metabolite-balancing constraint at a time to generate partial pathways that are vetted for independence against all prior generated such pathways. The algorithm implements the canonical approach, which in turn is a variant of the double-description method. The thesis demonstrates for the first time that graph-based approaches are viable for computing EFMs, and by natural extension, for computing the extreme rays of a convex cone. The main advantage of *gEFM* is utilizing the underlying structural information to derive a robust ordering for processing the metabolite balancing constraints. When applied to several test cases, *gEFM* is found to be competitive for several test cases when compared to other EFM computational methods.

Second, potential targets for engineering interventions are identified to improve production of a desired product from the given source metabolite using the *PreProPath* algorithm. The algorithm presented here can be adapted to utilize edge weights that correspond to a different desired objective. The method is "uncertainty-aware" as it considers degrees of freedom in the model and multiple metabolic states arising because of different uptake rates. The algorithm is based on guided search and avoids exhaustive exploration of all pathways in the network. The results of the case studies suggest that the algorithm can efficiently guide the search for pathway engineering targets.

Third, a bottleneck or limiting reactions from a given source metabolite to the desired destination metabolite in a metabolic networks are identified using the *Dominant-Edge Pathway* algorithm. It is shown that the approach is efficient because it does not enumerate all the pathways between source and destination metabolite. The analysis for thermodynamic bottlenecks for different test cases shows that the algorithm can be applied to large-scale models for which enumeration

of all the pathways is not feasible. Moreover, the algorithm can be used to identify other types of bottlenecks such as flux bottleneck by changing the edge-weighting scheme.

6.2 Future Research Directions

This thesis covers three graph-based pathway analysis techniques. The following are some possible extensions of the work discussed in the thesis:

- Adjacency test is the most computationally expensive step in EFM analysis. Implementation of adjacency test on graphical processing units (GPUs) can improve performance of EFM analysis because of availability of huge number of concurrent threads.

- Network partitioning for scalability of EFM analysis can be explored. Partitioning of network for *gEFM* algorithm is natural. Performing EFM analysis on partitions and then merging the partitions may improve the performance of EFM analysis.

- EFM analysis identifies pathways that can represent all possible steady-states. In reality, some of the EFMs are not biologically feasible. There is a need for computational methods to remove biologically infeasible pathways by incorporating more constraints such as physiological constraints and regulatory constraints.

- For identification of potential engineering targets, flux variations in the network are used as edge weights in *PreProPath* algorithm. By incorporating regulatory information in edge weights, predictability of *PreProPath* algorithm

can be further improved.

- Enumerating all the pathways is a fundamentally computationally intractable problem. To avoid enumeration of all the pathways approximate methods such as probabilistic (random walk) can be used to identify a set of pathways that represent the feasible flux space.

- Adapting algorithms to high performance computing platforms can be explored to improve the performance of existing algorithms for pathway analysis. This includes mapping complex data structures to the memory models of high performance computing platforms and modification of algorithms to support concurrent execution.

- Advances in proteomics, metabolomics and fluxomics have provided large volumes of data that can be used to improve existing pathway analysis approaches. EFM analysis can benefit from these advances by identifying the most significant EFMs based on the 'omics' data.

Bibliography

[1] Julien Gagneur and Steffen Klamt. Computation of elementary modes: a unifying framework and the new binary approach. *BMC bioinformatics*, 5:175, 2004.

[2] C. T. Trinh, P. Unrean, and F. Srienc. Minimal *Escherichia coli* cell for the most efficient production of ethanol from hexoses and pentoses. *Appl Environ Microbiol*, 74(12):3634–43, 2008.

[3] Ilana S. Aldor and Jay D. Keasling. Process design for microbial plastic factories: metabolic engineering of polyhydroxyalkanoates. *Current opinion in biotechnology*, 14(5):475–483, 2003.

[4] C. E. Nakamura and G. M. Whited. Metabolic engineering for the microbial production of 1,3-propanediol. *Curr Opin Biotechnol*, 14(5):454–9, 2003.

[5] Eric J. Steen, Yisheng Kang, Gregory Bokinsky, Zhihao Hu, Andreas Schirmer, Amy McClure, Stephen B. del Cardayre, and Jay D. Keasling. Microbial production of fatty-acid-derived fuels and chemicals from plant biomass. *Nature*, 463(7280):559–562, 2010.

[6] M. C. Chang and J. D. Keasling. Production of isoprenoid pharmaceuticals by engineered microbes. *Nat Chem Biol*, 2(12):674–81, 2006.

[7] Vincent J. J. Martin, Douglas J. Piteral, Sydnor T. Withers, Jack D. Newman, and Jay D. Keasling. Engineering a mevalonate pathway in *Escherichia coli* for production of terpenoids. *Nature biotechnology*, 21(7):796–802, 2003.

[8] D. J. Pitera, C. J. Paddon, J. D. Newman, and J. D. Keasling. Balancing a heterologous mevalonate pathway for improved isoprenoid production in *Escherichia coli*. *Metab Eng*, 9(2):193–207, 2007.

[9] Kevin T. Watts, Benjamin N. Mijts, and Claudia Schmidt-Dannert. Current and emerging approaches for natural product biosynthesis in microbial cells. *Advanced Synthesis & Catalysis*, 347(7-8):927–940, 2005.

[10] Salvador Peir, Hugo G. Menzella, Eduardo Rodriguez, John Carney, and Hugo Gramajo. Production of the potent antibacterial polyketide erythromycin c in *Escherichia coli*. *Applied and Environmental Microbiology*, 71(5):2539–2547, 2005.

[11] Blaine A. Pfeifer, Suzanne J. Admiraal, Hugo Gramajo, David E. Cane, and Chaitan Khosla. Biosynthesis of complex polyketides in a metabolically engineered strain of *E. coli*. *Science*, 291(5509):1790, 2001.

[12] S. Atsumi, T. Hanai, and J. C. Liao. Non-fermentative pathways for synthesis of branched-chain higher alcohols as biofuels. *Nature*, 451(7174):86–9, 2008.

[13] T. Hanai, S. Atsumi, and J. C. Liao. Engineered synthetic pathway for isopropanol production in *Escherichia coli*. *Appl Environ Microbiol*, 73(24):7814–8, 2007.

[14] M. Inui, M. Suda, S. Kimura, K. Yasuda, H. Suzuki, H. Toda, S. Yamamoto, S. Okino, N. Suzuki, and H. Yukawa. Expression of *Clostridium acetobutylicum* butanol synthetic genes in *Escherichia coli*. *Appl Microbiol Biotechnol*, 77(6):1305–16, 2008.

[15] L. P. Yomano, S. W. York, K. T. Shanmugam, and L. O. Ingram. Deletion of methylglyoxal synthase gene (mgsa) increased sugar co-metabolism in ethanol-producing *Escherichia coli*. *Biotechnol Lett*, 31(9):1389–98, 2009.

[16] A. F. Cann and J. C. Liao. Production of 2-methyl-1-butanol in engineered *Escherichia coli*. *Appl Microbiol Biotechnol*, 81(1):89–98, 2008.

[17] D. Densmore and S. Hassoun. Design automation for synthetic biological systems. *IEEE Design and Test of Computers, to appear*, 2012.

[18] D. A. Fell and J. R. Small. Fat synthesis in adipose tissue. An examination of stoichiometric constraints. *Biochem J*, 238(3):781–6, 1986.

[19] R. Mahadevan and C. H. Schilling. The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. *Metab Eng*, 5(4):264–76, 2003.

[20] YeeWen Choon, MohdSaberi Mohamad, Safaai Deris, RosliMd. Illias, ChuiiKhim Chong, and LianEn Chai. A hybrid of bees algorithm and flux balance

analysis with optknock as a platform for in silico optimization of microbial strains. *Bioprocess and Biosystems Engineering*, 37(3):521–532, 2014.

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

[22] P. Pharkya, A. P. Burgard, and C. D. Maranas. Optstrain: a computational framework for redesign of microbial production systems. *Genome Res*, 14(11):2367–76, 2004.

[23] Priti Pharkya and Costas D. Maranas. An optimization framework for identifying reaction activation/inhibition or elimination candidates for overproduction in microbial systems. *Metabolic Engineering*, 8(1):1 – 13, 2006.

[24] Monica L. Mo, Neema Jamshidi, and Bernhard O. Palsson. A genome-scale, constraint-based approach to systems biology of human metabolism. *Mol. BioSyst.*, 3:598–603, 2007.

[25] Jennifer L. Reed and Bernhard . Palsson. Thirteen years of building constraint-based in silico models of *Escherichia coli*. *Journal of Bacteriology*, 185(9):2692–2699, 2003.

[26] Mona Yousofshahi, Ehsan Ullah, Russell Stern, and Soha Hassoun. MC3: a steady-state model and constraint consistency checker for biochemical networks. *BMC Systems Biology*, 7(1), 2013.

[27] S. Schuster and C. Hilgetag. On elementary flux modes in biochemical reaction systems at steady state. *J. Biol. Syst*, 2:165–182, 1994.

[28] R. Carlson and F. Srienc. Fundamental *Escherichia coli* biochemical pathways for biomass and energy production: creation of overall flux states. *Biotechnol Bioeng*, 86(2):149–62, 2004.

[29] Vicente Acuna, Flavio Chierichetti, Vincent Lacroix, Alberto Marchetti-Spaccamela, Marie-France Sagot, and Leen Stougie. Modes and cuts in metabolic networks: complexity and algorithms. *Bio Systems*, 95:51–60, 2009.

[30] J. Stelling, S. Klamt, K. Bettenbrock, S. Schuster, and E. D. Gilles. Metabolic network structure determines key aspects of functionality and regulation. *Nature*, 420(6912):190–3, 2002.

[31] J.A. Papin, N.D. Price, J.S. Edwards, and B.O. Palsson. The genome-scale metabolic extreme pathway structure in haemophilus influenzae shows significant network redundancy. *Journal of Theoretical Biology*, 215:67–82, 2002.

[32] N. Vijayasankaran, R. Carlson, and F. Srienc. Metabolic pathway structures for recombinant protein synthesis in *Escherichia coli*. *Appl Microbiol Biotechnol*, 68(6):737–46, 2005.

[33] F. Llaneras and J. Pic. An interval approach for dealing with flux distributions and elementary modes activity patterns. *Journal of theoretical biology*, 246:290–308, 2007.

[34] S. Klamt and E. D. Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–34, 2004.

[35] A. P. Burgard, E. V. Nikolaev, C. H. Schilling, and C. D. Maranas. Flux coupling analysis of genome-scale metabolic network reconstructions. *Genome Res*, 14(2):301–12, 2004.

[36] S. Schuster, S. Klamt, W. Weckwerth, F. Moldenhauer, and T. Pfeiffer. Use of network analysis of metabolic systems in bioengineering. *Bioprocess and Biosystems Engineering*, 24(6):363–372, 2002.

[37] J. A. Papin, N. D. Price, and B. O. Palsson. Extreme pathway lengths and reaction participation in genome-scale metabolic networks. *Genome Res*, 12(12):1889–900, 2002.

[38] Ross P Carlson. Decomposition of complex microbial behaviors into resource-based stress responses. *Bioinformatics (Oxford, England)*, 25:90–7, 2009.

[39] Bas Teusink, Anne Wiersma, Douwe Molenaar, Christof Francke, Willem M. de Vos, Roland J. Siezen, and Eddy J. Smid. Analysis of growth of *Lactobacillus plantarum* WCFS1 on a complex medium using a genome-scale metabolic model. *Journal of Biological Chemistry*, 281(52):40041–40048, 2006.

[40] Gautham V Sridharan, Ehsan Ullah, Soha Hassoun, and Kyongbum Lee. Discovery of substrate cycles in large scale metabolic networks using hierarchical modularity. Manuscript, 2014.

[41] J. Schwender, F. Goffman, J. B. Ohlrogge, and Y. Shachar-Hill. Rubisco without the calvin cycle improves the carbon efficiency of developing green seeds. *Nature*, 432(7018):779–82, 2004.

[42] F. Noika, J. Guddat, H. Hollatz, and B. Bank. *Theorie der linearen parametrischen Optimierung. 312 S., Berlin 1974. Akademie-Verlag. Preis 52,- M*, volume 56, pages 125–126. WILEY-VCH Verlag, 1976.

[43] S Schuster, C Hilgetag, J Woods, and DA. Fell. Elementary modes of functioning in biochemical reaction networks. aspects of interpretation and application,. *Computation in Cellular and Molecular Biological Systems (Cuthbertson, R, Holcombe, M and Paton, R, Eds.), World Scientific: Singapore*, pages 151–165, 1996.

[44] C. Wagner. Nullspace approach to determine the elementary modes of chemical reaction systems. *Journal of Physical Chemistry B*, 108(7):2425–2431, 2004.

[45] R. Urbanczik and C. Wagner. An improved algorithm for stoichiometric network analysis: theory and applications. *Bioinformatics*, 21(7):1203–10, 2005.

[46] T S Motzkin, H Raiffa, G L Thompson, and R M Thrall. The double description method, 1953.

[47] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities. *Zh. Vychisl. Mat. Mat. Fiz.*, 5(2):334–337, 1965.

[48] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[49] Komei Fukuda and Alain Prodon. Double description method revisited. *Combinatorics and Computer Science*, 1:91–111, 1996.

[50] A. von Kamp and S. Schuster. Metatool 5.0: fast and flexible elementary modes analysis. *Bioinformatics*, 22(15):1930–1, 2006.

[51] Marco Terzer and J. Stelling. Elementary flux modes state-of-the-art implementation and scope of application. *BMC Systems Biology*, 1:P2, 2007.

[52] M.L. Mavrovouniotis, G. Stephanopoulos, and Stephanopoulos. G. Computer-aided synthesis of biochemical pathways. *Biotechn. Bioeng*, 36:1119–1132, 1990.

[53] M.L. Mavrovouniotis. Synthesis of reaction mechanisms consisting of reversible and irre- versible steps. 1. a synthesis approach in the context of simple examples. *Ind. Eng. Chem*, 31:1625–1637, 1992.

[54] M.L. Mavrovouniotis. Synthesis of reaction mechanisms consisting of reversible and irre- versible steps. 2. formalization and analysis of the synthesis algorithm. *Ind. Eng. Chem*, 31:1637–1653, 1992.

[55] Harold N. Gabow and Robert E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.

[56] Bruce L. Clarke. *Stability of Complex Reaction Networks*, pages 1–215. John Wiley & Sons, Inc., 1980.

[57] Alex Seressiotis and James E. Bailey. Mps: An algorithm and data base for metabolic pathway synthesis. *BIOTECHNOLOGY LETTERS*, 8(12):837–842, 1986.

[58] C. H. Schilling, D. Letscher, and B. O. Palsson. Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. *J Theor Biol*, 203(3):229–48, 2000.

[59] S. Schuster, C. Hilgetag, JH Woods, and DA Fell. Reaction routes in biochemical reaction systems: algebraic properties, validated calculation procedure and example from nucleotide metabolism. *Journal of mathematical biology*, 45:153–181, 2002.

[60] K. H. Borgwardt. Average-case analysis of the double description method and the beneath-beyond algorithm. *Discrete & Computational Geometry*, 37(2):175–204, 2007.

[61] B. A. Burton. Optimizing the double description method for normal surface enumeration. *Mathematics of Computation*, 79(269):453–84, 2010.

[62] Dimitrije Jevremovic, C.T. Trinh, Friedrich Srienc, and Daniel Boley. On algebraic properties of extreme pathways in metabolic networks. *Journal of Computational Biology*, 17:107–119, 2010.

[63] S.J. Wiback, Iman Famili, H.J. Greenberg, and B.O. Palsson. Monte carlo sampling can be used to determine the size and shape of the steady-state flux space. *Journal of theoretical biology*, 228:437–447, 2004.

[64] R. Schwarz, P. Musch, A. von Kamp, B. Engels, H. Schirmer, S. Schuster, and T. Dandekar. Yana - a software tool for analyzing flux modes, gene-expression and enzyme activities. *BMC Bioinformatics*, 6:135, 2005.

[65] J. M. Schwartz and M. Kanehisa. Quantitative elementary mode analysis of metabolic pathways: the example of yeast glycolysis. *BMC Bioinformatics*, 7:186, 2006.

[66] D. Segre, D. Vitkup, and G. M. Church. Analysis of optimality in natural and perturbed metabolic networks. *Proc Natl Acad Sci U S A*, 99(23):15112–7, 2002.

[67] N.D. Price, J.A. Papin, and B.O. Palsson. Determination of redundancy and systems properties of the metabolic network of *Helicobacter pylori* using genome-scale extreme pathway analysis. *Genome Research*, 12:760–769, 2002.

[68] S. P. Reidy and J. M. Weber. Accelerated substrate cycling: a new energy-wasting role for leptin in vivo. *Am J Physiol Endocrinol Metab*, 282(2):E312–7, 2002.

[69] J. Gebauer, S. Schuster, L. F. de Figueiredo, and C. Kaleta. Detecting and investigating substrate cycles in a genome-scale human metabolic network. *FEBS J*, 279(17):3192–202, 2012.

[70] C. Kaleta, L.F. de Figueiredo, J. Behre, and S. Schuster. Efmevolver: Computing elementary fux modes in genome-scale metabolic networks. *14th German Conference on Bioinformatics (GCB). Gesellschaft f. Informatik e.V., Bonn.*, pages 180–190, 2009.

[71] Gautham Vivek Sridharan. *Modularity Analysis of Metabolic Networks Based on Shortest Retroactive Distances (ShReD).* PhD thesis, Tufts University,

2013.

[72] GauthamVivek Sridharan, Michael Yi, Soha Hassoun, and Kyongbum Lee. Metabolic flux-based modularity using shortest retroactive distances. *BMC Systems Biology*, 6(1), 2012.

[73] Marco Terzer and J. Stelling. Accelerating the computation of elementary modes using pattern trees. *Algorithms in Bioinformatics*, 4175:333–343, 2006.

[74] ChristianL Barrett, MarkusJ Herrgard, and Bernhard Palsson. Decomposing complex reaction networks using random sampling, principal component analysis and basis rotation. *BMC Systems Biology*, 3(1), 2009.

[75] Markus W. Covert, Eric M. Knight, Jennifer L. Reed, Markus J. Herrgard, and Bernhard O. Palsson. Integrating high-throughput and computational data elucidates bacterial networks. *Nature*, 429:92–96, 2004.

[76] C. Kaleta, L. F. de Figueiredo, and S. Schuster. Can the whole be less than the sum of its parts? pathway analysis in genome-scale metabolic networks using elementary flux patterns. *Genome Res*, 19(10):1872–83, 2009.

[77] S. Klamt. Generalized concept of minimal cut sets in biochemical networks. *Biosystems*, 83(2-3):233–47, 2006.

[78] Egon Balas and William Pulleyblank. The perfectly matchable subgraph polytope of a bipartite graph. *Networks*, 13(4):495–516, 1983.

[79] Sangbum Lee, Chan Phalakornkule, Michael M Domach, and Ignacio E Grossmann. Recursive MILP model for finding all the alternate optima in LP models

for metabolic networks. *Computers & Chemical Engineering*, 24(27):711 – 716, 2000.

[80] Yu-Sin Jang, Jong Myoung Park, Sol Choi, Yong Jun Choi, Do Young Seung, Jung Hee Cho, and Sang Yup Lee. Engineering of microorganisms for the production of biofuels and perspectives based on systems metabolic engineering approaches. *Biotechnology advances*, 2011.

[81] W. C. Ruder, T. Lu, and J. J. Collins. Synthetic biology moving into the clinic. *Science*, 333:1248–1252, 2011.

[82] S. Schuster, D. A. Fell, and T. Dandekar. A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. *Nat Biotechnol*, 18(3):326–32, 2000.

[83] Marco Terzer. *Large scale methods to enumerate extreme rays and elementary modes*. PhD thesis, Swiss Federal Institute of Technology, Zurich, 2009.

[84] Yaguang Si, Jeongah Yoon, and Kyongbum Lee. Flux profile and modularity analysis of time-dependent metabolic changes of de novo adipocyte formation. *American Journal of Physiology - Endocrinology and Metabolism*, 292(6):E1637–E1646, 2007.

[85] LE Quek, S Dietmair, JO Kromer, and LK. Nielsen. Metabolic flux analysis in mammalian cell culture. *Metab Eng*, 12(2):161–71, 2010.

[86] Ines Thiele, Thuy D. Vo, Nathan D. Price, and Bernhard Palsson. Expanded metabolic reconstruction of *Helicobacter pylori* (iIT341 GSM/GPR): an in

silico genome-scale characterization of single- and double-deletion mutants. *Journal of Bacteriology*, 187(16):5818–5830, 2005.

[87] Natalie C. Duarte, Markus J. Herrgard, and Bernhard . Palsson. Reconstruction and validation of *Saccharomyces cerevisiae* iND750, a fully compartmentalized genome-scale metabolic model. *Genome Res.*, 14(7):1298–1309, 2004.

[88] Nanette R Boyle and John A Morgan. Flux balance analysis of primary metabolism in chlamydomonas reinhardtii. *BMC Syst Biol.*, 3(4), 2009.

[89] M. Terzer and J. Stelling. Elementary flux mode tool (efmtool), December 2009.

[90] M. E. Dyer and L. G. Proll. An algorithm for determining all extreme points of a convex polytope. *Mathematical Programming*, 12(1):81–96, December 1977.

[91] Vicente Acuna, Alberto Marchetti-Spaccamela, Marie-France Sagot, and Leen Stougie. A note on the complexity of finding and enumerating elementary modes. *Bio Systems*, 99(3):210–4, 2010.

[92] Hendrik P. J. Bonarius, Georg Schmid, and Johannes Tramper. Flux analysis of underdetermined metabolic networks: the quest for the missing constraints. *Trends in Biotechnology*, 15(8):308–314, 1997.

[93] Amit Varma and Bernhard O. Palsson. Metabolic flux balancing: Basic concepts, scientific and practical use. *Nat Biotech*, 12(10):994–998, 1994.

[94] D. Croes, F. Couche, S. J. Wodak, and J. van Helden. Metabolic pathfinding: inferring relevant pathways in biochemical networks. *Nucleic Acids Res*, 33(Web Server issue):W326–30, 2005.

[95] P. Gerlee, L. Lizana, and K. Sneppen. Pathway identification by network pruning in the metabolic network of *Escherichia coli*. *Bioinformatics*, 25(24):3282–3288, 2009.

[96] J. M. Clomburg and R. Gonzalez. Biofuel production in *Escherichia coli*: the role of metabolic engineering and synthetic biology. *Appl Microbiol Biotechnol*, 86(2):419–34, 2010.

[97] R. Carlson and F. Srienc. Fundamental *Escherichia coli* biochemical pathways for biomass and energy production: identification of reactions. *Biotechnol Bioeng*, 85(1):1–19, 2004.

[98] R. Carlson and F. Srienc. Fundamental *Escherichia coli* biochemical pathways for biomass and energy production: creation of overall flux states. *Biotechnol Bioeng*, 86(2):149–62, 2004.

[99] Adam M. Feist, Christopher S. Henry, Jennifer L. Reed, Markus Krummenacker, Andrew R. Joyce, Peter D. Karp, Linda J. Broadbelt, Vassily Hatzimanikatis, and Bernhard O. Palsson. A genome-scale metabolic reconstruction for *Escherichia coli* K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information. *Mol Syst Biol*, 3, 2007.

[100] S. H. Hong and S. Y. Lee. Importance of redox balance on the production of succinic acid by metabolically engineered *Escherichia coli*. *Appl Microbiol Biotechnol*, 58(3):286–90, 2002.

[101] A. M. Sanchez, G. N. Bennett, and K. Y. San. Batch culture characterization and metabolic flux analysis of succinate-producing *Escherichia coli* strains.

Metab Eng, 8(3):209–26, 2006.

[102] S. Ranganathan, P. F. Suthers, and C. D. Maranas. Optforce: an optimization procedure for identifying all genetic manipulations leading to targeted overproductions. *PLoS Comput Biol*, 6(4):e1000744, 2010.

[103] D. A. Fell. Metabolic control analysis: a survey of its theoretical and experimental development. *Biochem J*, 286 (Pt 2):313–30, 1992.

[104] J. Rodriguez, J. M. Lema, and R. Kleerebezem. Energy-based models for environmental biotechnology. *Trends Biotechnol*, 26(7):366–74, 2008.

[105] A. P. Wlaschin, C. T. Trinh, R. Carlson, and F. Srienc. The fractional contributions of elementary modes to the metabolism of *Escherichia coli* and their estimation from reaction entropies. *Metab Eng*, 8(4):338–52, 2006.

[106] Jr. Forsythe, R. G., P. D. Karp, and M. L. Mavrovouniotis. Estimation of equilibrium constants using automated group contribution methods. *Comput Appl Biosci*, 13(5):537–43, 1997.

[107] Maurice Pollack. The maximum capacity through a network. *INFORMS*, 8(5):733–736, 1960.

[108] Matthias A.F. Peinhardt Volker Kaibel. On the bottleneck shortest path problem. *ZIB-Report 06-22*, 2006.

[109] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, pages 269–271, 1959.

[110] M. Terzer and J. Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24(19):2229–35, 2008.

[111] Matthew D. Jankowski, Christopher S. Henry, Linda J. Broadbelt, and Vassily Hatzimanikatis. Group contribution method for thermodynamic analysis of complex metabolic networks. *Biophys J*, 95(3):1487–99, 2008.

[112] M. M. Altintas, C. K. Eddy, M. Zhang, J. D. McMillan, and D. S. Kompala. Kinetic modeling to optimize pentose fermentation in *Zymomonas mobilis*. *Biotechnol Bioeng*, 94(2):273–95, 2006.

[113] R. P. Nolan, A. P. Fenley, and K. Lee. Identification of distributed metabolic objectives in the hypermetabolic liver by flux and energy balance analysis. *Metab Eng*, 8(1):30–45, 2006.

Appendices

Appendix A

Detailed code of *gEFM*

Algorithm 1: EFMGenerator class

Description: This class removes metabolites to generate elementary flux modes

```
1  class EFMGenerator
2  begin
3      List<Pathway > pathways
4      boolean [ ] metatbolitesRemoved
5      EFMGenerator (List<Pathway > reactions)
6      begin
7          pathways ← reactions
8          metatbolitesRemoved ← new boolean [MetaboliteCount]
9          for i ← 0 to (MetaboliteCount − 1) do
10             if Metabolite i is external then
11                 metatbolitesRemoved [i] ← true
12             end
13             else
14                 metatbolitesRemoved [i] ← false
15             end
16         end
17     end
18     generateEFMs ()
19     begin
20         while There are unprocessed metabolites do
21             m ← Unprocessed metabolite with minimum number of combinations
22             Pathway [ ] inputs ← All pathways in pathways for which m is a reactant
23             Remove inputs from pathways
24             ReversibleTree inputTree ← new ReversibleTree (inputs)
25             BitPatternTree bptIn ← a new BitPatternTree of inputs
26             Pathway [ ] outputs ← All pathways in pathways for which m is a product
27             Remove outputs from pathways
28             ReversibleTree outputTree ← new ReversibleTree (outputs)
29             BitPatternTree bptOut ← a new BitPatternTree of outputs
30             BitPatternTree bptNon ← a new BitPatternTree of pathways
31             generateCombinations (bptIn, bptOut, bptNon, inputTree.root, outputTree.root)
32             metatbolitesRemoved [m] ← true
33         end
34     end
35 end
```

Algorithm 2: EFMGenerator class

Description: This class removes metabolites to generate elementary flux modes

```
1  class EFMGenerator
2  begin
3      void generateCombinations (BitPatternTree bptIn, BitPatternTree bptOut,
       BitPatternTree bptNon, ReversibleTreeNode input, ReversibleTreeNode output, int m)
4      begin
5          activeReversibleReactions ← input.activeReversibleReactions | output.activeReversibleReactions
6          if activeReversibleReactions has reversible reaction pair then
7              return
8          end
9          if (input is not leaf) and (output is not leaf) then
10             generateCombinations (bptIn, bptOut, bptNon, input.child0, output.child0, m)
11             generateCombinations (bptIn, bptOut, bptNon, input.child0, output.child1, m)
12             generateCombinations (bptIn, bptOut, bptNon, input.child1, output.child0, m)
13             generateCombinations (bptIn, bptOut, bptNon, input.child1, output.child1, m)
14         end
15         else if (input is not leaf) and (output is leaf) then
16             generateCombinations (bptIn, bptOut, bptNon, input.child0, output, m)
17             generateCombinations (bptIn, bptOut, bptNon, input.child1, output, m)
18         end
19         else if (input is leaf) and (output is not leaf) then
20             generateCombinations (bptIn, bptOut, bptNon, input, output.child0, m)
21             generateCombinations (bptIn, bptOut, bptNon, input, output.child1, m)
22         end
23         else
24             foreach Pathway i in input do
25                 foreach Pathway o in output do
26                     Pathway combo ← new Pathway (i, o)
27                     if !bptIn.isSuperset (combo, i) and !bptOut.isSuperset (combo, o) and
                       !bptNon.isSuperset (combo) then
28                         combo.updateMetaboliteCoefficients (m)
29                         pathways.add(combo)
30                     end
31                 end
32             end
33         end
34     end
35 end
```

Algorithm 3: BitPatternTree class

Description: This class represents a bit pattern tree

```
1  class BitPatternTree
2  begin
3      boolean bitsUsed [ ]
4      BitPatternTreeNode root

5      BitPatternTree()
6      begin
7          root ← new BitPatternTreeNode ()
8          bitsUsed ← new boolean [ReactionCount]
9      end

10     void addPathway(Pathway p)
11     begin
12         for i ← 0 to (ReactionCount − 1) do
13             bitsUsed ← false
14         end
15         root.addPathway(p, bitsUsed)
16     end

17     boolean isSuperset(Pathway p)
18     begin
19         return root.isSuperset(p)
20     end
21 end
```

Algorithm 4: BitPatternTreeNode class

Description: This class represents a bit pattern tree node

```
1  class BitPatternTreeNode
2  begin
3      boolean leaf
4      BitPatternTreeNode child0, child1
5      int splitBit
6      List<Pathway > pathways
7      ReactionBitData label
8      BitPatternTreeNode()
9      begin
10         leaf ← true
11         child0 ← null
12         child1 ← null
13         splitBit ← −1
14         pathways ← new List<Pathway > ()
15     end
16     boolean isSuperset(Pathway p)
17     begin
18         if p.isSupersetOf(label ) then
19             return true
20         end
21         if leaf then
22             foreach Pathway pth in pathways do
23                 if p = pth then
24                     continue
25                 end
26                 if p.isSupersetOf(pth) then
27                     return true
28                 end
29             end
30             return false
31         end
32         else
33             if p.reactions[splitBit ] = 1 then
34                 if child1.isSuperset(p) then
35                     return true
36                 end
37             end
38             return child0.isSuperset(p)
39         end
40     end
41 end
```

Algorithm 5: BitPatternTreeNode class contd.

```
39 class BitPatternTreeNode
40 begin
   |
41 |  void addPathway(Pathway p, boolean [ ] bitsUsed)
42 |  begin
43 |  |  if leaf then
44 |  |  |  pathways.add(p)
45 |  |  |  if pathways.size > MAX_PATHWAYS then
46 |  |  |  |  split (bitsUsed)
47 |  |  |  end
48 |  |  |  label ← label and p.reactions
49 |  |  end
50 |  |  else
51 |  |  |  bitsUsed[splitBit ] ← true
52 |  |  |  if p.reactions[splitBit ] = 1 then
53 |  |  |  |  child1.pathways.add(p, bitsUsed)
54 |  |  |  end
55 |  |  |  else
56 |  |  |  |  child0.pathways.add(p, bitsUsed)
57 |  |  |  end
58 |  |  |  label ← label and p.reactions
59 |  |  end
60 |  end
   |
61 |  void split (boolean [ ] bitsUsed)
62 |  begin
63 |  |  leaf ← false
64 |  |  child0 ← new BitPatternTreeNode ()
65 |  |  child1 ← new BitPatternTreeNode ()
66 |  |  splitBit ← Unused bit with min difference of 1 and 0 count in pathways
67 |  |  foreach Pathway p in pathways do
68 |  |  |  if p.reactions[splitBit ] = 1 then
69 |  |  |  |  child1.pathways.add(p)
70 |  |  |  |  child1.label ← child1.label and p.reactions
71 |  |  |  end
72 |  |  |  else
73 |  |  |  |  child0.pathways.add(p)
74 |  |  |  |  child0.label ← child0.label and p.reactions
75 |  |  |  end
76 |  |  end
77 |  end
78 end
```

Algorithm 6: ReversibleTree class

Description: This class represents a tree which partitions on basis of reversible reaction pairs

```
1  class ReversibleTree
2  begin
3      ReversibleTreeNode root
4      ReversibleTree(Pathway [ ] pathways)
5      begin
6          bitsCannotBeUsed ← getBitsCannotBeUsedToSplit (pathways)
7          root ← new ReversibleTreeNode (pathways, 0, pathways.size, bitsCannotBeUsed, new
           ReactionBitData ())
8      end
9      ReactionBitData getBitsCannotBeUsedToSplit (Pathway [ ] pathways)
10     begin
11         ReactionBitData bitsCannotBeUsed ← new ReactionBitData ()
12         foreach reaction r in the network do
13             if r is not in any reversible reaction pair then
14                 bitsCannotBeUsed [r] ← true
15             end
16             else if r is not active in all pathways then
17                 bitsCannotBeUsed [r] ← true
18             end
19             else
20                 bitsCannotBeUsed [r] ← false
21             end
22         end
23         return bitsCannotBeUsed
24     end
25 end
```

Algorithm 7: ReversibleTreeNode class

Description: This class represents a node in **ReversibleTreeNode**

```
1  class ReversibleTreeNode
2  begin
3      Pathway [ ] pathways
4      boolean leaf
5      int start, end, splitBit
6      ReversibleTreeNode child0, child1
7      ReactionBitData activeReversibleReactions
8      ReactionBitData label
9      ReversibleTreeNode(Pathway [ ] p, int s, int e, ReactionBitData bitsCannotBeUsed,
       ReactionBitData a)
10     begin
11         pathways ← p
12         leaf ← true
13         start ← s
14         end ← e
15         activeReversibleReactions ← a
16         if a reversible reaction pair is active in activeReversibleReactions then
17             end ← start
18             return
19         end
20         splitBit ← Bit, which can be used, with min difference of 1 and 0 count in pathways
21         if splitBit < 0 then
22             label ← bitwise and of reactions in the pathways region [start,end)
23             return
24         end
25         activeReversibleReactions [splitBit ] ← true
26         bitsCannotBeUsed[splitBit ] ← true
27         leaf ← false
28         middle ← partition ()
29         child0 ← new ReversibleTreeNode (pathways, start, middle, bitsCannotBeUsed,
           activeReversibleReactions.clone())
30         child1 ← new ReversibleTreeNode (pathways, middle, end, bitsCannotBeUsed,
           activeReversibleReactions.clone())
31         label ← child0.label and child1.label;
32     end
33     int partition ()
34     begin
35         move all the pathways with inactive splitBit towards begining of pathways region [start,end)
36         return index of split point
37     end
38 end
```

Algorithm 8: Pathway class

Description: This class represents a pathway in the network

```
1  class Pathway
2  begin
3      double metCoeff [ ]
4      ReactionBitData reactions
5      Pathway parent1, parent2
6      Pathway(Pathway a, Pathway b)
7      begin
8          parent1 ← a
9          parent2 ← b
10         reactions ← parent1.reactions | parent2.reactions
11     end
12     void updateMetaboliteCoefficients(int m)
13     begin
14         metCoeff ← new double [UnprocessedMetaboliteCount]
15         double scalingFactor ← −parent1.metCoeff [m] / parent2.metCoeff [m]
16         for i ← 0 to (UnprocessedMetaboliteCount − 1) do
17             metCoeff [m] ← parent1.metCoeff [i] + scalingFactor ∗ parent2.metCoeff [i]
18         end
19     end
20     boolean isReactantMetabolite(int m)
21     begin
22         return metCoeff [m] < 0
23     end
24     boolean isProductMetabolite(int m)
25     begin
26         return metCoeff [m] > 0
27     end
28     boolean isSubsetOf(Pathway p)
29     begin
30         return (reactions = (reactions & p.reactions))
31     end
32     boolean isSupersetOf(Pathway p)
33     begin
34         return (p.reactions = (reactions & p.reactions))
35     end
36 end
```

Appendix B

Supplementary Data for

Ethanol Production in *E. coli*

Appendix C

Supplementary Data for

Succinate Production in *E. coli*