2011

ENUMERATING ALTERNATE OPTIMAL FLUX DISTRIBUTIONS FOR METABOLIC RECONSTRUCTIONS

Umaporn Siangphoe
Virginia Commonwealth University

Virginia Commonwealth University Life Sciences
Virginia Commonwealth University

This is to certify that the thesis prepared by Umaporn Siangphoe entitled ENUMERATING ALTERNATE OPTIMAL FLUX DISTRIBUTIONS FOR METABOLIC RECONSTRUCTIONS has been approved by her committee as satisfactory completion of the thesis requirement for the degree of Masters of Science.

Dr. James Paul Brooks, Department of Statistical Sciences and Operation Research

Dr. Stephen S. Fong, Chemical and Life Sciences Engineering, School of Engineering

Dr. Maria C. Rivera, Department of Biology, VCU Life Science

Dr. Gregory A. Buck, Director of Center for the Study of Biological Complexity

Dr. Thomas Huff, Dean of VCU Life Sciences

Dr. F. Douglas Boudinot, Dean of the Graduate School

August 17, 2011

ENUMERATING ALTERNATE OPTIMAL FLUX DISTRIBUTIONS FOR METABOLIC RECONSTRUCTIONS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Bioinformatics in Quantitative/Statistics Track at Virginia Commonwealth University.

by

UMAPORN SIANGPHOE,
Master of Science in Bioinformatics, VCU Life Science, Virginia Commonwealth University, August 2011

Director: Dr. James Paul Brooks
Assistant Professor, Department of Statistical Sciences and Operation Research

Virginia Commonwealth University
Richmond, Virginia
August, 2011

Acknowledgment

I would like to express my sincere gratitude and deep appreciation to my major advisor Dr. James Paul Brooks for giving me an opportunity to work on this project, including his guidance, invaluable advice, precious time, helpful and encouragement throughout which his help enabled me to complete this thesis.

I am also grateful to Dr. Stephen Fong and Dr.Maria Rivera, my co-advisors for their constructive comments, generous suggestions, and correction of this thesis.

Also, I would like to special thank to Dr.Herschell Emery for his advice in my study in this program and all professors who helped me to build and broaden my knowledge in bioinformatics.

Finally, I am always thankful to my mother, sister, nephews, relatives, and friends for their encouragement and support for my education.

TABLE OF CONTENTS

List of Tables

List of Tables

List of Figures

Abstract

ENUMERATING ALTERNATE OPTIMAL FLUX DISTRIBUTIONS FOR METABOLIC RECONSTRUCTIONS

By Umaporn Siangphoe, M.Sc.

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Bioinformatics in Quantitative/Statistics Track at Virginia Commonwealth University.

Virginia Commonwealth University, 2011.

Major Advisor: Dr. James Paul Brooks

Assistant Professor, Department of Statistical Sciences and Operation Research

Metabolites consumed and produced by microorganisms for mass and energy conservation may cause changes in a microorganism's environment. The microorganisms are unable to tolerate a particular environment for a long period. They may leave their old existence to find a new environment to sustain life. Essentially, organisms need to maintain their metabolic processes to survive in the new environment. Limitations of experimental studies to explore cell functions and regulations in detail result in insufficient information to explain processes of metabolic expressions under

environments of organisms. Consequently, mathematical modeling and computer simulations have been conducted to combine all possible cellular metabolic fluxes into single or multiple connected networks. Metabolic modeling based on linear programming (LP) subjected to constraints with an optimization approach is often applied metabolic reconstruction. The LP objective function is maximized to obtain an optimal value of biomass flux. Optimal solutions in LP problems can be used to explain how metabolites function in metabolic reactions. As an LP problem may have many optimal solutions, this study proposes a method for enumerating all alternate optimal solutions to evaluate important reactions of metabolic pathways in microorganisms. The algorithm for generating alternate optimal solutions is implemented in MetModelGUI, a Java-based software for creating and analyzing metabolic reconstructions. The algorithm is applied to models of five microorganisms: *Trypanosoma cruzi, Thermobifida fusca, Helicobacter pylori***,** *Cryptococcus neoformans* and *Clostridium thermocellum.* The results are analyzed using principal component analysis, and insight into the essential and non-essential pathways for each organism is derived.

CHAPTER 1 INTRODUCTION

In evolutionary theory, metabolites consumed and produced by microorganisms for mass and energy conservation may cause changes in a microorganism's environments. Internal and external effects related to the microorganisms and their environments may be another reason of these changes. They are unable to tolerate a particular environment for a long period of time. Similarly, when they are isolated from their natural environment, their living capabilities are rather restricted (1,2). A lack of energy resource is an example that the microorganisms deal with environmental changes to sustain life. This can be determined by higher survival rates of staying in appropriate environment than staying in a particular environment for a long time (2).

Generally, wild-type genes in a living cell are anticipated to perform their functions well and optimize their growth rates in a normal environment, whereas genes with mutations and evolutions may contribute to the internal and external environmental changes. Cellular functions for metabolisms regulated by those mutated genes are interrupted and affected by the environmental limitation. Essentially, organisms need to maintain their metabolisms to survive in the altered environment (3).

In the past decade, as high-throughput and biological technologies have greatly advanced, genomes studies have been increasingly developed. The developments generate new knowledge and provide systematic strategies to understand phenotypic

characteristics of microorganisms and cellular systems. Cell architectures and their products in the cell system were separately described based on their functions and locations rather than the entire genome system (4, 5). Therefore, it is essential to understand interrelationships between all of the cell elements and their related functions over the system. It is also important to connect the prior biocellular knowledge to the novel one and support claims made by scientific discoveries or advances of technologies. Since the potential benefits gained depend on prior knowledge, several researches involved cellular functions and metabolisms in both experimental and non-experimental studies have been widely published (3, 5).

Limitations of experimental studies to explore cell functions and regulations in details result in insufficient information to explain processes of metabolic expressions under environments of organisms (4, 6). Meanwhile, annotations of genome sequences are more available in the past decade and it becomes possible to reconstruct artificial biochemical networks (2). Consequently, mathematical modeling and computer simulations have been conducted to combine all possible cellular metabolic fluxes into single or multiple connected networks. Nowadays, there are several mathematical applications existing for analysis of metabolic fluxes. Mathematical theories used for supporting the applications are also developed and verified by experimental and observational studies. Most of the mathematical methods involved concentrations of substrates, products and co-factors related to the metabolic fluxes in the cellular pathways. Once the information of the mathematical models is known, the metabolic reconstruction can be built to predict the entire metabolic networks of the system. Additionally, we can analyze dynamic flux distributions, flux steady states and flux

variability by the mathematical models in order to interpret the whole functions and activities in the living cell (4, 5).

Mathematical models have been developed to describe characters of problems defined from several areas to be convenient for solving and analysis. The mathematical models play important roles in many aspects in biological sciences such as atomic models, genomic models, proteomic models, etc. They support us to decide and clarify situations of the problems. Such models are presented in terms of mathematical symbols and expressions. The models are related to quantifiable decisions called as an objective function, e.g. $Z = 3x_1 + 2x_2 + x_3$. The decisions are assigned as variables with restrict values to measure performances under given conditions, e.g. $5x_1 - 2x_2 + x_3 \leq 10$. The mathematical expressions for the restrictions are known as constraints of the models. Constraints and coefficients in the objective functions are determined as parameters of the models. Thus, in order to accomplish mathematical models with genetic and molecular functions, the restricted values of the decision variables are investigated to optimize the objective functions, conditional upon specific constraints. By the investigation, small variations of the parameters have possibly occurred. We propose the better the models, the better the functions are clarified (7, 8, 9).

To accomplish applications of mathematical models in various decision problems, theories of linear models are developed. An approach of the linear models to obtain optimal solutions in such problems is linear programming (LP). LP originated from mathematical programming or mathematical modeling paradigm, where variables represent quantifiable decision. Linear programming is also called linear objective function or linear inequalities-equalities for constraints. LP is typically applied in

operations research as a technique to evaluate problems of concern, to derive solutions of the problems, and to obtain optimal results. The optimization refers to both maximize and minimize of objective functions. The operations research involves conducting and coordinating operations. The operations research has been extensively applied in business and public service fields. However, they resemble the operation in conducting scientific research (7, 8, 9).

Although, not all decisions of problems are represented by a linear programming method, a number of important components in the problems are approximated by linear functions. Compared to non-linear programming, the linear program is less complicated as their linear functions consist of a single variable in each component and they have powers of 1 for each variable. The objective functions based on linear programming are more likely to accomplish the problem solving than those based on complex models or non-linear programming. They facilitate the overall structures of the problems making it more comprehensible and enable investigation into the interrelationships of the entire problem. With the goal of maximization of objective functions, linear programming is currently used to provide suitable resulting decisions on the problems of concern worthwhile (7, 9).

In general, cellular metabolic networks and metabolic capabilities in particular conditions can be explained by modeling and simulations in biological analysis. A study defined the modeling and simulations in two approaches: dynamic and static approaches (10). The dynamic approach is composed of kinetic equations, their parameters, and reaction rate conditions. The static approach consists of stoichiometric reactions in metabolic networks, used in large-scale metabolic analysis to present particular states

under genotypic and environmental conditions in organisms. These two approaches play a vital role in reconstructing genome-scale metabolic networks. The reconstruction becomes available when using gene annotation to explain the stoichiometric reactions of cellular metabolisms, setting linear mass balance equations for metabolites, completing the networks by the literature and experiment data, and then validating the results of modeling and simulations by comparing results from real experiments (5, 11).

Several algorithms for metabolic network reconstructions have substantially developed based on purposes in identifications targets and cell functions. A summary of algorithms with their corresponding mathematical methods for metabolic simulations can be obtained in Park JM et al. (2009) (11). In this study, we present a genome-scale network reconstruction based on flux balance analysis with constraint-based modeling.

1.1. Flux balance analysis

Flux balance analysis (FBA) is an approach based on linear optimization principles. It has been used to determine metabolic characteristics of genotypic and metabolic conditions with their environments. It also has been applied to predict metabolic flux distributions after genes, pathways, or regulatory circuits in microbial cells have been modified. Typically, metabolic fluxes take a few minutes to adapt themselves to altered environments. Thus, complex regulatory features of the dynamic fluxes are required to complete descriptions of the adaptations. This can occur when metabolic flux balances are assumed. As the nature of metabolic systems has limited

information for reconstruction, the FBA under given constraints has been developed to analyze the metabolic pathways in a steady state and to reduce flux solution spaces in real organisms. The flux balance analysis is particularly appropriate to study the complex system in cell metabolism. The FBA can be compared to genome experiments in order to verify the complex information and improve accuracy of appropriate constraints used under specific and investigated circumstances (3, 4, 6).

Edwards SJ and et. al, (2002) suggested six basic steps to implement flux-balance analysis in studying activities of elements in a cell system. First, a metabolic network that will represent cellular activities is reconstructed. In the metabolic network, metabolites or components along with their internal and external fluxes are drawn as a diagram with a system bound to understand the relationships of those elements. Second, balance equations, serving as dynamic mass balance of each component or metabolite and containing internal and external fluxes, such as nutrient intake and uptake rates of substrate and product concentration in biosynthesis, are built. Given, S is a stoichiometric matrix containing coefficients of flux balance equations and v is a vector of internal and external metabolic reaction rates. $S * v$ obtains a total net of metabolic consumptions and productions in the cell. In other word, linear programming enables dimension reduction of flux balance equations. Different organisms required different biosynthesis and different metabolic maintenances. The requirements can be obtained from experimental studies and previous knowledge of the organisms. Third, steady state mass balances are assumed to continue flux-balance analysis when information in some kinetic reactions is limited. Then, physicochemical constraints involving the internal and external fluxes are

conditioned using the circumstance information and certain factors related to the systems. Thus, the internal fluxes, such as rates of changes or growth rates are limited because of the steady state and the constraints gained. Due to the internal fluxes unknown and the number of possible fluxes is greater than the number of metabolites in the metabolic system; therefore, an objective function containing unknown parameters of the internal fluxes is defined to solve the problems of balance equations. Linear programming subjected to the constraints with an optimization approach is created to estimate maximum rates of change or other similar rates, such as maximum growth rates, minimum waste products, and maximum product formations. Each step provides scopes of information in which the possible solution spaces are narrower and more convenient to understand how the metabolites function. In addition to finding the set of optimized values, alternate methods are also considered to interpret all possible activities over the complex cell system. The altered methods may be to add or remove the metabolic reactions, change the internal and external flux parameters, and define different constraints and objective functions (3). Some strategies with utilizing the linear programming have been developed to create all possible alternative optimizations e.g. a mixed-integer linear programming (MILP), CoI-based weighted fluxes (CoI: Coefficient of Importance), quadratic programming (QP). The altered methods generate some hypothesis which can be verified using computer simulation (3, 4, 12, 13, 14).

Although, flux-balance analysis is applied in several aspects in physical and biological sciences, there are some limitations caused by gene mutations and evolutions. When genes are mutated, their functions are possibly changed. Their metabolites and other proteins may be decreased. The dynamic fluxes of those genes in the system are

ceased intermediately. The FBA may fail due to the absent and nonfunctional genes. For instance, E.coli MG1655 *in silico* was mutated by gene deletions. The metabolic network of the E.coli mutant was examined using FBA. Given the reason of gene mutation, the FBA failed to analyze in the part of the network in which toxic metabolic intermediates appeared. Apart from the mutation, the FBA may be restricted due to unknown or improper constraints or algorithms for conduction of flux balance systems and insufficient information for completion of metabolic pathways. These cannot improve accuracy of *in silico* experiments and lead to misinterpretation of flux results. (1, 3, 4).

1.2. Constrained-based modeling

Constraints are applied in genome-scale models to control limitation of cell function and consequences of cell behaviors (5). Constraints are used to allow possibilities of steady states in metabolic flux balance analysis so that we can obtain substantial information in biological reactions which lead to defining cell functions. The constrains-based approaches provide biochemically and genetically consistent frameworks to generate a hypothesis for testing microbial cell functions. They are often used in metabolic engineering experiments in order to predict metabolic capabilities in real an organism, which cannot be studied in reality (1). They are also created flux balance equations or basic functions for *in silico* analysis of microorganisms (2).

In silico is a computer representation of cellular metabolic simulation constituents, their interactions and their integrated functions as a whole. This phase was

coined in 1989 as an analogous word to *in vivo*. The *in silico* can be inferred as an experiment study of living organisms that takes place outside organisms. This analysis method enables stimulations of microbial growths and behaviors. The *in silico* representation can be formed as an *in silico* organism using metabolic network reconstructions based on genomic data as a backbone. Flux balance analysis and refined information of genotypic and phenotypic characteristics of genome on genome-metabolic network reconstructions are easily to study when *in silico* modeling and analysis of microbial metabolism in organisms were developed. (15, 16, 17)

Constraint-based modeling and simulations was often implemented using optimization techniques with various constraints to improve simulation results (1). Structures of constraint-based modeling are important to mention in these six parts: general structure, type of objective function, type of constraints, selection of algorithms, flux solution spaces, model validation and simulations.

1.2.1. General structure of constraint-based models is composed of variables, constraints and objectives.

Variable is a changeable value with respect to the given information under different conditions.

Decision variable or control variable is a value under the control of decision maker.

Constraints are the conditions that must be satisfied during the simulation to obtain an optimal solution. The constraints are classified into equality and inequality constraints. This is an example of constraints.

$$\alpha \cdot y_i \le a_i \le \beta \cdot y_i \qquad y_i \in \{0,1\} \tag{1.1}$$

α and β are constraints that indicate the upper and lower limits

a_i is a continuous variable

y_i is a discrete variable having a binary value of 0 and 1

Objectives are described by mathematical functions composed of decision variables, representing the purpose of decision marker. The equation of objective function is presented as follows:

$$\text{Maximize/Minimize } f(x) = \left(b_{r,1}x_1 + b_{r,2}x_2 + ...+ b_{r,n}x_n\right)^k \text{ , for all r} \tag{1.2}$$

$f(x)$ is the form of objective presents types of simulation.

Number of objectives $f(x)$ determines whether the system can have a single or multiple objectives.

b and *k* are constants.

k determines whether the system are linear or non-linear

Steps for solving optimization problems are the following:

1. Determine the decision variables
2. Formulate all proper objectives
3. Formulate constraints
4. Maximize/minimize the objective functions

In genome-scale metabolic models, substrates and products are converted to each other according to the stoichiometric reactions in their metabolic network. For metabolic flux balance, difference between rate of consumption and production for a specific metabolite equals to the change of metabolite concentration over a period of time. The equation of metabolic flux balance can be presented by

$$\frac{dX_i}{dt} = s_{ij}.v_j \qquad \text{where } \alpha_j \le v_j \le \beta_j \tag{1.3}$$

X denotes concentration of metabolite

i and *j* denote the indices of metabolites and reaction

S denotes the stoichiometric $m \times n$ metrix

m is number of metabolites

n is number of reactions

v is a vector representing the fluxes of reactions that consume and produce the metabolites

If concentration X is a substrate of the reaction, the stoichiometric coefficients will be negative values. *v* is subjected to upper and lower bound constraints presented by

α and β. When assuming a steady state reaction, the time derivative can be eliminated. Then the equation of metabolic flux balance results in a system of linear equations as follow:

$$s_{ij} . v_j = 0 \qquad \text{where } \alpha_j \le v_j \le \beta_j \tag{1.4}$$

1.2.2. Type of objective function

The form of objective functions or the method for solving the optimization problems is diverse with respect to metabolic purposes. The metabolic purpose may be maximizing growth rate, maximizing by-product formation, or maximizing ATP and reducing power. The type of objective function can be linear programming, quadratic programming, mixed integer linear programming, or others (1).

Single and multiple objective functions in constraint-based model were determined by number of objective functions $f(x)$ in the metabolic system. The single objective function is often used for maximizing growth rate and the multiple objective functions are applied for many features of biochemical formations in the metabolic system. The multiple objective functions are more useful for implementing these cases in ecological system: cell-to-cell interactions or two different organisms with two *in silico* models (1).

Linear programming (LP) is applied to solve the optimization problems in flux balance analysis under a pseudo-steady state as equation (4) including flux variability analysis (FVA), flux coupling analysis (FCA), and flux sensitivity analysis (FSA). The

FVA performs testing of maximize and minimize fluxes and comparing difference of flux solution spaces of each reaction. The FCA evaluates relationships of pairwise combinations in the network. Also, the FSA compares changes of objective functions to the changes in other fluxes (1).

For other desired purposes, minimization of metabolic adjustment (MOMA) and regulatory on/off minimization (ROOM) were used to examine physiological characteristics of organisms under gene knockout conditions. MOMA uses the same constraints with FBA but defines flux distributions with quadratic programming (QP). MOMA has a purpose to find a flux distribution that is unique and similar to the wild-type flux distribution. ROOM uses mixed integer linear programming (MILP) to minimize number of significant flux changes. Additionally, alternative pathways may take place when cell metabolisms were changed by gene deletions or mutations. The conventional methods, restricted to essential genes and reactions, need to be modified to support their changes. The summation of incoming and outgoing fluxes (flux-sum) at around particular metabolite was applied to evaluate in such case. In addition, if organisms have complex regulatory systems then steady-state regulatory flux balance analysis (SR-FBA) with MILP were applied to take into account regulatory mechanisms of a binary condition of genes, proteins and reactions. If flux balance analysis were performed based on thermodynamics then thermodynamics-based metabolic flux analysis (TMFA) was considered. In intracellular cytoplasm, enzymes catalyzing in a particular reaction were compared with other enzymes in a limited space. FBA with molecular crowding (FBAwMC) was applied to predict rate of each reaction in this case, which provide the results correlated to the data that was evaluated by experiments.

Furthermore, a bilevel optimization algorithm (OptKnock) allows for identifying gene knockout and overproduction of metabolites. OptReg is another multiple objective function, derived from the steady-state fluxes with upward and downward departures, and linear programming with several constraints. The OptReg optimizes a framework to examine multiple activations/inhibitions and drop-out of target candidates. OptGene, an extension of OptKnock, uses genetic algorithms and applies genetic modification to *in silico* genome-scale model. Lastly, an optimal metabolic network identification (OMINI) produces flux distribution and explores minimization of discrepancies between experimental and *in silico* data. Table 1 presents objective functions for particular algorithms used for analysis in genome-scale network reconstructions (1).

Table 1. Frameworks of algorithms and description of objective functions

Algorithm	Objective function	Solver	Descriptions
FBA	max/min v_j	LP	Usually maximizing the growth rate
MOMA	min $\sqrt{\sum_{j=1}^{M}(w_j - x_j)^2}$	QP	Minimizing the Euclidian distance from a wild type flux distribution under knockout condition
ROOM	min y_j	MILP	Minimizing the number of significant flux changes from a wild type flux distribution under knockout condition
OptKnock	max $v_{biochemical}$	MILP	Bilevel optimization framework: biomass, biochemical
OptReg	max $v_{biochemical}$	MILP	Determining the activation/inhibition and elimination
OptStrain	Step1: max $MW_i \sum_{j=1}^{M} s_{ij} v_j$, $i = p$ Step2: min $\sum_{j \in M_{non-naive}} y_j$	LP, MILP	Determining the maximum yield of the desired biochemical and minimizing the number of non-native reactions needed to meet the maximum yield of desired biochemical production
OMNI	Min $\sum_j we_j \left\| v_j^{opt} - v_j^{\exp} \right\|$	MILP	Identifying the reaction set that leads the best agreement between prediction and experiment

Note: The symbols for explaining each algorithm. i: index of metabolites, j: index of reactions, d: index of deleted reaction, N: the set of metabolites, M: the set of reactions, R: the set of substrates, P: the set of desired biochemical, A: the set of deleted reactions, E: the set of experimentally measured reactions, S: $m \times n$ stoichiometric matrix (m: number of metabolites; n: number of reactions), v: fluxes of reactions, $v_{biochemical}$: fluxes of biochemical production, v^{opt} and v^{ecp}: optimal and experimentally measured fluxes, MW : molecular weight of metabolites, we weight for measured fluxes.

1.2.3. Type of constraints

Types of constraints are different based on their cell functions. *Physico-chemical* constraints are constraints acting on mass, charge and energy conservations, such as biochemical reactions or local concentration rates inside cell. These constraints could not be destroyed from the cell systems. *Topobiological* constraints are constraints affecting both forms and functions. The topobiological constraints works depend on size, movement and interactions of cells, e.g. DNA physical arrangement or molecular distribution. *Environment* constraints are time and condition environmental constraints. The examples of the environmental constraints are nutrient availability, pH, or osmolarity. The last type of the constraint is the *regulatory* constraint. This constraint is self-imposed and works under evolutionary changes, such as gene expression or enzyme inhibitors (2, 5).

Apart from biological types of constraints, constraints can be defined in terms of mathematical forms for two basic types: *balance* and *bound* constraints. Balances are

constraints related to conserved quantities, such as mass, energy, momentum, electroneutrality, or osmotic pressure. Bounds are the greatest possible degree of numeric ranges of mathematical variables and parameters. The examples of bound constraints are concentrations, fluxes, or kinetic constants. In a genome-reconstruction model, balance constraints are applied in a steady state of fluxes in a metabolic network, no accumulation or metabolite quantities, while bounds are upper and lower limits of enzyme reactions, concentration values, collision frequency, or transmembrane potentials in the individual fluxes. The balances and bounds together are allowable to describe possible solution spaces of fluxes in a reconstructed network. Genome-scales network reconstructions with constrain-based approach were typically crated for a particular organism, which allow studying capabilities and phenotypic characteristics of organisms (2, 5).

Algorithms of constraint-based modeling and simulations are important for reconstructing genome-scale metabolic models. Typically, flux solution spaces by *in silico* simulations are too wide, while the flux solution spaces by experiment data are rather small, such as fluxes of cellular regulation, robustness, and homeostasis. Suitable constraints can provide flux solution spaces closer to the flux solution spaces in real organisms. Thus, appropriate constraints or relative constraints enable to reduce also solution spaces of simulation (1, 11).

1.2.4. Selection of algorithms of constraint-based modeling

Constraint-based model conductions are depended on study objectives and given conditions. Algorithms for the conductions are also selected based on these reasons. Incorrect algorithms may cause model errorless and misconception (1). Park JM et al (2009) provides a flowchart to select algorithms and objective functions using the study purposes and model conditions (figure 1).

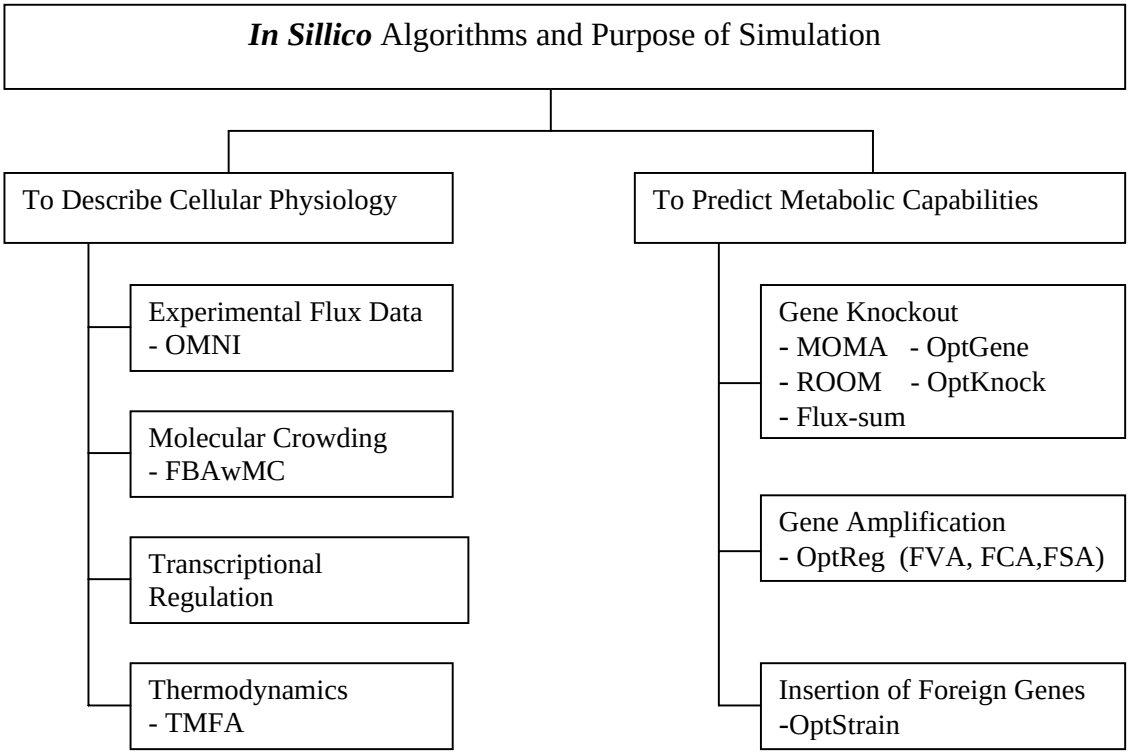

Figure 1. A flowchart to select algorithms by purposes of simulation and given conditions

The guideline to find appropriate algorithms was started by choosing a study purpose to describe cellular physiology or predict metabolic capabilities after genetic perturbations. In real organisms, several algorithms and constraints can be concurrently

selected such as regulation, robustness, and homeostasis as they were used to support in describing the cellular physiology simultaneously. In metabolic engineering studies, some algorithms, such as gene knockout, gene amplification, gene expression, or conduction of foreign genes are concurrently considered to find knockout candidates and describe metabolic properties. In drug discoverers, flux-sum is applied for metabolic modeling of new drug developments on gene targets in the conditions of gene perturbation and gene knockout. Other algorithms with different objectives were applied as provided in the guideline (Figure 1) (1).

1.2.5. Flux solution spaces

Flux solution spaces represent all possible states in metabolic networks in constraint-based modeling under genetic and environmental conditions. Flux solution space calculates maximum rate of biochemical and by-product productions under a specific growth rate and other objective rates. Given information by the flux solution spaces are also used to design metabolic engineering experiments and examine existences of engineering organisms whether they have satisfied design criteria. Overall, the flux solution spaces provide useful information to determine metabolic capabilities. Moreover, the flux solution spaces can become more realistic by adding some constraints, adding new metabolic reactions for expanding metabolic aspects in the network or removing some interrupted reactions for increasing correctness of predictions (1).

1.2.6. Validation of constraint-based modeling

The purposes of genome-scale metabolic network reconstruction are to conduct metabolic network systems consisting of sophisticated information of all elements and functional activities in the cellular systems and to make more realistic predictions of metabolic characteristics and their activities. One way to accomplish the purposes are to validate successful results by the reconstructed networks to real results by experimental studies for an acceptance of numbers to confirm whether the metabolic modeling are appropriate under given constraints (1).

There are two possible ways for validation of genome-scale metabolic network reconstructions. First, we can measure numbers of true positive (TP), true negative (TN), false positive (FP), or false negative (FN) results, where one assumes an experiment is a standard method and a genome metabolic reconstruction is an alternative method. The TF determines the number of results found in both experiment and simulation and the TN determines the number of no results in both sources. In contrast, the FP indicates results found in the simulation, but not in the experiment and the FN shows results found in the experiment, but not in the simulation. Both FP and FN indicates false predictions and inconsistency of the constraint-based modeling and genome-scale metabolic network reconstructions. The false predictions are also considered for the reconstructions with each condition or overall conditions of a full model (15). Alternative method for the validations is using a literature review. With limitations of experiments, several reconstructed genome-scale studies uses direct physical evidences from the literature review to ensure their discovered knowledge in their studies (1, 16, 17).

1.3. Linear programming

1.3.1. History of linear programming

Linear programming is a short name of “Linear programming problems” or “LP problems”, which is a field of applied mathematics concerned with problems. For example, diet problems are required for satisfying between energy and sources of nutrients involving other factors, such as serving size and price per serving. This problem can be stated by linear programming with requirements of equalities or inequalities between the satisfactions in order to get all energies needed in everyday life (7).

Linear programming was known by G. B. Dantzig in 1947. He designed a simplex method to solve the linear programming formulation of U.S. Air Force planning problems. However, the field of linear programming was studied as early as 1947 for the duality theorem of linear programming; a restricted class of LP; the system of linear inequalities, investigated by Fourier in 1826; and rudimentary algorithms for their solutions in 1939. The studies were about an essence of mathematical theories. The linear programming was also applied in other unrelated problems of production managements. In economic field, the LP was applied in the renowned system by L. Walras in 1847. At the same time, T.C. Koopmans recommended that the LP was an effective framework in analysis of classical economic theories. The LP led to pure mathematical theories, such as geometry of convex sets and theory of two-person games. Afterwards, it became popular in several areas and was increasingly considered as an efficient way for the entire operational system.

At important periods, the linear programming was well-known when L.V Kantorovich and T.C. Koopmans received the Novel Prize award in economic field for their contributions to the theory of optimum allocation of resources by the Royal Sweden Academy of Sciences in 1975. However, in academic areas, G. B. Dantzig was universally recognized to be the father of linear programming as he provided the first-known contribution. The second period, mathematicians made an attempt to create a solution to solve linear programming. The simplex method was finally discovered as an efficient algorithm, satisfied in both theoretical and practical. Nowadays, modern computation technologies have made linear programming easier in applying in real work. Problems could be remarkably presented by linear programming and solved by the simplex method with information based on experience and intuition (7, 20).

1.3.2. Theory of linear programming

Definition

A linear programming problem is the problem of maximizing (or minimizing) a linear function subject to a finite number of linear constraints.

Examples

Example 1: Maximize $5x_1 + 4x_2 + 3X_3$

Subject to $2x_1 + 3x_2 + x_3 \leq 5$

$4x_1 + x_2 + 2x_3 \leq 11$ (1.5)

$3x_1 + 4x_2 + 2x_3 \leq 8$

$$x_1, x_2, x_3 \geq 0$$

Example 2: Minimize $3x_1 - x_2$

Subject to $-x_1 + 6x_2 - x_3 + x_4 \leq 5$

$$7x_2 + \quad +2x_4 \leq 11 \qquad (1.6)$$

$$x_1 + x_2 + x_3 \quad = 5$$

$$x_3 + x_4 \leq 2$$

$$x_2, x_3 \geq 0$$

Linear function

If $c_1, c_2, ..., c_n$ are real numbers, then the function of f of real variables $x_1, x_2, ..., x_n$ defined by

$$f(x_1, x_2, ..., x_n) = c_1x_1 + c_2x_2 + ... + c_nx_n = \sum_{j=1}^{n} c_j x_j \qquad (1.7)$$

This function is called a **linear function**.

Linear equation

If f is a linear function and if b is a real number, then the equation is called a **linear equation**

$$f(x_1, x_2, ..., x_n) = b$$

And these equations are called **linear inequalities**

$$f(x_1, x_2, ..., x_n) \leq b$$

$$f(x_1, x_2, ..., x_n) \geq b$$

Linear constraints

Linear equations and linear inequalities are both referred to as **linear constraints.**

Standard form

Several authors call the standard form of linear programming as **canonical** or **symmetric** forms. These are the examples of standard forms of linear programming.

$$\text{Maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1,\ 2,\ ...,\ m) \qquad (1.8)$$

$$x_j \geq 0 \qquad (j = 1,\ 2,\ ...,\ n)$$

Where i are different subscripts to different constraints and j are different subscripts to different variables

The standard form is different from other LP forms, such as the example in (1.5) and (1.6) that all of their constraints are linear inequalities and the last n of $m+n$ constraints (1.8) are very special. In other words, none of the n variables are assumed negative values and such constraints are called *nonnegativity constraints*. Obviously, the

constraints in 1.6 are two linear equations and the variable x_1, x_4 may assume negative values.

Objective function

The objective function is the linear function that is to be maximized or minimized in an LP problem. For example, the function z of variables $x_1, x_2, x_3, x_4, x_5, x_6$ is the objective function, defined by

$$z(x_1, x_2, \ ..., \ x_6) = 3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6$$

Feasible solution

Number $x_1, x_2, \ ..., \ x_n$ that satisfy all the constraints of an LP problem are set to a feasible solution. Some LP problems may have many feasible solutions in some senses. Some may not have any feasible solutions at all. The latter LP problems are call **infeasible**

Optimal solution

An optimal solution is a feasible solution that maximizes or minimizes the objective function. For the maximizing or minimizing, it depends on the form of the problem. The corresponding values in the objective function maximized are called the **optimal values** of the problem.

For example, the unique optimal solution of a LP problem is

$x_1 = c_1,\ x_2 = c_2,\ x_3 = c_3,\ x_4 = c_4,\ x_5 = c_5,\ x_6 = c_6$; where c_i are any constants, or simply $(c_1,\ c_2,\ c_3,\ c_4,\ c_5,\ c_6)$

There are three possibilities of optimal solutions for a LP problem

1. Unique optimal solution
2. Many different optimal solutions
3. No optimal solutions.

The third possibility may have resulted from non-feasible solutions or too many different feasible solutions of the LP problem (or unbounded solution). The LP problem with the unbounded solution has many feasible solutions but none of them is the optimal solution or the best solution. These are the examples of the LP problem without optimal solutions.

The LP problem with *infeasible* solutions:

$$\begin{aligned} \text{Maximize} \quad & 3x_1 - x_2 \\ \text{Subject to} \quad & x_1 + x_2 \le 2 \\ & -2x_1 - 2x_2 \le -10 \\ & x_1,\ x_2 \ge 0 \end{aligned} \tag{1.9}$$

The LP problem with *unbounded* solutions:

$$\begin{aligned} \text{Maximize} \quad & x_1 - x_2 \\ \text{Subject to} \quad & -2x_1 + x_2 \le 1 \end{aligned}$$

$$-x_1 - 2x_2 \leq -2 \qquad (1.10)$$

$$x_1,\ x_2 \geq 0$$

In summary, there are three categories of linear programming problems.

1. The LP problem has an optimal solution or many solutions

2. The LP problem has infeasible

3. The LP problem is unbounded

1.4. Simplex method

1.4.1. Examples of using simplex method in LP problem

The simplex method for solving the LP problem can be described as

1. A linear programming problem can be stated by these following.

Maximize $5x_1 + 4x_2 + 3x_3$

Subject to

$$\begin{aligned} 2x_1 + 3x_2 + x_3 &\leq 5 \\ 4x_1 + x_2 + 2x_3 &\leq 11 \\ 3x_1 + 4x_2 + 2x_3 &\leq 8 \\ x_1,\ x_2,\ x_3 &\geq 0 \end{aligned} \qquad (1.11)$$

2. Slack variables which are analogous to each constraint and the z function are defined.

Every feasible solution $x_1,\ x_2,\ x_3$, the value of the left-hand side is at most value of the

values of the right-hand side for each constraint. The slack variables and z function are as follows:

$$\begin{aligned} x_4 &= 5 - 2x_1 - 3x_2 - x_3 \\ x_5 &= 11 - 4x_1 - x_2 - 2x_3 \\ x_6 &= 8 - 3x_1 - 4x_2 - 2x_3 \\ z &= 5x_1 + 4x_2 + 3x_3 \end{aligned} \tag{1.12}$$

$x_1,\ x_2,\ x_3$ are called decision variables, while $x_4,\ x_5,\ x_6$ are called as slack variables

The LP problem can be restated as

minimize z subject to $x_1,\ x_2,\ x_3\ , x_4,\ x_5,\ x_6 \geq 0.$ (1.13)

The relationships among (1.11), (1.12), and (1.13) can be described in that.

- Every feasible solution $x_1,\ x_2,\ x_3$ of (1.11) can be extended into a feasible solution $x_1,\ x_2,\ ...,\ x_6$ of (1.13).
- Every feasible solution $x_1,\ x_2,\ ...,\ x_6$ of (1.13) can be restricted into a feasible solution $x_1,\ x_2,\ x_3$ of (1.11).
- The feasible solutions of (1.11) and (1.13) carries optimal solutions of (1.11) onto the optimal solutions of (1.13) and vice versa.

3. The core of the simplex method is the successive improvement having found some feasible solution $x_1,\ x_2,\ ...,\ x_6$ of (1.13). We should find the feasible solution of

$\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_6$ and repeat the process a finite number of times. We can find the optimal solution eventually. The feasible solution of $\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_6$ can be stated

$$5\bar{x}_1 + 4\bar{x}_2 + 3\bar{x}_3 > \ 5x_1 + 4x_2 + 3x_3$$

4. According to the core of simplex method, we find some feasible solution by setting the decision variables x_1, x_2, x_3 at zero and evaluate the slack variable x_4, x_5, x_6

$$x_1 = 0,\ x_2 = 0,\ x_3 = 0,\ x_4 = 5,\ x_5 = 11,\ x_6 = 8 \qquad (1.14)$$

yields $z = 0$

5. Find the next solution to obtain a higher value of z. We keep $x_2 = x_3 = 0$, increase x_1 to obtain z and each result of slack variables x_4, x_5, x_6

$x_1 = 1,\ x_2 = 0,\ x_3 = 0,\ x_4 = 3,\ x_5 = 7,\ x_6 = 5$, which obtain $z = 5$

$x_1 = 2,\ x_2 = 0,\ x_3 = 0,\ x_4 = 1,\ x_5 = 3,\ x_6 = 2$, which obtain $z = 10$

$x_1 = 3,\ x_2 = 0,\ x_3 = 0,\ x_4 = -1,\ x_5 = -1,\ x_6 = -1$, which obtain $z = 15$

However, we cannot increase $x_1 = 3$ because it requires that $x_i \geq 0$. Thus, the strategies are that increasing x_i up to the bound, keeping $x_2 = x_3 = 0$, still maintain feasibility that $x_4,\ x_5,\ x_6 \geq 0$

6. We evaluate the conditions or constraints to find bounds of x_1. Given these condition,

$$x_4 = \quad 5 - 2x_1 - 3x_2 - \quad x_3 \ \geq \ 0 \text{ implies } x_1 \ \leq \frac{5}{2};$$

$$x_5 \geq 0 \text{ implies } x_1 \ \leq \frac{11}{4}$$

$$x_6 \geq 0 \text{ implies } x_1 \ \leq \frac{8}{3}$$

Thus, the next solution is

$$x_1 = \frac{5}{2}, \ x_2 = 0, \ x_3 = 0, \ x_4 = 0, \ x_5 = 1, \ x_6 = \frac{1}{2} \text{ for } z = \frac{25}{2} \qquad (1.15)$$

In this case, z is increased from the maximum of z we have evaluated so far.

7. As we still do not know that $z = \frac{25}{2}$ is the highest value of z, we should continue in a similar way to look for a feasible solution to better the feasible solution in the above section. Thus, we should conduct a new system of linear equations that relates to (1.15), such as the system (1.12) relates to (1.14).

The new system should present the variables that assume positive values (x_1 , x_5 , x_6) in (1.15) in terms of the variables that equal zero (x_2 , x_3 , x_4) in (1.15). Notice that the variable x_1 was changed from zero in (1.14) to positive values in (1.15). This implies that for the new system, we can express x_1 in the left-hand side and x_2 , x_3 , and x_4 in the right-hand side. We still evaluate the same constraint in (1.12). However, to express

x_5 , x_6 in terms of x_2 , x_3 , x_4 , we also need to replace x_1 by x_2 , x_3 , x_4 in the second and third constraints in (1.12). Thus, the new system is

$$\begin{aligned} x_1 &= \frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4 \\ x_5 &= 1 + 5x_2 + 2x_4 \\ x_6 &= \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{3}{2}x_4 \\ z &= \frac{25}{2} - \frac{7}{2}x_2 + \frac{1}{2}x_3 - \frac{5}{2}x_4 \end{aligned} \qquad (1.16)$$

We find the next solution to obtain a higher value of z . In the new system, the values of z can be increased only by the increment of x_3. If we keep $x_2 = x_4 = 0$, then increase x_3 by considering the first, second, and third constraints together. It implies that $x_3 \leq 5$ for $x_1 \geq 0$; no x_3 values for $x_5 \geq 0$; $x_3 \leq 1$ for $x_6 \geq 0$. Among the possible ranges of x_3 , the highest value of x_3 should be 1. By the new system, when keeping $x_2 = x_4 = 0$, the results of other variables will be $x_3 = 1$, $x_5 = 1$, $x_6 = 0$. Therefore, the new feasible solution is

$$x_1 = 2,\ x_2 = 0,\ x_3 = 1,\ x_4 = 0,\ x_5 = 1,\ x_6 = 0 \text{ , obtain } z = 13 \qquad (1.17)$$

The new system improves the z value from 12.5 to 13

8) The new solution (1.17) can obtain the highest value of z. However, this solution should present along with their system of linear equations. To construct the new system,

we use the information in (1.17). x_2 , x_4 , x_6 are the zero variables, while x_1 , x_3 , x_5 are the positive-value variables. The zero variables should be in the right-hand side and the positive-value variable should be on the left-hand side in the new system of linear equation. As the number of x_3 variables in the linear equations in (1.16) is smaller than the other variables, we start with the third constraints and substitute for x_3 in the first and second constraints. Therefore, the new system of linear equation is presented as follows:

$$\begin{aligned} x_3 &= 1 + x_2 + 3x_4 - 2x_6 \\ x_1 &= 2 - 2x_2 - 2x_4 + x_6 \\ x_5 &= 1 + 5x_2 + 2x_4 \\ z &= 13 - 3x_2 - x_4 - x_6 \end{aligned} \qquad (1.18)$$

9) After we have the new system of linear equations, we should do iteration to increase the value of z as we did in (1.12) and (1.16). However, there are no values that can increase z. If we increase any variables of x_2 , x_4 , x_6 , it will decrease the value of z instated. Also, by the constraints, the values of x_2 , x_4 , x_6 cannot be negative values. It seems that we are able to obtain the highest value of z ($z = 13$). The solution (1.17) is the optimal solution among all feasible solutions which satisfied the inequality $z \leq 13$.

1.4.2. Dictionary for solving linear programming

1. Given a LP problem in general

$$\text{Maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1,\ 2,\ ...,\ m) \tag{1.19}$$

$$x_j \geq 0 \qquad (j = 1,\ 2,\ ...,\ n)$$

2. We first introduced the slack variables $x_{n+1},\ x_{n+2},\ ...,\ x_{n+m}$ and denote the objective function by z. The standard form in (1.19) is defined as follow:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad (i = 1,\ 2,\ ...,\ m)$$

$$z = \sum_{j=1}^{n} c_j x_j \tag{1.20}$$

3. To be convenient, we can associate a system of linear equations with each of the feasible solutions to find the improved solutions. In each iteration, the simplex method moves from some feasible solution $x_1,\ x_2,\ ...,\ x_{n+m}$ to another feasible solution $\bar{x}_1,\ \bar{x}_1,\ ...,\ \bar{x}_{n+m}$, which is

$$\sum_{j=1}^{n} c_j \bar{x}_j \geq \sum_{j=1}^{n} c_j x_j \tag{1.21}$$

4. The improved feasible solutions are translated in choice of values of right-hand side variables into the corresponding values of the left-hand side variable and of the objective function. J.E.Strum (1972) refers to the systems as **dictionaries.** Thus, every dictionary

associated with (1.19) will be a system of linear equations in the variables $x_1,\ x_2,\ ...,\ x_{n+m}$ and z. However, not every system of linear equations in the variables constitutes a dictionary (7).

5. We have defined $x_{n+1},\ x_{n+2},\ ...,\ x_{n+m}$ and z in terms of $x_1,\ x_2,\ ...,\ x_n$, and so the n + m +1 variables are interdependent. For example, the three dictionaries in the first example are as follows:

$x_1,\ x_2,\ x_3\ ,x_4,\ x_5,\ \ x_6$ and z constitute a solution of (1.12)
$x_1,\ x_2,\ x_3\ ,x_4,\ x_5,\ \ x_6$ and z constitute a solution of (1.16)
$x_1,\ x_2,\ x_3\ ,x_4,\ x_5,\ \ x_6$ and z constitute a solution of (1.18)

These three dictionaries contain the same information of the seven variables. However, they present the seven variables in their own ways. For instance, the system of (1.12) the variables $x_1,\ x_2,\ x_3$ are independent and the slack variable $x_4,\ x_5,\ x_6$ and z are dependent on them. In addition, every solution of (1.12) is a solution of (1.16) and (1.18), and vice versa.

We can define properties of dictionaries in these following:

1. Every solution of the set of equations comprising a dictionary must be also a solution of (1.20) and vice versa.
2. The equations of every dictionary express m of the variable $x_1,\ x_2,\ ...,\ x_{n+m}$ and the objective function z in terms of the remaining n variables.

3. When setting the right-hand side variables at zero and evaluating the left-hand side variables, we can obtain a feasible solution. This property is call *feasible dictionaries.*

    - Every feasible dictionary has a feasible solution.
    - Not all feasible solution is described by a feasible dictionary, e.g.

      $x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 2,\ x_5 = 5,\ x_6 = 3$ of (1.5)

The feasible solutions that can be described by dictionaries are called *basic*. In other wards, the variable x_j on the left-side of a dictionary are the basic, while the variable x_j on the right-side of a dictionary are *nonbasic.* The important characteristic of the simplex method is that it work properly with basic feasible solutions and neglect all other feasible solutions.

1.4.3. Revised Simplex Method

1.4.3.1. Standard and revised simplex method

The revised simplex method is known as recreating a new solution directly from the original data and the new solution was found without any reference to dictionaries. In each iteration, the old solution is represented by a dictionary and the new solution can be easily found. For each iteration, the revised simplex method will solve two systems of linear equations, used some device to update their solutions. The device is referred to the product form of the inverse, created by G.B.Dantig and W.Orchrd-Hays (1954).

Conversely, the standard simplex method is the implementation of the simplex method that updates the dictionary in each iteration. The capabilities of implementations between the revised simplex method and the standard simplex method depend on a particular purpose of implementation and the nature of the data. The period of implementations may be different because of the reasons. However, the revised simplex method works faster than the standard simplex methods in the general large and dense LP problems. Thus, the modern computational programs always use the concepts of the revised simplex method to solve the LP problems (7).

1.4.3.2. Matrix Description of Dictionaries or the standard simplex method

1. We consider the dictionary, which was developed from the relationship between dictionaries and the original data.

$$\begin{aligned} x_1 &= 54 - 0.5x_2 - 0.5x_4 - 0.5x_5 + 0.5x_6 \\ x_2 &= 63 - 0.5x_2 - 0.5x_4 + 0.5x_5 + 1.5x_6 \\ x_7 &= 15 + 0.5x_2 - 0.5x_4 + 0.5x_5 + 2.5x_6 \\ \text{---} & \text{---} \\ z &= 1782 - 2.5x_2 + 1.5x_4 - 3.5x_5 - 8.5x_6 \end{aligned} \qquad (1.22)$$

arising from the linear programming problem,

$$
\begin{aligned}
\text{Maximize} \quad & 19x_1 + 13x_2 + 12x_3 + 17x_4 \\
\text{Subject to} \quad & 3x_1 + 2x_2 + x_3 + 2x_4 \le 255 \\
& x_1 + x_2 + x_3 + x_4 \le 117 \\
& 4x_1 + 3x_2 + 3x_3 + 4x_4 \le 420 \\
& x_1 , x_2, x_3, x_4 \ge 0
\end{aligned} \tag{1.23}
$$

2. The top three equations in the dictionary are modified to add three slack variables of the LP problem.

$$
\begin{aligned}
3x_1 + 2x_2 + x_3 + 2x_4 + x_5 &= 255 \\
x_1 + x_2 + x_3 + x_4 + x_6 &= 117 \\
4x_1 + 3x_2 + 3x_3 + 4x_4 + x_7 &= 420
\end{aligned} \tag{1.24}
$$

3. The three equations in (1.24) are converted to matrix terms. The matrixes of the dictionary (1.24) are $Ax = b$.

$$
\mathbf{A} = \begin{bmatrix} 3 & 2 & 1 & 2 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 4 & 3 & 3 & 4 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 225 \\ 117 \\ 420 \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}
$$

4. Solving the LP problem with the standard simplex method, we found the basic variables of the dictionary are $x_1,\ x_3,\ x_7$. Thus we should take these three variables in an unknown matrix.

$$\mathbf{A}_B = \begin{bmatrix} 3 & 2 & 1 \\ 1 & 1 & 1 \\ 4 & 3 & 3 \end{bmatrix}, \qquad \mathbf{A}_N = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 3 & 4 & 0 & 0 \end{bmatrix}, \qquad \mathbf{x}_B = \begin{bmatrix} x_1 \\ x_3 \\ x_7 \end{bmatrix}, \qquad \mathbf{x}_N = \begin{bmatrix} x_2 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

5. The $\mathbf{Ax} = \mathbf{b}$ can be converted to $\mathbf{A}_B\mathbf{x}_B = \mathbf{b} - \mathbf{A}_N\mathbf{x}_N$. Also, the square matrix $\mathbf{A}_B$ is a nonsingular matrix. The nonsingular matrix has their inverse matrix and their determinant is not equal to zero. $\mathbf{A}_B$ and $\mathbf{A}_N$ contain coefficients of basic and coefficient of non-basic variables, respectively. $\mathbf{x}_B$ and $\mathbf{x}_N$ contain basic and non-basic variables. Thus, we can solve the linear equations in terms of matrix to find a feasible solution that included the unknown variables ($x_1,\ x_3,\ x_7$).

Let $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{Ax} = \mathbf{A}_B\mathbf{x}_B + \mathbf{A}_N\mathbf{x}_N$, we obtain

$$\mathbf{A}_B\mathbf{x}_B + \mathbf{A}_N\mathbf{x}_N = \mathbf{b}$$

$$\mathbf{A}_B\mathbf{x}_B = \mathbf{b} - \mathbf{A}_N\mathbf{x}_N \qquad (1.25)$$

Given $\mathbf{A}_B^{-1}$ is an inverse matrix of $\mathbf{A}_B$, we multiplied by $\mathbf{A}_B^{-1}$ on the lift.

$$\mathbf{A}_B^{-1}\mathbf{A}_B\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b} - \mathbf{A}_B^{-1}\mathbf{A}_N\mathbf{x}_N$$

$$\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b} - \mathbf{A}_B^{-1}\mathbf{A}_N\mathbf{x}_N \qquad (1.26)$$

6. We should also obtain the objective function $\mathbf{z}$ as $\mathbf{cx}$ and $\mathbf{c}_B\mathbf{x}_B + \mathbf{c}_N\mathbf{x}_N$ matrix

$$\mathbf{c} = \begin{bmatrix} 19 & 13 & 12 & 17 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{c}_B = \begin{bmatrix} 19 & 12 & 0 \end{bmatrix}$$

$$\mathbf{c}_N = \begin{bmatrix} 13 & 17 & 0 & 0 \end{bmatrix}$$

7. We substitute for $\mathbf{x}_B$ and express z as the matrix terms.

$$\mathbf{z} = \mathbf{cx}$$

$$\mathbf{z} = \mathbf{c}_B\mathbf{x}_B + \mathbf{c}_N\mathbf{x}_N$$

$$\mathbf{z} = \mathbf{c}_B(\mathbf{A}_B^{-1}\mathbf{b} - \mathbf{A}_B^{-1}\mathbf{A}_N\mathbf{x}_N) + \mathbf{c}_N\mathbf{x}_N$$

$$\mathbf{z} = \mathbf{c}_B\mathbf{A}_B^{-1}\mathbf{b} - \mathbf{c}_B\mathbf{A}_B^{-1}\mathbf{A}_N\mathbf{x}_N + \mathbf{c}_N\mathbf{x}_N$$

$$\mathbf{z} = \mathbf{c}_B\mathbf{A}_B^{-1}\mathbf{b} + (\mathbf{c}_N - \mathbf{c}_B\mathbf{A}_B^{-1}\mathbf{A}_N)\mathbf{x}_N$$

8. We can summarize the dictionary of the LP problem in (1.23) in the matrix terms as

$$\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b} - \mathbf{A}_B^{-1}\mathbf{A}_N\mathbf{x}_N$$

$$\mathbf{z} = \mathbf{c}_B\mathbf{A}_B^{-1}\mathbf{b} + (\mathbf{c}_N - \mathbf{c}_B\mathbf{A}_B^{-1}\mathbf{A}_N)\mathbf{x}_N \qquad (1.27)$$

9. According to the dictionary, we record the standard from of a LP problem in (1.19) as

Maximize $\sum_{j=1}^{n} c_j x_j$ $\mathbf{cx}$

Subject to $\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad (i = 1, 2, ..., m)$ $\mathbf{Ax} = \mathbf{b}$

$x_j \ge 0 \quad (j = 1, 2, ..., n)$ $\mathbf{x}_j \ge \mathbf{0}$

Each matrix has number of rows and columns in these following:

$\mathbf{c}_{(n+m) \times 1}$, $\mathbf{x}_{1 \times (n+m)}$, $\mathbf{A}_{\mathrm{m} \times (\mathrm{n+m})}$, and $\mathbf{b}_{1 \times \mathrm{m}}$

10. Another purpose to use matrixes to describe the dictionary

We can use partitions of matrix to distinguish between basic and non-basic variables. We have the partition of $\mathbf{A}$ is $\mathbf{A}_B$ and $\mathbf{A}_N$, the partition of $\mathbf{x}$ is $\mathbf{x}_B$ and $\mathbf{x}_N$, and the partition of $\mathbf{c}$ is $\mathbf{c}_B$ and $\mathbf{c}_N$. At this moment, we can present

$\mathbf{A}_B$ is a nonsingular matrix (1.28)

We assume $\mathbf{x}^*$ is the basic feasible solution and partitions $x_1, x_2, ..., x_{n+m}$ in m basic and n non-basic variables.

As the basic feasible solution $\mathbf{x}^*$ satisfies $\mathbf{Ax}^* = \mathbf{b}$ and $\mathbf{x}_N^* = \mathbf{0}$,

$$\mathbf{A}_B\mathbf{x}_B^* = \mathbf{Ax}^* - \mathbf{A}_N\mathbf{x}_N^* = \mathbf{b}$$

$$\mathbf{A}_B\mathbf{x}_B^* = \mathbf{A}\mathbf{x}^* = \mathbf{b}$$

where $\mathbf{x}_B^*$ is the current values of the basic variables

Likewise, an arbitrary vector $\tilde{\mathbf{x}}$ satisfies $\mathbf{A}_B\tilde{\mathbf{x}}_B = \mathbf{b}$

$$\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}_B\tilde{\mathbf{x}}_B + \mathbf{A}_N\tilde{\mathbf{x}}_N = \mathbf{b}$$

As $\tilde{\mathbf{x}}_N = \mathbf{0}$, implies $\tilde{\mathbf{x}}_B = \mathbf{x}_B^*$, the results proof that $\mathbf{A}_B$ is a nonsingular matrix.

We call $\mathbf{A}_B$ as the basis matrix or simply the basis. To correspond with the name of basis, we assume the matrix $\mathbf{A}_B$ by the matrix $\mathbf{B}$ in (1.27).

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{A}_N\mathbf{x}_N$$

$$\mathbf{z} = \mathbf{c}_B\mathbf{B}^{-1}\mathbf{b} + (\mathbf{c}_N - \mathbf{c}_B\mathbf{B}^{-1}\mathbf{A}_N)\mathbf{x}_N \qquad (1.29)$$

1.4.3.3. The Revised Simplex Method

The revised simplex method is the alternative methods of the standard simplex method. To illustrate, we shall apply the revised simplex method to the feasible dictionary (1.22). The main steps of the revised simple method are

1. Choose the entering variable
2. Find the leaving variable

3. Update the current basic feasible

1. Choose the entering variable

To begin with, the entering variable is the basic feasible solution

$$\mathbf{x}_B^* = \begin{bmatrix} x_1^* \\ x_3^* \\ x_7^* \end{bmatrix} = \begin{bmatrix} 54 \\ 63 \\ 15 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix}$$

The entering variable may be any non-basic variable with a positive coefficient in the last row of the dictionary.

In (1.29), the last row column is a vector of coefficients $\mathbf{c}_N - \mathbf{c}_B\mathbf{B}^{-1}\mathbf{A}_N$ or

$$z = \ldots - 2.5x_2 + 1.5x_4 - 3.5x_5 - 8.5x_6 \qquad (1.30)$$

In the revised simplex method, the vector $\mathbf{c}_N - \mathbf{c}_B\mathbf{B}^{-1}\mathbf{A}_N$ is computed in two steps:

1. Find $\mathbf{y} = \mathbf{c}_B\mathbf{B}^{-1}$ from $\mathbf{yB} = \mathbf{c}_B$

For example, in (1.22) $\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \cdot \begin{bmatrix} 3 & 1 & 0 \\ 1 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 19 & 12 & 0 \end{bmatrix}$

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} 3.5 & 8.5 & 0 \end{bmatrix}$$

2. Calculate $\mathbf{c}_N - \mathbf{y}\mathbf{A}_N$ to fine the vector feature in (1.30).

$$\begin{bmatrix} 13 & 17 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 3.5 & 8.5 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 3 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -2.5 & 1.5 & -3.5 & -8.5 \end{bmatrix}$$

As the second component of the vector $\mathbf{x}_N = [x_1 \quad x_2 \quad x_3 \quad x_4]^T$ in (1.29) is a positive component, the second component enters the basis. Thus, the entering variable can be the non-basic variable. In addition, if a non-basic variable x_j corresponds to a component c_j of $\mathbf{c}_N$ and to a column $\mathbf{a}$ of $\mathbf{A}_N$ then the corresponding component of $\mathbf{c}_N - \mathbf{y}\mathbf{A}_N$ equals to $\mathbf{c}_j - \mathbf{y}\mathbf{a}$. Thus, if $\mathbf{y}\mathbf{a} < c_j$, the entering variable can be any non-basic variable x_j. The corresponding column a of $\mathbf{A}$ is called the entering column.

2. Find the leaving variable

We assume the value t as a value in the entering variable, which ranges from zero to some positive numbers. To find the leaving variable, we increase t, while maintaining the remaining non-basic variables at their zero levels and adjust the value of basic variables to maintain the constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$. We increase t until a value in the basic variables is first dropped below zero. The t value that the basic variable is dropped to zero is the largest admissible value of t and the basic variable is the leaving variable.

In the standard simplex method, determining the leaving variable is applied as

$$\begin{aligned} x_1 &= 54 \ldots - 0.5x_4 \ldots \\ x_3 &= 63 \ldots - 0.5x_4 \ldots \\ x_7 &= 15 \ldots - 0.5x_4 \ldots \end{aligned} \quad \text{as} \quad \begin{aligned} x_1 &= 54 \ldots - 0.5t \\ x_3 &= 63 \ldots - 0.5t \ldots \\ x_7 &= 15 \ldots - 0.5t \ldots \end{aligned} \qquad (1.31)$$

or in terms of the matrixes (1.29) $\mathbf{x}_B = \mathbf{x}_B^* - \mathbf{B}^{-1}\mathbf{A}_N\mathbf{x}_N$. Thus

$$\mathbf{x}_B = \mathbf{x}_B^* - t\mathbf{d}$$

, where $\mathbf{d} = \mathbf{B}^{-1}\mathbf{A}_N$ or $\mathbf{d} = \mathbf{B}^{-1}\mathbf{a}$

If the revised simplex method, information is available only $\mathbf{x}_B^*$, while the values in matrix $\mathbf{d}$ can be obtained by $\mathbf{Bd} = \mathbf{a}$

$$\begin{bmatrix} 3 & 1 & 0 \\ 1 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix} \cdot \mathbf{d} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}, \quad \text{so} \quad \mathbf{d} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

From $\mathbf{x}_B = \mathbf{x}_B^* - t\mathbf{d}$,

$$\begin{aligned} 54 - 0.5t &= 39 \\ 63 - 0.5t &= 48 \\ 15 - 0.5t &= 0 \end{aligned}$$

We find easily that the largest values of t is 30 and x_7 is the leaving variable.

Obviously, we may spend some time to manually update the entire dictionary in the standard simplex method, while we can create computations to update the values in the dictionary for the next iteration in the revised simplex method. For example, the next iteration of basic variables and the basis matrix B is updated to

$$\mathbf{x}_B^* = \begin{bmatrix} x_1^* \\ x_3^* \\ x_4^* \end{bmatrix} = \begin{bmatrix} 39 \\ 48 \\ 30 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 1 & 1 \\ 4 & 3 & 4 \end{bmatrix}$$

For these computations, the order of rows of $\mathbf{x}_B^*$ and column of $\mathbf{B}$ are not restricted. It needs only $\mathbf{x}_B^*$ should correspond to $\mathbf{B}$ in the computations. Also, the actual order of the m columns of $\mathbf{B}$ that was specified by the order list of the basic variable is called basis

heading. To be convenient, we replace the leaving variable by the entering values in each update of the basis heading.

This is a summary of the revised simplex methods by Chvatal V (1983)

Step 1: Solve the system $\mathbf{yB} = \mathbf{c}_B$

Step 2: Choose an entering column. This is may be any column $\mathbf{a}$ of $\mathbf{A}_N$ that $\mathbf{ya}$ is less than the corresponding component of $\mathbf{c}_N$. If there is no such column, then the current solution is optimal.

Step 3: Solve the system $\mathbf{Bd} = \mathbf{a}$

Step 4: Find the largest t such that $\mathbf{x}_B^* - t\mathbf{d} \geq 0$. If there is no such t then the problem is unbound; otherwise at least one component of $\mathbf{x}_B^* - t\mathbf{d}$ equals zero and the corresponding variable is leaving the basis.

Step 5: Set the value of the entering variable at t and replace the values $\mathbf{x}_B^*$ of the basic variable by $\mathbf{x}_B^* - t\mathbf{d}$. Replace the leaving column of B by the entering column and, in the basis heading, replace the leaving variable by the entering variable.

The firs-two steps in each iteration are to check whether there is a current feasible solution $\mathbf{x}^*$ in the dictionary, while the step 1 and step 3 make the revised simplex method more effective. The efficiency of the implementation in the step 1: $\mathbf{yB} = \mathbf{c_B}$ and the step 3: $\mathbf{Bd} = \mathbf{a}$ depends on the solutions or devices for solving the two systems. We describe typically ideas of two devices, which are the simplest and popular ones in the

revised simplex method. Further information of the devices is described in Chvatal V (1983).

1.4.4. Explicit bounds on individual variables

Many linear programming problems involve explicit upper bounds on individual variables. The revised simplex methods can be made to work on directly in such a way that the size of each basis matrix is only m x m (7.1). This technique can be applied in the more general context of problems

Maximize $\sum_{j=1}^{n} c_j x_j$

Subject to $\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1,\ 2,\ ...,\ m)$

$l_j \leq x_j \leq u_j \qquad (j = 1,\ 2,\ ...,\ n)$

Each l_j is an earlier number or symbol $-\alpha$, meaning that no lower bound is imposed on x_j, and each u_j is either a number or the symbol $+\alpha$, meaning that no upper bound is imposed on x_j, We need only add the slack variables $x_{n+1}, x_{n+2}, ..., x_{n+m}$, with $l_{n+i} = 0$ and $u_{n+i} = +\alpha$ for all i. This is a convenient way to admit variable x_j with $l_j = u_j$.

In addition, we need only observe that minimizing $\sum c_j x_j$ is tantamount to maximizing $\sum(-c_j)x_j$ and that every inequality constraint can be converted into an equation by the introduction of an appropriate slack variable.

An iteration of the revised simplex method to handle explicit bound

Step 1. Solve the system $\mathrm{yB} = c_B$.

Step 2. Choose an entering variable x_j. This may be any nonbasic variable x_j such that, with **a** standing for the corresponding column of **A**, we have either $\mathbf{ya} < c_j$, $x_j^* < u_j$, or $\mathbf{ya} > c_j$, $x_j^* < l_j$ If there is no such variable then stop; the current solution x^* is optimal.

Step 3. Solve the system **Bd = a.**

Step 4. Define $x_j(t) = x_j^* + t$ and $\mathbf{x_B}(t) = \mathbf{x_B^*} - t\,\mathbf{d}$ in case $\mathbf{ya} < c_j$ and $x_j(t) = x_j^* + t$, $\mathbf{x_B}(t) = \mathbf{x_B^*} + t\,\mathbf{d}$ in case $\mathbf{ya} > c_j$. If the constraints $l_j \le x_j(t) \le u_j$, $\mathbf{l_B} \le \mathbf{x_B}(t) \le \mathbf{u_B}$ are satisfied for all positive t then stop; the problem is unbounded. Otherwise set t at the largest value allowed by these constraints. If the upper bound imposed on t by the constraints $\mathbf{l_B} \le \mathbf{x_B}(t) \le \mathbf{u_B}$ is sticker than the upper bound imposed by $l_j \le x_j(t) \le u_j$ alone is as strict as the upper bound imposed by all the constraints in $\mathbf{l_B} \le \mathbf{x_B}(t) \le \mathbf{u_B}$

Step 5. Replace x_j^* by $x_j(t)$ and $\mathbf{x_B^*}$ by $\mathbf{x_B}(t)$. If the value of the entering variable x_j has just switched from one of its bounds to the other, then proceed directly to step 2 of the next iteration. Otherwise, replace the leaving variable x_i by the entering variable x_j in the basis heading, and replace the leaving column of **B** by the entering column **a**.

1.4.5. Dual revised simplex method

Each LP problem, called as primal, is associated with its counterpart known as dual LP problem. Instead of primal, the dual LP problem has fewer constraints than the primal and involves maximization of an objective function. The dual LP problem is constructed by defining a new decision variable for each constraint in the primal problems and a new constraint for each variable in the primal. The dual problem consists of coefficients of decision variables in the primal objective function. The coefficients matrix of the dual is the transpose of the primal coefficient matrix. Maximizing the primal problem is equivalent to minimizing the dual and their optimal values are exactly equal. The dual simplex method is similar to the simple method except for the criteria for selecting the entering and leaving basic variables and for stopping the algorithm. The first step of iteration is to determine entering basic variables, which is equivalent to determine the leaving basic variable in the simplex method. The negative coefficient with the largest absolute value of the dual problem corresponds to the negative variable with the largest absolute value in the simple method The second step is determine the leaving basic variable which is equivalent to determining entering variables in the simplex problem. The variable in the dual problem that reaches zero first corresponds to the coefficient that reaches zero first. These two criteria for stopping the algorithms are also complementary. The dual simplex method is useful for solving large linear programming problems because less artificial variables are introduced to construct the initial basis solutions and required fewer number of iteration (7, 22).

In a large sparse problem, the standard simplex method takes more time than the revised simplex method for execution in each iteration. Total estimate time per iteration is approximately $32m+10n$ for the revised simplex method versus $mn/4$ for the standard simplex method, where *m* is the number of columns and *n* is the number of rows. Typically, *n* is usually considered more than *m* as $n \geq 2m$ for a LP problem. In practical, the large sparse problem is assumed as at least 2000 rows per a LP problem. When, the large sparse problem is analyzed, the memory spaces are also considered as another problem related to the executed time. Thus, the time for execution estimated by the core memory may need to be adjusted from the peripheral memory as well. In addition to time for execution and memory spaces, mathematical algorithms applied in the revised simplex method are also a factor related to this consideration. For these reasons, the revised simplex method is applied in computer programs for solving the LP problems. In some cases, the revised simplex method may take more time than the standard simplex method when the bases $\mathbf{B}_k$ are completely dense and $n < 2m$.

Zero tolerances (ε_i) are important for selecting an entering variable. If small negative numbers or zeros are rounded to small positive numbers, the corresponding component of non-basic variable $\mathbf{c}_N - \mathbf{y}\mathbf{A}_N$ for entering the basis may cause some errors. We may define the zero tolerances in advance. For example, the component of $\mathbf{c}_N - \mathbf{y}\mathbf{A}_N$ is considered positive if its computed value exceeds ε_1. Results of nominators, divided by an extremely small numbers are valid when its value is less than ε_2, e.g diagonal elements in the eta columns. Also, the zero tolerance is used for comparing between two different numbers or vectors. The zero tolerances are defined based on prior knowledge

for particular issues. Murtagh B.A. (1981) suggested appropriate choices of the zero tolerances of $\varepsilon_1 = 10^{-5}$, $\varepsilon_2 = 10^{-8}$, or $\varepsilon_3 = 10^{-6}$ for computed values of 15 decimal digit numbers (7).

1.5. MetModel and MetModel GUI

1.5.1 MetModel

MetModel is a Python-based framework for flux balance analysis of cellular metabolism. MetModel applies the algorithm of optimization-based linear programming for analyzing metabolic reconstruction models of cellular organisms. The MetModel stands for Metabolic Modeling and was first developed in the Python-based framework and was modified to MetModelGUI by Burns W, Roberts S, Brooks P, Fong S. at VCU (24, 25, 26). Both versions have been applied for many cellular metabolic reconstruction projects at VCU. These packages were used to evaluate effects of genes/proteins deletion and genes knockout analysis in organism cell and also investigate gap analysis or gap filing to build the flux balance models based on the optimization of linear programming. In some previous studies, MetModel was applied to build an initial genome-scale metabolic model of Cryptosporidium hominis, in which 52 essential metabolic reactions were found and used to predict fluxes for each reaction of C. hominis (25). The MetModel was also applied in the field of metabolic engineering, such as increasing of enzymes activities to produce biofuel products. In such case, the MetModel enabled to

investigate metabolic pathways of Thermobifida fusca. The FBA of T.fusca model found 320 unique reactions, approximately 50% of the true reactions and metabolites in the entire pathways. The final reactions for T.fusca metabolic model are anticipated to be used for conducting a novel biofuel agent (26). In addition to C. hominis and T.fusca, the MetModel was applied to study a genome-scale metabolic model of the fungal pathogen, Cryptococus neoformans. The software performed the *in silico* gene deletion simulation to create metabolic pathways to understand how individuals infect with the cryptococcosis (27).

Basically, mathematical representation of metabolic reconstruction was used to obtain accuracy of subsequent computations of biochemical transformation (28). The first part of MetModel investigates specificity of metabolites and correctness of substrates, gene products, enzymes, or coenzymes and identity the molecular formula of the metabolites. The chemical formulae estimate stoichiometric coefficients of the reaction, balances of the elements and charges of the reactions. Directionality of reactions and cellular compartment where the reaction takes place was determined in the primary process of the MetModel (28).

MetModel consists of two tab-delimited text input files and twelve Python-based programs. There following are information of each program and Figure 2 is a summary of MetModel structure.

1. **Reactionsnew.txt** is a database of reactions storing in a tab-delimited file with six data columns as follows:

- Types of reaction, e.g. transport, extracellular, NAD metabolism, fatty acid biosynthesis, etc…
- Enzyme Commission or EC number is the enzyme identifier of KEGG database (Kyoto Encyclopedia of Genes and Genomes) for linking the same enzymes outside databases, such as IUBMB, ExplorEnz, BRENDA, ExPASy, UM-BBD, ERGO. For example, EC 3.1.3.36 is referred to phosphoinositide 5-phosphatase.
- Names of reactions used for calling in programming, e.g. R_NAGAly or R_ATPS.
- True/False directions: true for both irreversible and reversible directions and false for irreversible directions.
- Chemical name of reactions, name of catalysts and/or location of the reaction, such as ATPase,cytosolic, Glycolatedehydrogenase(NAD) or IMPdehydrogenase.

2. **LowCostMetabolites.txt** is a database of low-cost exchanges also storing in a tab-delimited file with 6 data columns:

- ID: sign of metabolites used for programming, e.g. ala-L[e], h2o[e]
- Name of metabolites: L-Alanine, L-Aspartate, Inosine, and etc…
- Charges ranged from -13 to 4
- Sources: yes/no for substrates
- Escapes: yes/no for products
- Category: types of metabolite, such as amino acid or nucleotides

3. **Builddb.py** is a Python program used for updating reaction database by combining reactions in the previous database and the reactions in the model of each organism into a new database with the same structure as the previous database, named reactionsnew.txt.

4. **BooleanParser.py** is a custom Python module to match the name of genes or proteins with the gene-protein reaction statement, such as “alphas” matched with “alphanums” by eliminating “(“, “)”, “&”, “|”, whitespace, or extensions (e.g. 6733.1 become 6733), and ignoring with providing an error message for the arguments with more than 255 characters.

5. **Eq_current.py** was used to parse reaction statement to a data structure that will be used in computations: “True” indicates both irreversible and reversible reactions and “False” indicates irreversible reaction. Coefficients were adjusted to an integer number. Signs of direction were categorized in the reversible and irreversible groups. Reactants or products with no compartments, single compartments, or multiple compartments were evaluated and converted to a new form in order to add in that structure. Special characters in the name of metabolites were changed to alphabet characters with unique and readable names. For example, e.g leu-L[c]' was converted to 'M_leu_DASH_L_c. Lower alphabets in the parenthesis indicate the cellular compartments where the reaction takes place. That conversion can be interchangeable as internal and external representations. These are the list of cellular compartments and signs used in referring to the compartments (Reed JL, et.al, 2006).

[b]: extraorganism, [c]: cytoplasm, [e]: extracellular, [g]: golgi aparatus, [h]: chloroplast or flagellum, [l]: lysosome or reservosome, [m]: mitochondria, [n]: nucleus, [p]: periplasm or between inner and outer mitocondria, [r]: endoplasmic reticulum, [v]: vacuole or acidocalcisome, [x]: peroxisome or glycosome, and [y]: glycosome.

6. MapGPR_current.py is a custom Python module, which has functions similar to the eq_current.py program but was used to classify different levels of genes and proteins expressions: high (1), moderate (0), low (-1) expressions, suggested by Shlomi T, et.al (2008) for the tissue-specific activity of metabolic disease-causing genes. The disease-causing gens may be more likely to be expressed in a specific tissue than genes not associated with the disease. Their study presented different metabolic functions in different tissues.

7. Rxn.py is a custom Python module used for defining components for a chemical formula, balancing the components, and expressing of reaction equations. In this file, csv and pyparsing, kegg modules were imported. The csv file was used to read data files in the CSV format The CSV file contains a number of rows, each row containing a number of columns, separated by commas. The pyparsing module provides a library of classes that parses input names to constructed grammar and expresses results of the grammar directly in Python code like regular expression in Perl. The kegg module were retrieved the codes of KEGG compound, a collection of small molecules, biopolymers, and other chemical substances that are relevant to biological systems to linl to other KEGG databases.

8. Metmodel_current.py is the main Python program for creating and analyzing metabolic models. The program was implemented with importing four standard Python modules: os, re, time, and pickle, three special Python modules: glpsol, libSBML, pyparsing, and three custom Python modules: mapGPR_current.py, eq_current.py and booleanParser.py. These are descriptions of functions of each module.

os - this module provides miscellaneous operating system interfaces, a portable way of using operating system dependent functionality, such as manipulation of paths (29).

re - this provides regular expression that matches with operations (29)

time - this provides a number of functions to deal with dates and the time within a day. (29)

pickle - this module implements a basic but useful algorithm for serializing and de-serializing a Python object structure. Pickling is the process that a Python object hierarchy is converted into a byte stream and unpickling is the inverse operation (29).

glpsol - this is the free optimization model engine of glpk package (GNU Linear Programming Kit). This package was used to solve linear programming (LP), mixed integer programming (MIP), and other related problems. GLPK uses the revised simplex method and the primal-dual interior point method for non-integer problems and the branch-and-bound algorithm together with Gomory's mixed integer cuts for (mixed) integer problems. This package was developed in several computer languages such as C, Java, or Python. The glpsol stands for GLPK linear programming/MIP solver, which can be used for a wide variety of optimization problems (30).

libSBML - is an application programming interface (API) library for reading, writing and manipulating files and data streams containing Systems Biology Markup Language (SBML) content. One of the features of libSBML is used for manipulating mathematical formulas at differences SBML Level and in both text-string and MathML forms, including presenting mathematical formulas regardless of their original format. The libSBML is also used to validate input files and data streams to verify correctness of the models (31).

pyparsing – this module was used to import names, define grammar, use the grammar to parse the input text, and process the results from parsing the input text. This is an alternative approach of using traditional lex/yacc approach, or regular expressions (32).

mapGPR_current.py – this was used to read, parse, and evaluate boolean GPR statements

eq_current.py – this works on parsing reaction equations, metabolites, compartments, and other elements to create a data structure prepared for computations.

booleanParser.py – this module matches the name of genes or proteins with the gene-protein reaction statement.

This program is used to analyze a constraint-based metabolic model with a data structure that contains model ID, model name, compartments, species, reactions and coefficient of reactions derived by eq_current.py. The main class (“cb”) of the program performed these following:

- Create and manipulate the information, such as setting, adding, deleting, updating each element in the species’s constraint-based model and reactions in the model,

selecting minimize or maximize objectives, setting default flux limits, and writing the constraint-based metabolic model of the species into *.lp file, which consists of problem header (FBA), objective functions (minimize or maximize), subject to (lists of mass balance equations), and bound (flux constraints).

- Call “glpsol” to solve the linear programming problems using optimization method, find the solutions, retrieve the values of objective functions and write them to *.xls or temp file.

- If the problem was solved, print the reactions from the current model into *.wil file. The information in that file contains reaction ID, name of reactions, EC numbers, true/false for reverse reaction, name and location of metabolites, chemical reactions of gene-protein reactions or protein-reaction-relation, pathway, and confidence.

- Use pickle to write a pickled object containing current model reaction constraints. Load model constraints from the pickled object. Overwrite any existing constraints if there is a constraint in the pickled object.

- Add source fluxes for all sources, escape fluxes for all escapes, exchange fluxes for all exchanges. Read and build initial model by defining biomass equation, source, escape, exchange metabolites, constraints, and gene-protein reaction.

- Build model from model text files downloaded from mm2

- Calculate gene presence/absence, calculate reaction presence/absence

- Write an *.xml output file for a flux distribution to build a map of the model by CellDesigner program.

- Write a SBML format file for the constrain-based metabolic model. This SBML file can be used to work with Cytoscape program.

- Write a *.dat file for the constrain-based metabolic model in a data structure of "S * v, which consists of 'mets': keys for metabolites, values for (reactionID, coef), and skip boundary metabolites if metabolite ends in '_b'.

- Delete specified set of genes. Check constraints. Solve unchanged model. Record default constraints. Make sure if genes were from gene-protein reactions of the model. Check whether gene deletion affects to reactions.

9. **Metmodel_gurobi.py** was used as a subclass of “cb” class in the metmodel_current.py and when deletions functions in the “cb” class was called. The Metmodel_gurobi.py can be alternatively used of glpk.py to do deletions of current basis. These are some modules used in metmodel_gurobi.py:

deepcopy – this module provides generic (shallow and deep) copying operations.

matrix and linalg – this provides from numpy for numerical computing in terms of matrix and linear algebra.

scipy.linalg – this module was used for solving linear systems of equations

scipy.sparse – this module was used for sparing two-dimension matrix in rows, columns, diagonals, or others.

sys - this module was used for accessing to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

string – this module was used for string operations

math – the module was used for the mathematical functions and used with complex numbers.

defaultdict – this is a part of the collections module. defaultdict is similar to regular dictionaries except for taking an extra first argument and when a dictionary key is encountered for the first time, the default factory function is called and the result used to initialize the dictionary value.

yaml – is a data serialization format designed for readability and interaction with scripting languages such as Perl and Python. yaml is optimized for data serialization, formatted dumping, configuration files, log files, Internet messaging and filtering.

gurobipy - This is a script used to run Gurobi Python programs within Python environment (33).

"gurobicb" was the main class in this program and was prepared for applying Gurobi optimizer, a state-of-the-art solver for linear programming (LP), quadratic programming (QP) and mixed-integer programming (MILP and MIQP) (33). The "gurobicb" class was used as a subclass of the "cb" class in the metmodel_current.py and has functions such as building a model from a yaml file, solving a model with Gurobi package, finding a minimum-cost and -size sets of sources, escapes, and reactions, finding the set of un-producible metabolites, generating all alternate optimal solutions.

10. **Wil2metmodel.py** was used to read all reaction models of each organism in *.wil file and generate each part of the models to different file types: *.biomass, *.reaction, *.source, and *.escape.

11. **Bouncertest.py** is a driver program to call each program to function and analyze the constraint-base metabolic model. Begin with metmodel_current.py, wil2metmodelpy.py,

metmodel_gurobi.py, or input file of organism (*.wil), and can be called functions from each imported program to evaluate results such as build_from_textfiles, set_escapes, bouncer2, functions in the metmodel_gurobi.py.

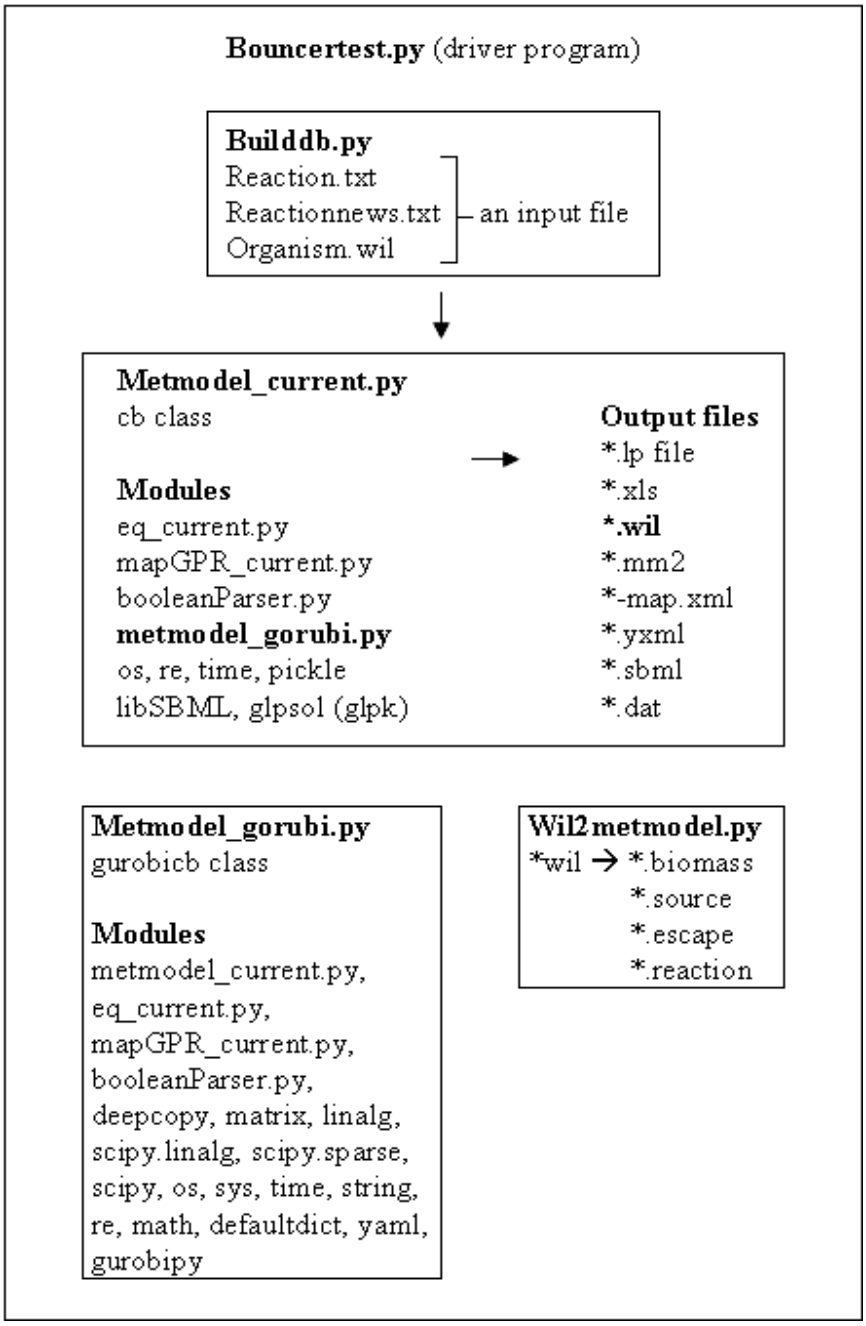

Figure 2. Summary of MetModel-based Python framework

1.5.2. MetModel GUI

MetModel GUI is Java-based Graphical User Interface for creating and analyzing linear programming-based models of cellular metabolism. The software allows users to build network-based models of cellular metabolism. The purpose of the program is to reconstruct models that the cells can make the most benefits of their resources such as metabolites to support their needs. For example, glucose is the element essential for microorganisms. The MetModelGUI can maximize the metabolic flux through reactions producing amount of glucose appropriate for the needs of the organisms. The MetModel GUI is composed of 22 java processed files.

Table 2. Twenty-two Java-based programs in MetModel GUI and imported programs for implementing in each program.

Item	Java programs	Item of imported program
1	BiomassObjective.java	7, 9, 11
2	Controller.java	7, 9, 11, 13, 18
3	EssentialityFrame.java	6, 7, 15, 16, 18
4	EssentialityOutputFrame.java	7, 10
5	GapFrame.java	7, 9, 11, 18
6	GeneNode.java	7
7	Handler.java	1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15,16, 17, 18, 19, 20, 21
8	Menu.java	1, 2, 3, 5, 7, 8, 11, 12, 13, 19
9	MetaboliteNode.java	7
10	MetModel.java	1, 2, 3, 4, 5, 7, 8, 11, 13, 14, 16, 17, 19, 20, 21
11	Model.java	6, 9, 11, 15, 16, 18, 21
12	MyOptionPane.java	7
13	Optimization.java	7, 9, 11, 13, 18
14	OutputFrame.java	7
15	ProteinNode.java	6
16	ReactionDatabase.java	9, 11, 18
17	ReactionDetails.java	7, 9, 18

18	ReactionNode.java	9
19	ReadGPRFrame.java	7
20	SortableTable.java	5, 7, 11, 17, 16, 18
21	TransportFrame.java	7, 9, 11, 21
22	VisualizationFrame.java	7, 11

These following are explanation for the programs in Table 2.

1. BiomassObjective.java: creates “Biomass Objective” frame, read animo acids, nucleotides, cofactors, and biomass equations from reaction nodes, protein nodes, metabolite nodes and reactions nodes, and display the data in the window. This program allows user to update coefficients of the genes, proteins, metabolites, and a biomass equation. Changes in coefficients in this window will not change coefficients in the reaction database.

2. Controller.java: creates functions of menu bar such as opening reaction database and reading line information, setting gene transportation, updating biomass metabolites, saving biomass equations to the same and new files, closing current database, and exiting the program.

3. EssentialityFrame.java: creates “Gene Essentiality” frame. The program checks whether gene or protein knockouts affect to reactions. The program will knockout the reactions by deletions of one gene for single reaction at a time and double genes or double proteins out of a reaction at a time and then will evaluated biomass fluxes values. If the biomass fluxes less than -1.0, the genes or proteins are unessential for the reactions and pathways. If the biomass flux results close to zero values in both positive and negative directions, the genes and proteins are essential for pathways. Selections between single and double relations are based on cellular metabolic information of each organism.

4. EssentialityOutputFrame.java: creates “Output” frame that provides essential outputs of MetModelGUI and contains results of coefficients of biomass fluxes, reaction fluxes, and transport fluxes. The results will be displayed when selecting “Run Model” from menu bar. The results were performed by processing of Handler program.

5. GapFrame.java: creates “Gap Analysis” frame. Gap analysis will be performed when biomass fluxes present a result of zero value. Gap data (*.-gap.dat) will be exported. The program allows user to select some sources and escapes to fill in gaps. The program will fill in the gaps by searching for sources and escapes in database and calculating cost of metabolites using data in LowCostMetabolites.txt.

6. GeneNode.java: get gene names and checks whether the genes are presented in the database. The program will return “true” if the names are present and “false” if the names are absent.

7. Handler.java. This is a child class for handling steps of MetModelGUI. For examples, closing of all java programs, adding of modules, setting of biomass equations, adding of model rows for LP analysis, updating of metabolites, adding of reactions from a reaction database, updating of biomass metabolites, updating of reactions, removing of metabolite rows, showing of reaction details. This file provide sub-functions used for each main function, such as setting of biomass coefficients, showing and hiding of biomass objectives, receiving and returning biomass equations, adding of model rows, receiving of lists of metabolites, reactions, adding of the lists to databases, getting file names, paths of gene-protein reactions, metabolite paths, executing of metabolite models, executing of gene-protein reaction models, printing outputs and essentiality output, retuning the lists of reactions, reaction names, fluxes, transport name, transports and

fluxes to optimization modules, saving gene names, protein names, reactions, sources, escapes, transports and fluxes of metabolic model.

8. Menu.java: creates menus and submenus in the menu toolbars using functions created by Handler.java, Controller.java, Optimization, Handler,java, and MyOptionPane.java. There are four main menu types: “File”, “Build”, “Tool”, and “GPR” menus. “File” menu works for creating a new file (*.wil file), opening, saving, and closing the *wil file. “Build” menu works for modifying for transports, setting of biomass equations, and running the model to file the optimization solution. “Tool” menu works for gap analysis or gap filling, exporting a map file for visualizing the essential reactions. “GPR” menu works for reading gene-protein reaction data, evaluating gene essentiality in pathways and eliminating genes or proteins unnecessary in the pathways. Each menu provides working control-keys as shown in the right-side for each menu.

9. MetaboliteNode.java: sets metabolite values such as ATP, NAD, and substrates and product bounds, sets compartments, number of reactions, and coefficients of reactions,

10. MetModel.java: create a driver program for running the MetModel 1.0. This program call to other java programs of the MetModel 1.0 for creating and analyzing the metabolic models based on the optimization-based linear programming.

11. Model.java: create functions for setting a metabolic model, which consists of list of reactions, metabolites, compartments, sources, escapees, and gene, protein as well as setting and removing transporsts, getting reactions, adding, and updating metabolites.

12. MyOptionPane.java: creates “Biomass equation” window to view current biomass reactions.

13. Optimization.java: provides linear programming solving. The program imports qs java library, QSopt's Java Callable function library. This library is a port of an alpha version of the C callable function library and compatible with the java runtime environment version 1.4 and up (28). The “qs” provide functions to read and write a LP problem from and to a file, build a LP problem and solve LP problems.

Arguments were declared, including waiting time from java runtime class. Lists of model information: metabolites, metabolite names, reactions, sources and escapes were assigned. Matrix rows and columns of linear programming problem were created using one-dimensional arrays. Metabolite and reaction data were read from the ReactionNode list and MetaboliteNode list. Lower and upper bounds of each metabolite were set. A massage error will be reported if the LP cannot be completely set up. The LP problem was solved by “opt_dual” function, dual simplex algorithm in the “QSopt” library. If the LP is flux, the “opt_dual” function returns solutions of the LP and “get_status” function obtains the solution status that found an optimization solution by the “QS.LP_OPTIMAL” function and printout of the optimization solution by Handler file. If the LP is GPR, the get_objval function will return a current objective function. However, if no optimal solutions are found, a massage of error in getting an optimal solution and the LP unable for solving is appeared. Row names and constraint names were listed to the outputs. Reaction names, reaction fluxes, metabolite names, metabolite fluxes were printed to the output window created by OutputFrame.java file.

14. OutputFrame.java: create “Output” frame to present results of the MetModelGUI. The outputs are provided from the Handler.java file such as lists of updated reactions, fluxes, metabolite names, metabolite fluxes, sources and escape names

15. ProteinNode.java. This file creates functions for adding protein names to the list of genes and checking whether the proteins are presented in the list of genes

16. ReactionDatabase.java: This file reads data in the reaction database, and reactions in a current model, adds new reactions and removes duplicated reactions, works with ReactionNode.java, and Model programs. It returns specific reactions, list of reaction nodes, metabolite nodes and length of reactions that will be used in analysis processes.

17. ReactionDetails.java: This program works for linking information of reaction nodes and metabolite nodes and updating the reaction information. The program displays reaction details in “Reaction Database” and “Current Model” frame. These following are the information: pathway, EC numbers, short reaction names, chemical reaction names, irreversible or revisable relations, pattern of reactions. This file will call sorttableTable program, when users click at header columns to sort data in columns. Fucntions in JInternalFrame class were applies to develop the frame and create layouts of the outputs in the frame.

18. ReactionNode.java: This file creates functions for making reaction nodes. Each reaction node contains path, ecNum, reaction name, irreversible and revisable directions, gene/protein names, metabolites names, coefficients, compartments, position, lower and upper bound, including functions for updating coefficients, metabolites, compartments, and reactions. This file creates some functions to check the data with the database data, print output of coefficients, irreversible or revisable reactions, reset bounds, lower bounds and upper bsound and knockout reaction the bounds

19. ReadGPRFrame.java: create “GPR Data” frame to provide functions to use for creating and updating frames such as buttons for browsing, adding, exiting, and layout of reactions, and adding constraints for analyzing gene-protein reactions. This program reads genes and proteins in gene-protein reactions and provides the information to the Handler file.

20. SortableTable.java: provide sort functions to order data displayed in program frames and model analysis. Indexes of rows are served as row numbers or model numbers of the database.

21. TransportFrame.java: create “Transports” frame, called by “Modify Transports” menu. The program allow user to add and remove metabolites from the analysis and to re-set lower and upper limits of bounds. A total number of metabolites will be printed to screen output when running this file.

22. VisualizationFrame.java: create “Visualization” frame to export a *-map.xml file containing essential genes and proteins to create pathways in CallDesigner program or in KEGG website. The functions in this program are to create layouts, buttons, and functions to create that file. This program applies functions in JInternalFrame to develop the frame as other frame programs do.

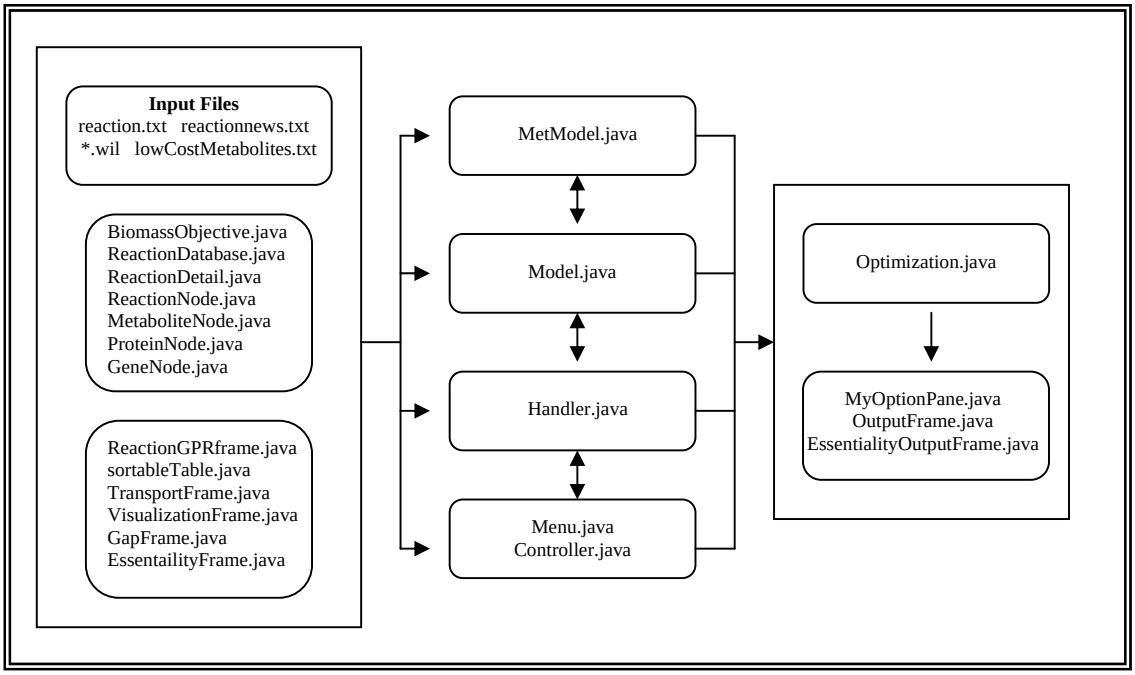

Figure 3. Summary of MetmodelGUI - based java framework

Figure 3 displays a general process of 22 java classes in MetModelGUI. MetModel.java was worked as a driver program and stated by importing information of reactions and metabolites in the databases to the process. Then the other programs in the same block of the information and the Model.java were called to built components of the model, while the Menu.java was functioned by users and send requirements to Controller.java and Handler.java to perform process as required. Meanwhile, Optimization.java was called to solve problems by Handler.java and then alternate optimal solutions were reported to a tab-delimited file by OutputFram.java.

This study is a continue project for MetModelGUI to enumerate alternate optimal solutions in which microorganisms make optimal use of their resources in metabolic pathways.

CHAPTER 2 ALGORITHM

2.1. Steps of an algorithm

To alternate optimal solutions, MetModelGUI was modified by adding another program, BouncerFrame.java. The BouncerFrame class was a subclass of MetModel, Menu, and Handler classes. In the MetModel class, we created an object to call the BouncerFrame class and implement two methods in this class to open and close the Bouncer window. In the Menu class, we added an object to create a Bouncer menu in the menu bar to access the BouncerFrame class to open the Bouncer window and implement methods in the BouncerFrame class. In the Handler class, after we created a Bouncer object, we created a Bouncer constructor as a module of Handler functions and added two methods to call open and cancel methods in the MetModel class.

In the BouncerFrame class, five public methods were created with names: BouncerFrame, init, initLayout, setFilename. These classes were used to construct templates of the Bouncer window and to access methods from other classes. A location of the output file and limitation of execute time was used to generate the solutions inputted by users. In addition, another three public methods were created to implement the simplex method and alternate optimal solutions named as: bouncers, getActiveReaction,

and actionPerformed. The dual simplex method by QS java language was applied to solve the problem. QS library was imported and used to provide a set of functions for creating, manipulating, and solving linear-programming problems (28).

In this study, there are two important parts. The first part is to get active reactions. The active reactions are defined as the set of reactions that can have non-zero flux. We evaluated the active reactions by using QSopt to maximize the flux through each reaction. If the optimal values are positive numbers, the reactions are active. If the optimal values equal zero, the reaction are inactive.. Then we removed the inactive reactions from the LP problem and reset a new problem. The new problem consists of the active reactions, a biomass reaction, sources or substrates, escapes or products, coefficients of metabolites in each reaction, and lower and upper bounds of solutions. We created matrix A, matrix c, matrix of lower and upper bounds. The matrix A of size mxn contained coefficients of metabolites (rows) for each reaction (columns). The matrix c of size $1xn$ contained coefficients of an objective function. In our study, our aim was to maximize the biomass reaction so the coefficient of biomass equal one and the others equal zero values. In other words, the optimal value (z) of the problem equals the optimal value of biomass reaction that is subject to each constraint and bounds. The lower and upper bounds were imported from the reaction database in the form of an input file or were entered by users to their matrices. The lower and upper bounds were determined by reversible and irreversible reactions. The upper bound of all reactions was limited at 1000, whereas the lower bound was -1000 for reversible reactions and zero for irreversible reactions. In the second part, that information was restructured to be a LP problem to import to the QSopt java

application. By the QSopt application, we obtained indexes of basic and non-basic reactions, indexes of zero-cost values, index of basic metabolites, and a set of optimal solutions. We began to enumerate alternate optimal solutions. We created another matrix containing coefficients of metabolites of basic reactions, matrix B. This is a non-singular matrix with the row and column sizes equal to the size of indexes of basic metabolites and indexes of basic reactions. Indexes of non-basic variables that reduced cost to zero were assigned for new entering variables in further iterations. Leaving variables were also defined as conditions in page 47. In each iteration, the list of entering variables, the lists of vectors of basic reaction indexes, the lists of solution vectors were pushed into three stack objects. We retrieved the last element of each stack and used it to implement the algorithm in each iteration until the stack was empty or the time ran out. After the iterations were stopped all alternate optimal solutions stored in a stack were printed to an output file. These following are implementation of the algorithm in BouncerFrame.java.

2.1.1. An Outline of the Algorithm.

1. The original biomass in the model is stored as a temporary reaction.
2. The original biomass in the model is removed from the model.
3. The temporary biomass reaction is added in the list of reactions.
4. Each reaction is transformed into an LP problem and each problem is solved at a time.
5. The reactions with non-zero flux are added in the list of active reaction.
6. The temporary biomass reaction is removed from the list of regular reactions.
7. The LP problem is restructured. The active reactions are included in the problem instead of all reactions. The new problem is solved to obtain a set of optimal solutions.
8. The indexes of basic rows and columns are stored.
9. The nonbasic variables with reduced cost zero are stored as possible entering variables.
10. A matrix A with coefficients of active reactions, biomass equation, sources, and escape is created.
11. A matrix c with coefficients of an objective function (biomass) is created.
12. Matrices of lower and upper limits of x solution are created.
13. The indexes of entering variables, lists of basic variables and solutions are added to stacks
14. The additional two stacks are created to store unique solutions and unique basis.
15. The process of iteration to alternate optimal solutions is started.
    - The last elements of each stack in 13 are popped out.

- The index of entering variable, lists of basic columns and solutions are prepared.
- A leaving variable in the basis is determined
    - A matrix B with the corresponding basic columns of A is created
    - A matrix 'a' with the corresponding column of entering variable of A is created.
    - Determine entering direction
    - Find the minimum t that the x + td is in the bounded interval. If there is such t in the basic variables, there is a leaving variable
- The list of x solutions and basic columns are updated for the entering and leaving variables
- Determine if the current basis and solution are the new basis or new solution.
- If the current basis is new, the current basis is added to their stack in 14.
- If the current solution is new, the current solution is added to their stack in 14.
- Determine if there are new entering variables or non-basis with reduced cost zero.
- If there are the new entering variables, they are added to the stack of entering variable.
- If there are the new entering variables, the current basis and new solution are added to their stack in 13.
- Iteration is executed until the stacks in 13 are empty or time ran out.

16. The list of solutions in 14 are printed to an output file

These following are all of the details of the algorithm

Bouncer() method:

1. Initialize run time and user input time to implement the algorithm.
2. Store biomass reaction ("BiomassTemp") to a string object
3. Delete the original biomass reaction using a removing method in the Handler class
4. Add biomass as a regular reaction by creating a reaction node and using an adding reaction method in Handler class
5. Get active reaction method: getActiveReaction();
6. Delete the temporary biomass reaction ("BiomassTemp")
7. Delete inactive reactions using the removing reaction method in the Handler class
8. Add original biomass equation back using set biomass method in Handler class
9. Solve new problem without inactive reactions

    Call QSopt functions in Optimization class to solve LP problem

10. Set a LP new problem

11. Assign an epsilon equal to 0.000001
12. Apply QSopt functions to obtain basic reaction indexes, basic metabolite indexes, and non-basic index with reduced cost zero for new entering variables.
13. Use indexes of basic metabolite to get name of basic metabolites
14. Set matrix B by the list of indexes of basic metabolites (r) and basic reactions (c)

```
For (each reaction)
    If (c [i] == '1')
        The indexes were added into the list of basic reaction indexes
    Else if (c [i] != '1')
        The indexes were added into the list of non-basic reaction indexes
For (each constraint)
    If (r [i] == '1')
        The indexes were added into the list of basic metabolite indexes
For (each element in rc [])                    //reduced cost zero
    If (abs(rc[i]) <= epsilon)
        The indexes were added into the list of reduced cost zero variables
For (each element in c [i] and rc [])
    If (col [i] != '1' && (abs(rc[i]) <= epsilon))
        The indexes were added into the list of entering variables
```

15. Get coefficients of active reactions in matrix A with columns for active reactions, containing biomass reaction and rows for basic metabolites.
16. Create two vectors for lower and upper bounds with column sizes equal to column sizes of matrix A.

```
For (each basic reaction)
        For (each basic metabolite)
            A[i][j] = 0.0;
For (each active reactions) {
        Set biomass equation for each reaction
        low[i] = lower bound of each reaction
        upp[i] = lower bound of each reaction
        names[i] =reaction names
        For (each metabolite) {
            Get metabolite names in the reaction
            Find indexes of basis metabolites by matching the metabolite name with
            the basic metabolite names.
            If (the index >= 0) {
                A[index][i] = get coefficient of the basic metabolite for this
                reaction
            }
        }
    }
```

17. Get coefficients of sources and escapes for Matrix A

```
    For (each source} {
        low[i] = lower bound of the source
        upp[i] = lower bound of the source
```

For (each metabolite) {

Find indexes of basis metabolites by the metabolite name and the basic metabolite names.

If (the index >= 0)

A[i][j] = 1.0;

}

An increment for a column size

}

For (each escape} {

low[i] = lower bound of the escape

upp[i] = lower bound of the escape

For (each metabolite) {

Find indexes of basis metabolites by the metabolite name and the basic metabolite names.

If (the index >= 0)

A[i][j] = -1.0;

}

An increment for a column size

}

18. Enumerate alternate optimal solution

For (each entering variable)

Push entering variable to a stack for entering variables (enterX)

For (each index of basic reaction)

```
        Add indexes of basic reactions to an array list (vParent_)
Push vParent_ to the stack of basic variables vParent
For (each x solution)
        Add x values to an array list (xSol_)
Push xSol_ to the stack of basic variables (xSol)
For (number of entering variable - 1) {
        Push xSol_ to xSol
        Push vParent_ to vParent
 }
For (each x solution)
        Add x values to an array list (solutions_)
Push solutions_ to the stack of alternate optima solutions (solutions).
For (each index of basic reaction)
        Add basic reaction indexes to an array list (vBases_)
Push vBases_ to the stack of unique basic reaction (vBases)

While (the stack of enter variable is not empty) {
        Pop the last element in the stack of index of entering variables (enter)
        If (vParent stack is not empty) {
                Pop the last vector in the stack of the indexes of basic variables (basic
                reactions)
                Convert the list of basic variable to an array (vB)
         }
```

```
If (xSol stack is not empty) {
        Pop the last vector in the stack of the x solutions (solutions)
       Convert the list of basic variable to an array (x)

//Create Matrix B by the list of vB
For (each row in Matrix A)
       For (each column in Matrix A)
              Bmatrix[i][j] = Amatrix[i][vB[j]]

// determine if entering variable is at lower or upper bound
If (absolute of differences of x values and its lower bound is less than epsilon)
        Lower bound = 1

// Find entering direction and solve Bd = a
For (each row in AMatrix)
        a[i][0] = AMatrix[i][entering variable]

// Find entering direction and solve Bd= a
   Convert the Bmatrix[][] to Matrix object call B_m
   Convert the a[][] to Matrix object call a_m
   Apply JAMA function to solve d
   Convert Matrix d to an array object d
```

```
 // Find the minimum t that x + td is less than ratio of upper to lower bounds
 Leave = enter
 Diff = upp[enter] - low[enter]
 For (each vB array) {
        If (absolute value of d < epsilon) {increment for-loop and skip else-if)
        Else if (lower bound & d > 0) then find t; x – lower bound = t d
        Else if (lower bound & d < 0) then find t; x – upper bound = t d
        Else if (upper bound & d > 0) then find t; upper bound – x = t d
        Else if (upper bound & d < 0) then find t; lower bound – x = t d
        If (t less than diff) {diff= t, leaving variable index = e, and increase m}
  }

// Update new X
For (each a in x solution) {
        If (a in vB_)
                If (lower bound) then newX = x – td
                Else newX  = x + td
        Else newX  = x
}
// Update xnew_ list
For (each newX)
        Add newX in an array list, newX_
```

```
// Compare basis lists and solution lists to their previous lists
If (enter = leave)
        Newbase == 0
Else {
        Find indexes of leaving variable in vB (b)
        Assign vB[b] = indexes of entering variable
        Update basic reaction index in Matrix B
        Sort elements of basis reaction array
        For (each row in Matrix A)
                B = A[i][entering variable]
        Copy stack of vBases to a new stack of (vBaseslists)
        While (vBaseslists is not empty) {
                Pop vBaseslist into an array list (vBaseslists_)
                For (each basic variable) {
                        Get new basic variables to basis []
                        If (basis [] = vB[]) {
                                Newbase = 0
                                Stop this while-loop
                         }
                }
        } // end while
}// end else
```

```
// Create a new list of new basic variable
For (each basic variable)
        Add vB in the new list (vB2_)

// Check new solutions
Copy stack of array list of solutions to a new stack (tempsols)
While (tempsols is not empty) {
        Pop the last vector of x solutions to array list (tempsols_)
        For (each x solution)
                Get element of x solution
        Convert newX[] to matrix object newX_m
        Convert solution[] to matrix object, solutions_m
        Find differences between newX_m and solutions_m
        Convert the difference to matrix objects and get array of the difference
        For (each column)
                Sum of absolute values of the difference
        If (Sum < epsilon) {
                New solution = 0;
                Stop while-loop;
        }
}
// Add new basic variables to vBases stack
If (new basis) {
```

```
        Push vB2_ to vBases stack
}
If (new solution)
        Push xnew_ to solutions stack
If (new basis or new solution) {
        // Solve  y; where yB = c
        Create matrix c;
        If (biomass index in vB_)
                c = 1.0;
        Convert B to matrix object B_m
        Convert c to matrix object c_m
        Use solve function in JAMA java class to find y
        Convert matrix y to array y
        For (each j column of A)
                If (absolute of y < epsilon)
                        Add j to an array list (zero_cost)
        For (each j column of A)
                If (j not in vB_)
                        If (j in zero_cost) {
                                Push j into enterX stack
                                Push vB_ into vParant stack
                                Push newX_ into xSol stack
                        }
```

```
            } // end if
            Call system current time function to take end time in seconds
            Calculate runtime if less then user-input time then stop while-loop
        } // end while-loop; where entering variable empty and runtime less than input time

//Print solutions stack to output file
Copy solutions stack to a new stack (h)
try {
    Initialize PrintWriter object and specify filename included organisms name and
    current date time of execution (file)
        Print variable names (reaction names)
        while (h is not empty){
            Pop h to an array list (k)
            For (each k)
                Print k element to the output file
        }
    }
    Close output file
    } catch (throw Exception) { }
 Print run time used and number of iterations.
```

Void getActiveReaction():

```
    For (each reaction)
```

Get each reaction (w) from the list of reactions

Set biomass equation for each w

Find optimal value (z) for each w

If z > epsilon then add w to a list of active reactions

Else if w is reversible reaction then reverse the reaction

Set biomass equation for each w

Find optimal value (z) for each w

If z > epsilon then add w to a list of active reactions

If z <= epsilon then add w to a list of inactive reactions

Void actionPerformed (Get event):

Get user command

If (“Browse”)

Get direction for the output file

Else if (“Run”)

Bouncer()

Else if (“Cancel”)

Close Bouncer Frame()

2.1.2. Program Testing

We checked the program in two parts. The first part is to check correctness of the method of getting active reactions and creating matrix A. We used print statements to extract information such as number of all reactions, sources or escapes, coefficients of biomass reactions, reaction and metabolite names when each statement was executed step-by-step before, during and after getting active reactions. We then export matrix A to oversee whether the reactions (variables in columns) correspond to the metabolites (constraints in rows). Each coefficient in the matrix was multiplied by itself and the results for each row and each column was summed to see whether there are any rows or columns containing all zero values. Results of checking this part was provided in the results section.

The second part is to test the algorithm in enumerating alternate optimal solutions. We applied two small LP problems that were manually solved on papers to test in the Bouncer program.

Example 1: Maximize $2x_1 + x_2$

Subject to $4x_1 + 2x_2 \leq 8$

$x_1 \leq 0 \quad , \quad x_2 \leq 0$

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 8 \\ 3 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 2 & 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{L} = [0 \quad 0 \quad 0 \quad 0]$$

$$\mathbf{X} = \begin{bmatrix} 2 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{U} = [10 \ 10 \ 10 \ 10]$$

Entering variable is x_2 and basis variables x_1 and x_3. The alternate optimal solution for this problem was $x = \begin{bmatrix} 1 & 2 & 0 & 0 \end{bmatrix}$

Example 2: Maximize $x_1 + -x_2 + x_3 + x_4$

Subject to $2x_1 + 2x_2 + 2x_3 + 2x_4 = 2$

$x_1 \geq 0 \qquad -1 \leq x_2 \leq 1$

$x_3 \geq 0 \qquad x_4 \geq 0$

$$\mathbf{A} = \begin{bmatrix} 2 & 2 & 2 & 2 & 1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 2 \end{bmatrix}$$

$$\mathbf{C} = [1 \ \ -1 \ \ 1 \ \ 1 \ \ 0] \qquad \mathbf{L} = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{X} = [0 \ \ -1 \ \ 0 \ \ 0 \ \ 4] \qquad \mathbf{U} = \begin{bmatrix} 4 & 1 & 4 & 4 & 0 \end{bmatrix}$$

Entering variable is x_1, x_3, x_4 and basis variables x_5. The alternate optimal solution for this problem were

$\mathbf{x} = [2 \ -1 \ 0 \ 0 \ 0]$ for entering variable x_1

$\mathbf{x} = [0 \ -1 \ 2 \ 0 \ 0]$ for entering variable x_3

$\mathbf{x} = [0 \ -1 \ 0 \ 2 \ 0]$ for entering variable x_4

2.2. Instructions for MetModelGUI

MetModelGUI have four main sections on menu bar: File, Build, Tool, and GPR. We can obtain information of biomass equations, reactions, sources and escapes of a microorganism using the open button. Save, save as, and exit buttons are available for user to save data file in other locations and exit program. The build button has functions to modify transport, set or update biomass equations, the run model (solve LP) to find optimal solutions and optimal values of biomass equation. The tool button contains options to view biomass equation, implement gap analysis, and create a file to visually evaluate metabolic pathways. The bouncer option was added in the Tool menu. Users can specify locations of solutions, and input time in seconds to execute the while-loop. The number of iterations of while-loop is depended on the input time, computer memory spaces, type and size of LP problems or number of variables and constraints. The GPR button has two options, read gene-protein reaction data (GPR) and delete some repeated reactions of genes and proteins. When a gene functions for more than one protein on its pathway users can select single or double deletions to analyze only a unique and essential pathway.

To enumerate alternate optimal solutions, users open an organism file (*.wil file) and may first run model to find the solution of all reactions to check if information in the *wil file is correctly structured. Then, users can open Bouncer window on Tool menu, specify location of the output file, input time of iterations in second and press run button. If not the specific location, the output file is located in the same directory as the organism file. Total execution time and total number of iterations of while-loop including total number of solutions can be obtained from the output filenames. Note that the current version of this program is unavailable for organism

having boundary metabolites. These metabolites have a linage across cell barriers which exist both inside and outside cells. In this version, we may convert those metabolites to the metabolites functioning outside cells and treat them as sources and escapes to balance equations to enumerate the alternate optimal solutions from the existing reactions.

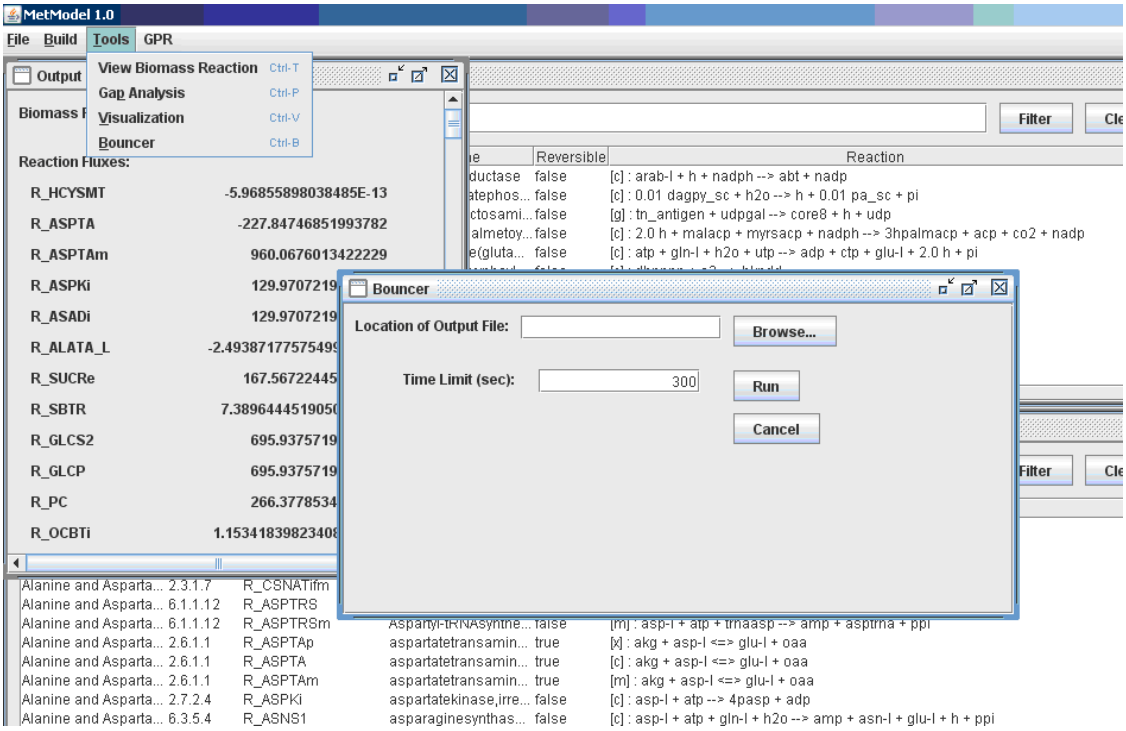

Figure 4. Bouncer window on MetModelGUI

CHAPTER 3 APPLICATION

3.1 Implementation

We applied MetModelGUI with the proposed “bouncer” algorithm to analyze the existing metabolic reconstructions with data available for five microorganisms: *trypanosoma cruzi*, *thermobifida fusca*, *helicobacter pylori*, *cryptococcus neoformans and clostridium thermocellum.* Data input files and results obtained from Java-based MetModelGUI are summarized in Table 3.

In analysis, principal component analysis (PCA) was used to reduce a number of variables (reactions) into a smaller number of artificial variables called principal components. The principal components can be defined as a linear combination of optimally-weighted variables as some of variables may be correlated with one another. The smaller number of principal components account for most of the variance in the data sets. The first components tend to explain relatively larger amounts of variance, while the later components are prone to explain relatively smaller amounts (37). The aim of the analysis is to present the reactions in a small dimensional space so that we can understand how variable the optimal solutions are among the reactions and the pathways. The analysis was preformed using proc factor statement with principal method and promax rotation in SAS 9.2 (SAS Institute, Cary, NC, USA). Number of components, eigenvalues, and rotated factor patterns were presented. The eigenvalue represents the

amount of variance that is accounted for by a given components. The rotated factor patterns are correlations between the variables and the factor. We imported the alternate optimal solutions obtained by MetModel and MetModelGUI and constructed the first and second principal components. Bi-plots of the first two components were created to describe linear combinations of the original variables to summarize a variety of the reactions and evaluate differences between the reactions. Reactions in the first principal component with rotated factor patterns higher than or equal 0.9; and less than or equal -0.9 were considered as important variables (reactions). The reactions at extreme ends of the bi-plots are highly variable among the alternate optimal solutions generated. The reactions with high correlations would be essential reactions that play critical rules in metabolic pathways of the microorganisms. In addition to the essential reactions, we evaluated reactions and pathways consistently remaining in a specific level or inactive while other reactions function, including flux directions of the essential reactions that were determined by mean values of the reactions in each solution group.

The data input file for an organism includes a biomass equation, a list of reactions, sources, and escapes. The data file was directly imported to MetModelGUI, except for *H. pylori* which contains a number of boundary metabolites. We converted those metabolites to extracellular metabolites and treated them as sources and escapes in the data input file. The *T. cruzi* data contained 146 reactions and *C. neoformans* contained 706 reactions. The other organisms had similar number of reactions in the input file. Each organism had fewer than 15 sources and escapes, except for *H. pylori* containing 74 sources and 74 escapes. The number of metabolites were little more than number of reactions. The *H. pylori* model produced the highest biomass flux (71.5) while

the *C. neoformans* model produced the lowest biomass flux (2.4). *T. cruzi, T. fusca and C. thermocellum* models provided biomass flux equal to 8.3, 15.7 and 11. The number of active reactions in *T. cruzi* and *H. pylori* model was not much different from the total number of reactions, while *T. fusc, C. neoformans,* and *C. thermocellum* models had high numbers of inactive reactions (196, 295, and 161, respectively). The size of matrix A (see Chapter 2) corresponds to the number of active reactions, sources, escapes, and basic metabolite indexes. The initial number of entering variables is the number of nonbasic reaction indexes that have reduced cost zero. We had 140 initial entering variables for *H. pylori* data, and 9 initial entering variables for *T. cruzi* data. The number of iterations for generating alternate optimal solutions may depend on the size of basic reaction indexes, basic metabolites indexes, and time input. Allowing the program longer time produces more iterations and solutions. In considering for number of iterations, *T. cruzi* data performed the highest iteration, whereas *H. pylori* data provided the lowest iteration. However, the *H. pylori* had the highest rate of generating new solutions (50.7%). Based on the same time interval, *T. fusca* data provided the most number of solutions. In 25-minute execution, *T. fusca, H. pylori, C. neoformans C. thermocellum* data provided 2547, 1185, 1436, and 1227 solutions. In the same memory capacities, *T. fusca* and *C. neoformans model* can be analyzed longer than *H. pylori and C. thermocellum models.* Those solutions are unrepeated and satisfied the lower and upper bounds on reaction fluxes. The values of $S * v$ for all models are equal or less than the epsilon value. Among five metabolic models, the *T. cruzi model* contained the highest number of turn-on reactions (54/131, 41.2%). The *C. neoformans* model contained 116 turn-on reactions (28.4%) and the other models contained 10-15%. In PCA results, 116 and 84 reactions in

C. neoformans and *H. pylori* data accounted for the first component, whereas the number of important reactions as defined was 28 and 43 reactions, respectively. *T. cruzi and C. thermocellum* data presented a few important reactions. More details can be seen in Tables 4, 6, 8, 10, and 12.

3.2. Validation of Implementation

The bouncer algorithm and alternate optimal solutions between MetModel and MetModeGUI were compared. We tested the two programs using the same data input file of *T. cruzi.* The same number of epsilon (0.000001) was used to protect distinct precisions due to rounding in floating point arithmetic in the bouncer algorithm and between the two programming languages. Numbers of all reactions, sources, escapes, metabolites were the same. Both biomass fluxes were equal to 8.265146. The solutions before getting active reactions in the two programs were different for 26 and 30 variables in twice testing. The Gurobi python module and QSopt java library may use different methods of selections of entering variables to the basis. The numbers of active reactions and sizes of matrix A were the same. The coefficients in matrix A in MetModel and MetModelGUI were correctly created for reactions, biomass, sources, escapes and their metabolites. We multiplied the first solution back to the coefficients in matrix A after we had active reactions and solved the LP problem to check whether $S_i * v_i$ of each constraint equal to zero as the condition set for the LP problem ($S_i * v_i$; where i = 0 to n, n is column size of matrix A). The summation of each constraint should be zero and we found all the summations in 120 constraints equal to zero in both programs. In finding alternate

optimal solutions, number of entering variables, basic reaction indexes, and basic metabolite indexes between the two programs were the same. MetModel provided 183 unique solutions from 24,182 iterations for ten minute of execution, while MetModelGUI provided 720 unrepeated solutions from 16,849 iterations within the seven-minute execution time. It seems that the MetModel may generate alternate optimal solutions faster than the MetModelGUI. All of the 183 solutions in MetModel appear in the 720 solutions in MetModelGUI. In the 183 solutions by MetModel, 38 reactions from 16 pathways were considered as important reactions by PCA analysis. Conversely, among 720 solutions by MetModelGUI, 5 reactions from 3 pathways were important reactions in *T. cruzi* metabolism. Possibly larger numbers of solutions may increase variations in the data on the factor spaces.

Table 3. Application of MetModelGUI for *Trypanosoma cruzi*. *Thermobifida fusca, Helicobacter pylori, Cryptococcus neoformans and Clostridium thermocellum*

Information	*T. cruzi*	*T. fusca*	*H. pylori*	*C.neoformans*	*C. thermocellum*
1. N of all reactions	146	423	479	706	560
2. N of sources	4	11	74	5	13
3. N of escapes	3	1	74	4	9
4. N of metabolites	160	485	485	839	569
5. Biomass flux	8.265146	15.722562	71.4681328	2.380329	11.360386
6. N of active reactions	124	286	413	400	399
7. Size of matrix A	120 x 131	252 x 298	383 x 561	354 x 409	350 x 420
8. Correctness of matrix A	Yes	Yes	Yes	Yes	Yes
9. SV of each constraint	0	0	0	0	0
10. N of entering variables	9	33	140	47	35
11. N of basic reaction indexes	120	252	383	354	350
12. N of basic metabolite indexes	120	252	383	354	350
13. N of iterations for 10 minutes	-	5298	1924	3931	3329
14. N of solutions for 10 minutes	-	2187	970	936	1014
15. Time at last execution (minutes)	6.8	86.5	25.7	50.7	26.1
16. N of iterations at last execution	16849	6684	2337	6891	4123
17. N of solutions at last execution	720	2792	1185	1698	1231
18. Solutions for each iteration	Unrepeated, in bounds	Unrepeated, in bounds	Unrepeated, in bounds	Unrepeated, in bounds	Unrepeated, in bounds
19. N of turn-on reactions	54	43	84	116	44
20. N of turn-off reactions at 0	37	87	200	237	110
21. N of turn-off reactions at a specific level	40	168	277	56	267
22. N of variable contribute to the first component	54	43	84	116	15
23. N of important reactions	5	16	43	28	5
24. N of pathway in #23	3	9	15	8	3

Note: Executions were stopped with out of memory for the results in No.15, 16, and 17, except for *T. cruzi* model which was completely executed.

Trends in obtaining new optimal solutions by the bouncer algorithm are different among the five complex metabolic models. All models are more likely to highly enumerate new optimal solutions in the beginning period of executions. The *T. cruzi* model seems to rapidly alternate optimal solutions in the first-fifteen seconds. The *H. pylori* and *C. thermocellum* models had highly producing new solutions in the first-ten minutes. Meanwhile, the *T. fusca* and *C.neoformans* models are more likely to enumerate well new solutions in the first-fifteen minutes. The rate of obtaining new solutions in the *T. cruzi, T. fusca H. pylori, C. thermocellum and C.neoformans* models are approximately 29, 3, 2, 2, and 1 solution per second. After the early period of executions, the identification of new solutions in the complex models tends to decrease. The number of new solutions was found less than those found in the early period. The *T. cruzi* model was completely executions, which can be used to represent the tendency of enumerating alternate optimal solutions in the bouncer algorithm. When no new solutions are generated, number of entering variables in the stack collection is decreased until the collection is empty. Concurrently, the bouncer algorithm in the other models presents new solutions are consistently generated when new entering variables are consistently added into the collection. In addition, we found number of iterations corresponds to the time of execution. The number of new solutions frequently increases by numbers of iterations for all models. More details of these results can be seen in Figure 5-9.

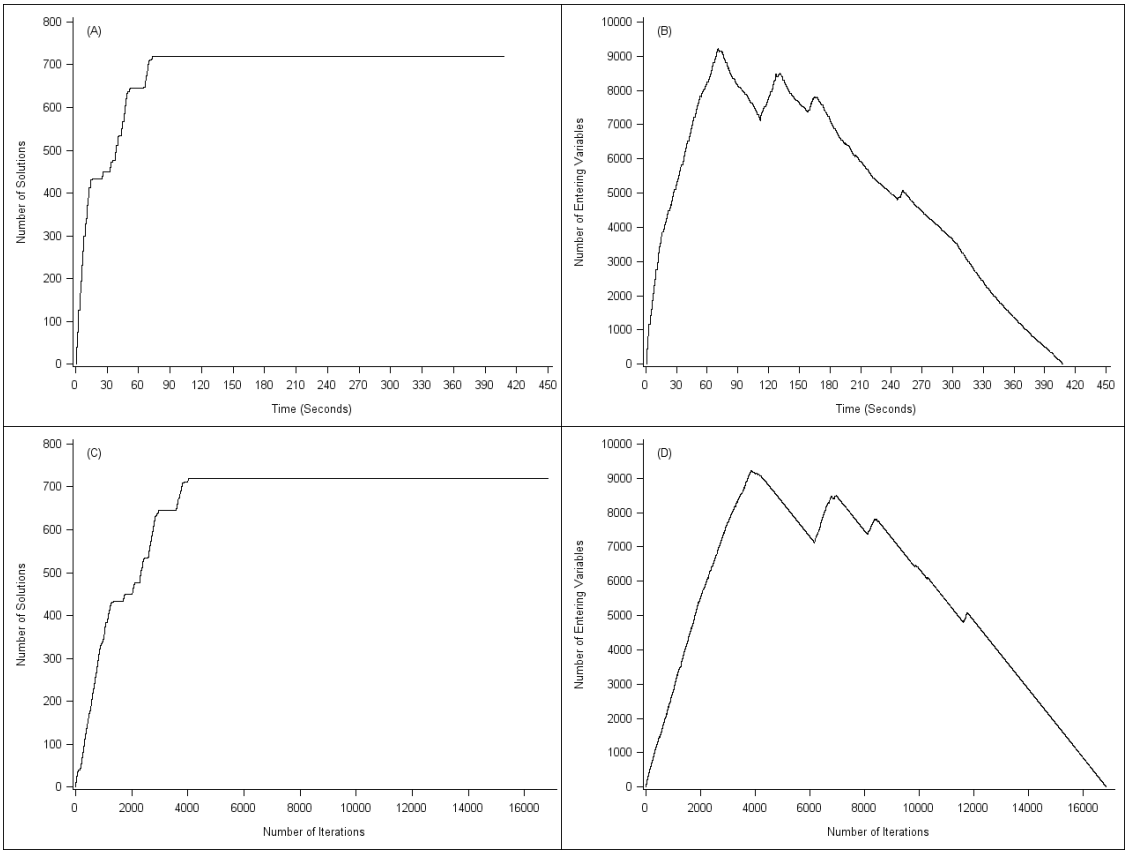

Figure 5. Line plots of number of solutions and number of entering variables over execution time (A and B) and number of iterations (C and D) in *Trypanosoma cruzi* model analysis

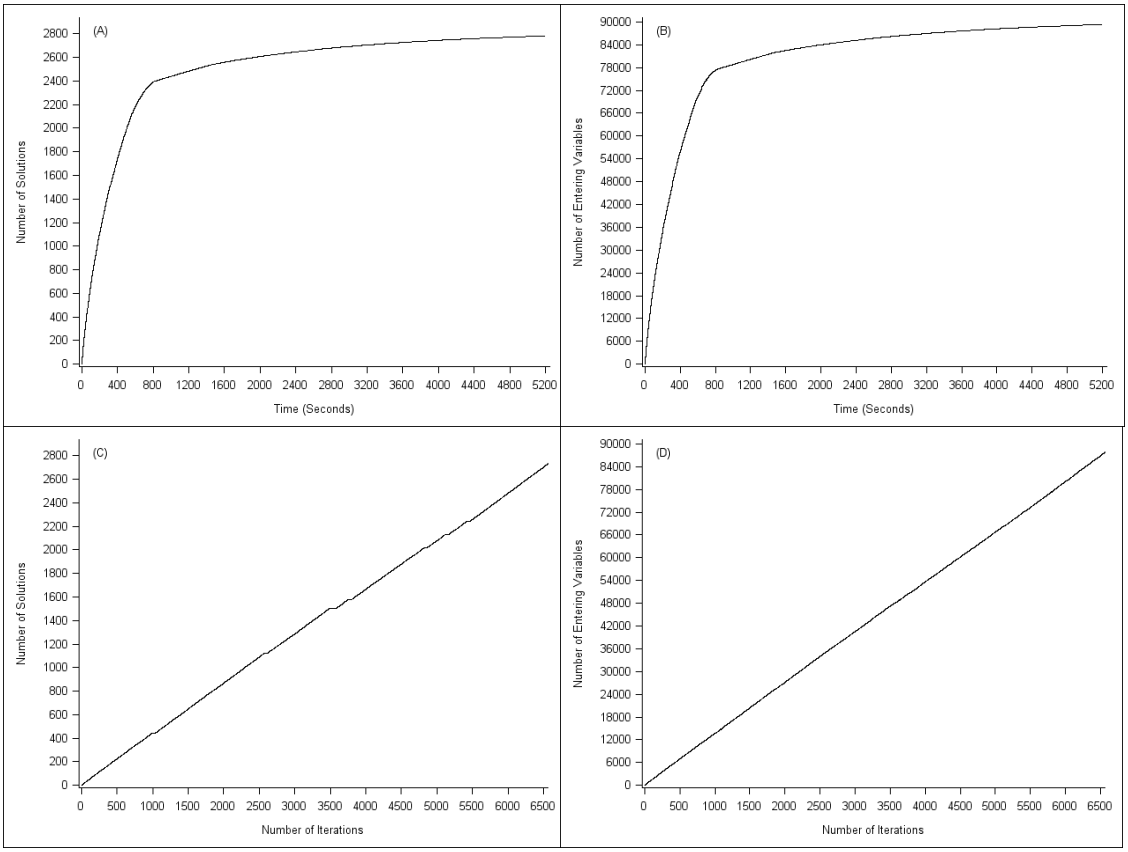

Figure 6. Line plots of number of solutions and number of entering variables over execution time (A and B) and number of iterations (C and D) in *Thermobifida fusca* model analysis

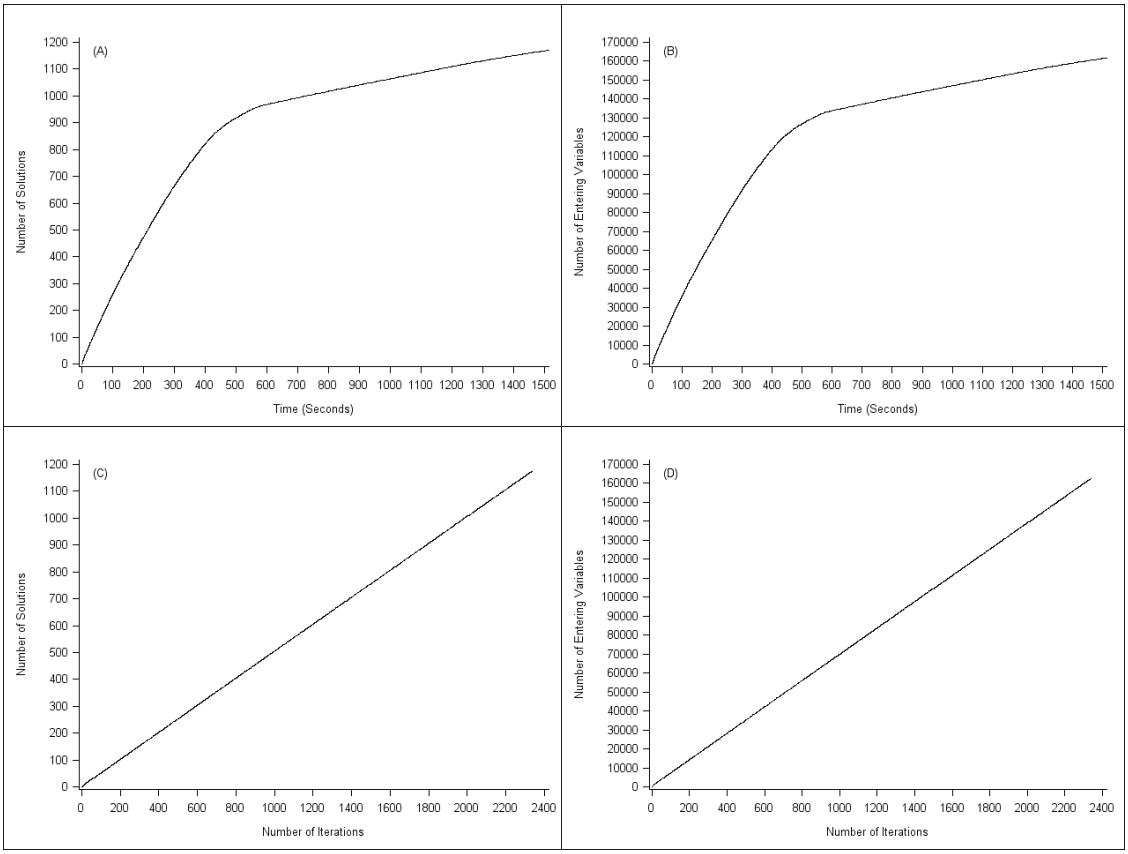

Figure 7. Line plots of number of solutions and number of entering variables over execution time (A and B) and number of iterations (C and D) in *Helicobacter pylori* model analysis

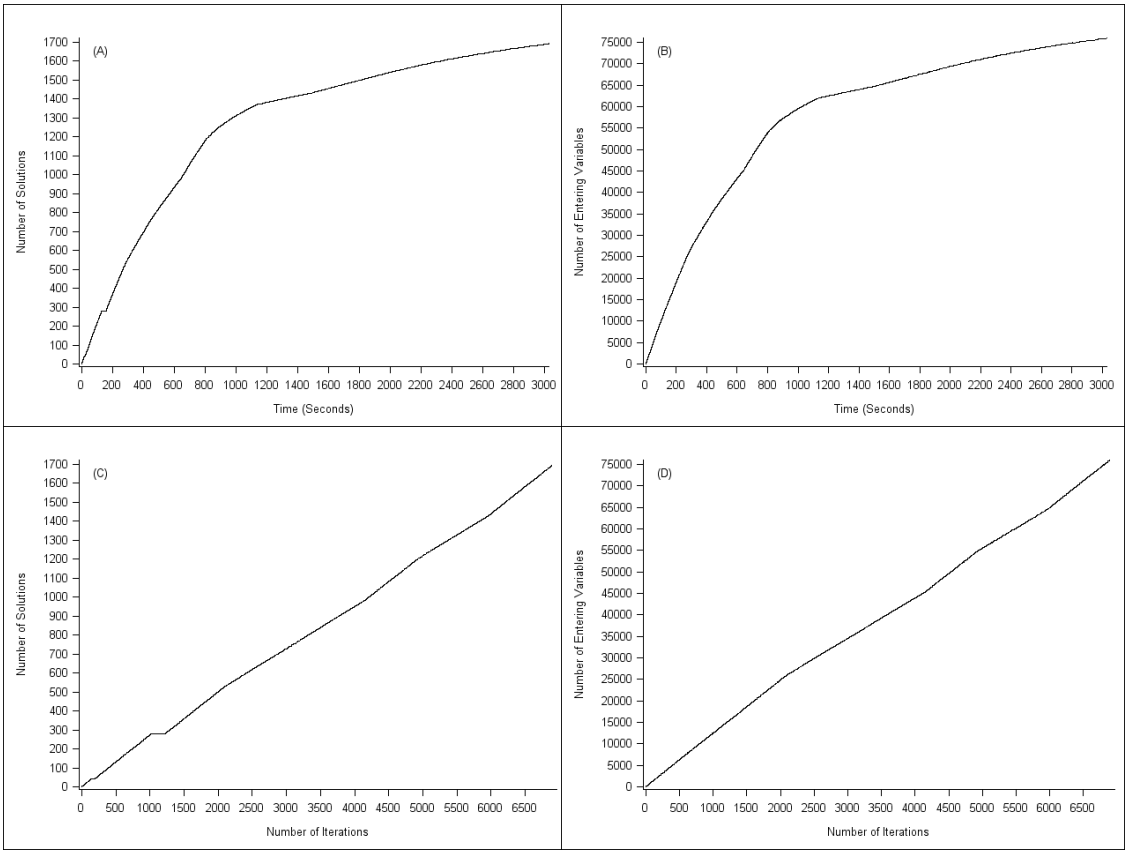

Figure 8. Line plots of number of solutions and number of entering variables over execution time (A and B) and number of iterations (C and D) in *Cryptococcus neoformans* model analysis

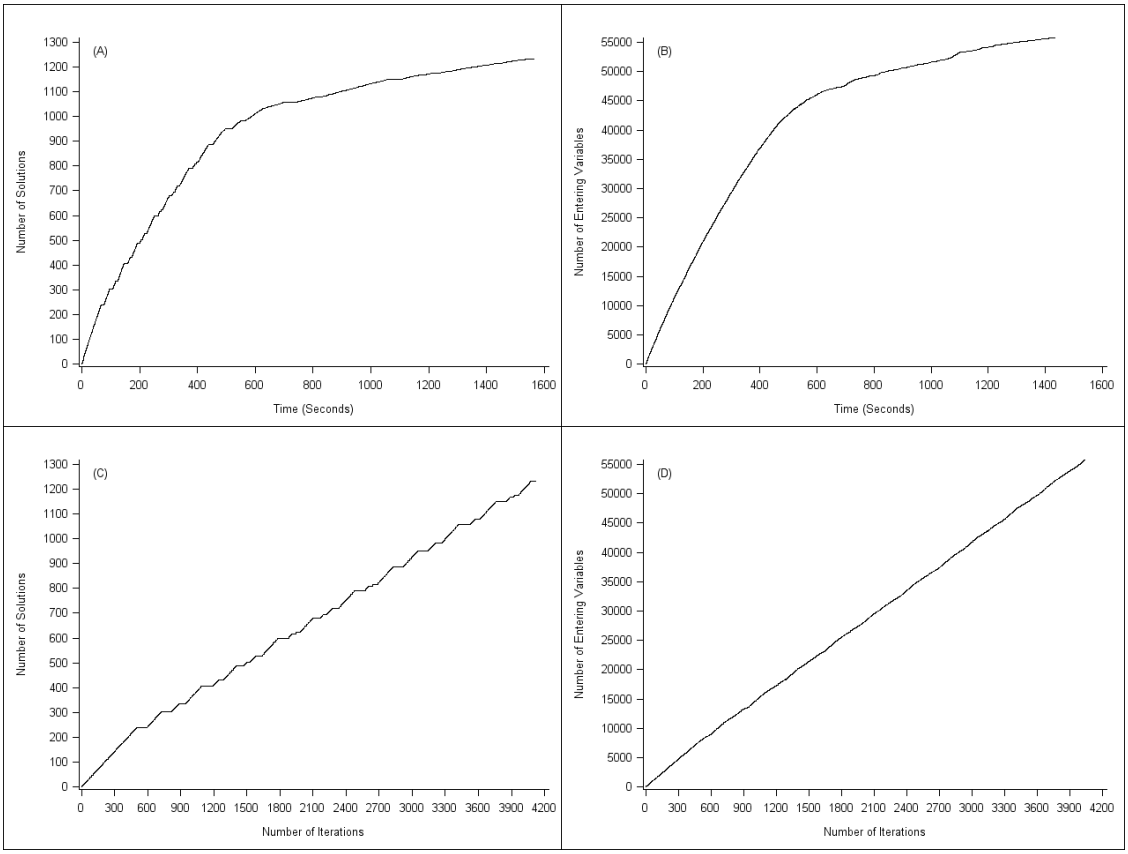

Figure 9. Line plots of number of solutions and number of entering variables over execution time (A and B) and number of iterations (C and D) in *Clostridium thermocellum* model analysis

3.3 Detailed Analysis of Alternate Optimal Solutions for Five Microorganisms

Alternation optimal solutions generated by MetModelGUI with complete executions for *T cruzi* model and at last executions for *T. fusca, H. pylori, C. neoformans and C. thermocellum* model were analyzed by PCA. The PCA results are presented in details as follows:

1. Eigenvalues and proportions of variance accounted for
2. List of essential reactions and their pathways
3. List of reactions with very little variability
4. Characters of solutions obtained by PCA analysis
5. Directions of essential reactions on the characters of solutions

1. Trypanosoma cruzi

Trypanosoma cruzi is a species of parasitic euglenoid trypanosomes. *T.cruzi* is known as a cause of the Chagas' disease, which is a major public health problem in endemic countries. Understanding the biology and biochemistry of *t.cruzi* can help improvement antichagasic agents to treat the Chagas's disease (38).

There were nine components with an eigenvalue greater than one. The eigenvalues for the component 1, 2, and 3 were 15.023, 9.656, and 6.284. The last component displays an eigenvalue of 2.428. The first, second, and third components account for 27.8%, 17.9% and 11.6% of the total variance. The eigenvalues and the proportions of variance accounted for in this analysis appear in Figure 10.

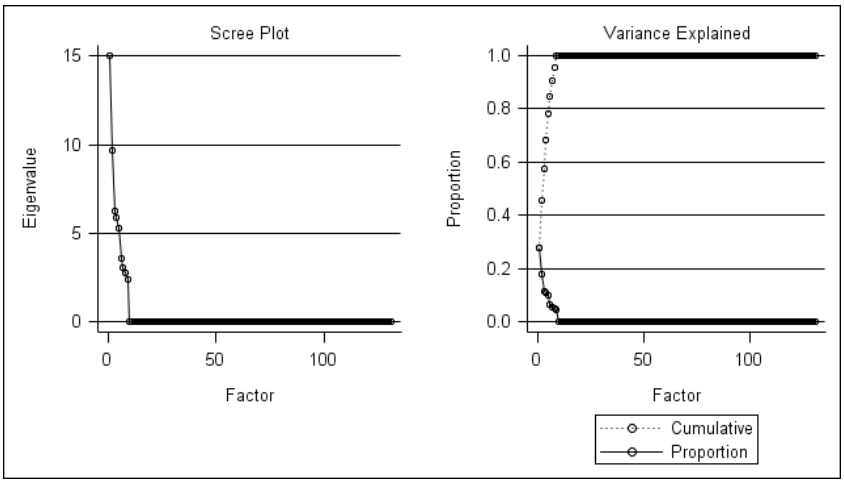

Figure 10. Scree plot of eigenvalues from principal component analysis of 124 active reactions, 4 sources, and 3 escapes in the *Trypanosoma.cruzi* model

The following are associations of reactions in the first and second principal components.

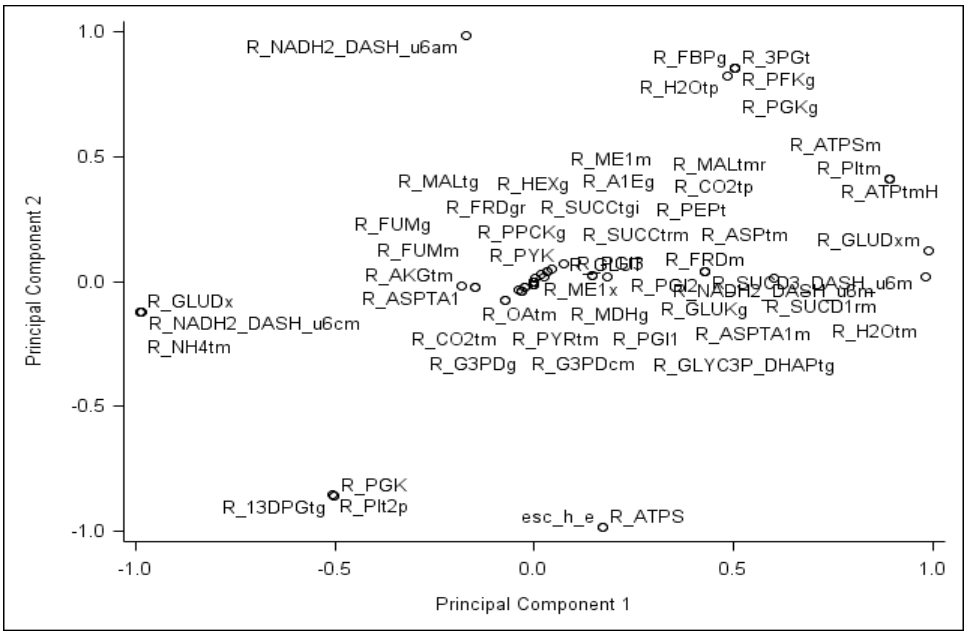

Figure 11. Bi-plot of the rotated factor pattern in the first and second components of the metabolic reactions in *Trypanosoma cruzi* by alternate optimal solutions conducted by MetModelGUI.

In Figure 11, we can see the metabolic reconstruction of *T.cruzi* consists of many metabolic reactions (54 reactions from 30 pathways). Reactions correlated to other reactions were closely presented to same dimensions on the factor spaces e.g. R_13DPGtg, R_PGK, and R_PIt2p. It seems that there are six groups of reactions. The reactions in each group may have some relationships or functions together in the pathways. The reactions presented in the figure exist in these pathways: ATP synthesis, alanine and aspartate metabolism, citric acid cycle, glutamate metabolism, glycolysis and gluconeogenesis, oxidative phosphorylation, pyruvate metabolism, and transportation in extracellular, glycosomal and mitochondrial.

Table 4. Essential reactions and their pathways in *Trypanosoma cruzi* in the first principal component by alternate optimal solutions generated from MetModelGUI

Pathways	Reaction names (Enzymes)	Reactions	Rotate factor Pattern
Glutamate metabolism	Glutamatedehydrogenase (NAD)	R_GLUDx	-0.98677
Glutamate metabolism	Glutamatedehydrogenase (NAD) (Mitochondrial)	R_GLUDxm	0.98677
Oxidative Phosphorylation19	NAD hdehydrogenase	R_NADH2_DASH_u6cm	-0.98490
Transport, Mitochondrial	H_2O transport in mitochondrial	R_H2Otm	0.97994
Transport, Mitochondrial	NH3 mitochondrial transport	R_NH4tm	-0.98677

Two reactions were found at the extremely right side: R_GLUDxm, and R_H2Otm, and three reactions found at the extremely left side: R_GLUDx, R_NADH2_DASH_u6cm, and R_GLUDx. These reactions may provide the major energy sources of metabolism for *T.cruzi* (Figure 11, Table 4). Our results are similar to a study that suggested some glutamate metabolic pathways could effect on survival of *T.cruzi* as the glutamate pathways were inhibited, which led to stress conditions such as

nutritional starvation and oxidative stress. Specially, NADP+-link glutamate dehydrogenase was a catalyst of the activities (39).

We also listed the reactions with very little variability in the first component in Table 5. These reactions may relate to the reactions with high variability in Table 4.

Table 5. Reactions with very little variability in *Trypanosoma cruzi* model

Pathways	Reaction names (Enzymes)	Reactions
Citric Acid Cycle19	Fumarase,mitochondrial	R_FUMm
Glycolysis/Gluconeogenesis11	Aldose1epimerase-Likeprotein	R_A1Eg
Glycolysis/Gluconeogenesis12	Glucokinase	R_GLUKg
Glycolysis/Gluconeogenesis13	Hexokinase(D-Glucose:atp)	R_HEXg
Glycolysis/Gluconeogenesis15	Glucose-6-Phosphateisomerase,glycosome	R_PGI1
Glycolysis/Gluconeogenesis16	Glucose-6-Phosphateisomerase,glycosome	R_PGI3
Glycolysis/Gluconeogenesis17	Glucose-6-Phosphateisomerase,glycosome	R_PGI2
Glycolysis/Gluconeogenesis46	Fumarase,glycosome	R_FUMg
Glycolysis/Gluconeogenesis47	Fumaratereductase,nad,glycosome	R_FRDgr
Transport, Glycosomal	Cytoplasm2glycosome,succ	R_SUCCtgi
Transport, Mitochondrial	Cytoplasm2mitochondrion,succ	R_SUCCtrm

Note: Rotated pattern factor < 0.01 or > - 0.01 and not included the zero value.

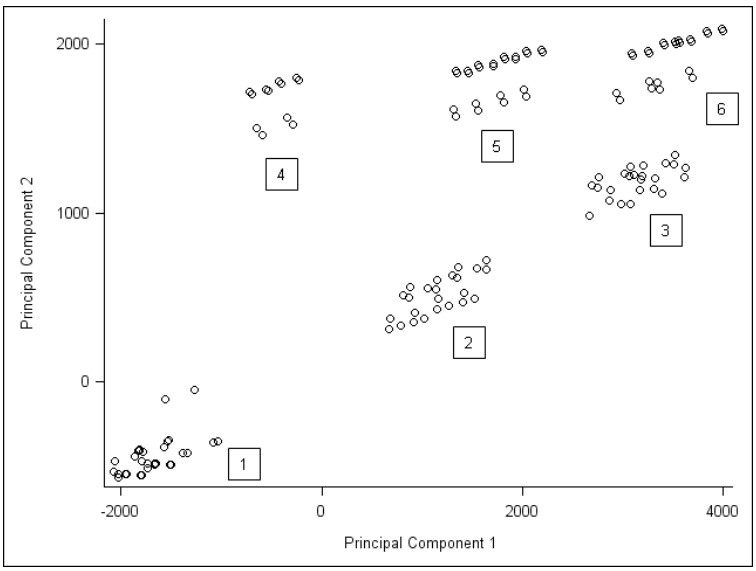

Figure 12. Bi-plot of PCA scores on the first two principal components for 720 solutions in the *Trypanosoma cruzi* model.

Table 6. Flux directions of five essential reactions for each solution group in Figure 12.

Essential Reactions	Solution Groups					
	1 (n = 180)	2 (n = 120)	3 (n = 110)	4 (n = 60)	5 (n = 100)	6 (n = 100)
R_GLUDx	586.68	222.13	-35.04	546.35	222.13	14.76
R_GLUDxm	-44.21	320.34	577.51	-3.88	320.34	527.71
R_H2Otm	-699.26	-428.73	-233.56	-729.16	-465.73	-297.69
R_NADH2_DASH_u6cm	623.87	255.90	0.00	580.12	255.90	48.07
R_NH4tm	44.21	-320.34	-577.51	3.88	-320.34	-527.71

Note: Mean values of each solution group were presented

It seems that the PCA scores of the first two components of 720 solutions in the *T cruzi* model were categorized to six groups (Figure 12). The flux directions of all the five essential reactions have the same high and low fluxes in Group 1 and 4; Group 2 and 5. Four in five essential reactions have the same fluxes in Group 3 and 6, except for R_GLUDx presenting low fluxes in Group 3 but high fluxes in Group 6 (Table 6).

2 Thermobifida fusca

Thermobifida fusca is a rod shaped and thermophilic organism found in decaying organic matter and is a major degrader of plant cell wall. Its former name is Thermomonaspora fusca. This organism produces multiple extracellular enzymes for the decomposition of cellulose and lignocellulose residues, which are important for the breakdown of agricultural and urban wastes. *T. fusca* usually presents in plant materials and bioplolymer substrates of natural origin. It contributes to the environment by decomposing organic matter (40, 41, 42).

There were 14 components with an eigenvalue greater than zero, and of these, 12 components with an eigenvalue greater than one. The eigenvalues for the component 1, 2, and 3 were 16.628, 4.015, and 3.261. The last component displays an eigenvalue of 0.305. The first, second, and third components account for 38.7%, 9.3%, and 7.6% of the total variance. The eigenvalues and the proportions of variance account for in this analysis appear in Figure 13.

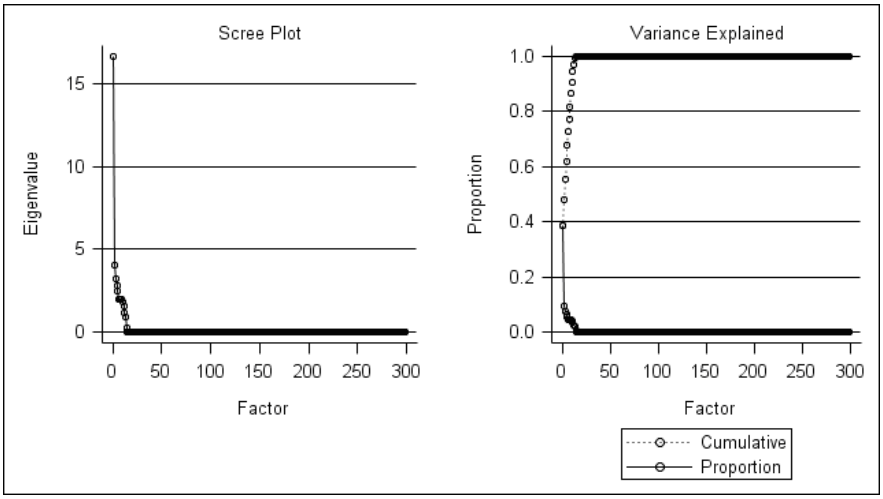

Figure 13. Scree plot of eigenvalues from principal component analysis of 286 active reactions, 11 sources, and 1 escapes in the *Thermobifida.fusca* model

These following are associations of the reactions in the first and second principal components.

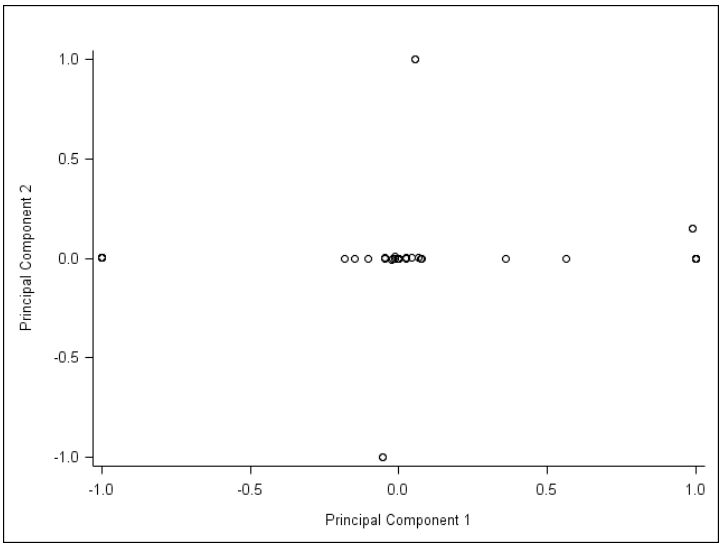

Figure 14. Bi-plot of the rotated factor pattern in the first and second components of the metabolic reactions in *Thermobifida fusca* by alternate optimal solutions conducted by MetModelGUI.

The first two principal components of *T. fusca* metabolic reactions can be partitioned into six groups. There are 43 reactions from 15 pathways contributed to the components. The majority of the reactions with low correlations to the first factor are nucleotides: cytidylatekinase, nucleoside-diphosphatase, and umpkinase.

Table 7. Essential reactions and their pathways in *Thermobifida fusca* in the first principal component by alternate optimal solutions generated from MetModelGUI

Pathways	Reaction names (Enzymes)	Reactions	Rotate factor Pattern
Alanine and Aspartate Metabolism	Argininosuccinatelyase	R_ARGSL	-0.99973
Alanine and Aspartate Metabolism	Aspartatetransaminase	R_ASPTA	0.999728
Amino Acid Metabolism	Acetylornithinetransaminase	R_ACOTA	-0.99973
Amino Acid Metabolism	Acetylglutamatekinase	R_ACGK	0.999728
Arginine and Proline Metabolism	Ornithinetransacetylase	R_ORNTAC	0.999728
Arginine and Proline Metabolism	Argininosuccinatesynthase,reversible	R_ARGSSr	-0.99973
Arginine and Proline Metabolism	Ornithinetransaminase	R_ORNTA	0.999728
Arginine and Proline Metabolism	1-Pyrroline-5-Carboxylatedehydrogenase	R_P5CD	0.999728
Argininedeiminase	Argininedeiminase	R_ARGDr	-0.99973
Central Metabolism	Pyruvate,phosphatedikinase.	R_PPDK	0.988718
Citric Acid Cycle	Malatedehydrogenase	R_MDH	-0.99973
Citric Acid Cycle	Fumarase	R_FUM	-0.99973
Glutamate metabolism	Glutamatedehydrogenase(Nadp)	R_GLUDy	0.999728
Glycolysis/Gluconeogenesis	Pyruvatekinase	R_PYK	0.988718
Urea Cycle	N-Acetyl-G-Glutamyl-Phosphatereductase	R_AGPR	-0.99973
Urea cycle/amino group metabolism	L-Glutamate5-Semialdehydedehydratase(Spontaneous)	R_G5SADs	0.999728

Of the 43 reactions, 16 reactions from 9 pathways were the important reactions in *T. fusca* metabolism. These pathways of the reactions were in the extremely right side: alanine and aspartate metabolism, amino acid metabolism, central metabolism, glutamate metabolism, glycolysis/gluconeogenesis, and urea cycle/amino group metabolism; and in the extremely left side: alanine and aspartate metabolism, amino acid metabolism, arginine and proline metabolism, citric acid cycle, and urea cycle (Figure 9, Table 7). As described, source of energy of *T.* fusca may depend on several metabolic pathways as said in the study of proteomic analysis of *T fusca* for metabolic pathways of cellulose utilization (43).

Below the list of reactions with very little variability was in the first component. This information may be useful to evaluate how the reactions and pathways function.

Table 8. Reactions with very little variability in *Thermobifida fusca* model

Pathways	Reaction names (Enzymes)	Reactions
Fatty Acid Metabolism	Formatedehydrogenase	R_FDH
Glyoxylate Metabolism	Formatedehydrogenase	R_FDHr
IMP Synthesis 7	Phosphoribosylaminoimidazolecarboxylase	R_AIRC2r
Methionine Metabolism	Adenosylhomocysteinase	R_AHCi
Methionine Metabolism	Adenosylhomocysteinase	R_AHC

Note: Rotated pattern factor < 0.01 or > - 0.01 and not included the zero value.

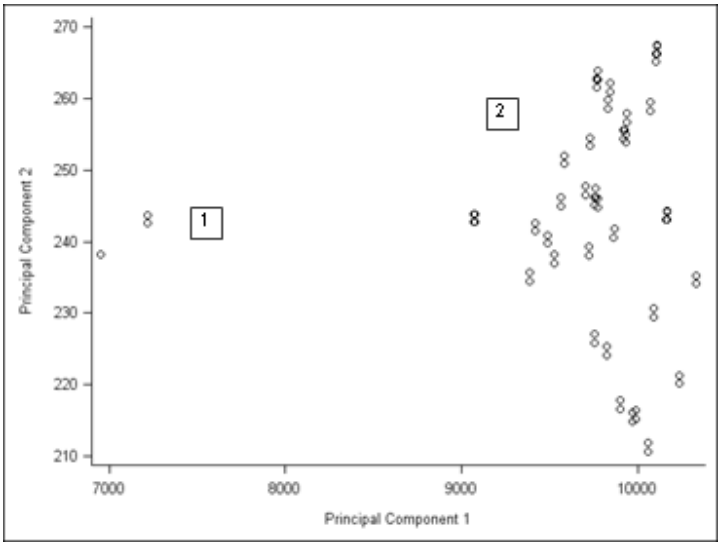

Figure 15. Bi-plot of PCA scores on the first two principal components for 2792 solutions in the *Thermobifida fusca* model.

Table 9. Flux directions of 16 essential reactions for each solution group in Figure 15.

Essential Reactions	Group 1 (n = 24)	Group 2 (n = 2768)
R_ACGK	16.00	21.53
R_ACOTA	-16.00	-21.53
R_AGPR	-16.00	-21.53
R_ARGDr	-994.47	-1000.00
R_ARGSL	-991.69	-997.22
R_ARGSSr	-991.69	-997.22
R_ASPTA	972.65	978.18
R_FUM	-979.26	-984.79
R_G5SADs	13.22	18.76
R_GLUDy	895.79	901.33
R_MDH	-979.26	-984.79
R_ORNTA	13.22	18.76
R_ORNTAC	16.00	21.53
R_P5CD	11.07	16.60
R_PPDK	871.73	877.31
R_PYK	833.47	839.05

Note: Mean values of each solution group were presented

In figure 15, 2792 solutions for in the *T fusca* model were categorized to two groups by PCA scores of the first two components. Flux directions of all the 16 essential reactions in the two groups have the same fluxes directions: 9 high fluxes and 17 low fluxes (Table 9).

3 Helicobacter pylori

Helicobacter pylorus is a gram-negative, microaerophilic bacterium that causes chronic inflammation of the stomach or the peptic ulcers diseases in human. This bacterium is transmitted by ingesting contaminated food and water from *H pylorus* .infected people. It requires oxygen at low concentration. It contains a hydrogenase which can be used to obtain energy by oxiding molecular hydrogen produced by intestinal bacteria. It also produces oxidase, catalase, and urease and capable of forming biofilms (44).

There were 24 components with an eigenvalue greater than zero and of these, 10 components had an eigenvalue greater than one. The eigenvalues for the component 1, 2, and 3 were 48.607, 9.475, and 4.925. The last component displays an eigenvalue of 0.156. The first, second, and third components account for 57.9%, 11.3%, 5.9% of the total variance. The eigenvalues and the proportions of variance accounted for from this analysis appear in Figure 16.

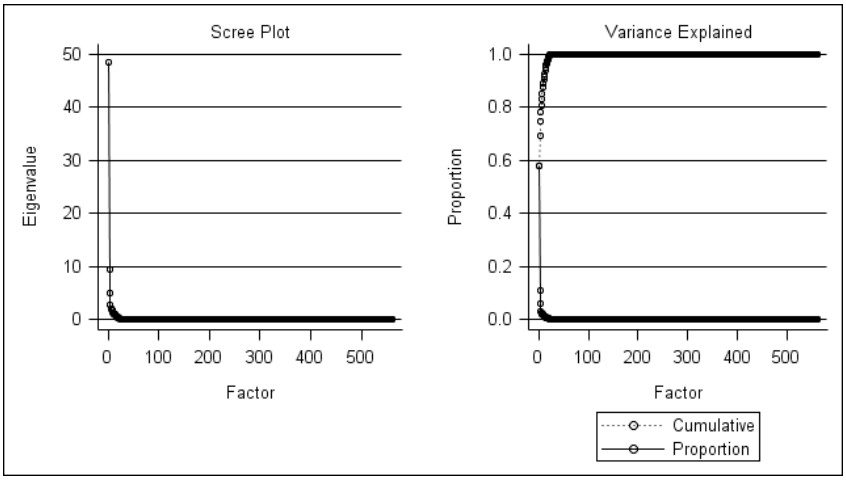

Figure 16. Scree plot of eigenvalues from principal component analysis of 413 active reactions, 74 sources, and 74 escapes in the *Helicobacter pylorus* model

Figure 17 are associations of the reactions in the first and second principal components. In *H. pylorus* data, we found 37 reactions, 20 sources and 27 escapes contributed to the principal components. 35 reactions, 4 sources and 4 escapes were considered as important metabolic reactions, substrates and products in *H. pylorus* metabolism (Table 8). The 35 reactions were in these pathways: alanine and aspartate metabolism, citric acid cycle, glutamate metabolism, glycine and serine metabolism, glycolysis/gluconeogenesis, pentose phosphate pathway, respiratory chain, respiratory chain, tricarboxylic acid (TCA), tryptophan metabolism, transport, and extracellular transport. This study found some reactions involved in *H. pylorus* metabolism as similar as reported in the study of citric acid cycle in *H. pylorus*: aconitase, isocitratede hydrogenase (HADP), fumarase, 2-oxoglutaratereversible transport via symport, NADH dehydrogenase (menaquinone) (Complex), malate dehydrogenase (menaquinone-6 acceptor), succinate, and fumarate antiporter (45).

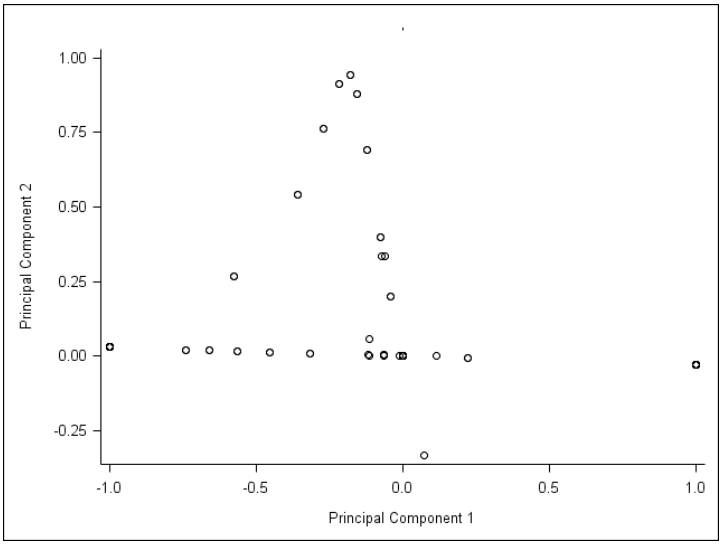

Figure 17. Bi-plot of the rotated factor pattern in the first and second components of the metabolic reactions in *Helicobacter pylorus* by alternate optimal solutions conducted by MetModelGUI.

Table 10. Essential reactions and their pathways in *Helicobacter pylorus* in the first principal component by alternate optimal solutions generated from MetModelGUI

Pathways	Reaction names (Enzymes)	Reactions	Rotate factor Pattern
Tryptophan metabolism	Acetyl-Coac-Acetyltransferase	R_ACACT1r	-0.9993
Transport, Extracellular	Acetoacetatetransportviaprotonsymport	R_ACACt2	0.9993
Citric Acid Cycle	Aconitase	R_ACONT	0.9993
Transport, Extracellular	2-Oxoglutaratereversibletransportviasymport	R_AKGt2r	-0.9993
Alanine and Aspartate Metabolism	L-Aspartase	R_ASPT	0.9993
Alanine and Aspartate Metabolism	Aspartatetransaminase	R_ASPTA	-0.9993
Citric Acid Cycle	Citratesynthase	R_CS	0.9993
Transport, Extracellular	L-Cysteinetransportviaabcsystem	R_CYSabc	0.9993
Transport, Extracellular	L-Cysteine/L-Glutaminereversibleexchanger	R_CYSGLUexR	-0.9993
Pentose Phosphate	2-Dehydro-3-Deoxy-Phosphogluconatealdolase	R_EDA	0.9993
Pentose Phosphate	6-Phosphogluconatedehydratase	R_EDD	0.9993
TCA cycle 6	Fumaratereductase	R_FRD5	0.9993
Respiratory chain	Frdo	R_FRDO	0.9993
Citric Acid Cycle	Fumarase	R_FUM	0.9993
Transport	Fumaratetransportoutviaprotonantiport	R_FUMt3	0.9993

Table 10. Essential reactions and their pathways in *Helicobacter pylorus* in the first principal component by alternate optimal solutions generated from MetModelGUI (cont.)

Pathways	Reaction names (Enzymes)	Reactions	Rotate factor Pattern
Pentose Phosphate	Glucose6-Phosphatedehydrogenase	R_G6PDH2	-0.9993
Glutamate metabolism	Glutaminesynthetase	R_GLNS	-0.9993
Pentose Phosphate	Phosphogluconatedehydrogenase	R_GND	-0.9993
Respiratory chain	Hyda	R_HYDA1	0.9993
Citric Acid Cycle	Isocitratedehydrogenase(Nadp)	R_ICDHyr	0.9993
Transport, Extracellular	L-Malatereversibletransportviaprotonsymport	R_MALt2r	0.9993
TCA cycle 8	Malatedehydrogenase (Menaquinone6asacceptor)	R_MDH4	0.9993
Respiratory chain 1	Nadhdehydrogenase(Menaquinone) (Complexi)	R_NDH_DASH _1	-0.9993
TCA cycle 5	3-Oxoacidcoa-Transferase(Succinyl-Coa:acetoacetate)	R_OCOAT1	0.9993
TCA cycle 4	Ferredoxinoxidoreductase	R_OOR	0.9993
Glycolysis/Gluconeogenesis	Glucose-6-Phosphateisomerase	R_PGI	0.9993
Pentose Phosphate	_6-Phosphogluconolactonase	R_PGL	-0.9993
Pentose Phosphate	Ribulose5-Phosphate3-Epimerase	R_RPE	-0.9993
Pentose Phosphate	Ribose-5-Phosphateisomerase	R_RPI	0.9993
Glycine and Serine Metabolism	L-Serinedeaminase	R_SERD_L	-0.9993
Transport, Extracellular	L-Serinereversibletransportviaprotonsymport	R_SERt2r	-0.9993
Transport, Extracellular	Succinate:fumarateantiporter	R_SUCFUMt	0.9993
Pentose Phosphate	Transaldolase	R_TALA	-0.9993
Pentose Phosphate	Transketolase	R_TKT1	-0.9993
Pentose Phosphate	Transketolase	R_TKT2	-0.9993
Escape		R_ESC_fum_e	-0.9993
Escape		R_ESC_mal_l_e	-0.9993
Escape		R_ESC_ser_l_e	0.9993
Escape		R_ESC_succ_e	0.9993
Source		R_SRC_acac_e	0.9993
Source		R_SRC_akg_e	-0.9993
Source		R_SRC_gln_l_e	0.9993
Source		R_SRC_h2_e	0.9993

Below the list of reactions with very little variability was in the first component.

Table 11. Reactions with very little variability in *Helicobacter pylorus* model

Pathways	Reaction names (Enzymes)	Reactions
Transport, Extracellular	Notransport (Diffusion)	R_NOt
Source		R_SRC_lys_l_e
Source		R_SRC_met_l_e
Source		R_SRC_thymd_e
Escape		R_ESC_lys_l_e
Escape		R_ESC_met_l_e
Escape		R_ESC_ni2_e
Escape		R_ESC_no_e
Escape		R_ESC_thm_e
Escape		R_ESC_thymd_e
Escape		R_ESC_uri_e

Note: Rotated pattern factor < 0.1 or > - 0.1 and not included the zero value.

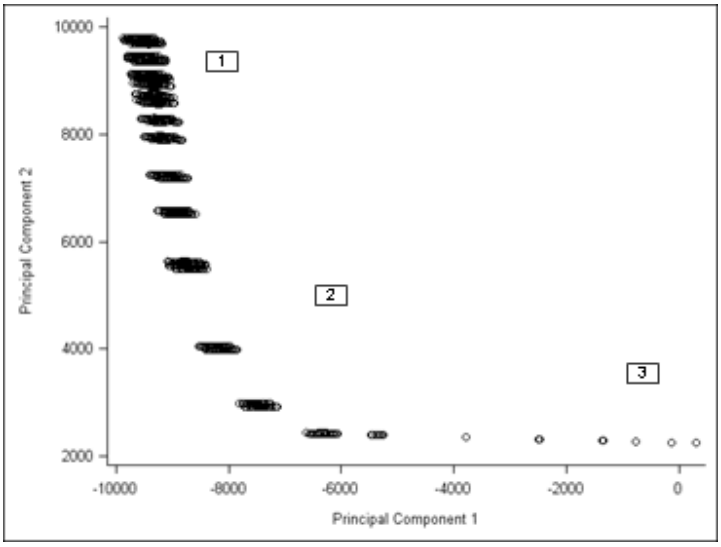

Figure 18. Bi-plot of PCA scores on the first two principal components for 1185 solutions in the *Helicobacter pylorus* model.

Table 12. Flux directions of 43 essential reactions for each solution group in Figure 18.

Essential Reactions	Group 1 (n = 1061)	Group 2 (n = 116)	Group 3 (n = 8)
R_ACACT1r	-402.31	-403.33	-414.22
R_ACACt2	402.31	403.33	414.22
R_ACONT	586.80	588.85	610.62
R_AKGt2r	853.54	852.00	835.68
R_ASPT	4.70	5.21	10.65
R_ASPTA	-29.51	-30.03	-35.47
R_CS	586.80	588.85	610.62
R_CYSabc	0.69	1.20	6.64
R_CYSGLUexR	-5.22	-5.73	-11.17
R_FRDO	404.28	405.31	416.19
R_FUM	-820.34	-819.83	-814.39
R_G6PDH2	885.55	883.50	861.74
R_GLNS	19.50	18.98	13.54
R_GND	419.63	416.55	383.90
R_HYDA1	**0.00**	**2.82**	**32.75**
R_ICDHyr	586.80	588.85	610.62
R_MALt2r	640.04	642.09	663.86
R_MDH4	616.31	618.88	646.08
R_NDH_DASH_1	928.80	926.74	904.98
R_OCOAT1	402.31	403.33	414.22
R_OOR	404.28	405.31	416.19
R_PGI	-889.37	-887.32	-865.55
R_PGL	885.55	883.50	861.74
R_RPE	899.92	897.87	876.10
R_RPI	480.89	481.92	492.80
R_SERD_L	368.08	367.05	356.17
R_SERt2r	-487.25	-488.27	-499.16
R_SUCFUMt	-50.61	-46.25	0.00
R_TALA	450.55	449.52	438.64
R_TKT1	452.95	451.92	441.04
R_TKT2	446.97	445.95	435.06
src_acac_e	402.31	403.33	414.22
src_akg_e	853.54	852.00	835.68
src_gln_l_e	5.22	5.73	11.17
src_h2_e	**0.00**	**2.82**	**32.75**
esc_fum_e	279.93	276.60	241.23
esc_mal_l_e	359.96	357.91	336.14
esc_ser_l_e	487.25	488.27	499.16
esc_succ_e	949.39	953.75	1000.00

Note: Mean values of each solution group were presented

In PCA analysis of the 1185 solutions for the *H. pylorus* model, the majority of the solutions were grouped together in the left side in the figure 18, which were the highly negative scores of the first principal component. Some other solutions display PCA scores far away from the majority group. We categorized those solutions into three groups according to the PCA scores of the first principal component (Group 1: PCA scores < -9000, Group 2: PCA scores -2000 to -9000, and Group 3: PCA scores > -2000). Almost all of the essential reactions had similar flux directions. Only a small difference of some reaction fluxes was found among the three groups. For instance, H_2 and Hyda reactions had zero flux in the Group 1, while they showed non-zero fluxes in the Group 2, and 3. More details of flux directions of other essential reactions in the *H. pylorus* model can be seen in Table 12.

4 Cryptococcus neoformans

Cryptococus neoformans is a fungal pathogen that causes respiratory and neurological infections in immunocompromised individuals and HIV/AID patients. *C. neoformans* is currently reported as the most common opportunistic infection in HIV/AIDS patients in sub-Saharan Africa more than tuberculosis and as a critical disease in some countries such as India. We may see *C. neoformans* in the environments such as soil, water, milk, fruits, bird nests, bats, and etc but when they are inhaled in human or animal body, they disseminate the infection to their host via bloodstream and cause meningoencephalitis. It grows well in alkaline, neutral, and acidic environments of human body. *C.neoformans* infections can be treated by antifungal but there is an increase of drug resistance (46, 47). Better understanding the metabolic reactions and

pathways of this pathogen would help to decrease morbidity and mortality in the endemic areas.

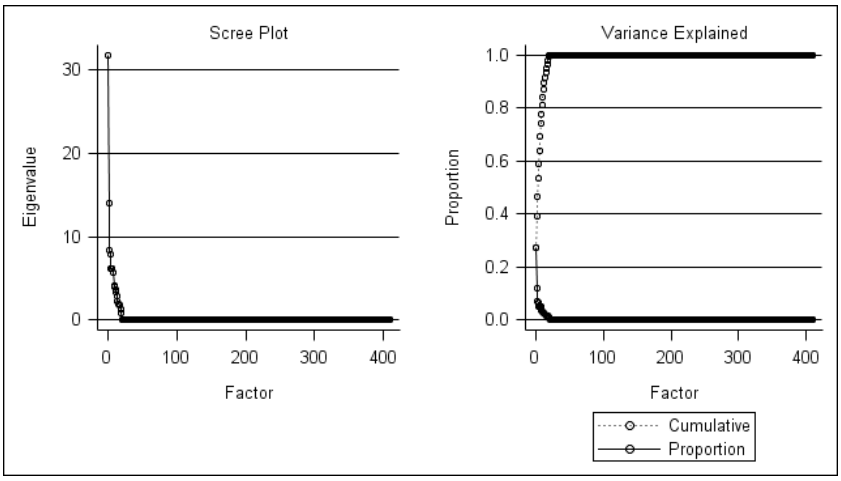

Figure 19. Scree plot of eigenvalues from principal component analysis of 400 active reactions, 5 sources and 4 escapes in the *Cryptococus neoformans* model

There were 20 components with an eigenvalue greater than zero, 19 of them with an eigenvalue greater than one. The eigenvalues for the component 1, 2, and 3 were 31.714, 14.045, and 8.422. The last component displays an eigenvalue of 0.816. The first, second, and third components account for 27.3%, 12.1%, and 7.3% of the total variance. The eigenvalues and the proportions of variance accounted for from this analysis appear in Figure 19.

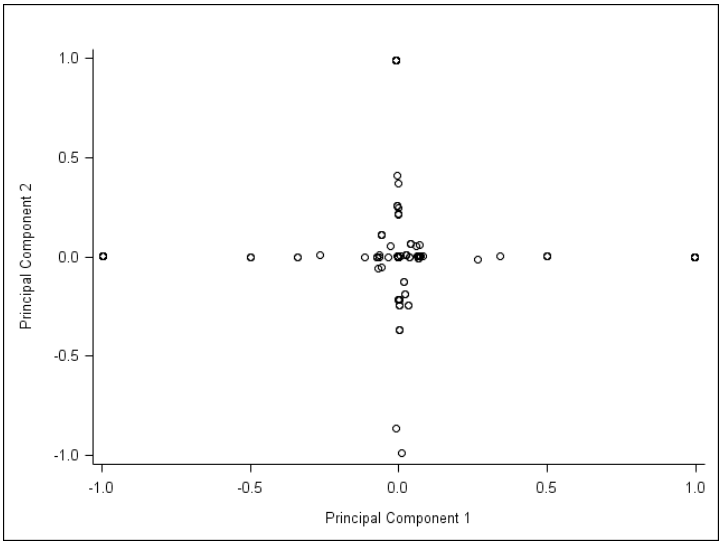

Figure 20. Bi-plot of the rotated factor pattern in the first and second components of the metabolic reactions in *Cryptococcus neoformans* by alternate optimal solutions conducted by MetModelGUI.

In the analysis of *Cryptococus neoformans,* we found 116 reactions from 31 pathways contributed to the first component and it seems they were organized to 5 groups. However, 28 essential reactions from 8 pathways were extracted from the list of the reactions. 17 in the 28 reactions exist in the extremely right side and 11 exist in the extremely left side. The essential reactions contained in these pathways: fructose and mannose metabolism, galactose metabolism, glycolysis/gluconeogenesis, pentose phosphate pathway, pyruvate metabolism, extracellular transport, sources and escape fluxes.

Table 13. Essential reactions and their pathways in *Cryptococcus neoformans* in the first principal component by alternate optimal solutions generated from MetModelGUI

Pathways	Reaction names (Enzymes)	Reactions	Rotated Factor Pattern
Fructose and Mannose Metabolism	Hexokinase (D-Fructose:atp)	R_HEX7	0.99751
Galactose metabolism	Sucrosehydrolyzingenxyme,extracellular	R_SUCRe	0.99751
Glycolysis/Gluconeogenesis	Aldehydedehydrogenase (Acetaldehyde,nadp)	R_ALDD2y	0.99751
Glycolysis/Gluconeogenesis	Enolase	R_ENO	0.99751
Glycolysis/Gluconeogenesis	Fructose-Bisphosphatealdolase	R_FBA	0.99751
Glycolysis/Gluconeogenesis	Glucose-6-Phosphateisomerase	R_PGI	0.99751
Glycolysis/Gluconeogenesis	Glyceraldehyde-3-Phosphatedehydrogenase	R_GAPD	0.99751
Glycolysis/Gluconeogenesis	Hexokinase (D-Glucose:atp)	R_HEX1	0.99751
Glycolysis/Gluconeogenesis	Phosphofructokinase	R_PFK	0.99751
Glycolysis/Gluconeogenesis	Phosphoglyceratekinase	R_PGK	-0.99751
Glycolysis/Gluconeogenesis	Phosphoglyceratemutase	R_PGM	-0.99751
Glycolysis/Gluconeogenesis	Pyruvatekinase	R_PYK	0.99751
Glycolysis/Gluconeogenesis	Triose-Phosphateisomerase	R_TPI	0.99751
Pentose Phosphate Pathway	6-Phosphogluconolactonase	R_PGL	-0.99751
Pentose Phosphate Pathway	Glucose6-Phosphatedehydrogenase	R_G6PDH2	-0.99751
Pentose Phosphate Pathway	Phosphogluconatedehydrogenase	R_GND	-0.99751
Pentose Phosphate Pathway	Ribose-5-Phosphateisomerase	R_RPI	0.99751
Pentose Phosphate Pathway	Ribulose5-Phosphate3-Epimerase	R_RPE	-0.99751
Pentose Phosphate Pathway	Transaldolase	R_TALA	-0.99751
Pentose Phosphate Pathway	Transketolase	R_TKT1	-0.99751
Pentose Phosphate Pathway	Transketolase	R_TKT2	-0.99751
Pyruvate Metabolism	Pyruvatedecarboxylase	R_PYRDC	0.99751
Transport, Extracellular	Acetatereversibletransportviaprotonsymport	R_ACt2r	-0.99751
Transport, Extracellular	D-Fructosetransportinviaprotonsymport	R_FRUt2	0.99751
Transport, Extracellular	Glucosetransport (Uniport)	R_GLCt1	0.99751
EscapeFlux	EscapeFlux	R_ESC_h2o_e	-0.99751
EscapeFlux	EscapeFlux	R_ESC_ac_e	0.99751
SourceFlux	SourceFlux	R_SRC_sucr_e	0.99751

In Table 13, we found pentose phosphate pathway is an important pathway for C. *neoformans* metabolism. This result is similar to a study which suggested some cellular processes such as cell wall maintenance, stress and virulence were important for target genes of C. *neoformans* (48). The pentose phosphate pathway was a highly conserved pathway and was important for reductive biochemistry during oxidative stress in many

organism. For C. *neoformans,* the pentose phosphate pathways were essential for the ability of resistance and adaptation to high levels of oxidative stress. Another study suggested that gluconeogenesis and glycolysis metabolic pathways for carbon utilization in C. *neoformans,* which restricted for growth of lactate and glucose (49). The pathwars were also found as important pathways for energy sources in this analysis. In addition, the pyruvate decarboxylase was a reaction found in the process of conversion from pyruvate to acetate. The acetate production was relevant to the pathogenesis of *C. neoformans.* It was presented as one of the major metabolites present in infected tissue (50).

Below the list of reactions with very little variability was in the first component.

Table 14. Reactions with very little variability in *Cryptococcus neoformans* model

Pathways	Reaction names (Enzymes)	Reactions
Alanine and Aspartate Metabolism	Homocysteines-Methyltransferase	R_HCYSMT
Alternate Carbon Metabolism	Glycogenphosphorylase	R_GLCP
Alternate Carbon Metabolism	Glycogensynthase(Udpglc)	R_GLCS2
Cholesterol Metabolism	Acetyl-Coac-Acetyltransferase	R_ACACT1
Cofactor and Prosthetic Group Biosynthesis	Nicotinicacidmononucleotidepyrophosphorylase	R_NAMNPP
Fatty Acid Biosynthesis	Fatty-Acid--Coaligase(Hexadecanoate)	R_FACOAL160
Fatty Acid Biosynthesis	Fatty-Acid--Coaligase(Tetradecanoate)	R_FACOAL140
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydratase(3-Hydroxydecanoyl-Coa)	R_ECOAH4
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydratase(3-Hydroxyhexadecanoyl-Coa)	R_ECOAH7
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydratase(3-Hydroxyhexanoyl-Coa)	R_ECOAH2
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydratase(3-Hydroxyoctanoyl-Coa)	R_ECOAH3
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydratase(3-Hydroxytetradecanoyl-Coa)	R_ECOAH6
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydrogenase(3-Oxodecanoyl-Coa)	R_HACD4
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydrogenase(3-Oxododecanoyl-Coa)	R_HACD5
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydrogenase(3-Oxohexadecanoyl-Coa)	R_HACD7
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydrogenase(3-Oxohexanoyl-Coa)	R_HACD2
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydrogenase(3-Oxooctanoyl-Coa)	R_HACD3
Fatty Acid Degradation	_3-Hydroxyacyl-Coadehydrogenase(3-Oxotetradecanoyl-Coa)	R_HACD6
Fatty Acid Degradation	Acetyl-Coac-Acyltransferase(Butanoyl-Coa)(R)	R_ACACT2r
Fatty Acid Degradation	Acetyl-Coac-Acyltransferase(Hexanoyl-Coa)(R)	R_ACACT3r
Fatty Acid Degradation	Acetyl-Coac-Acyltransferase(Octanoyl-Coa)(R)	R_ACACT4r

Table 14. Reactions with very little variability in *Cryptococcus neoformans* model (cont.)

Pathways	Reaction names (Enzymes)	Reactions
Fatty Acid Degradation	Acetyl-Coac-Acyltransferase(Tetradecanoyl-Coa)(R)	R_ACACT7r
Fatty Acid Degradation	Acyl-Coadehydrogenase(Butanoyl-Coa)	R_ACOAD1
Fatty Acid Degradation	Acyl-Coadehydrogenase(Hexanoyl-Coa)	R_ACOAD2
Fatty Acid Degradation	Acyl-Coadehydrogenase(Octanoyl-Coa)	R_ACOAD3
Fatty Acid Degradation	Acyl-Coadehydrogenase(Tetradecanoyl-Coa)	R_ACOAD6
Fatty acid elongation	B-Ketoacylsynthetase(Palmitate,n-C16:0)	R_KAS8
Fatty acid elongation	Fattyacyl-Coasynthase(N-C10:0coa)	R_FAS100COA
Fatty acid elongation	Fatty-Acyl-Coasynthase(N-C12:0coa)	R_FAS120COA
Fatty acid elongation	Fatty-Acyl-Coasynthase(N-C14:0coa)	R_FAS140COA
Fatty acid elongation	Fatty-Acyl-Coasynthase(N-C16:0coa)	R_FAS160COA
Fatty acid elongation	Fattyacyl-Coasynthase(N-C8:0coa),lumpedreaction	R_FAS80COA_L
Galactose metabolism	Utp-Glucose-1-Phosphateuridylyltransferase	R_GALU
IMP Biosynthesis	Phosphoribosylaminoimidazolecarboxylase	R_AIRCr
IMP Synthesis 7	Phosphoribosylaminoimidazolecarboxylase	R_AIRC2r
Membrane Lipid Metabolism	Acetyl-Coacarboxylase,reversiblereaction	R_ACCOACr
Methionine Metabolism	Adenosylhomocysteinase	R_AHCi
Methionine Metabolism	Methionineadenosyltransferase	R_METAT
NAD Biosynthesis	Naprtase(Rev)	R_NAPRTr
Nucleotide Salvage Pathway	Adentylatekinase(Itp)	R_ADK4
Nucleotide Salvage Pathway	Uridylatekinase(Dump)	R_URIDK2r
Nucleotides	Adenosinekinase	R_ADNK1
Nucleotides	Guanylatekinase(Gmp:atp)	R_GK1
Nucleotides	Nucleoside-Diphosphatekinase(Atp:dadp)	R_NDPK8
Nucleotides	Nucleoside-Diphosphatekinase(Atp:dudp)	R_NDPK6
Nucleotides	Nucleoside-Diphosphatekinase(Atp:idp)	R_NDPK9
Nucleotides	Nucleoside-Diphosphatekinase(Atp:udp)	R_NDPK2
Purine and Pyrimidine Biosynthesis	Dutpdiphosphatase	R_DUTPDP
Purine and Pyrimidine Biosynthesis	Guanylatekinase(Gmp:datp)	R_GK2
Purine and Pyrimidine Biosynthesis	Phosphoribosylaminoimidazolecarboxylase(Mutaserxn)	R_AIRC3
Threonine and Lysine Metabolism	Threoninealdolase	R_THRA
Threonine and Lysine Metabolism	Threoninealdolase	R_THRAr
Transport, Extracellular	Atpase,cytosolic	R_ATPS
Transport, Extracellular	Cytosinetransportinviaprotonsymport	R_CSNt2
Transport, Extracellular	Cytosinetransportviafacilateddiffusion	R_CSNt
Transport, Extracellular	Pyruvatereversibletransportviaprotonsymport	R_PYRt2r
Transport, Extracellular	Pyruvatetransportinviaprotonsymport	R_PYRt2
Transport, Mitochondrial	Dicarboxylatetransport,mitochondrial	R_DICtm
Transport, Mitochondrial	L-Glutamatereversibletransportviaprotonsymport,mitochondrial	R_GLUt2m
Transport, Mitochondrial	L-Glutamatetransportintomitochondriaviahydroxideionantiport	R_GLUt5m
Transport, Mitochondrial	Malatetransport,mitochondrial	R_MALtm
Transport, Mitochondrial	Phosphatetransporter,mitochondrial	R_PIt2m
Transport, Mitochondrial	Phosphatetransportviahydroxideionsymport,mitochondrial	R_PIt5m
Transport, Mitochondrial	Succinatetransport,mitochondrial	R_SUCCtm
Tryptophan metabolism	Acetyl-Coac-Acetyltransferase	R_ACACT1r

Table 14. Reactions with very little variability in *Cryptococcus neoformans* model (cont.)

Pathways	Reaction names (Enzymes)	Reactions
Tyrosine, Tryptophan, and Phenylalanine Metabolism	Tyrosinetransaminase,irreversible	R_TYRTAi
	_3-Hydroxyacyl-Coadehydratase(3-Hydroxybutanoyl-Coa)	R_ECOAH1
.	_3-Hydroxyacyl-Coadehydratase(3-Hydroxydodecanoyl-Coa)	R_ECOAH5
.	_3-Hydroxyacyl-Coadehydrogenase(Acetoacetyl-Coa)	R_HACD1
Pathways	Reaction names (Enzymes)	Reactions
.	Acetyl-Coac-Acyltransferase(Decanoyl-Coa)(R)	R_ACACT5r
.	Acetyl-Coac-Acyltransferase(Dodecanoyl-Coa)(R)	R_ACACT6r
.	Acyl-Coadehydrogenase(Decanoyl-Coa)	R_ACOAD4
.	Acyl-Coadehydrogenase(Dodecanoyl-Coa)	R_ACOAD5
.	Acyl-Coadehydrogenase(Hexadecanoyl-Coa)	R_ACOAD7
.	B-Ketoacylsynthetase(N-C14:0)	R_KAS2

Note: Rotated pattern factor < 0.01 or > - 0.01 and not included the zero value.

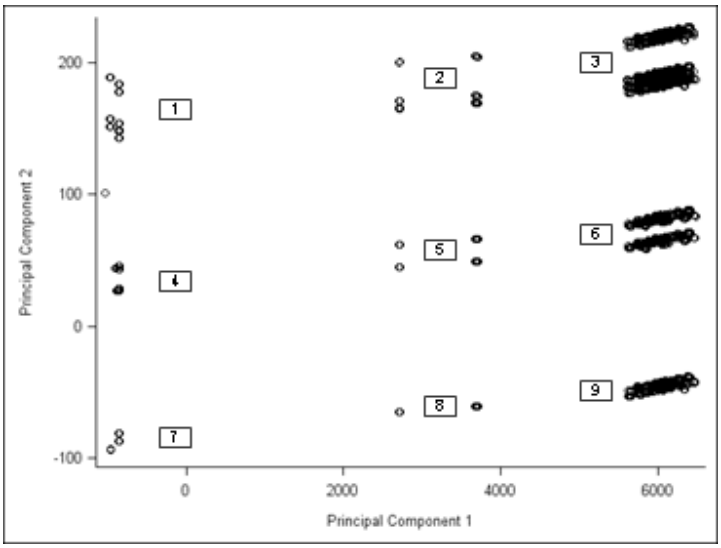

Figure 21. Bi-plot of PCA scores on the first two principal components for 1698 solutions in the *Cryptococus neoformans* model.

Table 15. Flux directions of 28 essential reactions for each solution group in Figure 21.

Essential Reactions	Solution Groups								
	1 (n = 25)	2 (n = 12)	3 (n = 6)	4 (n = 24)	5 (n = 12)	6 (n = 6)	7 (n = 922)	8 (n = 461)	9 (n = 230)
R_ACt2r	-16.56	-18.03	-16.71	-18.68	-18.68	-18.68	-18.68	-18.68	-18.68
R_ALDD2y	126.59	128.07	126.75	128.72	128.72	128.72	128.72	128.72	128.72
R_ENO	400.34	401.81	400.49	402.46	402.46	402.46	402.46	402.46	402.46
R_FBA	239.13	240.11	239.23	240.55	240.55	240.55	240.55	240.55	240.55
R_FRUt2	167.21	167.46	167.24	167.57	167.57	167.57	167.57	167.57	167.57
R_G6PDH2	**2.13**	**0.66**	**1.97**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
R_GAPD	403.91	405.39	404.07	406.04	406.04	406.04	406.04	406.04	406.04
R_GLCt1	167.21	167.46	167.24	167.57	167.57	167.57	167.57	167.57	167.57
R_GND	**2.13**	**0.66**	**1.97**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
R_HEX1	167.21	167.46	167.24	167.57	167.57	167.57	167.57	167.57	167.57
R_HEX7	167.21	167.46	167.24	167.57	167.57	167.57	167.57	167.57	167.57
R_PFK	239.13	240.11	239.23	240.55	240.55	240.55	240.55	240.55	240.55
R_PGI	163.39	165.11	163.58	165.87	165.87	165.87	165.87	165.87	165.87
R_PGK	-403.91	-405.39	-404.07	-406.04	-406.04	-406.04	-406.04	-406.04	-406.04
R_PGL	**2.13**	**0.66**	**1.97**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
R_PGM	-400.34	-401.81	-400.49	-402.46	-402.46	-402.46	-402.46	-402.46	-402.46
R_PYRDC	125.30	126.78	125.46	127.43	127.43	127.43	127.43	127.43	127.43
R_RPE	-91.48	-92.46	-91.58	-92.89	-92.89	-92.89	-92.89	-92.89	-92.89
R_RPI	-93.60	-93.11	-93.55	-92.89	-92.89	-92.89	-92.89	-92.89	-92.89
R_SUCRe	167.21	167.46	167.24	167.57	167.57	167.57	167.57	167.57	167.57
R_TALA	-45.27	-45.76	-45.32	-45.97	-45.97	-45.97	-45.97	-45.97	-45.97
R_TKT1	-45.27	-45.76	-45.32	-45.97	-45.97	-45.97	-45.97	-45.97	-45.97
R_TKT2	-46.21	-46.70	-46.26	-46.92	-46.92	-46.92	-46.92	-46.92	-46.92
R_TPI	212.02	213.00	212.12	213.43	213.43	213.43	213.43	213.43	213.43
esc_ac_e	16.56	18.03	16.71	18.68	18.68	18.68	18.68	18.68	18.68
esc_h2o_e	832.79	832.54	832.76	832.43	832.43	832.43	832.43	832.43	832.43
src_sucr_e	167.21	167.46	167.24	167.57	167.57	167.57	167.57	167.57	167.57

Note: Mean values of the solutions of each group were presented

In PCA analysis on solutions for the *C. neoformans* model, PCA scores of the first-two components classified 1698 solutions into 9 groups (Figure 21). The majority groups were displayed in the same level of the first principal component scores (Group 3, 6, and 9). However, when considering flux directions, most of the essential reactions had the similar flux directions. 20 essential reactions had high fluxes, whereas 8 had low fluxes.

Small differences of the mean solution values were found among Group 1, 2, and 3. More details in the analysis can be found in Table 16.

5. Clostridium thermocellum

Clostridium thermocellumc is one of the cellulolytic microorganisms and classified as a gram-positive bacterium. This bacterium is unable to convert cellulose biomass into ethanol and hydrogen. It is useful for the production of biofuel from biomass. The waste products of the process are generated such as hydrogen, carbon dioxide, acetate, and primarily ethanol.

In PCA analysis, there were 13 components with an eigenvalue greater than one and 3 component with an eigenvalue in between zero and one. The eigenvalues for the component 1, 2, and 3 were 12.241, 5.573, and 3.726. The last component displays an eigenvalue of 0.171. The first, second, and third components account for 27.8%, 12.7%, and 8.5% of the total variance. The eigenvalues and the proportions of variance accounted for from this analysis appear in Figure 22.

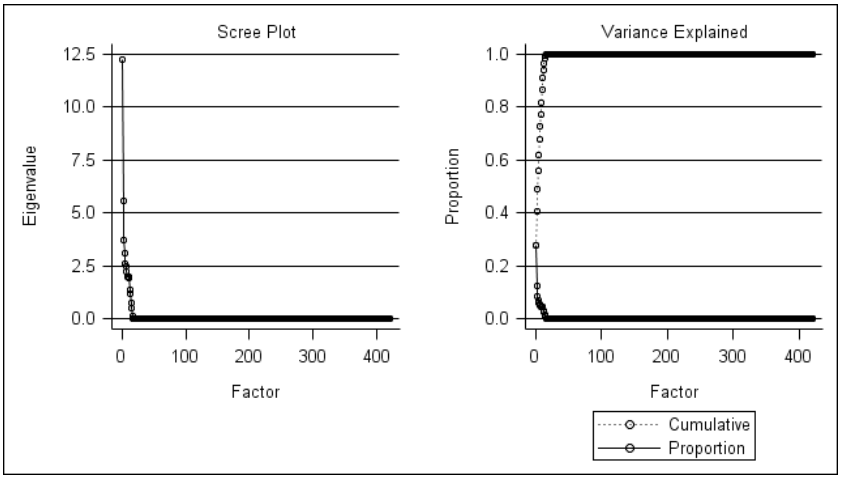

Figure 22. Scree plot of eigenvalues from principal component analysis of 399 active reactions, 13 sources and 9 escapes in the *Clostridium thermocellum* model

These following are reactions associated with *C. thermocellum.*

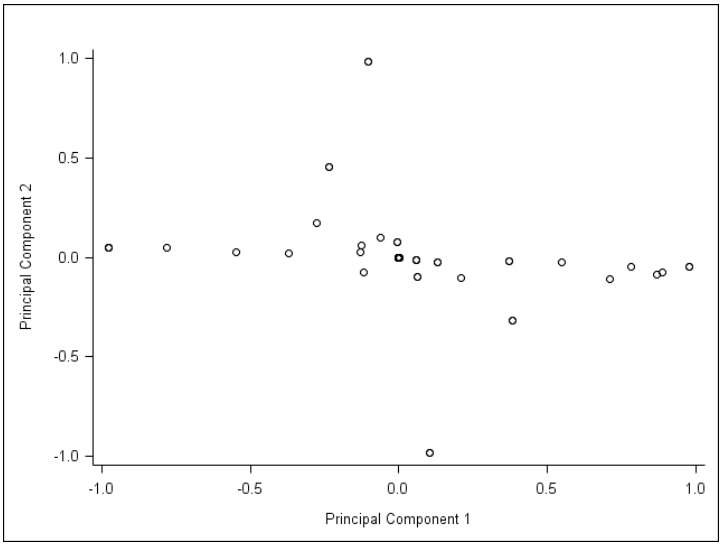

Figure 23. Bi-plot of the rotated factor pattern in the first and second components of the metabolic reactions in *Clostridium thermocellum* by alternate optimal solutions conducted by MetModelGUI.

Thirty-nine reactions, two sources, and three escapes contribute to the first two components. Of the 39 reactions, 5 reactions were listed as important reactions found in methionine and cysteine pathways. Cysteine and methionine are sulfur-containing amino acids. The cysteine was synthesized from serine in different pathways and metabolized to pyruvate with multiple routes. The methionine was an essential amino acid, which animals could not synthesize. In bacteria, methionine was synthesized from aspartate and produces S-Adenosylmethionine and ATP. The S-Adenosylmethionine was a methyl group in many important transfer reactions such as DNA methylation for regulation of gene expression (51). The reactions possibly release a number of energy for C. *thermocellum*.

Table 16. Essential reactions and their pathways in *Clostridium thermocellum* in the first principal component by alternate optimal solutions generated from MetModelGUI

Pathways	Reaction names (Enzymes)	Reactions	Rotated Factor Pattern
Methionine Metabolism	O-Acetyl homoserine(Thiol)-Lyase	R_AHSERL2	0.97715
Methionine Metabolism	Metb1(REV)	R_METB1r	-0.97715
Methionine Metabolism	O-Succinyl homoserinelyase(Elimination), reversible	R_SHSL4r	0.97715
Cysteine biosynthesis	Cystathionineg-Lyase	R_CYSTGL	-0.97715
Methionine biosynthesis	O-Succinyl homoserinelyase (H_2S)	R_SHSL2r	-0.97715

Below the list of reactions with very little variability was in the first component.

Table 17. Reactions with very little variability in *Clostridium thermocellum* model

Pathways	Reaction names (Enzymes)	Reactions
Nucleotides	Cytidylatekinase(Cmp,ctp)	R_CYTK6
Pyrimidine metabolism	Nucleoside-Diphosphatekinase(Atp:cdp)	R_NDPK3
Transport, Extracellular	Glycoaldehydyereversibletransport	R_GCALDt
EscapeFlux		R_ESC_co2_e
EscapeFlux		R_ESC_gcald_e
EscapeFlux		R_ESC_h2o_e
SourceFlux		R_SRC_co2_e
SourceFlux		R_SRC_h2o_e

Note: Rotated pattern factor < 0.01 or > - 0.01 and not included the zero value.

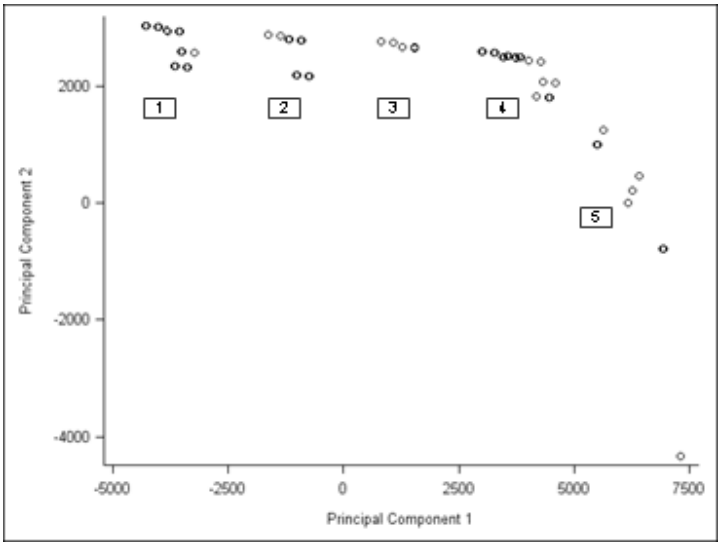

Figure 24. Bi-plot of PCA scores on the first two principal components for 1231 solutions in the *Clostridium thermocellum* model.

Table 18. Flux directions of five essential reactions for each solution group in Figure 24.

Essential Reactions	Group 1 (n = 456)	Group 2 (n = 343)	Group 3 (n = 96)	Group 4 (n = 224)	Group 5 (n = 112)
R_AHSERL2	0.00	0.00	501.27	501.27	501.27
R_CYSTGL	501.27	501.27	0.00	0.00	0.00
R_METB1r	0.00	0.00	-501.27	-501.27	-501.27
R_SHSL2r	-497.58	-497.58	-998.85	-998.85	-998.85
R_SHSL4r	-497.04	-497.04	4.22	4.22	4.22

Note: Mean values of each solution group were presented

In PCA analysis on solutions for the *C. thermocellum* model, 1231 solutions can be classified into 5 groups by the PCA scores in the first-two components (Figure 24). The number of solutions classified in each group was not much different. We found some different flux directions of the five essential reactions. R_AHSERL2 and R_SHSL4r had high fluxes in the Group 3, 4, and 5, while they had low fluxes in other groups. It seemed that R_CYSTGL had opposite functions with the R_AHSERL2, that is, the R_CYSTGL had high fluxes in Group 1 and 2, but low fluxes in Group 3, 4, and 5. Meanwhile, R_SHSL2r reaction showed low fluxes direction in all groups. More details in the analysis were presented in Table 19.

CHAPTER 4 CONCLUSION

This study applied linear programming subjected to constraints with an optimization approach in metabolic modeling. We implemented flux-balance analysis to study steady state metabolic reactions in cell systems of five microorganisms: *T. cruzi, T. fusca, H. pylori, C. neoformans and C. thermocellum.* An LP problem consists of an objective function, a list of constraints, upper and lower limits of solutions. The objective function was a biomass with a coefficient of one. The optimal value for the objective function was the biomass flux. A metabolic reaction was listed as a variable of the problem. The constraints require that the metabolites remain at a constant concentration. A metabolite may involve more than one reaction. We solved the LP problems using Gurobi Python and QSopt Java applications. A list of solutions and an optimal value for each problem can be used to explain how the metabolites function in the cell system.

As an LP problem may have many optimal solutions, the goal of this study is to implement and apply a method for enumerating alternate optimal solutions to evaluate important reactions of metabolic pathways in the microorganisms. We modified Java-based MetModelGUI and added a ”bouncer” algorithm to be convenient for future users. In the bouncer algorithm, we created two main processes. The first process was to select active reactions of the pathways, defined as the set of reactions that can have non-zero

flux. We then restructured the LP and solved the problems. The second process was to generate other optimal solutions using the solution of the new LP problem to start the algorithm and iterate steps until there was no new solution or finished run time. The alternate optimal solutions were analyzed by PCA. A number of variables (reactions) were reduced into a smaller number of components. The variables contribute to the first components with high variability were considered as important reactions of the pathways.

Metabolism in microorganisms is a complex biological process. We found five important chemical reactions in *T. cruzi* metabolism. They were chemical reactions involved glutamate and electron transports in mitochondrial. In *T. fusca* metabolism, sixteen important reactions involved in producing amino acids alanine, aspartate, arginine, and proline, and in the glycolysis pathways, citric acid cycle, urea cycle and central metabolism. Important chemical reactions in *H. pylori* metabolism were found in several pathways such as tryptophan metabolism, citric acid cycle, pentose phosphate, respiratory chain, glycolysis and gluconeogenesis pathways. For *C. neoformans* metabolism, twenty-eight important reactions were found in fructose and mannose metabolism, galactose metabolism, glycolysis/gluconeogenesis, pentose phosphate pathway, pyruvate metabolism, extracellular transport. In *C. thermocellum* metabolism, five important reactions were in methionine and cysteine pathways. The essential chemical reactions in several pathways indicate that energy sources in microorganisms might be contributed by different metabolic pathways. The results were summarized from alternate optimal solutions and principal component analysis so we can interpret all possible activities over the complex cell systems of the organisms (3, 4).

The selection of metabolic reactions with non-zero flux in stead state would provide solutions that are more convenient to determine essential metabolic reactions. In the total reactions of *T. cruzi, T. fusca, H. pylori, C. neoformans and C. thermocellum,* 15%, 32%, 14%, 42% and 28% were inactive reactions. *C. neoformans* data had higher number of inactive reactions than *H. pylori* and *C. thermocellum* but provided the similar number of alternate solutions. This process is useful to narrow scopes of information in the metabolisms and had no effect on number and correctness of alternate optimal solutions. It saves time and memory space in iteration algorithm.

In general, a number of alternate optimal solutions depend on sizes of interconnected pathways in their metabolic network (2). In our data, sizes of matrix A in *H. pylori, C. neoformans, and C. thermocellum* were similar to each other but *H. pylori* contained more sources and escapes. Most of them were boundary metabolites converted to extracellular metabolites or had lower interconnected pathways. The lower complexity of metabolic networks may be the reason that *H. pylori* had lower number of optimal solutions than the other organisms although the metabolism network of *H. pylori* had a larger size. Moreover, the number of optimal solutions in our data had been proved that did not depend on the number of entering variables, basic reaction indexes, basic metabolite indexes or biomass flux. *H. pylori* data had high number of entering variables but the number of solutions of *H. pylori* data was close to the number in *C. neoformans. C. thermocellum. T. fusca* data had the similar number of entering variables to the *C. neoformans* and *C. thermocellum* data but they provided more solutions. Therefore, number of alternate optimal solutions would depend on sizes of metabolite network and

types of constraints or environmental conditions (2). They may depend on types of objective functions but this cannot be proved as we use the same objective functions.

The bouncer algorithm shows some particular characteristics in enumerating alternate optimal solutions. All five models have high rate of generating the new solutions in the beginning period of execution. This may be because the early period has more suitable of nonbasic variables reduced cost zero than the later period. If new entering variables are frequently added to the stack collection, there is a consistent chance to obtain new optimal solutions. When no new entering variables are added into the basis for some period, this may predict time out of the execution process. Also, when the time of execution passes the early period, number of iterations tends to decrease. It is possible that memory capacities in computation are limited by increase of new solutions stored. The algorithm and execution process still need more evaluation in details.

The reduction of a number of reactions to a smaller number of principal components would be useful to search for important metabolic reactions. As the smaller number of principal components account for most of the variance in the data sets, we considered the reactions in the first component, which accounts for as much of the variability in the data as possible. The reduction would decrease uncertainty of actual reactions in the metabolic pathways. The important reactions were considered as the most informative reactions. However, these results would be validated by an experiment.

Alternate optimal solutions may vary in different algorithms. Apart from a variety of number of optimal solutions in a LP problem, the optimal solutions may vary by types and selections entering and basis variables in algorithm obtained. In addition, given the property of genome-scale network, a steady state in optimal metabolic networks is

assumed. The variation of alternate solutions may differ by the set of reactions used in the analysis (2, 52). A study suggested using random sampling of all elementary vectors of metabolic networks as we can estimate probability of possible outcomes and expect information of the outcomes. In considering possible reduction of state-space, the methods satisfied optimal criteria and provide systematic method in studying the set of alternate solutions (53).

Although MetModelGUI is ready for users, testing and applying the program for several organisms is necessary. The developed program have been created and implemented in five organisms. Results of important metabolic reactions remain unproved by an experiment or measuring *in vivo* fluxes.

This study can serve as a starting point for applying alternate optimal algorithms to metabolic reconstructions. A recursive mixed-integer LP algorithm, another alternative algorithm for alternate optimal algorithms with the same objective function and satisfy constraints would be applied to compare results to our algorithm.

Genome-scale network reconstruction has several redundant pathways. Many metabolites function both intracellular and extracellular fluxes. In the diverse conditions, flux variability analysis would be useful to understand the entire complex system of metabolic reconstruction. Also integration of several algorithms and constraints are encouraged for more accurate descriptions of the biological system.

List of References

List of References

1. Park JM, Kim TY, Lee, SY. Constrained-based genome-scale metabolic simulation for systems metabolic engineering. Biotechnology Advance 2009; 27: 979-988.

2. Price ND, Reed JL, Palsson BO. Genome-scale models of microbial cells: evaluating the consequence of constraints. Nature Reviews Microbiology 2004; 2: 886-897.

3. Varma A & Palsson BO. Metabolic Flux Balancing: Basic Concepts, Scientific and Practical Use. Nature Biotechnology 1994; 12: 994-998.

4. Edward JS, Covert M, Palsson B. Metabolic modeling of microbes the flux-balance approach. Environmental Microbiology 2002; 4(3), 133-140.

5. Mahadevan R, Burgard AP, Famili I, Dien SV, Schilling CH. Applications of metabolic modeling to drive bioprocess development for the production of value-added chemicals. Biotechnology and bioprocess engineering 2005, 10: 408-417.

6. Park JM, Kim TY, Lee SY. Prediction of metabolic fluxes by incorporating genomic context and flux-converging pattern analyzes. Proc Natl Acad Sci U S A. 2010 Aug 17;107(33):14931-6. Epub 2010 Aug 2.

7. Chvatal V. Linear Programming. New York: W.H. Freeman and Company; 1983.

8. Hillier FS, Lieberman. Introduction to operation research, seventh edition. McGraw-Hill New York; 2001.

9. Ignizio JP. Linear Programming in Single- & Multiple –Objective Systems. New Jensy: Hall, Inc.; 1982.

10. Yugi K, Nakayama Y, Kinoshita A, Tomita M. Hybrid dynamic/static method for large scale simulation of metabolism. Theor Bio Med Model 2005; 2: 42.

11. Palsson BO, Price ND, Papin JA. Development of network-based pathway definitions: the need to analyze real metabolic networks. Trends Biotechnol 2003; 21: 195-8

12. Mahadevan, R.; Schilling, C.H..The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. Metabolic Engineering, Oct 2003, Vol. 5 Issue 4, p264, 13p.

13. Shlomi T, Berkman O, Ruppin E. Regulatory on/off minimization of metabolic flux changes after genetic perturbations. Proc Natl Acad Sci U S A. 2005 May 24;102(21):7695-700. Epub 2005 May 16.

14. Burgard AP, Maranas CD. Optimization-based framework for inferring and testing hypothesized metabolic objective functions. Biotechnol Bioeng. 2003 Jun 20;82(6):670-7.

15. Ventura BD, Lemerle C, Michalodimitrakis K, Serrano L. From in vivo to in silico biology and back. Nature 443, 527-533.

16. Edwards JS, Ibarra RU, Palsson BO. In silico predictions of Escherichia coli metabolic capabilities are consistent with experimental data. Nature Biotechnology, Feb2001, Vol. 19 Issue 2, p125, 6p.

17. Covert MW, Schilling CH, Famili I, Edwards JS, Goryanin II, Selkov E, Palsson BO. Metabolic Modeling of Microbial Strains in silico. Biochemical Sciences 26(3):179-86, March 2001

18. Natalie C. Duarte,1,3 Markus J. Herrgård,1,2,3 and Bernhard Ø. Palsson. Reconstruction and Validation of Saccharomyces cerevisiae iND750, a Fully Compartmentalized Genome-Scale Metabolic Model. Genome Res. 2004 July; 14(7): 1298–1309.

19. Duarte NC, Becker SA, Jamshidi N, Thiele I, Mo ML, Vo TD, Srivas R, and Palsson BO. Global reconstruction of the human metabolic network based on genomic and bibliomic data. Proc Natl Acad Sci U S A. 2007 Feb 6;104(6):1777-82. Epub 2007 Jan 31.

20. Ates O, Oner ET, Arga KY. Genome-scale reconstruction of metabolic network for a halophilic extremophile, Chromohalobacter salexigens DSM 3043. BMC Syst Biol. 2011 Jan 21;5:12

21. Dantzig. G. B., A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear from under linear inequality restraints. Pacific Journal of Mathematics 5: 1995, 183-195.

22. Hillier F., and Lieberman G. Introduction to Operations Research. 7th Edition. Holden-Day, 2001.

23. Murtagh, B.A. Advanced Linear Programming: Computation and Practice. New York :McGraw-Hill. 1981.

24. Brooks P. MetModel GUI: Software for Building Optimization-based Models of Cellular Metabolism: Informs Washington D.C. - 2008 Interactive Session. Informs Annual Meeting: Washington D.C. 2008.

25. Vanee N, Roberts SB, Fong SS, Manque P, Buck GA. A genome-scale metabolic model of Cryptosporidium hominis. Chem Biodivers. 2010 May;7(5):1026-39.

26. Alvin J, Fong S, Estevez A. Investigation of Thermobifida fusca as a Novel Biofuel Agent. A report published in Bioinformatics and Bioengineering Summer Institute. Virginia Commonwealth University. August, 2008.

27. Gonyea S. Creation of a genome-scale metabolic model for the fungal pathogen. Cryptococus neoformans. A report published in Bioinformatics and Bioengineering Summer Institute. Virginia Commonwealth University. August, 2006-2007.

28. Reed JL, Famili I, Thiele I, Palsson BO. Towards multidimensional genome annotation. Nat Rev Genet. 2006 Feb;7(2):130-41. Review.

29. Python 2.7.1 documentation. Python Software Foundation. The Python Software Foundation is a non-profit corporation. 1990-2011.

30. GNU Linear Programming Kit. http://www.gnu.org/software/glpk/glpk.html

31. SBML Team. California Institute of Technology. http://sbml.org/Software/libSBML/docs/python-api/

32. McGuire P. Getting started with pyparsing. O'Reilly Media, 2007 http://pyparsing.wikispaces.com/

33. Gurobi Optimizer Quick Start Guide Version 4.0. Gurobi Optimization, Inc. 2011

34. Applegate D, Cook W, Dash S, and Mevenkamp M. QSopt Linear Programming Solver. QSopt Reference Manual. Version 1.0 - October 27, 2003. http://www2.isye.gatech.edu/~wcook/qsopt/downloads/users.pdf.

35. Roberts SB. Buck GA Python-based framework for building & analyzing constraint-based metabolic models and application to genome-scale model of streptococus sanguinis. 2007.

36. Strum JE. Introduction to Linear Programming. San Francisco: Holden-Day

37. Hatcher L, Stepanski E. A step-by-step approach to using the SAS system for univariate and multivariate statistics. Cary, NC: SAS Institute Inc. 1994

38. Uchiyama, N. Antichagasic activities of natural products against Trypanpsoma cruzi. Journal of Health Science 2009 Vol. 55 No. 1 pp. 31-39.

39. Magdaleno A, Suárez Mantilla B, Rocha SC, Pral EM, Silber AM. The Involvement of Glutamate Metabolism in the Resistance to Thermal, Nutritional, and Oxidative Stress in Trypanosoma cruzi. Enzyme Res. 2011;2011: 486928. Epub 2011 Apr 10.

40. DOE Joint Genome Institute, Thermobifida fusca YX project, Website: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=genomeprj&cmd=Retrieve&dopt=Overview&list_uids=94.

41. JGI Finished Genome, 2005. Website: http://genome.jgi-psf.org/finished_microbes/thefu/thefu.home.html.

42. Wilson DB. Studies of Thermobifida fusca Plant Cell Wall Degarding Enzymes. Chem Rec. 2004;4(2):72-82.

43. Adav SS, Ng CS, Sze SK. ITRAQ-based quantitative proteomic analysis of Thermobifida fusca reveals metabolic pathways of cellulose utilization. J Proteomics. 2011 Jun 17. [Epub ahead of print].

44. Liu ZF, Chen CY, Tang W, Zhang JY, Gong YQ, Jia JH (August 2006). "Gene-expression profiles in gastric epithelial cells stimulated with spiral and coccoid Helicobacter pylori". J Med Microbiol 55 (Pt 8): 1009–15.

45. Kelly DJ and Hughes NJ. Chapter 12The Citric Acid Cycle and Fatty Acid Biosynthesis.Washington (DC): ASM Press; 2001. Bookshelf ID: NBK2413 PMID: 21290715 http://www.ncbi.nlm.nih.gov/books/NBK2413/

46. Hast MA, Nichols CB, Armstrong SM, Kelly SM, Hellinga HW, Alspaugh JA, Beese LS. Structures of Cryptococcus neoformans protein farnesyltransferase reveal strategies for developing inhibitors that target fungal pathogens. J Biol Chem. 2011 Aug 4. [Epub ahead of print].

47. Garcia J, Shea J, Alvarez-Vasquez F, Qureshi A, Luberto C, Voit EO, Del Poeta M. Mathematical modeling of pathogenicity of Cryptococcus neoformans. Mol Syst Biol. 2008;4:183. Epub 2008 Apr 15.

48. Brown SM, Upadhya R, Shoemaker JD, Lodge JK. Isocitrate dehydrogenase is important for nitrosative stress resistance in Cryptococcus neoformans, but oxidative stress resistance is not dependent on glucose-6-phosphate dehydrogenase. Eukaryot Cell. 2010 Jun;9(6):971-80. Epub 2010 Apr 16.

49. Pricea MS, Quiroza MB, Pricea JL, Toffalettia DL, Voraa H, Hub G, Kronstadb, JW, Perfecta JR. Cryptococcus neoformans Requires a Functional Glycolytic Pathway for Disease but Not Persistence in the Host mBio vol. 2 no. 3 e00103-11.

50. Hu G, Cheng PY, Sham A, Perfect JR, Kronstad JW. Metabolic adaptation in Cryptococcus neoformans during early murine pulmonary infection. Mol Microbiol. 2008 Sep;69(6):1456-75. Epub 2008 Jul 30.

51. Kanehisa, M., Goto, S., Furumichi, M., Tanabe, M., and Hirakawa, M.; KEGG for representation and analysis of molecular networks involving diseases and drugs. Nucleic Acids Res. 38, D355-D360 (2010).

52. Urbanczik R. Enumerating constrained elementary flux vectors of metabolic networks. IET Syst Biol. 2007 Sep;1(5):274-9.

53. Lee, S., C. Phalakornkule, M.D. Domach and I.E. Grossmann, "Recursive MILP Model for finding all the Alternate Optima in LP models for Metabolic Networks," Computers and Chemical Engineering 2000; 24: 711-716.

VITA

Umaporn Siangphoe:

Education:

Ph.D., Biostatistics, Virginia Commonwealth University, VA, USA (2011-present)
M.Sc., Bioinformatics, (Quantitative/Statistics), Virginia Commonwealth University, VA, USA
M.Sc., Biostatistics, Mahidol University, Bangkok, Thailand
B.Ns., Nursing Science, Mahidol University, Bangkok, Thailand

Awards/ Scholarship:

Graduate Assistant Scholarship, Department of Biostatistics, School of Medicine, Virginia Commonwealths University, VA
Graduate Bioinformatics Scholarship 2010-11, School of Life Science, Virginia Commonwealths University, VA