RESEARCH AND DEVELOPMENT OF A GENETIC DESIGN TOOL

by

Brahmaji Mutthoju

A thesis submitted to the

Graduate School-Camden

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of Master of Science

Graduate Program in Computer Science

written under the direction of

Dr. Desmond S. Lun

and approved by

Dr. Desmond S. Lun

(Committee Member and Thesis Director)

Dr. Michael A. Palis

(Committee Member)

Dr. Suneeta Ramaswami

(Committee Member and Program Director)

Camden, New Jersey October 2013

THESIS ABSTRACT

RESEARCH AND DEVELOPMENT OF A GENETIC DESIGN TOOL

By Brahmaji Mutthoju

Thesis Director:

Dr. Desmond S. Lun

Computers are aiding biotechnology researchers to rapidly develop new engineered strains of organisms with favorable product accumulation. The organisms being engineered can be represented as constraint based metabolic models. These models are analyzed using computational methods to find optimal genetic manipulation strategies such as gene knockouts. Due to growing biological knowledge, there has been a corresponding increase in the size and complexity of the *in silico* metabolic models. Therefore the biotechnology research community needs access to an effective computational method and an efficient implementation of the method that reasonably quickly solves the problem of searching for genetic manipulations. Furthermore, the biotechnology researchers are of diverse backgrounds and their computer skills are different. This should not hinder the community from using these computational techniques. In order to address the above requirements, an efficient and user-friendly software solution has been proposed and developed for discovering strategies that may enhance the yield of chemical compounds of interest.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Dr. Desmond Lun, my thesis advisor and mentor, for his patient guidance, enthusiastic encouragement and useful critiques of my research and development work. I would also like to thank Mr. James Kelly for helping me with software design and for keeping my project files safe. My grateful thanks to Dr. Rajiv Gandhi, Dr. Alok Baveja and Dr. Andrei Nikiforov for motivating me to work hard and improve my knowledge in Computer Science.

My sincere thanks to Dr. Abhishek Bhattacharjee and Dr. Ricardo Bianchini of Department of Computer Science at Rutgers New Brunswick, for teaching me the fundamental concepts of computer architecture especially about memory hierarchy and dynamic scheduling of modern microprocessors which have been a great use in conducting my research and writing this thesis.

I would also like to extend my thanks to Dr. Suneeta Ramaswami for giving me an opportunity to be part of Department of Computer Science at Rutgers Camden.

Finally, I wish to thank my parents and my brother for their support and encouragement throughout my study.

DEDICATION

Dedicated to my Creator who taught me how to read and write

Contents

List of Figures

List of Tables

1 Introduction

Many organisms have the capability to produce compounds which can be used in chemicals, food and drugs. Micro-organism *E. coli* for instance can produce acetate and succinate. Succinate is an important chemical compound used in food and pharmaceutical industries. *Bacillus subtillis*, another micro-organism has the ability to produce antibiotics, high quality enzymes and proteins, nucleosides, and vitamins [1, 2]. Such organisms can be used as cell factories to sustainably produce bulk chemicals, pharmaceuticals, food ingredients, enzymes and other products. However, the product yields of these organisms have been identified to be very low as compared to their theoretically possible maximums [3].

In order to enhance the yield of these compounds, the organisms are genetically engineered to obtain new strains. Genetic engineering is a process of manipulating the genome (the hereditary information) of an organism. One of the types of genetic manipulations is the removal of genes called ”gene knockouts”. Such a manipulation is accomplished through the use of nuclease, an enzyme that can cleave certain bonds in Deoxyribonucleic Acid (DNA) or Ribonucleic Acid (RNA). These knockouts ensure that the production of the desired compound is an obligatory bi-product of growth [3]. Nevertheless, the space of all possible genetic manipulations is too vast to be practically tried out with actual organisms.

To search for favorable genetic manipulation strategies, a number of computational methods have been developed which are based on *in silico* (computer based) models of organisms' metabolic networks. Metabolic network is a network of reactions related to the organism's metabolic function of converting available raw materials to energy. The computational methods however have different computational complexities and hence different efficiencies. The methods are also not directly accessible to users unfamiliar with the computational methods.

To aid genetic engineering in identifying new strains of organisms, an accessible, efficient and user friendly genetic design tool would be beneficial. The genetic design tool must be available to all the potential users which improves accessibility and also enables contribution to the software. The tool must be based on a computational method that identifies genetic manipulations efficiently to enable rapid genetic engineering. The tool should be implemented efficiently to achieve good performance even on low-end or medium-end computers. Furthermore, the tool must be user-friendly to users with diverse backgrounds and different technical skills in computer usage.

1.1 Statement of the Problem

The challenges present in developing a genetic design software tool for identifying manipulation strategies which enhance an organism's capability to produce compounds of interest are related to three important aspects of a software system: Accessibility, efficiency, and usability. If the genetic design tool is not accessible, it would not be reachable to a wide range of users. Secondly, if the genetic design tool does not identify strategies efficiently utilizing available computational resources economically, the tool may not be able to serve rapid engineering and may in turn affect accessibility due to a demand for higher-end computing systems. Finally, if the genetic design tool is not usable by all the potential users of the system, the software system may render itself to be impractical for aiding genetic engineering.

Accessibility: Accessibility is an important facet of a software system because lack of this attribute creates an impediment to a wide range of users from using the system. Accessibility is defined as the degree to which a product, device, service, or environment is available to as many people as possible. A software system with good accessibility will be able to reach users in academic settings, users in industrial settings and hobbyists.

A genetic design tool that is based on proprietary computational method and a software implementation using proprietary tools and packages which need licensing fee may hinder its accessibility to a wider range of users.

Efficiency: In addition to having the ability to use the software system, the users may also benefit from a computationally efficient algorithm and performance optimized implementation of the algorithm. Efficiency is defined as the extent to which time, effort or cost is used well for the intended task or purpose. An efficient software system utilizes time more economically which may result in increased productivity or may even be able to support immediate needs. A genetic design tool is based on a computational method and a programming language based implementation of the computational method.

The computational method should be efficient to search the vast space of possible genetic manipulations. This space becomes larger as the size and complexity of the organisms being studied increases. An inefficient computational method may result in a genetic design software tool that is impractical for identifying genetic manipulation strategies for large and complex organisms.

Furthermore, an efficient computational method is necessary but may not be sufficient for obtaining an efficient software application. An inefficient implementation of the computational method will not be able to utilize the available computational resources economically and as a result it may demand faster computers which are often expensive. This may in turn affect the accessibility of the genetic design tool for users with medium-end computing systems.

Usability: Finally, an accessible, efficient software system when combined with usability may further benefit the users. Usability is defined as the ease of use and learnability of a human-made object such as a software application. A software system that is not usable may not serve the purpose for which it was constructed in providing service to users in need. The users of the genetic design tool may be of diverse backgrounds with different levels of familiarity with computational methods for metabolic engineering or with different technical skills in computer usage. As a consequence, an unfriendly genetic design tool may fail to aid effective metabolic engineering.

1.2 Background and Need

The research addressed three principle areas related to genetic design software application development. The first area dealt with improving accessibility. The second area dealt with efficiency and the third area addressed usability of the genetic design tool.

The first area addressed in the study was accessibility of the genetic design tool to a wide range of users. This may be accomplished by providing a software implementation that utilized freely accessible computational method, tools and supporting software libraries. The computational method implemented was Genetic Design through Branch and Bound (GDBB) which is accessible [4]. The programming language used was Java which has free support and the requisite software tools and packages which can be downloaded and installed for free. Finally, the computational method GDBB required Mixed Integer Linear Programming (MILP) solver and hence Gurobi v5.5 solver was employed which is free to use under academic license.

The second area that was dealt with in the study was efficiency of the genetic design tool. Two major aspects namely computational method and implementation performance that affect efficiency of the software system were addressed. GDBB is a computational method that relatively quickly identifies genetic manipulation strategies as compared to previous methods such as OptKnock [3] and Genetic Design through Local Search (GDLS) [5]. OptKnock is a method that uses global search which searches the entire space of all possible genetic manipulations and GDLS uses local search which searches for a fixed number of manipulations starting from a set of previously identified best strategies. On the other hand, GDBB uses *Branch and Bound* optimization method and reports all intermediate near optimal solutions with each solution having higher optimal value than the previously reported solutions.

Another aspect that affects efficiency of the genetic design tool is the implementation of the computational method. To utilize the available computational resources (such as CPU and memory) economically, effective code optimization principles used for numerical computations were applied [6]. This included optimizing code to improve data locality [7], utilizing super-scalar architecture and avoiding unnecessary operations to reduce load on the functional units of the CPU (such as adders, multipliers and dividers).

The third area that the study addressed was usability of the genetic design tool by a wide group of users. Irrespective of user's familiarity with computational methods and technical skill in computer usage, the software tool must be user-friendly to the potential users of the application. In order to address this aspect, user interface design principles [8] were applied to design the user interface and "User Testing Through Thinking Out Loud" [9] intervention was used to identify user interface bugs which guided in making necessary design changes to alleviate any trouble in the usage of the genetic design tool.

1.3 Research Questions

The following research questions corresponding to the three areas namely accessibility, efficiency and usability will be addressed in this study.

1. How can a genetic design tool be implemented efficiently?
2. What software packages and tools improve the accessibility of genetic design tool?
3. How can the usability of the genetic design tool be improved for the purpose of metabolic engineering research?

1.4 Purpose of Research

1.4.1 Purpose Statement

The purpose of the study was to use GDBB computational method, code performance optimization principles and user interface design principles to construct a software tool that identifies genetic manipulation strategies for aiding industrial and medical biotechnology research to optimize the production of certain compounds of interest.

1.4.2 Rationale for Research

Certain organisms are capable of producing compounds of interest which are used in food and drugs. In order to increase the yield of the compounds, the organisms are genetically engineered. To search for possible genetic manipulations efficiently, computational methods based on computer based models of metabolic networks of organisms have been developed. Among many computational methods, an efficient method that can quickly identify strategies for large and complex models is more likely to support rapid genetic engineering. Additionally, an efficient implementation of the computational method may be beneficial since an unoptimized implementation may not utilize computational resources economically which may increase the need for a faster computer system and in turn affect the availability of the tool to all potential users. Furthermore, these computational methods are not user-friendly to those users who are unfamiliar with the computational methods or who lack technical expertise in computer usage.

1.4.3 Description

In order to aid industrial and medical biotechnology research in identifying genetic manipulation strategies to increase production of compounds of interest, the researcher developed a software tool implementing GDBB computational method [4], code performance optimization principles [6, 7, 10] and user interface design principles [8]. GDBB identifies near-optimal gene knockout strategies in seconds or minutes [4]. The computational method was implemented in Java programming language and solved using Gurobi MILP solver (Gurobi Optimization).

Performance optimization principles such as a combination of compiler optimization techniques, and memory access optimization strategies which take into account the modern superscalar computer architecture and memory hierarchy, are expected to enhance the efficiency of the software implementation [6, 7, 10]. Subroutine level profiling was applied to identify inefficient code sections and for each construct, appropriate optimization technique listed in table A.1 was applied. Timing template as given in appendix B was used as an intervention to measure efficiency gain before and after the application of the optimization technique.

User interface design principles are intended to improve usability of the software for industrial and medical biotechnology research community considering their different levels of computer usage skills. The principle of user testing [11] which is a user interface design principle was used as an intervention for user interface testing. An undergraduate student in dentistry from India and two PhD students in Computational Biology department at Rutgers Camden were introduced to the principle and the results of the observation were recorded. Finally the narrative data was used to fix bugs in the user interface.

1.4.4 Expected Results

As a result of the implementation of GDBB, performance optimization principles for coding, optimal utilization of memory hierarchy of a computer system and user interface design principles, we are expected to obtain an easily accessible, efficient and user-friendly standalone software application for genetic design catering to the needs of biotechnological research.

1.5 Significance to the Field

An efficient computational method which is capable of quickly identifying near optimal gene knockout strategies and progressively finding better strategies with time has been implemented. This may result in rapid genetic engineering which may be beneficial to meet the demands of chemical and pharmaceutical industries.

Code optimization principles have improved the performance of the implementation which has resulted in utilizing the available computational resources more economically as compared to un-optimized implementation. This may decrease the demand for buying faster computers to run problems of higher complexity and size. As a result, the application may be able to run on medium-end machines which could improve the availability of the software to a wider range of users.

The user interface design principles and the User Testing Through Observation [9] intervention were able to enhance the user experience of the genetic design tool among the sample group of users. Hence the method may prove effective to further enhance the usability of the genetic design tool for real users of the application.

Finally, the packages utilized to develop the software are freely available through academic license which has enabled the software to be made available for download and use. This leads to improvement in accessibility and may even lead to contribution towards improvement of other aspects of the software such as the computational method and software efficiency.

1.6 Definitions

- Branch and bound: An algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization
- Combinatorial optimization: optimization problem of finding an optimal object from a finite set of objects.
- Discrete optimization: optimization problem in which variables are restricted to assume only discrete values such as integers.
- Gene: A gene is a molecular unit of heredity of a living organism. It is widely accepted by the scientific community as a name given to some stretches of DNA (Deoxyribonucleic Acid) and RNA (Ribonucleic Acid)

- Genetic engineering: manipulation of an organism's genome using biotechnology. For example, genes may be removed or knocked out using nuclease (an enzyme capable of cleaving certain bonds in nucleic acids).
- Genome: entirety of an organism's hereditary information. It is encoded in DNA or in RNA (in many viruses)
- Linear optimization: it is a technique for optimization of a linear objective function, subject to linear equality and linear inequality constraints.
- Memory hierarchy: It is the organization of a memory subsystem in the form of different levels based on response time. There are four major storage levels
  - Internal: Processor registers and cache
  - Main: System RAM and controller cards
  - Off-line bulk storage: Tertiary and off-line storage
  - On-line mass storage: Secondary storage
- Superscalar architecture: A superscalar CPU architecture implements a form of parallelism called instruction level parallelism within a single processor. Instruction level parallelism is a measure of how many of the operations in a computer program can be performed simultaneously.
- User interface (UI): It is the system by which users interact with a machine.

1.7 Limitations

One of the limitations of the computational method GDBB is that the algorithm is designed to find near optimal solutions in order to improve efficiency. Truncation time is used to define the time at which the optimization must terminate. Thus the solution obtained may not be the optimal solution for the problem.

Another limitation that affects external validity is the results pertaining to efficiency. The gain in efficiency was observed on a machine with Intel core i5 processor running windows 7 operating system. This machine is based on Reduced Instruction Set Computer (RISC) architecture and memory hierarchy with multiple levels of caches. Hence the results may not apply to computers based on other architectures and/or computers with no memory hierarchy.

The researcher has limited access to participants for usability testing. The participants are not directly associated with industrial or medical biotechnology research. Additionally, the sample size is small which includes one undergraduate student in dentistry and two computational biology students. As a result, the current results may affect external validity, i.e. the generalizability of the results to the actual setting of industrial or medical biotechnology research groups.

2 Review of the Literature

The literature review will address two areas related to the genetic design tool for identifying manipulation strategies that may lead to the overproduction of compounds of interest. The first section will address theory behind computer aided genetic design and research related to computational methods to identify genetic manipulation strategies. The second section will focus on theory related to architectural features that affect software performance and studies on effective code optimization principles.

2.1 Computational Methods for Genetic Manipulations

2.1.1 Applications of Computer-based Modeling of Biological Systems

One of the major reasons for building models of biological systems is to discover new genetic manipulation strategies for producing essential chemical compounds. The models constructed for this purpose are genome scale *in silico* models. Genome scale *in silico* models are used to drive engineering of biological systems which have applications in the fields of medical and industrial biotechnology.

In the field of medical biotechnology, *in silico* metabolic models have been used to aid the production of nutrients and dietary supplements and to improve the production of drugs. Consider the example of *Corynebacterium glutamicum*. This bacterium is used industrially to produce amino acids, L-lysine, L-glutamate and can produce organic acids under oxygen deprivation conditions. A genome-scale model (GEM) of the bacterium was used to find candidate gene deletions to increase organic acid production under oxygen-deprived conditions [12]. The model was also used to improve lactate production by interrupting succinate-producing reactions and by disrupting oxidative phosphorylation reactions.

Another bacterium *Bacillus subtilis* has the ability to produce antibiotics, high quality enzymes and proteins, nucleosides, and vitamins which makes it an important industrial organism [13, 14]. Similarly, the bacterium *Streptomyces coelicolor* produces antibiotics, immunosuppressants and anti-cancer agents.

An essential model that may be studied in the near future is the human metabolic GEM. Currently, 99% of human euchromatic[1] sequence has been completed [15] and if this task is accomplished, *in silico* metabolic analysis methods will be useful for studying human physiology and pathophysiology [16].

Two of the best studied microbial species to date are *E. coli* and *S. cerevisiae.* These organisms from which much about biology has been learned serve as critically important. *E. coli* metabolic GEM has been extensively used in applications including increased production of lycopene [17, 18], succinate [3, 19, 20], lactate [3, 21], malate [22], L-valine [23], L-threonine [24], additional amino acids [25], vanillin [26], 1,3-propanediol [3] using the metabolic models [27–29].

To identify genetic manipulation strategies for increasing the production of lactate and vanallin in *E. coli*, computational methods such as Opt-Knock [3] and OptStrain [26] were used. Quadruple gene deletions proposed by Opt-Knock were tested experimentally and this resulted in a strain capable of an increased lactate production of 0.87-1.75 g/L per 2 g/L of glucose [21]. OptStrain identified three reactions to be introduced into *E. coli* for vanillin production [26]. Then subsequently Opt-Knock was used to systematically search for gene deletions to enhance vanillin yield [3].

2.1.2 Computer-based Modeling of Biological Systems

Genome scale *in silico* models of the organisms employed for identifying genetic manipulation strategies are represented as constraint-based metabolic models. The approach for constructing these models is based on successive imposition of different physico-chemical, topobiological and environmental constraints.

[1]Euchromatin is a packed form of DNA, RNA or protein that is rich in gene concentration

Physico-chemical constraints exist due to dense packing of the interior of a cell. As a result, the viscosity of the cell is 100 to 1000 times that of water and consequently the diffusion rate for macromolecules is probably slow. Furthermore, the confinement of a large number of molecules within a semipermeable membrane causes high osmolarity [2], sodium-potassium pumps generate osmotic pressure and hence the cell requires a mechanism to balance osmolarity or the rigid cell wall (to physically withstand it). Additionally, the intracellular reaction rates are determined by concentration inside a cell. Finally, the biochemical reactions need to have negative free energy change to proceed in the forward direction.

Topobiological constraints are three dimensional constraints which arise due to crowding of molecules inside a cell. For example, the linear dimension of bacterial genome is on the order of 1000 times the length of a cell. Therefore the DNA must be tightly packed in the nucleus of the cell in an accessible and functional configuration because DNA is functional only if it is accessible. As a consequence, the physical arrangement of a bacterial genome is constrained.

Environmental constraints occur due to nutrient availability, pH, temperature and osmolarity which are condition dependent. Consider the example of *E. coli* whose life cycle is subject to sudden environmental changes. The environment outside the animal is ambient and there is a presence of ample amount of oxygen. When *E. coli* enters an animal's mouth it experiences a heat shock. Subsequently when *E. coli* reaches the animal's stomach it experiences an acid shock. After that, *E. coli* enters the animal's intestine and undergoes a pH shock and nutritionally rich anaerobic environment. In this environment, *E. coli* grows rapidly in the presence of other bacterial species. Finally *E. coli* receives a cold shock and ample oxygen with diminishing nutrients. As a consequence of the changing environment, *E. coli* must adjust its internal functional state in order to survive.

[2]Osmolarity is the solute concentration in a solution measured in osmoles per litre

2.1.2.1 Mathematical Representation of Constraints

The physico-chemical constraints, topobiological constraints and environmental constraints are mathematically defined as balances and bounds. Balances are equality constraints and bounds are inequality constraints. Consider conservation of mass for instance. This physico-chemical constraint is mathematically defined using the balance constraint. Under steady state condition, compounds neither accumulate nor deplete and hence for each compound x_i the production rate is equal to the consumption rate. This is mathematically represented as

$$\frac{d\mathbf{x}}{dt} = \mathbf{S}\mathbf{v} = 0 \tag{2.1}$$

where $\mathbf{S}$ is an $m \times n$ stoichiometric matrix as given below

$$\mathbf{S} = \begin{bmatrix} \ddots & \vdots & \\ \dots & S_{ij} & \dots \\ & \vdots & \ddots \end{bmatrix}$$

whose ij^{th} element S_{ij} is the coefficient of the metabolite i in a metabolic reaction j. $\mathbf{v}$ is an n element flux vector whose j^{th} element, v_j is the rate of reaction j with respect to time.

An important physico-chemical constraint that needs to be taken into account is the thermodynamic constraint. This constraint is represented as a bound on the flux through a metabolic reaction. The mathematical representation of the constraint for the fluxes through all the reactions in the metabolic network is represented as,

$$\mathbf{v}_{min} \leq \mathbf{v} \leq \mathbf{v}_{max} \tag{2.2}$$

where, $\mathbf{v}_{min}$ and $\mathbf{v}_{max}$ are n dimensional vectors of minimum and maximum flux values respectively. If a reaction is irreversible, then the minimum flux value is non-negative which is mathematically represented as $\mathbf{v}_{min} = \mathbf{0}$.

A third physico-chemical constraint is the solvent capacity constraint. Assuming the biological system is in a steady state, this constraint limits the concentration of the compounds that participate in metabolic reactions. Such a constraint can be defined by setting upper bounds on compound concentrations. This can be mathematically represented as,

$$x_i \leq x_{i,max} \tag{2.3}$$

where x_i is a metabolite and $x_{i,max}$ is the maximum concentration limit on the metabolite.

The fourth type of physico-chemical constraint is the kinetic constraint. The metabolic reactions are shown to be constrained by maximal reaction rates. Two factors affect the kinetic constraints. First, the kinetic constants must be numerically positive. Second, the collision frequency of molecules defines upper bound on the kinetic constants. Taking these factors into consideration, constraints on kinetic constants can mathematically be represented as

$$0 \leq k \leq k_{max} \tag{2.4}$$

Thus the physico-chemical, topobiological and environmental constraints limit the possible functional states that can be attained by the metabolic network. This constrained network forms a closed space whose properties can be studied.

A number of *in silico* methods have been developed to study the properties of genome-scale networks [30]. One of the methods is to find solutions of interest in the constrained metabolic network space. If the space is constrained by a set of linear equality and linear inequality constraints, the solution space is a polytope which is a space with flat sides. The solution to be found can be represented by an objective function. If the objective function is linear, the optimal solution to the objective function can be obtained by applying linear optimization (LP).

2.1.3 Computational Methods

An objective can be used to study the capability of an organism's metabolic network. One of the important objectives to consider is the maximization of a metabolite production. The problem under consideration is to study the biochemical capabilities of *E. coli*.

A method popularly studied to find the capabilities of metabolic networks is the Flux Balance Analysis (FBA). In this analysis, the restricted knowledge of the parameters of a biological system are combined with thermodynamic principles to generate predictions and verifiable hypotheses [31].

Consider a metabolic network with M metabolites. The sum of all production and consumption fluxes in steady state weighted by stoichiometric coefficients of the reactions is zero as given in the equation (2.1). The flux vector $\mathbf{v}$ in the case of FBA includes exchange fluxes which represent the rate of metabolite transport through the cell membrane. The reason for making the steady state approximation is that the metabolite concentration reaches equilibrium much faster than (in seconds) the rate of genetic regulation (minutes) [32, 33].

Additional constraints that correspond to nutrient availability and maximal fluxes are applied in the form of inequalities as follows,

$$\alpha_j \leq v_j \leq \beta_j \tag{2.5}$$

where α_j and β_j are j^{th} elements of n dimensional vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ respectively which contain either measured or imposed values.

The FBA problem can be represented mathematically by the equation given below.

$$\begin{aligned} \max \quad & \mathbf{f}^T\mathbf{v} \\ \text{subject to} \quad & \\ & \mathbf{S}\mathbf{v} = 0 \end{aligned} \tag{2.6}$$

$$\alpha \leq \mathbf{v} \leq \beta$$

where $\mathbf{f}$ is an n dimensional vector of coefficients that represents the objective function under consideration intended to be maximized under the given constraints (2.1) and (2.5).

FBA computes the flow of metabolites through the metabolic network. This can be used to predict the growth rate of an organism as well as important metabolites. the method was applied to predict the maximum growth rate of *E. coli* in the presence as well as in the absence of oxygen.

In order to apply FBA, a phenotype[3] must be defined as a biological objective which is relevant to the problem under consideration. Since we are predicting growth rate, the objective is biomass production. Biomass production can be defined as the rate of conversion of metabolic compounds into biomass constituents such as nucleic acids (DNA, RNA), proteins, lipids (fats, fat soluble vitamins A, D, E and K) and so on.

Biomass production can be mathematically defined by augmenting an artificial column of coefficients in the stoichiometric matrix. This column corresponds to the consumption of precursor metabolites. Precursor metabolites are compounds that participate in certain chemical reactions that produce monomers such as amino acids (which polymerize to form proteins) and nucleotides (which polymerize to form nucleic acids such as DNA and RNA).

The basis for defining the biomass reaction is the experimental measurements of biomass components. To make one gram of *E. coli* biomass, the precursor requirements are given as

$$\begin{aligned} f_{precursor} = &+0.205v_{G6P} + 0.071v_{F6P} + 0.898v_{R5P} \\ &+0.361v_{E4P} + 0.129v_{T3P} + 1.496v_{3PG} \\ &+0.519v_{PEP} + 2.833v_{PYR} + 3.748v_{AcCoA} \\ &+1.787v_{OAA} + 1.079v_{\alpha KG} \end{aligned} \tag{2.7}$$

[3]Phenotype is an organism's observable characteristic

where, G6P is D-glucose 6-phosphate; F6P is D-fructose 6-phosphate; R5P is Alpha-D-ribose 5-phosphate; E4P is D-erythrose 4-phosphate; T3P is Triose 3-phosphate; 3PG is 3-phospho-D-glycerate; PEP is Phosphoenolpyruvate; PYR is Pyruvate; AcCoA is Acetyl coenzyme A; OAA is Oxaloacetate and αKG is Alpha ketoglutarate.

Additionally, the cofactor requirements for synthesizing monomers such as amino acids (building blocks of proteins) and nuleotides (building blocks of DNA and RNA) from precursors and then subsequently for polymerizing the monomers to macromolecules is given by

$$f_{cofactors} = 42.703v_{ATP} - 3.547v_{NADH} + 18.22v_{NADPH} \tag{2.8}$$

where, ATP is Adenosine triphosphate; NADH is Nicotinamide adenine dinucleotide - reduced and NADPH is Nicotinamide adenine dinucleotide phosphate - reduced.

Therefore the mass and cofactor requirements needed to make *E. coli* biomass is the sum of the quantities of precursor requirements and the cofactor requirements as given in the equation (2.9) [34].

$$f = f_{precursors} + f_{cofactors} \tag{2.9}$$

FBA was applied to calculate the aerobic growth rate of *E. coli* assuming that the uptake of glucose, and not oxygen are the limiting constraints on growth. The calculation was carried out using the published model of *E. coli* metabolic network. This model includes glucose and oxygen uptake into the cell in addition to metabolic reaction and biomass reactions [35].

The assumptions were mathematically represented by setting the maximum uptake rate of glucose to a physiologically realistic level of 18.5 mmol gDW^{-1} h^{-1} (DW is Dry Weight) and the maximum uptake rate of oxygen to an arbitrarily high level.

The problem was solved using LP to obtain maximum possible flux through the biomass reaction. FBA under aerobic conditions (abundance of oxygen) predicted an exponential growth rate of 1.65 h^{-1} and under anaerobic conditions (absence of oxygen) it predicted a growth rate of 0.47 h^{-1}.

There are several limitations for FBA. First, FBA does not account for kinetic parameters. Second FBA cannot predict metabolic concentrations but only flux distribution at steady state. Third, FBA does not consider regulatory effects such as activation of enzymes by protein kinases and regulation of gene expression. As a result, the predictions made by FBA may not always be accurate.

FBA is applied to predict the effects of partial or complete gene deletions on the phenotype of an organism. To simulate gene knockouts, constraints can be modified by limiting the reactions that are affected by the gene knockouts to zero flux. This concept has been applied to devise methods that are based on FBA analysis to maximize metabolite production. This can be accomplished by exploring a trade-off between the conflicting objective of the organism and metabolite production.

2.1.3.1 Organisms as Cell Factories

The behavior of the networks of micro-organisms is governed by internal cellular objectives which compete with the targets of chemical overproduction. In order to deal with these challenges, a bi-level optimization framework termed OptKnock has been developed to suggest gene knockouts for biochemical overproduction while recognizing that the metabolic flux distributions are governed by internal cellular objectives [3].

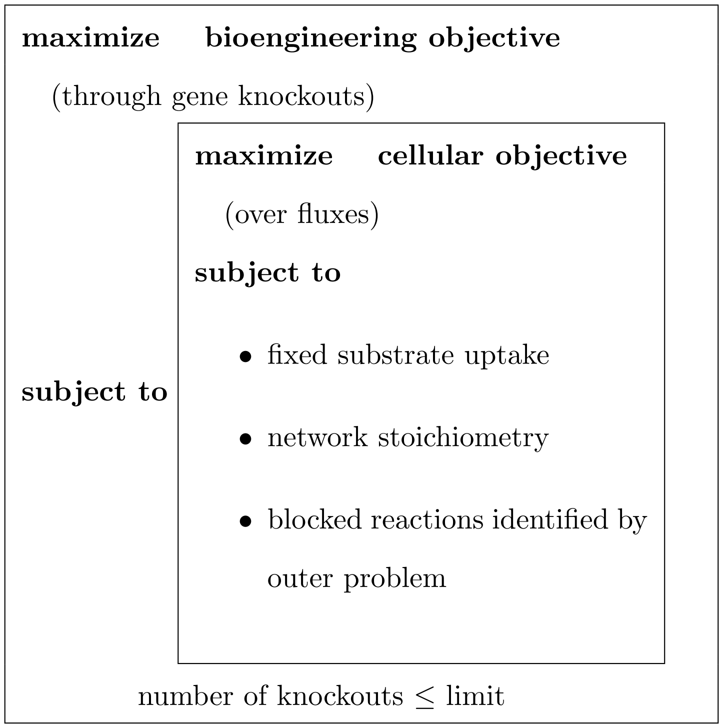
Two cellular objectives were considered for OptKnock. (1) Maximization of biomass yield and (2) minimization of metabolic adjustment (MOMA) since FBA models usually invoke optimization of specific cellular objectives such as ATP production [36, 37], biomass formation [38] and MOMA [31] which are subject to network stoichiometry. These objectives suggest a probable flux distribution.

The biomass maximization hypothesis has been shown to be successful in certain cases when applied to the stoichiometric models of *E. coli* metabolism. For instance in the prediction of the lethality of gene knockouts [27] and in the identification of correct sequence of byproduct secretion under increasing anaerobic conditions [39].

OptKnock framework was introduced for predicting knockout strategies resulting in overproduction of specific chemical compounds in *E. coli.* This is accomplished by making sure that the chemical of interest is an obligatory by-product of growth achieved as a consequence of re-shaping the connectivity of the metabolic network. OptKnock framework identifies and removes the metabolic reactions which can uncouple the cellular growth from chemical production. The model considered was an *in silico* abstraction of *E. coli* metabolic network of Palsson and co-workers [27].

OptKnock was applied for succinate, lactate and 1,3-propanediol production in *E. coli* with the cellular objective of maximizing biomass yield assuming a fixed quantity of uptake glucose. The results of the approach that maximized biomass yield were compared and contrasted with the results of the approach that hypothesized MOMA as the cellular objective.

In order to identify the combinations of multiple gene deletions that optimally couple cellular growth objectives with an imposed chemical production target, the following multilayered optimization framework given in the figure below was developed [3].


maximize bioengineering objective
(through gene knockouts)
subject to
maximize cellular objective
(over fluxes)
subject to
• fixed substrate uptake
• network stoichiometry
• blocked reactions identified by outer problem
number of knockouts ≤ limit

The framework takes into account two competing optimality strategies. First the cellular objective and second the chemical production. This multilayered optimization framework is referred to as the bi-level optimization problem [40].

The problem of maximizing cellular objective is quantified as an aggregate reaction flux for a steady state metabolic network consisting of $\mathcal{N} = \{1, \ldots, N\}$ metabolites and $\mathcal{M} = \{1, \ldots, M\}$ reactions. The network of reactions is fueled by glucose substrate. The mathematical representation of the problem is as follows,

$$\begin{aligned}
&\max \quad \mathbf{v}_{cellular\ objective} && (2.10)\\
&\text{subject to}\\
&\quad \sum_{j=1}^{M} S_{ij} v_j = 0, && \forall i \in \mathcal{N}\\
&\quad v_{pts} + v_{glk} = v_{glc\_uptake} && mmol/gDW.hr\\
&\quad v_{atp} \geq v_{atp\_main} && mmol/gDW.hr\\
&\quad v_{biomass} \geq v_{biomass}^{target} && 1/hr\\
&\quad v_j \leq 0, && \forall j \in \mathcal{M}_{irrev}
\end{aligned}$$

$$v_j \leq 0, \qquad \forall j \in \mathcal{M}_{secr_only}$$

$$v_j \in \Re, \qquad \forall j \in \mathcal{M}_{rev}$$

where, v_j is the flux through reaction j, $\mathbf{v}_{glc\_uptake}$ represents basic glucose uptake, $\mathbf{v}_{atp\_main}$ is the non-growth related ATP maintenance requirement; $\mathbf{v}_{biomass}^{target}$ is the minimum biomass production; $\mathbf{v}$ is a vector comprising of internal and transport reactions, where forward (positive) direction means uptake of particular metabolites and reverse (negative) direction represents secretion of metabolites; $\mathbf{v}_{pts}$ and $\mathbf{v}_{vglk}$ are rates of uptake of glucose through phosphotransferase system and glucokinase system respectively; $\mathcal{M}_{secr\_only}$ is a set of transport fluxes for those metabolites which can only be secreted from the network; $\mathcal{M}$ is the complete set of reactions; $\mathcal{M}_{rev}$ is a subset of $\mathcal{M}$ consisting of reversible reactions only; $\mathcal{M}_{irrev}$ is a subset of $\mathcal{M}$ comprising only irreversible reactions; cellular objective is assumed to be the drain of biosynthetic precursors in the ratios required for biomass formation [41]. The fluxes are reported as 1 gDW^{-1} h^{-1} and the biomass as g biomass produced gDW^{-1} h^{-1}.

Subsequently, the gene deletions were modeled using a vector $\mathbf{y}$ where its j^{th} element y_j is 1 if the flux through reaction j, v_j is active and 0 if the flux through reaction j, v_j is inactive for all j belonging to the set $\mathcal{M}$. The gene deletion was mathematically represented as

$$v_j^{min}.y_j \leq v_j \leq v_j^{max}.y_j \qquad \forall j \in \mathcal{M} \tag{2.11}$$

which ensures that the reaction flux v_j gets set to 0 if y_j is set to 0 and v_j is free to assume any value between lower limit v_j^{min} and upper limit v_j^{max} if y_j is set to 1. The values of v_j^{min} and v_j^{max} are found by minimizing and maximizing every flux which are subjected to the constraints of the primal problem (2.10) defined previously.

In order to identify optimal gene/reaction knockouts one must solve the bi-level optimization problem which chooses a set of reactions (by setting $y_j = 1$) such that, the optimization of cellular objective leads indirectly to the overproduction of the chemical of interest. Assuming the biomass formation as the cellular objective, the mathematical representation of the problem is as follows,

$$
\begin{aligned}
&\max_{y_j} \quad \mathbf{v}_{chemical} && (2.12)\\
&\text{subject to} \\
&\qquad \max_{v_j} \quad \mathbf{v}_{biomass} && (2.13)\\
&\qquad \text{subject to} \\
&\qquad\qquad \sum_{j=1}^{M} S_{ij} v_j = 0, \\
&\qquad\qquad v_{pts} + v_{glk} = v_{glc\_uptake} \\
&\qquad\qquad v_{atp} \geq v_{atp\_main} \\
&\qquad\qquad v_{biomass} \geq v_{biomass}^{target} \\
&\qquad\qquad v_j^{min}.y_j \leq v_j \leq v_j^{max}.y_j, \quad \forall j \in \mathcal{M} \\
&\qquad y_j = \{0,1\}, \qquad \forall j \in \mathcal{M} \\
&\qquad \sum_{j \in M} (1 - y_j) \leq K
\end{aligned}
$$

Many Computational methods were built based on the OptKnock framework to identify gene manipulation strategies. These methods search the space of all possible genetic manipulations to find strategies that may result in the desired metabolic network state. However, this space is vast and as a result the methods get computationally very intensive as the number of manipulations allowed in the design increase. For large models such as *E. coli* K-12 MG 1655 [42], iAF1260, runtime becomes very large for designs with more than a few manipulations. Consequently, as the number of reactions, metabolites and genes in the metabolic models continue to grow [43], more efficient computational techniques are required for effective *in silico* design.

In order to efficiently search for the space of possible genetic manipulations, a heuristic algorithm called GDLS was developed. This method has the capability to handle large models and allows for larger number of genetic manipulations in the design with runtime scaling linearly with total number of manipulations T. GDLS applies local search with multiple search paths in order to find strategies that are locally optimal. Beginning with a certain strategy as a starting point, GDLS uses MILP to search for best strategies that differ from the starting point by at most k additional manipulations. The total number of strategies maintained by GDLS at any point in time is limited to the size M.

For the next iteration, the current M best strategies are used as the starting point and the above process is repeated which results in at most M best strategies where each of the strategies differs from the previous M strategies by at most k additional manipulations. The procedure is repeated until no more strategies better than the existing ones can be found.

Thus, GDLS can find strategies with T total number of manipulations using computationally feasible increments (determined by k and M).

The paper limited the consideration of bi-level FBA framework to gene knockouts for simplicity but any bi-level FBA framework which can be transformed to MILP can be used such as up or down regulation of genes using OptReg [44].

GDLS implemented reductions to decrease the size of the FBA model without changing its properties. GDLS utilized gene-protein reaction (GPR) mappings which is a many-to-many mapping between genes and reactions. GPR mappings potentially reduces the search space which results in the reduction of search complexity and as a consequence the runtime. Other reductions included removing dead-end reactions and linked reactions [45, 46].

Three principle reductions were applied to the FBA model. (i) Removal of all dead-end reactions (ii) redefining linked reactions and (iii) successively maximizing and minimizing each flux subject to the constraints of the problem respectively.

1. Dead-end reactions are reactions in which the metabolites are associated with only one reaction and hence they do not carry any flux.

2. Linked-reactions are reactions in which the metabolites are associated with exactly two reactions. Since the metabolites are conserved, the fluxes of the two reactions will always be in the same ratio and hence can be reduced to single variable. Thus all

reactions of the form

$$S_{ij_1}v_{j_1} + S_{ij_2}v_{j_2} = 0 \tag{2.14}$$

can be transformed by reducing j_1 and j_2 to a single reaction since

$$v_{j_1} = -\frac{S_{ij_2}}{S_{ij_1}}v_{j_2} \tag{2.15}$$

3. Third type of reduction is accomplished by successively maximizing and minimizing each flux which are subject to the constraints of the problem. This problem is referred to as the max-min problem [47–49]. As a result, we obtain tighter bounds on the fluxes. Thus we solve,

$$\begin{aligned} &\max/\min\ v_j \\ &\text{subject to} \\ &\qquad \mathbf{Sv} = \mathbf{0} \\ &\qquad \mathbf{a} \leq \mathbf{v} \leq \mathbf{b} \end{aligned} \tag{2.16}$$

In order for GDLS to function at the genetic level, GPR mappings were implemented. GPR mappings is a mathematical definition of how certain genetic manipulations affect reactions in the network. If L is the number of genetic manipulations, then GPR mappings is an $L \times n$ matrix. Where, lj^{th} element G_{lj} of $\mathbf{G}$ is 1 if l^{th} genetic manipulation maps to reaction j and 0 if l^{th} genetic manipulation does not map to reaction j.

A previously defined problem for identifying genetic manipulation strategies which was based on the conversion of bi-level optimization problem to its equivalent MILP problem [3] was used. The bi-level optimization problem was defined as follows,

$$\max \quad \mathbf{g}^T\mathbf{v} \tag{2.17}$$

subject to

$$\sum_{l=1}^{L} y_j \leq C,$$
$$y_l = \{0, 1\},$$
$$\max \qquad \mathbf{f}^T\mathbf{v}$$
$$\text{subject to}$$
$$\mathbf{S}\mathbf{v} = \mathbf{0},$$
$$(1-\mathbf{y})^T G_j a_j \leq v_j \leq (1-\mathbf{y})^T G_j b_j, \quad j = 1, \ldots, n$$

where, **g** is the synthetic objective vector whose j^{th} element g_j is the weight of the reaction j in the synthetic objective; **y** is the knockout vector whose l^{th} element y_l is 1 if the genes involved in the manipulation l are knocked out and 0 if the genes in the manipulation are retained; G_j is the j^{th} column of **G** and C is the maximum number of knockouts being allowed.

The bi-level optimization problem (2.18) was converted to its equivalent MILP problem as defined in the case of OptKnock [3]. The dual of the inner biological objective [3, 50] can be obtained as follows

$$\begin{aligned} \max \qquad & \sum_{j=1}^{n} \nu_j b_j - \mu_j a_j \\ \text{subject to} \qquad & \\ & f_j - \sum_{i=1}^{m} \lambda_i S_{ij} - \nu_j + \mu_j - \xi_j = 0 \qquad j = 1, \ldots, n \\ & -D\mathbf{y}^T G_j \leq \xi_j \leq D\mathbf{y}^T G_j \\ & \boldsymbol{\mu}, \boldsymbol{\nu} \geq 0 \end{aligned} \tag{2.18}$$

where, $\boldsymbol{\lambda}$ is the dual variable for the equality constraints associated with the cellular objective (2.18); $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ are the dual variables for lower and upper bounds respectively; $\boldsymbol{\xi}$ is the dual variable for the constraints $v_j = 0$ and $y_j = 1$; and D is a scalar selected to be large enough to make sure that $\boldsymbol{\xi}$ is effectively not constrained when $\mathbf{y}^T G_j$ is not zero. D was chosen to be 100.

If the optimal solutions of the inner primal biological objective problem (2.18) and the corresponding dual problem (2.18) are bounded, the objective function values of both the problems should be equal to each other. Hence the objective functions can be equated to one another and the constraints of both the problems can be collected together. As a result, the bilevel formulation of the problem (2.18) can be transformed to a MILP problem as given below [3].

$$\begin{aligned}
\max \quad & \mathbf{g}^T\mathbf{v} && (2.19)\\
\text{subject to} \quad & && \\
& \sum_{l=1}^{L} y_l \leq C && \\
& y_l \in \{0,1\} && l = 1,\ldots,L \\
& \mathbf{S}\mathbf{v} = \mathbf{0} && \\
& (1-\mathbf{y})^T G_j a_j \leq v_j \leq (1-\mathbf{y})^T G_j b_j && j = 1,\ldots,n \\
& \mathbf{f}^T\mathbf{v} = \sum_{j=1}^{n} \nu_j b_j - \mu_j a_j && \\
& f_j - \sum_{i=1}^{m} \lambda_i S_{ij} - \nu_j + \mu_j - \xi_j = 0 && j = 1,\ldots,n \\
& -D\mathbf{y}^T G_j \leq \xi_j \leq D\mathbf{y}^T G_j && \\
& \mu, \nu \geq 0 &&
\end{aligned}$$

GDLS was applied for acetate and succinate overproduction problems in *E. coli* using iAF1260 as the FBA model. The model was given no available oxygen and 10 mmol gDW^{-1}.h^{-1} of available glucose since succinate and acetate fermentations are usually carried out under anaerobic conditions [51, 52] and the conditions are consistent with good *in silico* production.

GDLS achieved an improvement in the computational time for those solutions that yielded values comparable to the desired flux. The method searches the space of manipulations by propagating sets of best solutions. An increase in the number of search paths correspondingly increases the size of the set propagated which increases the runtime but has the following advantages. Increasing the number of search paths allows finding solutions with greater synthetic fluxes. It also makes the search more robust to inaccuracies in the MILP solver.

GDLS is based on the assumption that good strategies can be obtained by adding manipulations to existing good strategies. However this assumption was observed to be untrue for certain cases by another computational method called GDBB [4].

GDBB was developed as a solution to the problem of finding near-optimal solutions more efficiently as compared to OptKnock and GDLS. GDBB employs a constraint-based model of metabolism and predicts the production of desired metabolite when the organism's network is subject to genetic manipulations. In general, GDBB can be used to find near-optimal genetic manipulation strategies for a setup which has a bi-level optimization framework and which can be converted to a single level MILP problem. For example, near-optimal gene knockout strategies as given in OptKnock [3] and near optimal up and down regulation strategies as in OptReg [44]. The conversion of bi-level optimization problem has led to efficient computational methods for searching the space of genetic manipulations [3, 26, 49].

GDBB employs truncated branch and bound algorithm to handle bi-level optimization framework used in OptKnock [3], OptReg [44] and GDLS [5]. Truncated branch and bound is an adapted version of the standard *branch and bound* algorithm which is used to find solutions to optimization problems. Truncated branch and bound terminates processing after running for a period of time defined as an input. The termination may take place at a feasible near-optimal solution which can be considered sufficient for practical purposes [53].

GDBB was shown to perform better than the previous approaches for finding genetic manipulation strategies on bi-level optimization problems. The method finds near-optimal solutions in seconds or minutes as opposed to days taken by other approaches. The research pertaining to GDBB focused on finding knockout strategies that resulted in favorable metabolic phenotypes.

The definition of the problem for GDBB is the same as the one previously defined for GDLS. The MILP problem (2.19) was set up and passed to Gurobi MILP solver with the following configuration. The solution feasibility tolerance *FeasibilityTol* was set to 10^{-9}; integer feasibility tolerance *IntFeasTol* was set to 10^{-9} for achieving accurate solutions; *Heuristics* was set to 1.0 in order to produce better feasible solutions; *MIPFocus* was set to 1 to give higher priority to solution feasibility over optimality; *ImproveStartGap* was set to ∞ for gaining focus on producing more feasible solutions; and finally the *TimeLimit* was set to a finite value to specify the truncation time for the branch and bound algorithm.

The problem was solved using a truncated branch and bound implementation of Gurobi solver (Gurobi Optimization). Gurobi is a standalone, commercial solver used for LP, quadratic programming (QP) and mixed integer programming (MIP) which includes MILP & mixed integer quadratic programming (MIQP) problems.

The space of possible gene knockout strategies was searched for a maximum running time of 86400 *s* (24 *h*). The running time was an arbitrary choice. GDBB found solutions comparable to those of GDLS and global search with an improvement in its running time of one or two orders of magnitude. GDBB was able to find solutions within 24 *h*, which the other approaches were unable to find.

GDBB found a knockout strategy that resulted in an acetate production flux of 19.232 mmol gDW^{-1} h^{-1} which was an increase of 2.62278 mmol gDW^{-1} h^{-1}, a 14% over the best solution found by GDLS in 24 h period. GDBB was able to predict the strategy in just 81 seconds.

In the case of succinate production, GDBB was observed to show a smooth increase in the synthetic flux as the number of knockouts increased, hence it is believed that the maximum synthetic flux that can be achieved is somewhere around 12 mmol gDW^{-1} h^{-1}.

GDBB and GDLS can find similar solutions with a 14-16 knockout in 24 hour time period. All the knockout strategies discovered by GDBB were reached within 6 minutes and no further improvement was found in the synthetic flux by allowing the method to run for 24 hours. Additionally, GDBB found solutions which made more biological sense being more consistent with previous experimental findings as compared to GDLS.

The utility of GDBB could increase with more complex problems. The researchers predicted that the complexity of a genetic manipulation problem may increase in two ways. (1) The size of the metabolic models will increase as the biological knowledge about the organisms being studied grows or if a more complex organism than *E. coli* is being considered. (2) Modifications other than knockouts such as increasing gene expressions can be considered.

One of the limitations of GDBB is that the truncation time limits GDBB to finding near-optimal solutions only. Since the truncation time is defined with an arbitrary value it does not have any correlation with the rate at which GDBB discovers new strategies with higher flux values for a given number of maximum knockouts allowed.

2.1.4 Summary

Since GDBB was shown to perform better as compared to GDLS and OptKnock, it is more favorable as an effective computational method for an efficient software implementation to find near optimal gene manipulation strategies.

2.2 Theory Driving Efficiency Considerations

This section will address the aspects of computer hardware and software systems that will impact the performance of the genetic design software tool. The first subsection will discuss memory subsystem in a typical modern computer. The second subsection will deal with modern microprocessor architecture. The final subsection will address modern optimizing compilers and how they exploit the hardware features to produce an optimal software system.

2.2.1 Memory Hierarchy

The modern computer systems are based on RISC (Reduced Instruction Set Computer) design. Memory subsystem of RISCs are organized in a hierarchical fashion. This hierarchy can be visualized as a pyramid consisting of different layers as shown in the figure 2.1.

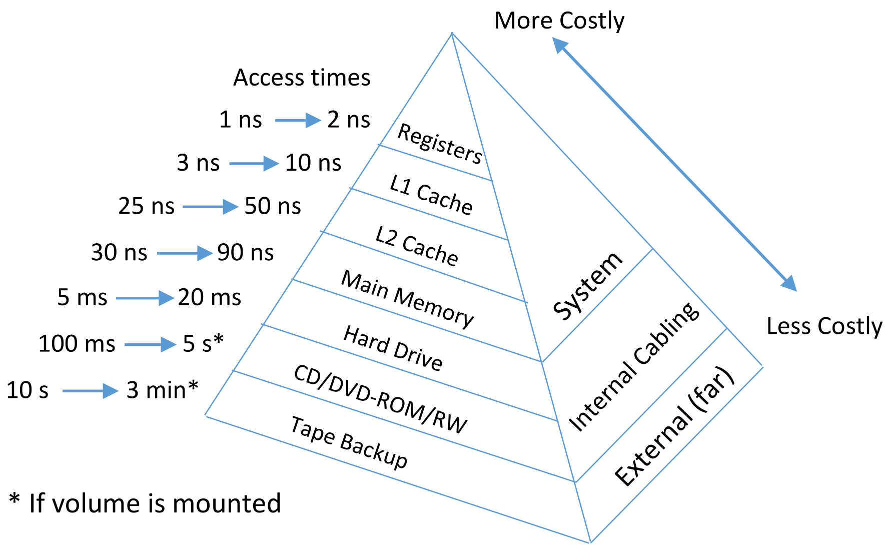


FIGURE 2.1: Memory Hierarchy

The layers of the memory hierarchy consists of registers, caches, main memory and secondary memory. The layers at the top of the pyramid are smaller in size but faster in access because they are closer to the central processing unit (CPU). The amount of data transfered between the bottom layers is larger as compared to the data transfered at the top layers.

The topmost memory module that is closest to the CPU is a set of registers. Registers are the fastest to access and smallest in size. Registers are placed inside the chip that houses the CPU. High end RISC processors have 32 floating point registers which can be accessed with no delay (zero cycles latency[4]). The RISC processors also possess 32 integer registers

[4]Latency is the time between the start and the completion of an event. Latency in the case of a register is the time taken to retrieve a unit of data

which are meant for integer arithmetic including address calculations of array elements that are loaded or stored.

Below the registers in the hierarchy are caches which are slower in access and larger in size as compared to the registers but faster in access and smaller in size as compared to memory. Typical machines today have multiple cache levels such as level-1 (L1) and level-2 (L2) caches. If data is available in L1 cache, then it is transfered from the cache to the register. The latency for the transfer is 1 cycle. But if data is not present in L1 cache, then L1 "cache miss" occurs and data must be transfered from lower level memory to L1 cache. If the system has L2 cache, then a transfer from L2 cache to L1 cache takes a few dozen cycles. If data is not available in L2 cache then L2 "cache miss" occurs and the data is transfered from memory to L2 cache which has a latency of approximately 100 cycles. If the machine does not contain L2 cache, then data is transfered directly from memory to L1 cache.

A cache data can be divided into blocks and lines. A cache block is a unit of data that can be transfered between cache and registers. A cache line which contains many cache blocks, is the smallest unit that can be loaded from memory to cache. Cache line size typically varies between 4 and 32 words. A cache is said to be p way associative if a cache line can be placed in any one of p possible slots.

Next in the hierarchy is main memory. Main memory has a high latency and hence higher access time as compared to cache.

After the main memory is the secondary memory. Secondary memory is very slow in access but very large in size as compared to the main memory. This memory is a real performance bottleneck because it is the greatest contributor to the degradation of performance. The reason for low performance is due to very large access latency as compared to the main memory. The typical access time of main memory is of the order of nanoseconds but hard drive which is the fastest secondary memory in the hierarchy has an access time in the order of milliseconds as given in the figure 2.1 which is 10^6 times slower as compared to main memory.

2.2.2 Modern Processor Architectures

Modern computers are based on superscalar architectures. A superscalar architecture is one that can perform multiple operations in one processor clock cycle. Consider an n-way superscalar processor. This processor can fetch n instructions simultaneously in one clock cycle. A typical processor is capable of performing a combination of adds, multiplies, loads/stores and branching in one clock cycle. This instruction-level parallelism is exploited in the hardware or by using a combination of hardware and software support.

Another technique that a superscalar processor employs is pipelining. Each functional unit in a pipelined processor executes independent instructions in such a way that the pipeline can output every cycle. This can be achieved with those instructions whose inputs do not depend on the outputs of the previous instructions in the program sequence. From a programmers perspective, a pipeline having a latency of s cycles has the same effect as s separate floating point units having the latency of one clock cycle.

2.2.3 Optimizing Compiler

A modern compiler can perform many optimizations which involves exploitation of memory hierarchy and superscalar architecture. A few of the optimizations that a compiler is capable of performing are enlisted in appendix A. A compiler can be set to optimize at the highest level of optimization. This may lead to an improvement in code performance but may also result in degradation of performance. This deterioration is because a compiler applies transformation rules which are based on heiristics which may improve performance in most cases but it is not guaranteed [6].

3 Methods

The following research questions were addressed in this research & development

- How can a genetic design tool be implemented efficiently?
- What software packages and tools improve the accessibility of the genetic design tool?
- How can the usability of the genetic design tool be improved for the purpose of metabolic engineering research?

This research employed a set of accessible software packages and tools to implement an efficient computational method GDBB [4] in Java programming language and applied effective code optimization principles and applicable user interface design principles to the software. The effect of applying the code optimization principles were measured using code profiling techniques and the principles that showed a significant improvement in the efficiency (i.e. which were effective), were retained. Furthermore, proven and/or applicable user interface design principles were employed and subsequently usability testing was conducted using an interview & observation protocol [9] with sample users. The narrative data collected through the observation was used to fix user interface bugs.

3.1 Design and Implementation of Computational Method

To implement the computational method, the mathematical definition of the GDBB [4] problem (2.19) defined previously, was represented in terms of matrices and matrix operations. This enabled compact representation of multiple linear equations and inequalities as fewer matrix operations. The redefinition of the problem is as follows,

$$\max \boldsymbol{\gamma}^T \mathbf{x} \tag{3.1}$$

Subject to

$$\mathbf{E}\mathbf{x} = \boldsymbol{\delta}$$

$$\mathbf{M}_a\mathbf{x} \geq \mathbf{A}$$

$$\mathbf{M}_b\mathbf{x} \leq \mathbf{B}$$

$$\boldsymbol{\alpha} \leq \mathbf{x} \leq \boldsymbol{\beta}$$

where,

$$\mathbf{M}_a = \begin{bmatrix} \mathbf{I}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times m} & \mathbf{a}_d\mathbf{G}^T \\ \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{I}_{n\times n} & \mathbf{0}_{n\times m} & D\mathbf{G}^T \end{bmatrix} \tag{3.2}$$

$$\mathbf{M}_b = \begin{bmatrix} \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times m} & \mathbf{1}_{1\times L} \\ \mathbf{I}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times m} & \mathbf{b}_dG^T \\ \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & \mathbf{I}_{n\times n} & \mathbf{0}_{n\times m} & -D\mathbf{G}^T \end{bmatrix} \tag{3.3}$$

$$\mathbf{E} = \begin{bmatrix} \mathbf{S} & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times m} & \mathbf{0}_{m\times L} \\ \mathbf{0}_{n\times n} & -\mathbf{I}_{n\times n} & \mathbf{I}_{n\times n} & \mathbf{I}_{n\times n} & \mathbf{S}^T & \mathbf{0}_{n\times L} \\ \mathbf{f}^T & \mathbf{a}^T & -\mathbf{b}^T & \mathbf{0}_{1\times n} & \mathbf{0}_{1\times m} & \mathbf{0}_{1\times L} \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{\gamma}^T = \begin{bmatrix} \mathbf{g}^T & \mathbf{0}_{1\times n} & \mathbf{0}_{1\times n} & \mathbf{0}_{1\times n} & \mathbf{0}_{1\times m} & \mathbf{0}_{1\times L} \end{bmatrix} \tag{3.5}$$

$$\mathbf{x}^T = \begin{bmatrix} \mathbf{v}^T & \boldsymbol{\mu}^T & \boldsymbol{\nu}^T & \boldsymbol{\xi}^T & \boldsymbol{\lambda} & \mathbf{y}^T \end{bmatrix} \tag{3.6}$$

$$\mathbf{A}^T = \begin{bmatrix} \mathbf{a}^T & \mathbf{0}_{1\times n} \end{bmatrix} \tag{3.7}$$

$$\mathbf{B}^T = \begin{bmatrix} C & \mathbf{b}^T & \mathbf{0}_{1\times n} \end{bmatrix} \tag{3.8}$$

$$\boldsymbol{\alpha}^T = \begin{bmatrix} -\boldsymbol{\infty}_{1\times n} & \mathbf{0}_{1\times n} & \mathbf{0}_{1\times n} & -\boldsymbol{\infty}_{1\times n} & -\boldsymbol{\infty}_{1\times n} & \mathbf{0}_{1\times L} \end{bmatrix} \tag{3.9}$$

$$\boldsymbol{\beta}^T = \begin{bmatrix} \boldsymbol{\infty}_{1\times n} & \boldsymbol{\infty}_{1\times n} & \boldsymbol{\infty}_{1\times n} & \boldsymbol{\infty}_{1\times n} & \boldsymbol{\infty}_{1\times n} & \mathbf{1}_{1\times L} \end{bmatrix} \tag{3.10}$$

$$\boldsymbol{\delta}^T = \begin{bmatrix} \mathbf{0}_{1\times m} & \mathbf{f} & \mathbf{0}_{1\times 1} \end{bmatrix} \tag{3.11}$$

$\mathbf{I}_{n\times n}$ is an $n \times n$ identity matrix of the form

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}_{n\times n} \tag{3.12}$$

$\mathbf{0}_{p\times q}$, for any $p, q \in \mathcal{Z}^+$ is a $p \times q$ zero matrix of the form

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}_{p\times q} \tag{3.13}$$

$\mathbf{1}_{p\times q}$, for any $p, q \in \mathcal{Z}^+$ is a $p \times q$ matrix whose elements are all one as given below

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \dots & & & & \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}_{p\times q} \tag{3.14}$$

$\boldsymbol{\infty}_{1\times p}$, for any $p \in \mathcal{Z}^+$ is a $1 \times p$ vector whose elements are all infinity

$$\begin{bmatrix} \infty & \infty & \infty & \dots & \infty \end{bmatrix}_{1\times p} \tag{3.15}$$

$\mathbf{a}_d$ is an $n \times n$ matrix whose elements are all zero except the diagonal elements which correspond to the elements of a as shown below

$$\begin{bmatrix} a_1 & 0 & 0 & \dots & 0 \\ 0 & a_2 & 0 & \dots & 0 \\ 0 & 0 & a_3 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & a_n \end{bmatrix}_{n \times n} \tag{3.16}$$

$\mathbf{b}_d$ is an $n \times n$ matrix similar to $\mathbf{a}_d$ but whose elements correspond to the elements of $\mathbf{b}$

The matrix representation of the GDBB problem is implemented as a set of sparse array lists in Java programming language. Matrices are constructed using ordinary for loops and then passed to a function that parses the matrices and constructs parameters for Gurobi Solver Java Application Programming Interface (API). Then an independent Java thread is employed to run the GDBB problem as shown in the figure 3.1.

While the main GDBB thread is solving the problem, another Java thread executing a Gurobi callback function, reports any intermediate solutions of the Gurobi solver. The intermediate solutions are queued to a solution queue which is shared between multiple threads. This solution queue is dequeued by another parallel thread which prepares a new database instance to save the intermediate solution and then publishes the result to another independent Graphical User Interface (GUI) thread. The new GUI thread displays the result on the user interface. This mechanism ensures that all the intermediate genetic manipulation strategies discovered by the Gurobi solver appear immediately and enables the user to use the intermediate near optimal solutions or terminate the software at a satisfactory result.

Furthermore, the Gurobi solver's Java API was used to set the following parameters for the MILP solver as previously defined for GDBB [4]. $FeasibilityTol$ was set to 10^{-9} and $IntFeasTol$ was set to 10^{-9}; $Heuristics$ was set to 1.0; $MIPFocus$ was set to 1; $ImproveStartGap$ was set to ∞ and $TimeLimit$ is set by the user as an input at runtime.

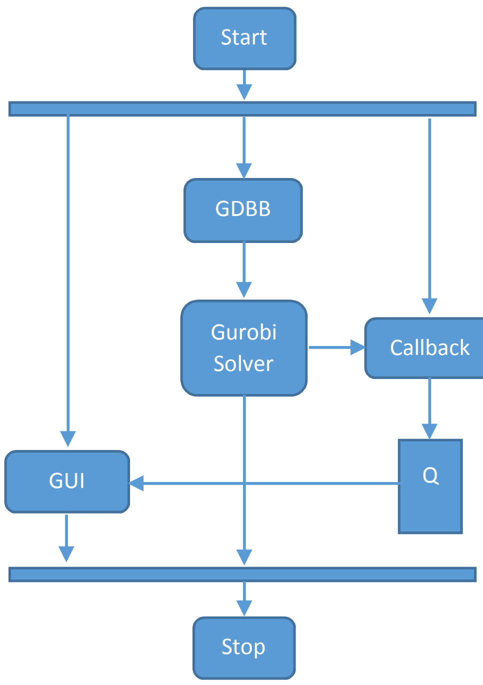


FIGURE 3.1: Genetic design tool Design

In order to verify the correctness, test the performance gain and usability of the genetic design tool, the researcher used the latest genome-scale model of *E. coli* iAF1260 [42]. This model consists of 1260 genes, 2077 reactions and 1039 unique metabolites.

3.1.1 Measurement Instruments

The truncated branch and bound implementation of Gurobi solver (Gurobi Optimization) was used to solve the GDBB problem. Although Gurobi needs commercial license, Gurobi solver is freely available for academic purposes. Gurobi provides Java API for LP, QP and MIP problems to support Java applications.

Gurobi solver also provides API for implementing callback mechanism. A callback function is invoked asynchronously by the solver to report events such as intermediate solutions or solver interruptions. The intermediate solutions for the truncated branch and bound algorithm are the near optimal solutions discovered thus far. Therefore the corresponding objective function value and variable values are reported to the user.

3.1.2 Data Collection

The data were collected by running the genetic design tool with truncation time set to ∞ and by running FBA respectively for various knockout configurations. The final solutions obtained in both the cases were plotted using a bar graph. The genetic design tool was run for $k = 1, 2, 3, \ldots, 6$ until at most five solutions were generated and the solutions in each case was recorded. The synthetic objective function was defined for maximization of acetate production. For each of the solutions obtained from the genetic design tool, the knockouts were simulated for FBA by setting the minimum flux value v_i^{min} and maximum flux value v_i^{max} to zero respectively for reaction i which is a reaction that is suppressed due to the knockout(s). Subsequently, FBA was run for each of the knockout strategies suggested by the genetic design tool and the corresponding optimal values of the objective function (2.6) were recorded. The problems were solved on a DELL XPS 15z machine with Intel core i5 processor with 6 GB RAM running Gurobi v5.5 and Java 7.

3.1.3 Data Analysis

The results of the Java implementation of genetic design tool were compared with the results of FBA. For each of the chemical compounds, the respective graphs were plotted. Each graph is a comparative plot between the results of the genetic design tool and the results of FBA for each of the knockouts respectively.

3.2 Code Optimization for Performance

Since an optimizing compiler does not guarantee a performance optimized code, applying hand optimization can be effective in improving the efficiency of the implementation [10].

In order to optimize matrix construction of the computational method, the following procedure was applied. A code profiler called JVM Monitor was run to identify the subroutine that is consuming maximum CPU time. Then a code snippet for which an optimization technique is applicable, was selected from the implementation. Optimization was then applied to the code snippet thus selected and the relative performance between the original code and optimized code was compared using the testing routine in appendix C. If the gain was greater than 0.0 %, the original code snippet in the application was replaced with its optimized version. If the gain was less than 0.0 %, then the original code was retained in the application.

The optimization techniques were applied directly to the code snippets in the application without extracting them to a different testing routine. To compare the performance, the testing routine was incorporated into the main application and two instances of the code snippet were prepared with the original version preceding the optimized version which is the order defined in the testing routine. An example of the application of an optimization technique is shown below.

```
long iterations = 100000000;

long startTime = System.nanoTime();
for (int p = 0; p < iterations; p++) {
}
long endTime = System.nanoTime();
long empty_loop_time = (endTime - startTime);

startTime = System.nanoTime();
for (int p = 0; p < iterations; p++) {
    // Unoptimized Code snippet
    for (int i = reactions_4_sMatrix; i < reactions_4_sMatrix_gprMatrix; i++)
    {
        varName = Integer.toString(i);
        GDBB.getSolver().setVar(varName, VarType.BINARY, 0, 1);
        this.varNames.add(varName);
    }
}
endTime = System.nanoTime();
long execution_time_normal = (endTime - startTime - empty_loop_time)/iterations;
System.out.println("execution time before optimization: " + execution_time_normal + " ns");
```

```
startTime = System.nanoTime();
for (int p = 0; p < iterations; p++) {
        // Optimized Code Snippet
        N = reactions_4_sMatrix_gprMatrix − this.model.getGprMatrix().size() \% 8;
        for (int i = reactions_4_sMatrix; i < N; i += 8) {
                GDBB.getSolver().setVar(Integer.toString(i), VarType.BINARY, 0, 1);
                GDBB.getSolver().setVar(Integer.toString(i + 1), VarType.BINARY, 0, 1);
                GDBB.getSolver().setVar(Integer.toString(i + 2), VarType.BINARY, 0, 1);
                GDBB.getSolver().setVar(Integer.toString(i + 3), VarType.BINARY, 0, 1);
                GDBB.getSolver().setVar(Integer.toString(i + 4), VarType.BINARY, 0, 1);
                GDBB.getSolver().setVar(Integer.toString(i + 5), VarType.BINARY, 0, 1);
                GDBB.getSolver().setVar(Integer.toString(i + 6), VarType.BINARY, 0, 1);
                GDBB.getSolver().setVar(Integer.toString(i + 7), VarType.BINARY, 0, 1);
        }

        for (int i = N; i < reactions_4_sMatrix_gprMatrix; i++) {
                GDBB.getSolver().setVar(Integer.toString(i), VarType.BINARY, 0, 1);
        }
}
endTime = System.nanoTime();
long execution_time_optimal = (endTime − startTime − empty_loop_time)/iterations;
System.out.println("execution time after optimization: " + execution_time_optimal + " ns");

System.out.println("gain: " + ((float)execution_time_normal / execution_time_optimal − 1.0)*100
                                        + " \%");
```

3.3 User Interface

User interface design was carried out in two phases. The first phase involved employing the applicable user interface design principles [8] as given in appendix A to the design and implementation of the genetic design tool. The second phase consisted of testing the user interface through user observation intervention [9] to determine any difficulties in the usability of the software tool. These difficulties were recorded and applicable changes were made to the user interface design to alleviate issues with usability.

3.3.1 Design and Implementation

The principles applied to design and implement the user interface are described below in detail. Each design change was made either to satisfy one of the principles or multiple principles simultaneously.

Principle of State Visualization: This principle states that the changes in behavior should be reflected in the appearance of the program. In order to implement this, the important changes from the perspective of the user were identified for the program. In the case of GDBB, the important changes were the generation of intermediate solutions. The solutions thus obtained consisted of, objective function value, gene knockouts and flux vector of the optimal solution. These solutions are immediately displayed on the GUI.

Principle of Focus: This principle states that some aspects of user interface attracts attention more than others do. The principle is applicable in the cases where the user must be informed about global state changes. In the case of GDBB, the global state change is the background processing that takes place while the problem is being solved. This background processing is conveyed to the user using a counter that indicates the total time elapsed since the GDBB computation was initiated.

Principle of Safety: The principle of safety states that the user must be allowed to develop confidence by providing a safety net. This principle was implemented by providing minimal options while the program is processing. This was achieved through a dialog that pops up when GDBB is selected from the main menu. The dialog does not allow the user to have access to the main dialog of the application so that the user may not simultaneously start other operations and lose track of the operation intended to be performed. In the dialog, only three options are provided namely *start*, *stop* and *close*. The *close* operation is disabled when the application starts until the user clicks *stop*. This avoids ambiguity regarding the purpose of *close* operation (the purpose is only to close the dialog and not to stop the process which is the function of *stop* button).

Principle of Context: The principle of context states that the user activity must be limited to one well-defined context. This principle is also taken care of when all the operations are limited to GDBB in the pop-up dialog and all and only the outputs generated are limited to the context of the GDBB problem.

Principle of Aesthetics: This principle states that, one must create a program of beauty. This requirement is taken care of by representing the solutions as a tree structure with all the intermediate solutions appearing as leaves to the model under consideration and inserting icons where ever it is appropriate. The counter output previously discussed also takes care of the requirement that the user's usually do not like programs that feel sluggish or slow and hence an animation was displayed to hide sluggishness in the program.

3.3.2 Testing

For user interface testing, the observation took place at the researcher's school located in Camden, New Jersey with two PhD students from the Center of Computational and Integrative biology (CCIB) department and remotely with one student at a residence located in Bangalore, Southern India. The User Observation Through Thinking out Loud [9] intervention was conducted on an individual basis. The instructions described in the principle were provided directly to the users at the researcher's school and remotely to the user located abroad using online video chat session.

3.3.2.1 Participants

The sampling procedure used by the researcher was purposive sampling. One set of participants were from researcher's school site who were PhD students from the CCIB department. The participants from the CCIB department were selected because they were representative of the group of industrial and medical biotechnology researchers. The students were studying and applying computational modeling of biological systems as well as metabolic engineering techniques in-vitro on microorganisms in laboratories at the school. Another participant remotely located at Bangalore, India was an undergraduate student in dentistry. The selection was made because the student represents a group of users from pure medical/ biological background and no academic background in systems biology and at the same time represents a group of users from different ethnicity and culture and whose requirements can be addressed remotely.

3.3.2.2 Intervention

User interface testing was carried out by recruiting help in spotting unavoidable defects in the program. To detect a user interface bug the researcher observed the sample of users use the application. A methodology called User Observation Through Thinking Out Loud [11, 54] was applied for conducting the observation and identifying difficulties in usage of the application.

The researcher introduced himself to a user and described the purpose of the observation. The researcher explained that he is looking for spots where it is difficult to use the product and that this would aid the application developer in improving the product. The researcher also made it clear to the participant that he/she could quit at any time if it is becoming uncomfortable.

Next, the researcher explained the purpose of using laptop and desktop computing machines for testing. The purpose is to enable real users to run the software comfortably on personal computers which usually have a similar configuration such as hardware and operating system.

After that, the researcher explained how to think loud. The participant was asked to say whatever comes to the mind as they used the software tool. The purpose behind thinking out loud was to enable the researcher to examine the expectations of the product. If the participant forgot to think aloud, the researcher reminded the user to continue talking.

Then, the researcher clarified that help will not be provided. When the participant had difficulty, the answer was not immediately provided but after three minutes. The researcher made sure that no more information is given than a true user of the software tool would be given. Information such as features offered by the product were explained before-hand. If the participant had questions in between, he/she was told to ask them in any case and that the answers would be provided at the end of the observation.

Subsequently, the tasks in the product were introduced. The participant was given a list of tasks to be performed in the form of instructions on a pdf document. These instructions are provided in Appendix D. The general functionality of the product excluding the functionality of the genetic design tool was demonstrated to the participant.

Next, any questions asked by the participant were clarified and the actual observation was started. The researcher noted down the information and concluded the observation once tasks were carried out as given in the list of instructions.

After that, the questions posed by the participant during the observation were answered and subsequently, the researcher discussed interesting behavior that the participant liked to see in the product.

Finally, the results of the observation were reviewed and areas where the participant faced trouble were identified. The tool was then modified to alleviate trouble faced by the participant.

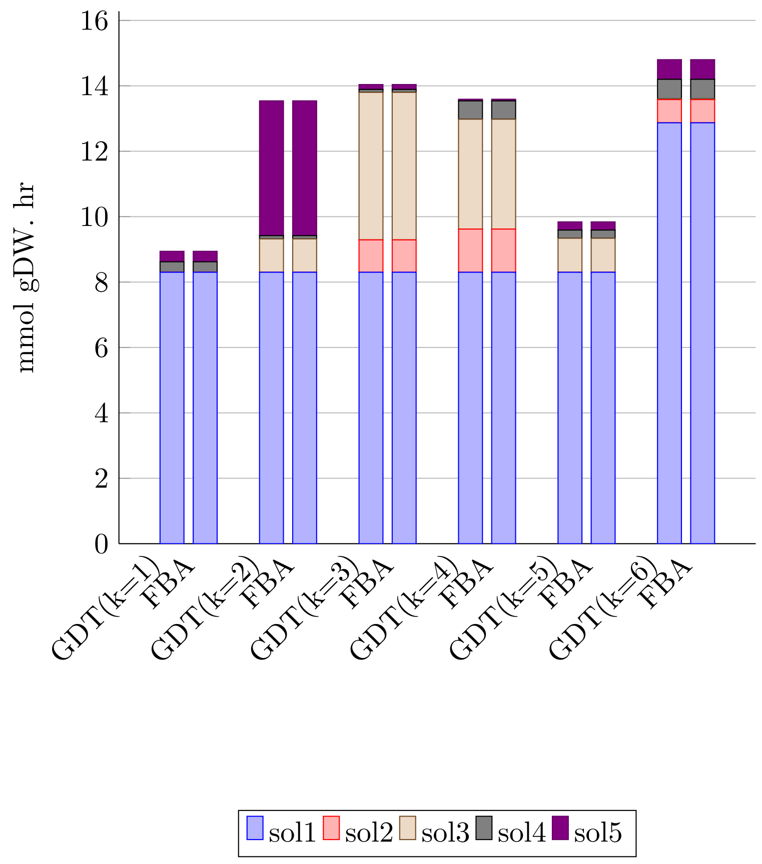
4 Testing and Results

Genetic design tool's capabilities related to three areas will be discussed in detail in the following sections. The first section deals with testing the strategies suggested by the genetic design tool against the results of FBA and presents results of running the genetic design tool. Figure 4.1 shows a comparative plot of the results obtained for the genetic design tool and FBA for different knockout configurations. Figure 4.2 displays results of the genetic design tool for one of the knockout values, $k = 2$. FBA analysis was performed by running a genetically modified model of *E. coli* as suggested by the genetic design tool. The second section deals with the performance gains obtained after applying optimization principles. Table C.1 shows the running times and gains obtained for each of the sample codes. Table 4.2, displays the results of optimizing the actual code of the genetic design tool. Finally, the last section shows the user interface of the genetic design tool obtained as a result of applying the user interface design principles [8]. A discussion of the narrative data obtained from user testing is also incorporated in this section.

4.1 Computational Method

To test the genetic design tool, a comparative analysis was performed with FBA. For each of the configurations of the genetic design tool, the optimizer was configured to run by setting the *TimeLimit* parameter of the MILP solver to infinity (no truncation time) to ensure that the truncation time does not stop the solver before finding at least five solutions. The knockouts suggested by the genetic design tool for each of the configurations were then simulated by setting the lower bound v_i^{min} and the upper bound v_i^{max} to zero for the corresponding metabolic reaction which gets suppressed as a result of the knockout(s). The resulting constraints were then used as inputs for FBA analysis. The results obtained in the case of genetic design tool and FBA analysis are plotted in figure 4.1.

Figure 4.1: Genetic design tool vs FBA



Genetic design tool and independent FBA analysis showed comparable results for knockout values $k = 1, 2, 3, \ldots, 6$.

Another plot in figure 4.2 shows six consecutive intermediate solutions produced by the genetic design tool for maximum knockout limit of $k = 2$. For this knockout limit, the strategies proposed by the genetic design tool are given in table 4.1.

Table 4.1: Strategies obtained for at most two knockout configuration

No	**Knockouts**	**Biological Objective**	**Synthetic Objective**
1	-	0.231056	8.301396
2	(b3846 or b2341)	0.231056	8.301396

Continued on next page

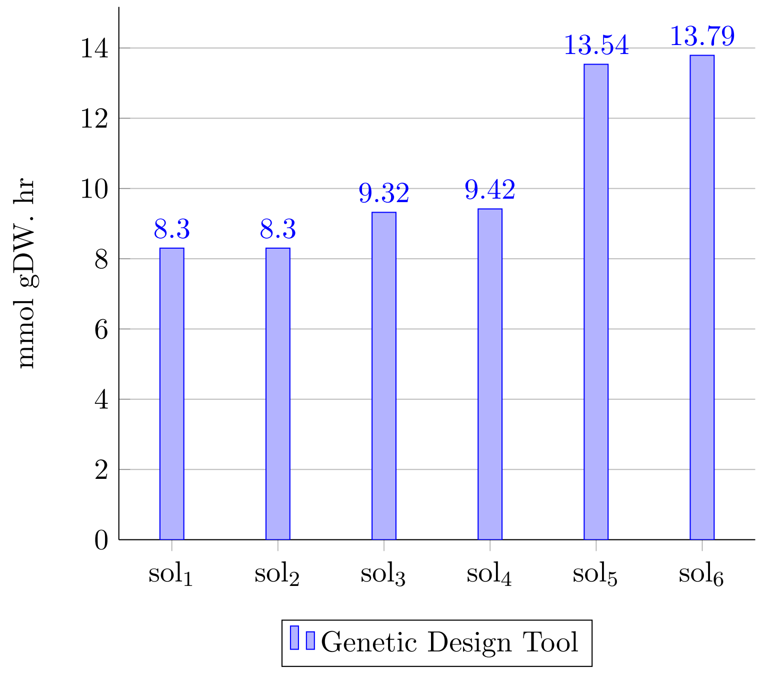
Table 4.1 – *Continued from previous page*

No	Knockouts	Biological Objective	Synthetic Objective
3	b1849 b2913	0.212228	9.320898
4	((b4079 and (b2481 and b2482 and b2483 and b2484 and b2485 and b2486 and b2487 and b2488 and b2489 and b2490)) or (b4079 and (b2719 and b2720 and b2721 and b2722 and b2723 and b2724))) b2913	0.181828	9.418173
5	(b0351 or b1241) (b2551 or b0870)	0.128993	13.535922
6	(b0351 or b1241) b1539	0.130347	13.791117

4.2 Code Optimization

Code optimization was performed in two stages. The first stage involved verifying whether a particular compiler optimization principle improves the performance of the sample code. The results of the first stage are displayed in appendix C. The second stage employed the principles which were effective in the first stage to the actual code snippets in the genetic design tool program. The results of applying the code optimization techniques to the genetic design tool code are reported in table 4.2.

FIGURE 4.2: Objective Values for k = 2



The Java compiler is an optimizing compiler, certain optimizations are performed during runtime and as a result of this, the optimization techniques such as loop invariant optimization, constant propagation, dead store elimination, dead variable elimination and boolean short circuiting are automatically handled by the compiler. Hence the principles had no effect on the performance of the sample code snippets. However, the Java compiler was unable to perform optimizations such as constant folding, loop unrolling and loop jamming respectively. The final code involved a combination of optimizations in order to improve performance but the compiler was unsuccessful in improving the performance as compared to the hand optimized code. The hand optimized code was able to run twice as fast as the unoptimized version.

Furthermore, the genetic design tool was run by setting the *iterations* parameter in the timing template to the value of one. As a result each of the unoptimized and optimized versions of the function ran only once. The process was repeated 10 times and the gains were recorded for each run respectively. The maximum gain observed was 16.44 %. The average of all 10 gains was 8.03 % ($SD = 5.20$). The average time taken by the unoptimized function was 0.9118 s and the average time taken by the optimized function was recorded to be 0.8445 s.

Finally, a function *setConstraints()* that builds the matrices for the problem was optimized. Subsequently, the function was tested for performance gain using the timing template in appendix C. The *iterations* parameter of the timing template was set to 10, 100, 200 and 500 respectively and the gains for each case were recorded as shown in the table 4.2. The gain for 10 iterations was significant but as the number of iterations increased, a drop in the gain was observed. The probable reason for this behavior is that repeated call to the function requires reconstruction of the same matrices which results in the accumulation of variables in memory.

TABLE 4.2: Results of optimizing $setConstrains()$ function

Iterations	**Execution time of unoptimized code, t_u (s)**	**Execution time of optimized code, t_o (s)**	**Gain (%)**
10	0.8311	0.6959	19.42
100	0.6028	0.5462	10.36
200	0.6236	0.5924	5.27
500	0.6128	0.5920	3.51

4.3 User Interface

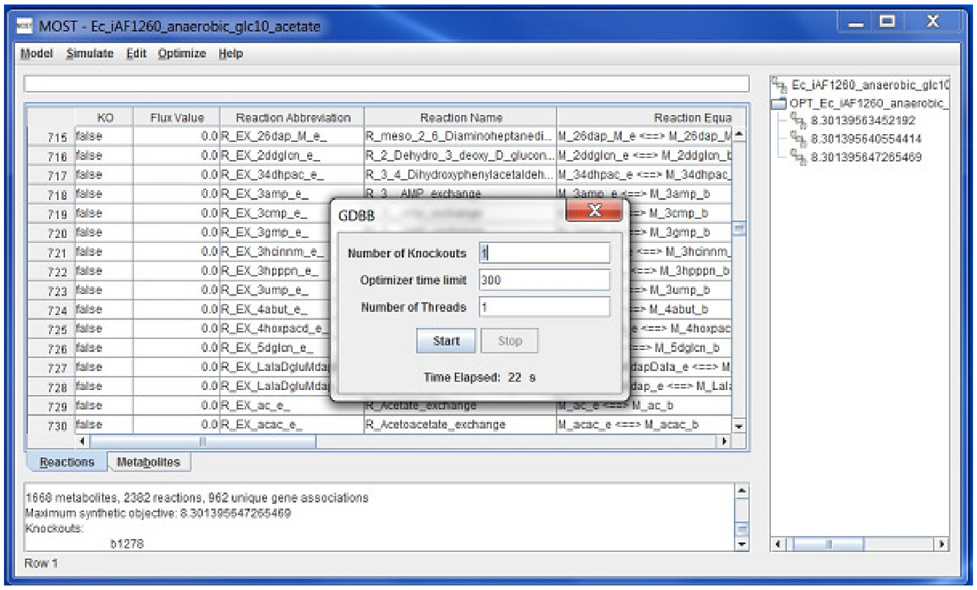
The user interface has been implemented following the user interface design principles [8] to the best extent possible. The pop-up dialog allows user to enter values for different parameters of the MIP Solver (Gurobi in this case). The right hand side panel displays a tree which contains intermediate solutions labeled by the synthetic objective value of the respective optimal solution. The console at the bottom scrolls down automatically and displays new solutions as and when they are obtained. The counter below the *start* and *stop* buttons keeps track of the time elapsed (in seconds) since the solver was started.

4.3.1 User Interface Testing Results

User observation intervention has enabled identification of a few trouble spots with the user interface of the application. The two PhD students from CCIB department of Rutgers Camden suggested that for a new user, opening the GDBB dialog and running the optimization directly without the need for setting the synthetic objective might be more intuitive. The second PhD student from the CCIB department had difficulty setting the synthetic objective because he expected to set it on the GDBB pop-up dialog. Finally a suggestion was made by both the participants to provide an option in the GDBB pop-up dialog for setting synthetic objective function.

The third user who is an undergraduate student of dentistry had trouble finding the exchange reaction that produced the compound of interest in the list of reactions. Additionally the user also expected the convenience of a search function to locate the reaction but the user was unable to locate the existing search function provided by the tool. The user also indicated that the names of the parameters in the GDBB dialog do not clearly indicate their functionality. Furthermore, the user was misled by a tree node that got created on the right side panel as soon as GDBB started to believe that a solution has been obtained. This led to the user terminating the optimization and faced further complications involving searching for the objective value on the main dialog. Next, the user accomplished the task of adding a new synthetic objective column but with the same name as the existing synthetic objective which resulted in no column being added to the table.

Furthermore, all the three users expected default values to be set for the parameters on the GDBB dialog which reduces the need to set them manually.



	KO	Flux Value	Reaction Abbreviation	Reaction Name	Reaction Equa
715	false	0.0	R_EX_26dap_M_e_	R_meso_2_6_Diaminoheptanedi...	M_26dap_M_e <==> M_26dap_M
716	false	0.0	R_EX_2ddglcn_e_	R_2_Dehydro_3_deoxy_D_glucon...	M_2ddglcn_e <==> M_2ddglcn_b
717	false	0.0	R_EX_34dhpac_e_	R_3_4_Dihydroxyphenylacetaldeh...	M_34dhpac_e <==> M_34dhpac_
718	false	0.0	R_EX_3amp_e_	R_3__AMP_exchange	M_3amp_e <==> M_3amp_b
719	false	0.0	R_EX_3cmp_e_		==> M_3cmp_b
720	false	0.0	R_EX_3gmp_e_		==> M_3gmp_b
721	false	0.0	R_EX_3hcinnm_e_		e <==> M_3hcinnm_
722	false	0.0	R_EX_3hpppn_e_		<==> M_3hpppn_b
723	false	0.0	R_EX_3ump_e_		==> M_3ump_b
724	false	0.0	R_EX_4abut_e_		==> M_4abut_b
725	false	0.0	R_EX_4hoxpacd_e_		e <==> M_4hoxpac
726	false	0.0	R_EX_5dglcn_e_		==> M_5dglcn_b
727	false	0.0	R_EX_LalaDgluMda		dapDala_e <==> M
728	false	0.0	R_EX_LalaDgluMda		dap_e <==> M_Lal
729	false	0.0	R_EX_ac_e_	R_Acetate_exchange	M_ac_e <==> M_ac_b
730	false	0.0	R_EX_acac_e_	R_Acetoacetate_exchange	M_acac_e <==> M_acac_b

GDBB

Number of Knockouts: 1

Optimizer time limit: 300

Number of Threads: 1

Time Elapsed: 22 s

Ec_iAF1260_anaerobic_glc10
OPT_Ec_iAF1260_anaerobic_
8.30139563452192
8.301395640554414
8.301395647265469

1668 metabolites, 2382 reactions, 962 unique gene associations
Maximum synthetic objective: 8.301395647265469
Knockouts:
b1278

Row 1

FIGURE 4.3: User Interface

5 Discussion

5.1 Limitations

Although a genetic design tool with accessible packages, improved performance and user feedback driven user interface has been developed, several limitations exist in this research. The first limitation is that the weaknesses that exist with the computational method GDBB also exist with the genetic design tool. GDBB uses bi-level optimization framework [3] which is based on the hypothesis that the organism optimizes biological growth along with the chemical compound as a bi-product. Another limitation is that the method though computationally efficient, produces only near optimal solutions which depends on the truncation time that limits the time for finding solutions. A third limitation is that the GDBB [4] method considered only knockouts and hence the implementation was limited to knockouts.

Other limitations are related to the performance of the genetic design tool. The genetic design tool was implemented in Java, a high level programming language compiled to a bytecode unlike other high level languages such as C which is compiled to machine code. As a result the tool needs virtual machine to run. This may limit the opportunity to apply hardware specific optimizations.

A second limitation is that certain performance optimizations are applicable only if the underlying hardware is based on superscalar architecture. Consider loop unrolling for instance, this optimization is effective only if multiple instructions can execute parallelly in a single clock cycle.

A third limitation is that certain optimizations assume memory hierarchy. Such a hierarchical design enables exploitation of spatial and temporal locality of access [7]. In other words, the program is likely to access the same data again (temporal locality) and adjacent data (spatial locality) in the near future respectively. The principle of loop fusion for instance, exploits the principle of locality. This limitation affects the external validity because the optimization principles that assume memory hierarchy may not be effective if the memory subsystem is not based on hierarchical organization.

A fourth limitation is related to the timing template (refer to Appendix A). The timing routine which calculates the gain after applying an optimization principle, is currently not considering the factors that affect the running time of the application. Factors such as paging, context switch between processes, threads and system time [55] affect the execution time of the routine and as a consequence the execution time of the code snippets being examined for performance improvement. This affects the internal validity because the timing routine measures wall clock time i.e., the total running time of the code snippet which may include time for paging, context switching in addition to the system time to service requests from the application and user application running time. Since these factors that affect the running time of the routine are unpredictable [6], the outputs of the timing template are slightly different each time it is run.

Other limitations are related to the selection of sample group and sample size for usability testing of the genetic design tool. The sample size was very small since there were three students in total, two students from CCIB and one undergraduate dental student. Although the user interface design was based on the sample group's narrative feedback, the usability of the application cannot be generalized to a larger group of real users.

5.2 Recommendation for Future Research

Extension of Computational Method: Based on the results of the research & development, there are several recommendations for future research. First, some of the limitations outlined in this research can be minimized or eliminated in a revised implementation of the computational method. The computational method can be extended to consider other cellular objectives and up and down regulation respectively. Since the optimization principles are independent of the computational method, the principles which turned out to be effective can be reapplied to the implementation. Similarly, the user interface design principles can also be re-applied to the extended implementation.

GPU based MILP Solver & GPU based Basic Linear Algebra Subprograms (BLAS) library: Since GDBB finds near optimal solutions in a finite amount of time, in order to improve the solution optimality a Graphics Processing Unit (GPU) implementation of MILP solver [56] can be used instead of a CPU based solver. A GPU is a processing unit that consists of hundreds or even thousands of cores. Each of the cores are capable of running a thread of execution independently of other cores and as a result a GPU can run a number of threads parallelly. This may lead to finding better optimal solutions more efficiently. Third in order to improve the efficiency of the construction of matrices, a GPU based linear algebra library can be used. An example of such a GPU library is CUBLAS (CUDA BLAS library) that runs on NVIDIA CUDA GPU. GPU based implementation may also lead to finding optimal solutions instead of near optimal solutions for higher values of k if the solver is run to its completion.

Code Performance Improvement: Other recommendations are related to improving the code efficiency of the genetic design tool. First limitation regarding the effectiveness of certain hardware dependent code optimization principles may improve if the problem is defined in a programming language that is directly compiled to machine dependent code such as C or C++ instead of bytecode as in the case of Java. Furthermore, a hotspot can be identified which is inefficient and a C or C++ function can be defined for it. The function can then be called from existing Java implementation of the genetic design tool using Java Native Interface (JNI), which is a programming framework that enables Java code to call the C or C++ function. Additionaly, to improve the efficiency of multiple threads of the genetic design tool, optimization principles such as data locality in shared memory and load balancing which is the distribution of work load among different parallel threads can be applied [6]. Finally, the accuracy of the timing routine (given in Appendix A) used for finding effective optimization principles may be improved by incorporating the execution time of the affecting factors into the calculation of efficiency gain.

Software Usability Enhancement: Final set of recommendations are to minimize the limitations related to the administration of User Observation Through Thinking Out Loud [9] Intervention. In order to increase the sample size of users, the intervention may be incorporated into the software which may pose questions to the end-users and record user experience. A random sampling of users can be taken and corresponding responses can be studied to improve the usability of the software. This improves the external validity of the tool since larger samples can be selected and the samples can represent real users. Another advantage is that since the genetic design tool is more accessible, users with diverse ethnicities and cultures can be considered for sampling which may further enhance the usability of the genetic design tool.

5.3 Conclusion

Four major conclusions can be drawn from this study. The first conclusion is that a computationally efficient genetic design algorithm GDBB has resulted in an interactive and efficient genetic design tool. Second conclusion is that the code optimization principles showed improvement in performance of the implementation compared to an unoptimized version. The third conclusion is that the User Observation Through Thinking Out Loud [9] intervention has been able to identify user interface bugs as well as receive user feedback for improving usability of the application. Fourth and final conclusion is that freely accessible tools could be used to develop an accessible software application.

The first conclusion is that the GDBB [4] has resulted in an interactive and efficient genetic design tool which can identify near optimal solutions found to be sufficient for practical purposes. The user can interactively define different input parameters such as maximum number of knockouts, finite or infinite truncation time for the solver, number of threads to take advantage of multiple cores of the CPU. Additionally, efficiency has been achieved since GDBB can identify intermediate solutions which are displayed on the user interface. This enables the user to utilize the intermediate knockout strategies without waiting for the final optimal solution.

The second conclusion is that the code optimization principles [6, 10] showed an improvement in the performance of the implementation as compared to the unoptimized version. Even though modern compilers can perform most of the optimizations, previous studies have shown that compiler may overlook opportunities for optimization [6, 10]. As a consequence of this, hand optimization has given improvement over unoptimized implementation. This may imply that the resulting genetic design tool can perform better and hence be more economical in terms of computer resource utilization.

The third conclusion is that the User Observation Through Thinking Out Loud [9] has been able to identify user interface bugs and enabled to receive user feedback. This has resulted in making necessary design and implementation changes which improved the usability of the application among the sample group of users. This may imply that, the intervention can be effective in identifying user interface problems and hence lead to improvement of usability of the genetic design tool for larger sample group of real users.

The fourth and final conclusion is that freely accessible tools enabled the implementation of an accessible genetic design tool. There are a few advantages in developing a freely accessible genetic design tool. First the application becomes available to anybody without involving any license fee and hence a larger group of users can access the application. Second since the tool is accessible, users can contribute towards improvement or in customization of the application. This may in turn result in the improvement of efficiency as well as usability of the genetic design tool in addition to maintaining accessibility.

5.4 Summary

The genetic design tool identifies genetic strategies quickly and at the same time it is interactive and accessible to a wide range of users, whether familiar or unfamiliar with the underlying computational methods, whether conducting research in laboratories or in large industries or using it for academic purposes, can aid in genetic engineering of organisms which act as alternative sources for improving the production of essential compounds used in drug and food supplements that can contribute in meeting the needs of today.

A Performance Optimization Techniques

Table A.1: Performance Optimization Techniques

Optimization Technique	**Example**	**Reason For Performance Improvement**
Constant folding	$x = 2.0 \times x \times 4.0$ to $x = 8 \times x$	Saves a floating point operation.
	$pi/2$ to $pi\_by\_2 = pi/2$	Saves a floating point load. Saves a floating point divide. Saving of tens of μ s
Loop invariant optimization	$x = 100;$ **while** $x > 0$ $x = x - (y + z);$ to $x = 100;$ $t = y + z;$ **while** $(x > 0)$ $x = x - t;$	Saves multiple integer addition operations

Loop induction elimination	**for** $(i = 1; i <= 10; i++)$
	
	    $a[i + 1] = 1;$
	to
	for $(i = 2; i <= 11; i++)$	

Continued on next page

Table A.1 – *Continued from previous page*

Optimization Technique	**Example**	**Reason For Performance Improvement**
	$a[i] = 1;$	Saves multiple integer operations
Removal of dead or unreachable code	to decrease memory utilization remove dead code or code that is not reachable	Saves unnecessary memory loads and stores
Constant Propagation	If there is an opportunity, change variable assignments to constant assignments It provides registerizartion opportunities Example: Consider the c code $x = 100;$ $y = x;$ Correspoinding assembly language is, LOAD R1, 100 STORE R1, &x LOAD R1, &x STORE R1 &y hence replace c code by, $x = 100;$ $y = 100;$ the assembly output is LOAD R1, 100	

Continued on next page

Table A.1 – *Continued from previous page*

Optimization Technique	**Example**	**Reason For Performance Improvement**
	STORE R1, &x STORE R1, &y	Saves memory loads and stores
Dead Store Elimination	If a variable value remains the same in a short span of code then do the following $t = y + z;$ $x = \text{FUNC}(t);$ to $x = \text{FUNC}(y + z);$	Saves a memory operation
Dead-Variable elimination	Definition: Live variable - A variable is live if its value gets subsequently used	Saves a memory operation
Short-Circuiting Boolean code	Test each sub-expression separately in compound boolean expression **if** $(x > 0 \&\& y > 0)$ $z = 1;$ to **if** $(x > 0)$ **if** $(y > 0)$ $z = 1;$	Saves integer operation
Loop Unrolling	Replace **for** $(i = 1; i <= 8; i++)$ $a[i] = a[i] * 8;$	

Continued on next page

Table A.1 – *Continued from previous page*

Optimization Technique	**Example**	**Reason For Performance Improvement**

	by
	$a[1] = a[1] * 8;$
	$a[2] = a[2] * 8;$
	$a[3] = a[3] * 8;$
	$a[4] = a[4] * 8;$
	$a[5] = a[5] * 8;$
	$a[6] = a[6] * 8;$
	or by
	for $(i = 1; i <= 6; i+ = 3)$
	{
	$a[i] = a[i] * 8;$
	$a[i+1] = a[i+1] * 8;$
	$a[i+2] = a[i+2] * 8;$
	}
		Exploits instruction level parallelism in a dynamically scheduled microprocessor

Loop Jamming	Combine similar loops as follows replace **for** $(i = 1; i <= 100; i++)$ $x[i] = y[i] * 8;$ **for** $(i = 1; i <= 100; i++)$ $z[i] = x[i] * y[i];$ by	

Continued on next page

Table A.1 – *Continued from previous page*

Optimization Technique	**Example**	**Reason For Performance Improvement**
	for $(i = 1; i <= 100; i++)$ $\{$ $x[i] = y[i] * 8;$ $z[i] = x[i] * y[i];$ $\}$	Saves additional loop overheads

B Testing Routine

```
long iterations = 100000000;
long startTime = System.nanoTime();
for (int p = 0; p < iterations; p++) {
}
long endTime = System.nanoTime();
long empty_loop_time = (endTime − startTime);

startTime = System.nanoTime();
for (int p = 0; p < iterations; p++) {
        // paste unoptimized code here
}
endTime = System.nanoTime();
long execution_time_normal = (endTime − startTime − empty_loop_time)/iterations;
System.out.println("execution time before optimization: " + execution_time_normal + " ns");

startTime = System.nanoTime();
for (int p = 0; p < iterations; p++) {
        // paste optimized code here
}
endTime = System.nanoTime();
long execution_time_optimal = (endTime − startTime − empty_loop_time)/iterations;
System.out.println("execution time after optimization: " + execution_time_optimal + " ns");

System.out.println("gain: " + ((float)execution_time_normal / execution_time_optimal − 1.0)*100
                        + " %");
```

C Sample Code Optimization Results

TABLE C.1: Results of optimizing sample code

Optimization Principle	**Example Code**	**Sample Code Execution Time, t_s (ns)**	**Optimized Code Execution Time, t_o (ns)**	**Gain = $[t_s/t_o - 1]100$ (%)**

Constant Folding	`// original code`
	`x = 2.0*x*4.0;`
	
	`// optimized code`
	`x = 8.0*x;`
		6	4	50.0

Loop Unrolling	`// original code`
	`for(i = 1; i <= 6; i++)`
	`        a[i] = a[i]*8;`
	
	`// optimized code`
	`a[1] = a[1]*8;`
	`a[2] = a[2]*8;`
	`a[3] = a[3]*8;`
	`a[4] = a[4]*8;`
	`a[5] = a[5]*8;`
	`a[6] = a[6]*8;`
		4	2	100.0

Continued on next page

Table C.1 – *Continued from previous page*

Optimization Principle	**Example Code**	**Sample Code Execution Time, t_s (ns)**	**Optimized Code Execution Time, t_o (ns)**	**Gain = $[t_s/t_o - 1]100$ (%)**

Loop Jamming	`// original code`
	`for(i = 1; i <= 100; i++)`
	`        x[i] = y[i]*8;`
	`for(i = 1; i <= 100; i++)`
	`        z[i] = x[i]*y[i];`
	
	`// optimized code`
	`for(i = 1; i <= 100; i++)`
	`{`
	`        x[i] = y[i]*8;`
	`        z[i] = x[i]*y[i];`
	`}`
		105	156	-32.75

		175.78 (SD = 2.49)	156.22 (SD = 1.48)	12.51 (SD = 0.63)

Continued on next page

Table C.1 – *Continued from previous page*

Optimization Principle	**Example Code**	**Sample Code Execution Time, t_s (ns)**	**Optimized Code Execution Time, t_o (ns)**	**Gain = $[t_s/t_o - 1]100$ (%)**

Combination	`// original code`
	`for(i = 1; i <= 3; i++)`
	`{`
	`        a[i] = 0;`
	`        a[i] = a[i] + 2*x;`
	`}`
	`for(k = 1; i <= 3; i++)`
	`        b[k] = b[k] + a[k] + 2*k*`
	`    k;`
		4	-	-

	`/* optimized code after pass one`
	`(Loop Jamming,`
	`Loop invariant removal,`
	`removal of extraneous code)`
	`*/`
	`long t = 2*x;`
	`for(j = 1; j <= 3; j++)`
	`{`
	`        a[j] = t;`
	`        b[j] = b[j] + a[j] + 2*j*j`
	`    ;`
	`}`
		4	7	-42.8571

Continued on next page

Table C.1 – *Continued from previous page*

Optimization Principle	**Example Code**	**Sample Code Execution Time, t_s (ns)**	**Optimized Code Execution Time, t_o (ns)**	**Gain $= [t_s/t_o - 1]100$ (%)**

	/∗ optimized code after pass two (Loop Unrolling)
	 ∗/
	 long t = 2∗x;
	 a[1] = t;
	 b[1] = b[1] + a[1] + 2∗1∗1;
	 a[2] = t;
	 b[2] = b[2] + a[2] + 2∗2∗2;
	 a[3] = t;
	 b[3] = b[3] + a[3] + 2∗3∗3;
		4	2	100.0

	/∗ optimized code after pass two (Constant Folding)
	 ∗/
	 long t = 2∗x;
	 a[1] = t;
	 b[1] = b[1] + a[1] + 2;
	 a[2] = t;
	 b[2] = b[2] + a[2] + 8;
	 a[3] = t;
	 b[3] = b[3] + a[3] + 18;
		4	2	100.0

D User Interface Testing Instructions

Step 1: Load an organism's model

Step 2: Set synthetic objective

Step 3: Open GDBB

Step 4: Set parameters for the problem

Step 5: Run the optimization

Step 6: Stop at a satisfactory solution

Step 7: Read and interpret the results

1. Obtain gene knockouts
2. Obtain maximum objective value
3. Obtain growth rate of the organism

Step 8: Save the optimal solution

Step 9: Delete a solution

Step 10: Delete all solutions

Step 11: Add a new synthetic objective column and re-run the optimization

Step 12: Set synthetic objective

Step 13: Optimize using the new synthetic objective

Bibliography

[1] Jamil B, Hasan F, Hameed A, and Ahmed S. Isolation of bacillus subtilis mh-4 from soil and its potential of polypeptidic antibiotic production. *Biotechnology Journal*, 4(1): 26–31, January 2007. URL `http://www.ncbi.nlm.nih.gov/pubmed?Db=pubmed&Cmd=ShowDetailView&TermToSearch=17337424`.

[2] Caroline B. Milne, Pan-Jun Kim, James A. Eddy, and Nathan D. Price. Accomplishments in genome-scale in silico modeling for industrial and medical biotechnology. *Biotechnology Journal*, 4(2):1653–1670, November 2012. URL `http://www.biotechnology-journal.com`.

[3] Anthony P. Burgard, Priti Pharkya, and Costas D. Maranas. Optknock: A bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Wiley InterScience*, 28(7):648–657, October 2003. URL `http://www.interscience.wiley.com`.

[4] Dennis Egen and Desmond S. Lun. Truncated branch and bound achieves efficient constraint-based genetic design. *Bioinformatics*, 28(12):4477–4479, December 2012. URL `http://bioinformatics.oxfordjournals.org`.

[5] Desmond S. Lun, Graham Rockwell, Nicholas J Guido, Michael Baym, Jonathan A Kelne, Bonnie Berger, James E Galagan, and George M Church. Large-scale identification of genetic design strategies using local search. *Molecular System Biology*, 5(18): 1–8, August 2009. URL `http://www.molecularsystemsbiology.com`.

[6] Stefan Goedecker and Adolfy Hoisie. *Performance Optimization of Numerically Intensive Codes*. Society for Industrial and Applied Mathematics, 2001.

[7] John L. Hennessy and David A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers, 4 edition, 2007.

[8] Talin. A summary of principles for user-interface design, 1999. URL `http://www.sylvantech.com/~talin/projects/ui_design.html`.

[9] Kathleen Gomell and Anne Nicol. User observation: Guidelines for apple developers. pages 1–26, April 1988.

[10] Phillip A. Laplante. *Real-Time Systems Design and Analysis*. John Wiley & Sons Inc Publication, 3 edition, 2008.

[11] Bruce Tognazzini. *TOG on Interface*. Addisson Wesley Publishing Company Inc, 1991.

[12] Yohei Shinfuku, Natee Sorpitiporn, Masahiro Sono, Chikara Furusawa, Takashi Hirasawa, and Hiroshi Shimizu. Development and experimental verification of a genome-scale metabolic model for corynebacterium glutamicum. *Microbial Cell Factories*, 43 (8), August 2009. URL `http://www.microbialcellfactories.com/content/8/1/43`.

[13] Oh YK, Palsson BO, Park SM, Schilling CH, and Mahadevan R. Genome-scale reconstruction of metabolic network in bacillus subtilis based on high-throughput phenotyping and gene essentiality data. *J Biol Chem*, 282(39):28791–28799, September 2007. URL http://www.ncbi.nlm.nih.gov/pubmed/17573341.

[14] Henry CS, Zinner JF, Cohoon MP, and Stevens RL. Genome-scale reconstruction of metabolic network in bacillus subtilis based on high-throughput phenotyping and gene essentiality data. *Genome Biol*, 10(6), June 2009. URL http://www.ncbi.nlm.nih.gov/pubmed/19555510.

[15] Hattori M. Finishing the euchromatic sequence of the human genome. *Nature*, 431 (7011), October 2004. URL http://www.ncbi.nlm.nih.gov/pubmed/15496913.

[16] Bernhard O. Palsson. *Systems Biology.* Cambridge University Press, 2009.

[17] Alper H, Jin YS, Moxley JF, and Stephanopoulos G. Identifying gene targets for the metabolic engineering of lycopene biosynthesis in escherichia coli. *Metab Eng*, 7(3): 155–164, May 2005. URL http://www.ncbi.nlm.nih.gov/pubmed/15885614.

[18] Alper H, Miyaoku K, and Stephanopoulos G. Construction of lycopene-overproducing e. coli strains by combining systematic and combinatorial gene knockout targets. *Nat Biotechnol*, 23(5):612–617, May 2005. URL http://www.ncbi.nlm.nih.gov/pubmed/15821729.

[19] Sang Jun Lee, Dong-Yup Lee, Tae Yong Kim, Byung Hun Kim, Jinwon Lee, and Sang Yup Lee. Metabolic engineering of escherichia coli for enhanced production of succinic acid, based on genome comparison and in silico gene knockout simulation. *Applied Environmental Microbiology*, 71(12):7880–7887, December 2005. URL http://www.ncbi.nlm.nih.gov/pubmed.

[20] Wang Q, Chen X, Yang Y, and Zhao X. Genome-scale in silico aided metabolic analysis and flux comparisons of escherichia coli to improve succinate production. *Appl Microbiol Biotechnol*, 73(4):887–894, December 2006. URL http://www.ncbi.nlm.nih.gov/pubmed.

[21] Fong SS, Burgard AP, Herring CD, Knight EM, Blattner FR, Maranas CD, and Palsson BO. In silico design and adaptive evolution of escherichia coli for production of lactic acid. *Biotechnol Bioeng*, 91(5):643–650, September 2005. URL http://www.ncbi.nlm.nih.gov/pubmed.

[22] Soo Yun Moon, Soon Ho Hong, Tae Yong Kim, and Sang Yup Lee. Metabolic engineering of escherichia coli for the production of malic acid. *Biochemical Engineering Journal*, 40(2):312–320, June 2008. URL http://dx.doi.org.proxy.libraries.rutgers.edu/10.1016/j.bej.2008.01.001.

[23] Park JH, Lee KH, Kim TY, and Lee SY. Metabolic engineering of escherichia coli for the production of l-valine based on transcriptome analysis and in silico gene knockout simulation. *Proc Natl Acad Sci U S A*, 104(19):7797–7802, May 2007. URL http://www.ncbi.nlm.nih.gov/pubmed.

[24] Kwang Ho Lee, Jin Hwan Park, Tae Yong Kim, Hyun Uk Kim, and Sang Yup Lee. Systems metabolic engineering of escherichia coli for l-threonine production. *Mol Syst Biol*, 3(143), December 2007. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2174629/`.

[25] Pharkya P, Burgard AP, and Maranas CD. Exploring the overproduction of amino acids using the bilevel optimization framework optknock. *Biotechnol Bioeng*, 84(7): 887–899, December 2003. URL `http://www.ncbi.nlm.nih.gov/pubmed/14708128`.

[26] Priti Pharkya, Anthony P. Burgard, and Costas D Maranas. Optstrain: A computational framework for redesign of microbial production systems. *Genome Research*, 14(11):2367–2376, November 2004. URL `http://www.ncbi.nlm.nih.gov/pubmed/15520298`.

[27] J. S. Edwards and B. O. Palsson. The escherichia coli mg1655 in silico metabolic genotype: Its definition, characteristics, and capabilities. *Proceedings of the National Academy of Sciences of the United States of America*, 97(10):5528–5533, May 2000. URL `http://www.pnas.org/content/97/10/5528.full`.

[28] Jennifer L Reed, Thuy D Vo, Christophe H Schilling, and Bernhard O Palsson. An expanded genome-scale model of escherichia coli k-12 (ijr904 gsm/gpr). *Genome Biology*, 4(R45), August 2003. URL `http://genomebiology.com/2003/4/9/R54`.

[29] Adam M Feist, Christopher S Henry, Jennifer L Reed, Markus Krummenacker, Andrew R Joyce, Peter D Karp, Linda J Broadbelt, Vassily Hatzimanikatis, and Bernhard Palsson. The escherichia coli mg1655 in silico metabolic genotype: Its definition, characteristics, and capabilities. *Mol Syst Biol*, 3(121):5528–5533, June 2007. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1911197/`.

[30] Nathan D Price, Jennifer L Reed, and Bernhard Palsson. Genome-scale models of microbial cells: evaluating the consequences of constraints. *Nature Reviews Microbiology*, 2(1):886–878, November 2004. URL `http://www.nature.com.proxy.libraries.rutgers.edu/nrmicro/journal/v2/n11/full/nrmicro1023.html`.

[31] Daniel Segre, Dennis Vitkup, and George M. Church. Analysis of optimality in natural and perturbed metabolic networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(23):15112–15117, November 2002. URL `www.pnas.com/cgi/doi/10.1073/pnas.232349399`.

[32] R Heinrich and S Schuster. *The Regulation of Cellular Systems*. Chapman and Hall, 1996.

[33] David Fell. *Understanding the Control of Metabolism*. Portland Press, 1999.

[34] Frederick C. Neidhardt, John L. Ingraham, and Moselio Schaechter. *Physiology of the Bacterial Cell: A Molecular Approach*. Sinauer Associates, 1990.

[35] Jeffrey D Orth, Ines Thiele, and Berhard O Palsson. What is flux balance analysis. *Nature Biotechnology*, 28(3):245–248, March 2010. URL `http://www.naturebiotechnology.com`.

[36] Majewski RA and Domach MM. Simple constrained-optimization view of acetate overflow in e. coli. *Biotechnol Bioeng*, 35(7):732–738, March 1990. URL `http://www.ncbi.nlm.nih.gov/pubmed`.

[37] Ramakrishna R, Edwards JS, McCulloch A, and Palsson BO. Flux-balance analysis of mitochondrial energy metabolism: consequences of systemic stoichiometric constraints. *Am J Physiol Regul Integr Comp Physiol*, 280(3):695–704, March 2001. URL `http://www.ncbi.nlm.nih.gov/pubmed/11171647`.

[38] Amit Varma and Bernhard O. Palsson. Metabolic capabilities of escherichia coli ii. optimal growth patterns. *Journal of Theoretical Biology*, 35(4):503–522, December 1993. URL `http://www.sciencedirect.com/science/article/pii/S0022519383712038`.

[39] Badarinarayana V, Estep PW 3rd, Shendure J, Edwards J, Tavazoie S, Lam F, and Church GM. Selection analyses of insertional mutants using subgenic-resolution arrays. *Nat Biotechnol*, 19(11):1060–1065, November 2001. URL `http://www.ncbi.nlm.nih.gov/pubmed/11689852`.

[40] Bard JF. *Practical Bilevel Optimization: Algorithms and Applications*. Kluwer Academic Publishers, 1998.

[41] Frederick C. Neidhardt. *Escherichia coli and Salmonella: Cellular and Molecular Biology*. ASM Press, 1996.

[42] Adam M Feist, Christopher S Henry, Jennifer L Reed, Markus Krummenacker, Andrew R Joyce, Peter D Karp, Linda J Broadbelt, Vassily Hatzimanikatis, and Bernhard Palsson. A genome-scale metabolic reconstruction for escherichia coli k-12 mg1655 that accounts for 1260 orfs and thermodynamic information. *Mol Syst Biol*, 3(121):994–998, June 2007. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1911197/`.

[43] Feist AM and Palsson B. The growing scope of applications of genome-scale metabolic reconstructions using escherichia coli. *Nat Biotechnol*, 26(6):659–667, June 2008. URL `http://www.ncbi.nlm.nih.gov/pubmed/18536691`.

[44] Priti Pharkya and Costas D. Maranas. An optimization framework for identifying reaction activation/inhibition or elimination candidates for overproduction in microbial systems. *Metabolic Engineering*, 28(30):1–13, September 2005. URL `http://www.elsevier.com/locate/ymben`.

[45] Patil KR, Rocha I, Frster J, and Nielsen J. Evolutionary programming as a platform for in silico metabolic engineering. *BMC Bioinformatics*, 6(308):994–998, December 2005. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1327682/pdf/1471-2105-6-308.pdf`.

[46] Miguel Rocha, Paulo Maia, Rui Mendes, Jos P Pinto, Eugnio C Ferreira, Jens Nielsen, Kiran Raosaheb Patil, and Isabel Rocha. Natural computation meta-heuristics for the in silico optimization of microbial strains. *BMC Bioinformatics*, 9(499), November 2008. URL `http://www.biomedcentral.com/1471-2105/9/499`.

[47] Burgard AP, Vaidyaraman S, and Maranas CD. Minimal reaction sets for escherichia coli metabolism under different growth requirements and uptake environments. *Biotechnol Prog*, 17(5):791–797, September-October 2001. URL `http://www.ncbi.nlm.nih.gov/pubmed/11587566`.

[48] Mahadevan R and Schilling CH. The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. *Metab Eng*, 5(4):264–276, October 2003. URL `http://www.ncbi.nlm.nih.gov/pubmed/14642354`.

[49] Maranas CD Pharkya P. An optimization framework for identifying reaction activation/inhibition or elimination candidates for overproduction in microbial systems. *Metab Eng*, 8(1):1–13, January 2006. URL `http://www.ncbi.nlm.nih.gov/pubmed/16199194`.

[50] Burgard AP, Vaidyaraman S, and Maranas CD. Minimal reaction sets for escherichia coli metabolism under different growth requirements and uptake environments. *Biotechnol Prog*, 17(5):791–797, September-October 2001. URL `http://www.ncbi.nlm.nih.gov/pubmed/11587566`.

[51] J. G. Zeikus, M. K. Jain, and P. Elankovan. Biotechnology of succinic acid production and markets for derived industrial products. *Applied Microbiology and Biotechnology*, 51(5), May 1999. URL `http://link.springer.com/article/10.1007%2Fs002530051431`.

[52] Causey TB, Zhou S, Shanmugam KT, and Ingram LO. Engineering the metabolism of escherichia coli w3110 for the conversion of sugar to redox-neutral and oxidized products: homoacetate production. *Proc Natl Acad Sci U S A*, 100(3):825–832, February 2003. URL `http://www.ncbi.nlm.nih.gov/pubmed/12556564`.

[53] Weixiong Zhang. Truncated branch-and-bound: A case study on the asymmetric tsp. *Proceedings of the AAAI-93 Spring Symposium on AI and NP-Hard Problems*, 51(5):160–166, December 1993. URL `http://www.researchgate.net/publication/2387219_Truncated_Branch-and-Bound_A_Case_Study_on_the_Asymmetric_TSP`.

[54] Brenda Laurel. *The Art of Human-Computer Interface Design.* Addison-Wesley, 1990.

[55] William Stallings. *Operating Systems.* Prentice Hall, 7 edition, 2012.

[56] Mohamed Esseghir Lalami and Didier El-Baz. Gpu implementation of the branch and bound method for knapsack problems. *IEEE Computer Society*, 26:1769–1777, 2012.