To my parents, Zhenlong Song and Meilu An
To my little sister, Rui Song
ACKNOWLEDGMENTS

First, I thank to my supervising committee chair, Associate Professor, Dr. Tamer Kahveci. As a nice professor, he gave me helpful and useful advices and suggestions when I was pursuing my Ph.D. Before I worked with him, I had no idea about how to pursue my Ph.D and how to do the research. After I joined his group, he was patient to guide me in my graduate study. He advised me on how to select research topics, solve the problems and present the papers. I am greatly influenced by his idea that “Do things in a smart way”. Everything has rules and people should follow these rules. For example, for research, he thinks it is much more important to learn the way to do research than doing the research itself. During pursuing Ph.D., I should become an independent researcher to find the problem and solve the problem. I learned a lot from him for my future career. Moreover, Dr. Kahveci is also a helpful friend. I met tough difficulties in my life when I was in school. He encouraged me a lot to overcome these difficulties. He inspired me to continue my research, finish my graduate study and create a bright future. He spent a lot of time to talk with me and illuminate the hope in my heart. He is really a nice professor and friend in my life. Without his help, I can not finish my graduate study.

Second, I am also thankful to Professor Sanjay Ranka as my committee cochair. He is also patient to guide my research topics and give me helpful suggestions in techniques. He provided useful ideas to organize my Ph.D. dissertation and research topics. I learned a lot from him in the technique parts. With his help, my research went smoothly. I have the courage to go further in my future career.

Thanks are also due to Professor Sartaj Sahni, Professor Jih-Kwon Peir and Professor Raymond Booth as my Ph.D. committee members. They gave me beneficial suggestions in my research topics and works. They fulfilled my Ph.D. dissertation.

Also, I am thankful to my officemates and friends who were with me in my graduate study. Ph.D. life is tough and hard. With my officemates and friends, my life became colorful. Thank
Xu Zhang, Padmavati Sridhar, Jayendra Venkateswaran, Lixia Chen, Yuchu Tong, Fei Xu, Leqian Zhang, Hechen Liu, Wenjie Yuan, Xiao Li, Ming Li, Gang Liu, Yan Li, Xuelian Xiao, Ferhat Ay, Nirmalya Bandyopadhyay, Gunhan Gulsoy and so on.

Finally, I greatly appreciated my family. When I felt frustrated in my life, my parents always accompanied me. My parents visited me when I was in the hard times. Their warm hearts and loves give me the courage and energy. Without their loves, I can not imagine that I am able to finish my dissertation. My mother is a great mother and she always says that “Everything will be well”. My father is a wonderful listener. He can solve all of my troubles. My little sister stands with me and supports me. With my little sister, I do not feel lonely in this world. Thanks to my family!
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGMENTS</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td></td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
</tr>
</tbody>
</table>

## CHAPTER

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>1.1 Overview of the Thesis</td>
<td></td>
</tr>
<tr>
<td>1.2 Overview of the Related Work</td>
<td></td>
</tr>
<tr>
<td>1.3 Overview of the Contributions</td>
<td></td>
</tr>
<tr>
<td>1.4 Outline of the Thesis</td>
<td></td>
</tr>
<tr>
<td>2 ENZYMATIC TARGET IDENTIFICATION USING BOOLEAN MODEL BY HEURISTIC METHODS</td>
<td></td>
</tr>
<tr>
<td>2.1 Motivation and Problem Definition</td>
<td></td>
</tr>
<tr>
<td>2.2 Methods</td>
<td></td>
</tr>
<tr>
<td>2.2.1 The Iteration Phase</td>
<td></td>
</tr>
<tr>
<td>2.2.2 The Fusion Phase</td>
<td></td>
</tr>
<tr>
<td>2.3 Results</td>
<td></td>
</tr>
<tr>
<td>2.3.1 Evaluation of the Damage Model on Real Drugs</td>
<td></td>
</tr>
<tr>
<td>2.3.2 Evaluation of the Iteration Phase</td>
<td></td>
</tr>
<tr>
<td>2.3.3 Evaluation of the Fusion Phase</td>
<td></td>
</tr>
<tr>
<td>2.4 Discussion</td>
<td></td>
</tr>
<tr>
<td>3 ENZYMATIC TARGET IDENTIFICATION USING LINEAR MODEL FOR MULTIPLE ENZYMES CATALYZE THE SAME REACTION</td>
<td></td>
</tr>
<tr>
<td>3.1 Motivation and Problem Definition</td>
<td></td>
</tr>
<tr>
<td>3.2 Computational Complexity</td>
<td></td>
</tr>
<tr>
<td>3.3 Methods</td>
<td></td>
</tr>
<tr>
<td>3.3.1 Single Enzyme</td>
<td></td>
</tr>
<tr>
<td>3.3.2 Multiple Enzymes</td>
<td></td>
</tr>
<tr>
<td>3.3.2.1 Substitute enzyme</td>
<td></td>
</tr>
<tr>
<td>3.3.2.2 Collaborate enzyme</td>
<td></td>
</tr>
<tr>
<td>3.3.2.3 Enzyme complex</td>
<td></td>
</tr>
<tr>
<td>3.4 Results</td>
<td></td>
</tr>
</tbody>
</table>
5.3.1 Notation ................................................................. 125
5.3.2 Exact Distance ...................................................... 126
5.3.3 Time-warping Distance .......................................... 127
5.3.4 Pattern Distance .................................................... 129
5.4 Methods ................................................................. 132
5.5 Experimental Results ............................................... 134
  5.5.1 Evaluation of the Performance .............................. 136
  5.5.2 Evaluation of the Accuracy ................................. 138
5.6 Discussion .............................................................. 139

6 INTEGRATING STRUCTURAL PROPERTIES OF PROTEINS AND BIOLOGICAL NETWORKS FOR COMPOUND SELECTION ......................................................... 143
  6.1 Motivation and Problem Definition ........................... 143
  6.2 Methods ................................................................. 147
    6.2.1 Protein Selection ............................................ 148
    6.2.2 Ranking Compounds ....................................... 149
      6.2.2.1 Ranking based on affinities .......................... 150
      6.2.2.2 Integrating networks in ranking ................. 152
  6.3 Results .............................................................. 156
    6.3.1 Accuracy in Estimation .................................. 160
    6.3.2 Accuracy in Ranking ..................................... 161
  6.4 Discussion ........................................................... 162

7 CONCLUSION ............................................................. 166
REFERENCES ............................................................... 168
BIOGRAPHICAL SKETCH .................................................. 177
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Iterative Steps for Figure fig/iter Eg.</td>
<td>51</td>
</tr>
<tr>
<td>2-2</td>
<td>Metabolic networks used in our experiments.</td>
<td>51</td>
</tr>
<tr>
<td>2-3</td>
<td>Comparison of the average damage values of solutions determined by the Iteration Phase versus that determined by the exhaustive search algorithm.</td>
<td>51</td>
</tr>
<tr>
<td>2-4</td>
<td>Comparison of the damage values found by the Iteration Phase and the Fusion Phase focusing on the queries in which the Fusion Phase improved the results of the Iteration Phase.</td>
<td>52</td>
</tr>
<tr>
<td>3-1</td>
<td>Running time in seconds of our binary method for the whole metabolism of <em>H. sapiens</em> from KEGG.</td>
<td>67</td>
</tr>
<tr>
<td>4-1</td>
<td>An example showing the generation of a child <em>Ch</em> from two hypothetical parents <em>F</em> and <em>M</em> for a pathway that contains nine enzymes. <em>?</em> denotes an undecided value.</td>
<td>112</td>
</tr>
<tr>
<td>4-2</td>
<td>The expected number of undecided enzymes for different pathways. <em>#E</em> denotes the number of enzymes.</td>
<td>113</td>
</tr>
<tr>
<td>4-3</td>
<td>Metabolic pathways from KEGG that are used in our experiments in this chapter.</td>
<td>113</td>
</tr>
<tr>
<td>4-4</td>
<td>Comparison of the truncated traversal method (maximum number of levels = 20) and the genetic algorithm for different pathways. <em>?</em> denotes that no SD values can be computed within one day.</td>
<td>116</td>
</tr>
<tr>
<td>4-5</td>
<td>The average SD value and the running time of the truncated traversal algorithm (with maximum number of levels = 23) and the Genetic Algorithm. The running time is reported in seconds.</td>
<td>117</td>
</tr>
<tr>
<td>4-6</td>
<td>The average running time of our Genetic Algorithm for large metabolism. The running time is reported in seconds.</td>
<td>117</td>
</tr>
<tr>
<td>4-7</td>
<td>The average SD values with different mutation rates <em>δ</em> in several pathways.</td>
<td>117</td>
</tr>
<tr>
<td>4-8</td>
<td>The estimated <em>λ</em> values and the cost time for different levels for the Glycolysis / Gluconeogenesis pathway. The average number of knocked out enzymes in the optimal solution of this pathway is 2.2.</td>
<td>117</td>
</tr>
<tr>
<td>4-9</td>
<td>The average SD value and the running time of Patil’s method and our Genetic Algorithm. The running time is reported in seconds.</td>
<td>118</td>
</tr>
<tr>
<td>5-1</td>
<td>Comparison of the average percentage of the intervals our algorithm generates for the exact and time-warping distance with and without splitting the dynamic states.</td>
<td>142</td>
</tr>
</tbody>
</table>
5-2 Accuracy of our algorithm using the three distance measures on datasets with different characteristics. The accuracy values are reported in terms of percentage of success. ................................. 142

6-1 Dataset information needed for the application to the small molecule pharmacologic chaperones of misfolded rhodopsin. ................................................................. 164

6-2 The accuracy of the activity level predictions of four strategies. Correlation coefficient shows the correlation between the actual and the predicted activity levels. $Ob_p$ and $Ob_a$ denote the predicted and the actual activity levels. ................................. 164

6-3 The ranking accuracy of three strategies. ................................................................. 164
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>A structure of this thesis</td>
<td>31</td>
</tr>
<tr>
<td>1-2</td>
<td>Graph representation of a metabolic pathway with four reactions $R_1$, $R_2$, $R_3$ and $R_4$, three enzymes $E_1$, $E_2$ and $E_3$, and five compounds $C_1$, $C_2$, $C_3$, $C_4$, $C_5$.</td>
<td>32</td>
</tr>
<tr>
<td>2-1</td>
<td>A graph constructed for a metabolic network with four reactions $R_1$, $R_2$, $R_3$, $R_4$, three enzymes $E_1$, $E_2$ and $E_3$, and five compounds $C_1$, $C_2$, $C_3$, $C_4$, $C_5$. $C_1$ is the target compound.</td>
<td>49</td>
</tr>
<tr>
<td>2-2</td>
<td>Evaluation of the Iteration Phase. (a) Average execution time in milliseconds. (b) Average number of iterations</td>
<td>50</td>
</tr>
<tr>
<td>3-1</td>
<td>Redraw Protein-reaction associations by Reed et al.</td>
<td>68</td>
</tr>
<tr>
<td>3-2</td>
<td>The average running time (in seconds) for the networks with single enzyme and multiple enzymes (substitute and collaborate).</td>
<td>68</td>
</tr>
<tr>
<td>3-3</td>
<td>The average running time (in seconds) of binary and continuous methods for the networks whose multiple enzymes are substitute ones.</td>
<td>69</td>
</tr>
<tr>
<td>3-4</td>
<td>The average running time (in seconds) of binary and continuous methods for the networks whose multiple enzymes are collaborate.</td>
<td>69</td>
</tr>
<tr>
<td>4-1</td>
<td>Graph representation of a metabolic pathway with three reactions $R_1$, $R_2$, $R_3$, two enzymes $E_1$ and $E_2$, and nine compounds $C_1$, $C_2$, $C_3$, $C_4$, $C_5$, $C_6$, $C_7$, $C_8$, $C_9$. Dashed lines show the impact of knocking out enzyme $E_1$.</td>
<td>112</td>
</tr>
<tr>
<td>4-2</td>
<td>Flux distribution of a hypothetical pathway.</td>
<td>112</td>
</tr>
<tr>
<td>4-3</td>
<td>Upper bound to the expected number of undecided enzymes for different $\lambda$ and pathway sizes ($N =</td>
<td>E</td>
</tr>
<tr>
<td>4-4</td>
<td>Glycolysis/Gluconeogenesis for <em>E.coli</em>. The enzymes highlighted in green are the enzymes that exist in <em>E.coli</em>.</td>
<td>114</td>
</tr>
<tr>
<td>4-5</td>
<td>Phenylalanine, tyrosine and tryptophan biosynthesis for <em>E.coli</em>. The enzymes highlighted in green are the enzymes that exist in <em>E.coli</em>.</td>
<td>115</td>
</tr>
<tr>
<td>4-6</td>
<td>Average SD values of the traversal method and exhaustive search for different pathways over multiple queries.</td>
<td>115</td>
</tr>
<tr>
<td>4-7</td>
<td>The average running time (in seconds) of the traversal method and exhaustive search over multiple queries for different pathways.</td>
<td>116</td>
</tr>
<tr>
<td>4-8</td>
<td>Distribution of the number of enzymes for Urea cycle and metabolism of amino groups.</td>
<td>116</td>
</tr>
</tbody>
</table>
5-1  The patterns $P_1$ and $P_2$ show the sequence of concentration of a compound resulting from two alternative manipulations to the metabolic network.  

5-2  Three transformations, stretch, scale and shift applied on a hypothetical dynamic state.  

5-3  An illustration of two dynamic states.  

5-4  An illustration of how partitioning the dynamic state $X$ into shorter intervals improves the running time.  

6-1  A framework for three compound selection strategies denoted by A, B, and C.  

6-2  Distribution of binding affinity for rhodopsin (PDB:1F88) and all the 1,990 compounds in diversity set.  

6-3  Retinol metabolism and Starch and sucrose metabolism. The dark solid circle shows the enzymes in the selected protein set.  

This thesis focuses on in silico methods for metabolic engineering. Metabolic engineering discusses methods to manipulate the metabolism to approach a given goal, e.g. increasing the cells’ production of a certain substance by operating the genetic and regulatory processes. It is widely applied to drug discovery, food industry and cosmetics.

In order to change the metabolism to the desired one, e.g. increasing the production of a certain compound, one metabolic engineering method is to use chemical compounds (i.e. drugs) to inhibit a set of enzymes. When an enzyme is inhibited, it cannot catalyze the reactions it is responsible from. As a result, the production of some of the compounds within the system may increase and some may decrease. The amount or the production rate of all the compounds determine the state of the metabolism. The enzymatic target identification problem aims to identify the set of enzymes whose knockouts lead to the state of the system close to the goal.

There are several models in the literature to describe the state of metabolism. In this dissertation we address the enzymatic target identification problem for a broad set of mathematical models. One of these models is Boolean model. We develop a scalable iterative method for the problem when Boolean model is used. This method consists of two phases: Iteration and Fusion Phases. The experiments on the E. coli metabolic network show that the average accuracy of the Iteration Phase alone deviates from that of the exhaustive search only by 0.02 %. The Iteration Phase is
highly scalable. It can solve the problem for the entire metabolic network of *Escherichia coli* in less than 10 seconds. The Fusion Phase improves the accuracy of the Iteration Phase by 19.3%.

Linear model describes metabolic networks as a system of linear equations rather than Boolean predicates. We prove that finding the enzyme knockout strategy by OptKnock framework is NP-hard and present methods considering multiple enzyme association. Though there exist a lot of articles to study the enzyme knockout strategy, there are few of these papers considering the enzyme association. We present the enzyme knockout strategies on FBA which can deal with the situation that a reaction is catalyzed by multiple enzymes. Considering the enzyme association, that is, “AND”, “OR” or a combination of them, we provide a binary method and continuous method for each association. Our experiments suggest that the enzyme associations influence the performance of linear programming method very much. We observe that our binary method runs much faster than continuous method. For the pathways of *H.sapiens* from KEGG, our binary method runs in less than one second for the entire metabolism. Therefore, our binary method is useful for the biological application.

Non-linear models can simulate the whole cell system and describe the complex interactions within a metabolic network that are impossible to explain using linear equations. We design two algorithms that solves the enzymatic target identification under non-linear as well as linear models. The first one is a traversal approach that explores possible solutions in a systematic way using a branch and bound method. The second one uses genetic algorithm to derive good solutions from a set of alternative solutions iteratively. Unlike the former one, this one can run for very large pathways. Our experiments show that our algorithms’ results follow those obtained in vitro in the literature from a number of applications. They also show that the traversal method is a good approximation of the exhaustive search algorithm and it is up to 11 times faster than the exhaustive one. This algorithm runs efficiently for pathways with up to 30 enzymes. For large pathways, our genetic algorithm can find good solutions in less than 10 minutes.
All the above three models, boolean, linear and non-linear suggest that metabolism reaches to steady state over time by changing its state dynamically. The sequence of all these states is the “dynamic state” of the system. Next, we consider the dynamic state of the system when we study the enzymatic target identification problem. We aim to find the set of enzyme knockouts that will produce a dynamic state similar to that goal pattern. In order to compare two dynamic states meaningfully, we propose three distance functions. These are the Euclidean distance, time-warping distance and pattern distance. Euclidean distance restricts the solution space to the exact goal state. Time-warping distance allows for stretching of the goal pattern in the time domain. Pattern distance allows scaling and shifting of the goal flux in addition to stretching in the time domain. We provide a branch and bound method to solve this problem. We also develop a partitioning strategy to reduce the running time of our method. This strategy avoids constructing the entire dynamic state by computing a lower bound to the distance between two dynamic states when the entire dynamic state is not available. Our experiments on the Purine metabolism show that our method runs accurately. They also show that our partitioning strategy reduces the number of time intervals computed for dynamic states by a factor of 2 to 6.

Once we identify which enzyme set should inhibit, the next step is to select chemical compounds (i.e. drugs) to alter the activity of these enzymes. One of the popular compound selection methods is to screen libraries of small compounds for their ability to bind to biological targets such as receptors and enzymes in silico. We develop two novel computational methods that rank a given set of compounds for a given target protein or enzyme. The major difference between our first method and traditional in-silico screening methods is that we consider additional proteins and enzymes while ranking compounds whereas existing strategies often focus only on the target protein alone. A drug compound can alter the state of the metabolic network. Our second method considers the impact of the drug compounds on the metabolic network by integrating the interactions among proteins in metabolic networks with the docking results. Experiments on
the pharmacologic chaperones of misfolded rhodopsin show that our method has better accuracy than the traditional methods that focus only on rhodopsin. Our results are in the top 5.7% of all possible rankings. For the same dataset, the traditional method’s results are in the top 81% of all possible rankings.
Metabolic engineering discusses methods to manipulate the metabolism to approach a given goal, e.g. increasing the cells’ production of a certain substance by operating the genetic and regulatory processes [20, 97, 104, 112]. It is widely applied to drug discovery, food industry and cosmetics. For example, fatty acid biosynthesis pathway converts fatty acids that are used in the cosmetic industry in creams and lotions [105, 115]. Butanoate metabolism produces poly-β-hydroxybutyrate which is essential for producing plastics [11]. Mevalonic acid pathway and MEP/DOXP pathway produce carotenoid that are often used as anti-oxidant in food industry [83]. The metabolisms of many organisms, such as bacteria, algea and plants naturally produce these compounds. A common practice is to extract them from these organisms. By manipulating the pathways of these organisms, the production of these compounds can be increased significantly by metabolic engineering. For example, we can cut the consumption of the desired compound by the underlying organism.

In drug discovery field, metabolic engineering plays a significant role too [10, 53]. A healthy metabolism keeps the status of organism at certain values. External affects or genetic mutations can change the production rate of a set of enzymes. They can even modify the structure of produced enzymes. Such unexpected enzyme behaviors can lead to aberrations in the metabolism. Low or missing activity of an enzyme may result in the blockage of the pathway. Furthermore, this can propagate to other parts of the pathway that need the compounds produced in the blocked part of the pathway. As a result, the production of some compounds may be increased and some of the others may be decreased [66, 84, 111]. Such aberrations in the metabolism can lead to severe diseases such as mental retardation, seizures, decreased muscle tone, organ failure and blindness [27, 89]. Thus, changing the metabolism back to a desired level is needed. Therefore, a metabolic engineering method is needed to alter the metabolism status.
Currently, in silico method is widely used in metabolic engineering [8, 22, 24, 25, 72, 77]. Computational methods have the following advantages. First, it has low costs. With the computational prediction and filter, Biologists do not need to build unnecessary in vivo or in vitro experiments [45, 60, 98]. Second, it reduces large amounts of time. For example, if we want to select drug candidates from a large compound library by real screening, that is, we build experiments for each compound, then it will cost quite a few years for a library with 10000 compounds. However, if we select them by virtual screening, that is, we screen these compounds by computer, we can test thousands of compounds in a day and finish a whole compound library with 10000 compounds for only several days. Third, with the help of computer, we may design a compound as drug which has not been synthesized yet.

Figure 1-1 shows the structure of this thesis which discusses the problems of metabolic engineering in the following chapters in detail.

1.1 Overview of the Thesis

Enzymes catalyze biological reactions [33, 91]. Reactions transform a set of compounds into another set of compounds. In order to change the metabolism to the desired one, e.g. increasing the production of a certain compound, one metabolic engineering method is to use chemical compounds (i.e. drugs) to inhibit a set of enzymes. When an enzyme is inhibited, it cannot catalyze the reactions it is responsible from. As a result, the production of the system may increase and some may decrease. The enzymatic target identification problem is how to identify the set of enzymes whose knockouts lead to the system status close to the goal. The size and the complex structure of the metabolic pathways make the enzymatic target identification problem computationally challenging. In order to change the metabolism to the desired one, we first evaluate the state of the metabolism. Based on different models, the state of the metabolism can be expressed in different ways, such as the yield of each compound, the flux of each compound or the concentration of each compound. The steady state means that the state that remains
unchanged over time. If we only consider the steady state of the system, there exist three popular models, Boolean model, linear model and non-linear model. The first part of this thesis discusses the enzymatic target identification problem for the steady state analysis with three models, Boolean model, linear model and non-linear model. This thesis also describes the methods for the enzymatic target identification problem with dynamic states. Finally, the thesis presents approaches to select compounds to inhibit enzymes.

**Enzymatic target identification using Boolean network models.** Sridhar et al. and Song et al. considered a Boolean model of the enzymatic target identification problem [94–96]. In their version, each entry of the state denotes whether the corresponding compound is present or not. For this simplified version, the goal, then, is to identify the set of enzymes whose knockouts eliminate all the “targeted compounds” while incurring minimum “damage”. Currently, Sridhar et al provided an optimal algorithm for this model [96]. Chapter 2 discusses a heuristic algorithm for the enzymatic target identification problem using Boolean model.

**Enzymatic target identification using linear model for multiple enzymes catalyze the same reaction.** In fact, Boolean model is too simple to describe the system. Flux Balance Analysis, (FBA) [7, 23, 29, 48, 57, 69, 71] is a popular linear model to describe the metabolism as linear programming equations. Web experiments show that it is a reasonable model for the flux distribution. For the linear model, we can use integer linear programming to tackle the enzymatic target identification problem. OptKnock is one example to the algorithms in this class [9]. However, OptKnock can not deal with the situation that the reaction is catalyzed by multiple enzymes not only one. Chapter 3 provides a method for the enzymatic target identification problem using linear model in the case that multiple enzymes catalyze the same reaction.

**Enzymatic target identification using non-linear model by manipulating the steady state.** Though the linear model works well for some cases, there exist more complex non-linear models to describe the metabolism. These non-linear models can simulate the cell system better than
the linear model. For example, S-systems [85, 86, 107] and GMA model [15, 43, 73, 107] are two popular non-linear models for the cell system simulation. For these non-linear models, the previous methods, e.g. linear programming method, can not apply to the enzymatic target identification problem directly. Therefore, Chapter 4 describes methods for the enzymatic target identification problem using the non-linear model.

**Enzymatic target identification with dynamic states.** If we consider the dynamic state to evaluate the system status, Chapter 5 discusses this situation. The process between two steady states is significant. If there exists two different “pathes” from the start state to the final state, the influence of these two pathes on the whole biological system may be significant different. For example, if we want to increase the blood sugar concentration of a biological system, one path is to gradually enhance the blood sugar concentration. Another way is to aggrandize the blood sugar concentration with a sharp curve then decrease it to the goal concentration. The first method may cost much time to reach the goal however the second method may bring dangerous side-effect. That is, the sugar concentration may reach a serious high extent and lead the organism to die. Thus, it is necessary to consider the dynamic process when we identify the enzyme set to change the state of the biological system. Therefore, Chapter 5 describes methods for the enzymatic target identification problem with dynamic states.

**Integrating structural properties of proteins and biological networks for compound selection.** Once we identify which enzyme set should inhibit, the next step is to select chemical compounds (i.e. drugs) to alter the activity of these enzymes. One of the popular compound selection methods is to screen libraries of small compounds for their ability to bind to biological targets such as receptors and enzymes [38] in silico. This process is also known as “docking” [59]. Despite some success in compound prediction in several applications, recent validation studies show that docking methods have a poor performance in compound selection [110]. In this thesis, we consider the compound selection problem. We define this problem as follows. Assume
that we are given a target protein or enzyme and a library of compounds. Compound selection problem aims to identify the compounds from this library that will bind to the target at a high rate and change the activity level of the target. More specifically, we develop a ranking algorithm that sorts the compounds in the compound library according to their probability of altering the activity of the target protein or enzyme.

1.2 Overview of the Related Work

Metabolic engineering has been studies for tens of years. However, in silico approaches for metabolic engineering is still a new field. In order to find new in silico approaches for metabolic engineering, the first step is to achieve a number of databases for metabolism. Fortunately, there are more and more public databases available. Some examples are KEGG [46], EcoCyc [52] and ENZYME [3]. There are also some Bionetwork and interaction databases such as, aMAZE [58], DIP [55] and BIND [2]. Enzymatic target identification is a new field in metabolic engineering. Sridhar et al. and Song et al. provided the enzymatic target identification problem [94–96]. If we consider the steady state of the system, we briefly summarize three classic models and corresponding methods for the enzymatic target identification problem.

For the enzymatic target identification problem using Boolean model, Sridhar et al. and Song et al. considered a simplified version of Boolean model [94–96]. In Boolean model, compounds are separated into two groups, Target compounds and Non-Target compounds. Target compounds are the ones that we aim to remove and Non-Target compounds are the remaining ones. All the compounds are two states, present and not present. We term the side-effects of inhibiting a given set of enzymes as the damage caused to the metabolic network. Formally, we define damage of inhibiting a set of enzymes as the number of non-target compounds whose productions are stopped due to the inhibition of those enzymes. Then, the enzymatic target identification problem based on Boolean model is to seek the set of enzymes whose inhibition eliminates all the target compounds and inflicts minimum damage on the rest of the network.
Figure 1-2 shows an example for Boolean model. Inhibiting $E_1$ knocks out the reactions $R_1$ and $R_2$ and compounds $C_1$, $C_2$, $C_3$ and $C_4$. Suppose $C_1$ is the target compound. Inhibiting $E_1$ stops three non-target compounds ($C_2$, $C_3$ and $C_4$). The damage is three. If we inhibit $E_2$ and $E_3$, we remove the reactions $R_3$, $R_4$ and $R_1$. Then, we remove the target compound $C_1$ and one non-target compound $C_5$. The damage is one. It comes the question which enzyme set is better and how to select the enzyme set with the minimum damage. Evaluating all the enzyme set combination is exponential.

Sridhar et al., used a branch and bound strategy, named OPMET to solve this problem for pathways with up to 32 enzymes in less than an hour [96]. OPMET expressed the search space as a binary tree. In the root node, all the enzymes are present in the network. OPMET started from the root node. When OPMET traversed the tree, first visit the current node, second the left child, third the right child. When OPMET visited the current node, it computed the current damage and maintained the global cut-off threshold. Based on the current damage and the global cut-off threshold, OPMET decided whether it activated the prune strategy to prune some unnecessary nodes to visit. Thus, it saved the computation time. OPMET also provided a prioritization strategy to order the enzymes to visit. The main idea is to push the good solutions to the top levels of the tree. Thus, more nodes would be pruned in the prune strategy. OPMET explored the search space dynamically. OPMET guaranteed an optimal solution with two filtering strategies, prune strategy and prioritization strategy. They compute an upper bound to the number of target compounds eliminated and a lower bound to the side-effect respectively. However, OPMET is an exponential method in the worst case. It can not deal with the large network e.g. more than 32 enzymes in the network.

For the enzymatic target identification problem using linear model, integer linear programming is a kind of approach to tackle this problem. OptKnock is one example for this approach [9]. These strategies simulate the metabolism using Flux Balance Analysis,
At a high level, they represent each flux as a variable and solve a linear equation with linear constraints on these variables as follows.

Maximize (or minimize) Objective function / Subject to steady state constraints

This formulation represents the metabolism using a stoichiometric matrix $S$ \[116\] where the rows and the columns correspond to compounds and fluxes respectively. Assume that $x = [x_1, x_2, \cdots, x_n]'$ denotes the flux vector for a network with $n$ fluxes. The objective function is typically to maximize a variable or a linear combination of a set of variables. Thus an objective function is typically $\sum_i c_i x_i$ where $c_i$ are given constants. The constraints define the steady state using stoichiometric model. The solution to the equation $Sx = 0$ is the set of all steady states in this model. Assume that $y = [y_1, y_2, \cdots, y_m]'$ denotes the vector for enzyme activity. $y_j$ is a binary variable which is equal to 0 if an enzyme is knocked out and 1 else. Then, there exists the constrains that $\mu_j^{\text{min}} y_j \leq x_j \leq \mu_j^{\text{max}} y_j$, where $\mu_j^{\text{max}}$ and $\mu_j^{\text{min}}$ is the maximum and minimum possible flow corresponding to flux $j$, which express that whether the enzyme $y_j$ catalyzes the reaction $x_j$.

There exist several non-linear models to express the metabolism. For example, S-systems \[85, 86, 107\] defines the steady state as the solution to the equation system

$$\dot{X}_i = \alpha_i \prod_j X_j^{g_{ij}} - \beta_i \prod_j X_j^{h_{ij}} = 0, \forall i.$$

Here, the variable $X_i$ represents the concentration of the $i$th molecule. $\dot{X}_i$ is the derivative of $X_i$. The constants $\alpha_i$, $\beta_i$ and $g_{ij}$, $h_{ij}$ denote the rate of the reaction and the rate at which each molecule contributes to a reaction. Clearly, the constraints are non-linear in the S-systems of equations. Taking the logarithm of the constraints linearizes the constraints as follows. Define $Y_i = \log X_i$. The constraints become

$$\log \alpha_i + \sum_j Y_j g_{ij} - (\log \beta_i + \sum_j Y_j h_{ij}) = 0, \forall i.$$
GMA model [15, 43, 73, 107] is another non-linear model. This model considers each reaction that a compound is a part of separately and represents the steady state using the following equations

\[ \dot{X}_i = \sum_h \gamma_{ih} \prod_j X_j^{f_{ij}} = 0, \forall i. \]

Here, the constants \( \gamma_{ih} \) and \( f_{ij} \) denote the rate of the reaction and the rate at which each molecule contributes to a reaction. In this model, not only the constraints are non-linear, but also the summation of the multiplicative terms make it impossible to take the logarithm of the constraints. Because these models are non-linear, we can not use the existing linear methods for these non-linear models.

Most of the existing methods are suited well for linear models or Boolean model. Thus they do not work when these more complex models are used to compute the steady state of the metabolic network.

Klamt et al. introduced a minimal cut set problem, which aimed to find a minimal set of reactions whose deletion leads to no feasible balanced flux distribution in the objective reaction [54]. The authors described several algorithms to solve the minimal cut set problem. The aim of this model is to block the objective reaction function which results in the removal of the objective metabolite synthesis. It can not be used when the aim is to partly decrease or increase the objective metabolites.

Extreme Pathway Analysis [78] uses FBA to find the path in a pathway that maximizes or minimizes the production of a given compound. This problem is similar to a special case of the enzyme target identification problem considered in this chapter. De et al. for example, consider the extreme pathway analysis problem [18]. In order to reduce the yield of a compound in a pathway, De et al. use FBA to compute the optimal pathway so that the yield of the target metabolite is minimum. They, then, change the concentration of the enzymes in other paths so that these paths are inactive except that optimal one. This method has two major drawbacks.
First, it requires changing the concentration of many enzymes. In practice, changing the enzyme concentration is a costly process. Therefore, the number of enzymes whose concentrations are altered should be kept low. Second, the alterations that change the production of a compound can affect the production of other compounds in that pathway. Thus, the solution found by this method can have significant side-effects. In addition to these drawbacks, extreme pathway analysis cannot solve the enzymatic target identification problem for non-linear model.

Patil et al presented an evolutionary programming method for finding optimal gene deletion strategies [72]. Their method generates a population of random solutions and use genetic algorithm to improve this population. Their method can be applied to non-linear models as well as linear models. However, it has several drawbacks. The enzymatic target identification problem looks for a set of enzymes that are connected over a complex network and interact through reactions over compounds. Patil et al’s method ignores these interactions while constructing the population of solutions as well as creating new generation of solutions using crossover. They instead create these solutions randomly. The search space of the enzyme target identification problem is exponential in the number of enzymes. As a result, their method fails to converge to a good solution. Furthermore, the solutions found by their method suggests knocking out unnecessarily large number of enzymes.

All the above mentioned methods consider only the steady state of the metabolism. They ignore the sequence of states the underlying network visits while reaching the steady state. As a result, although their solution may be optimal at the steady state, the intermediate states of their solutions can be undesirable.

There are several models that simulate the dynamic state of a given metabolic network. For example, Dynamic Flux Balance Analysis (DFBA) extended the traditional FBA to describe the change rate of the fluxes over a period of time [62, 64]. DFBA incorporates the time parameter which can predict the metabolite concentrations. It considers the entire time period and builds a
non-linear programming problem. It separates the time into several intervals. For each interval it employs a linear-programming method to estimate the flux values during that interval. Integrated dynamic FBA (idFBA) simulates the integrated system including signaling, metabolic and regulatory networks [56]. Similar to DFBA, idFBA separates the time into several intervals. For each interval, it applies FBA to compute the flux values. From these values, it decides which reactions will take place during the next interval. Integrated FBA (iFBA) model builds a dynamic simulation among metabolic, regulatory and signaling networks [16] along the same lines as DFBA and idFBA. It first separates the time to several intervals. It then applies ordinary differential equations (ODEs) [75] and Boolean regulatory model to constrain the FBA linear programming problem. It updates the biomass and external metabolite concentrations for use in subsequent time steps. All these methods aim to find the dynamic state of a given metabolic network. They however do not consider the dynamic enzymatic target identification problem, which is the focus of Chapter 5.

For compound selection problem, one of the popular methods is to screen libraries of small compounds for their ability to bind to biological targets such as receptors and enzymes [38]. This process is also known as “docking”. Docking algorithms estimate how two molecules can bind with each other to form a stable complex [59]. DOCK [26], Glide [30] and GOLD [41] are a few examples of existing docking software. These tools predict the binding affinity between each small molecule and the target protein. Once the docking software computes the affinity of each small molecule in a library of molecules, the next step is often to pick the ones that have high predicted affinity values and test them in the lab. This process has been successful in several applications. For example, Lyne et al used FlexX-Pharm [34] docking software to search about 200,000 compounds and identified four novel classes of inhibitor for Chk1 kinase [63]. Kellenberger et al. searched about 44,000 compounds by docking software, GOLD and Surflex [40, 49], and found novel non-peptide ligands for GPCR CCR5 [50].
1.3 Overview of the Contributions

The following briefly shows our main contributions in this thesis.

**Enzymatic target identification using Boolean network models.** In this model, each compound has boolean state, present or not. The goal is to identify the set of enzymes whose knockouts eliminate all the “targeted compounds” while incurring minimum “damage”. Targeted compounds are the ones whose productions need to be stopped. They have defined “damage” as the number of non-targeted compounds that are eliminated because of knockouts. Minimum damage is the minimum number of non-targeted compounds eliminated from the metabolism while eliminating the targeted compounds among all possible ways of eliminating the targeted compounds. Therefore, this model considers the toxicity while traditional drug development approaches have often focused more on the efficacy of drugs than toxicity.

**Contribution:** We developed a scalable iterative method which computes a sub-optimal solution within reasonable time-bounds. The method consists of two phases: Iteration and Fusion Phases. The experiments on the *E. coli* metabolic network show that the average accuracy of the Iteration Phase alone deviates from that of the exhaustive search only by 0.02 %. The Iteration Phase is highly scalable. It can solve the problem for the entire metabolic network of *Escherichia coli* in less than 10 seconds. The Fusion Phase improves the accuracy of the Iteration Phase by 19.3%.

**Enzymatic target identification using linear model for multiple enzymes catalyze the same reaction** Flux Balance Analysis, (FBA) [7, 29, 48] is a popular linear model to describe the metabolism as linear programming equations. Integer linear programming is a popular method to tackle the enzymatic target identification problem. OptKnock is one example to the algorithms in this class [9]. At a high level, they represent each flux as a variable and solve a linear equation with linear constraints on these variables as follows.

Maximize (or minimize) Objective function
Subject to steady state constraints

However, Optknock cannot deal with the situation that the reaction is catalyzed by multiple enzymes not only one. In fact, multiple enzymes may carry out the same reaction. For example, substitute enzymes denote that only one of these enzymes need to be present for the reaction to occur. There exists “OR” associations among these enzymes. Collaborate enzymes denote that all these enzymes have to be expressed for the reaction to occur. There exists “AND” associations among these enzymes. Moreover, the reaction is catalyzed by a set of enzymes whose association is a combination of “OR” and “AND”. Therefore, it is necessary to study how to apply the linear programming method to the multiple enzyme association.

Contribution: We prove that finding the enzyme knockout strategy by OptKnock framework is NP-hard. It is consistent with the experiment results that when the network size increases, the running time of OptKnock framework increases exponentially. We provide a binary method and continuous method for “AND” and “OR” association. Experiments show that the enzyme association influence the performance of linear programming method very much. We observe that our binary method runs much faster than continuous method. For the pathways of *H.sapiens* from KEGG, our binary method runs less than one second for the whole metabolism. Therefore, our binary method is useful for the biological application.

Enzymatic target identification using non-linear model by manipulating the steady state. The “state” of a metabolic pathway can be expressed as a vector, which denotes the yield of the compounds [106] or the flux [71] in the pathway at a given time. Yield of a compound is the amount of product obtained in the chemical reaction [106]. The flux of a reaction is the rate at which each compound is produced or consumed by that reaction [71]. “Steady state” is the state that remains unchanged over time. The “Enzymatic Target Identification Problem” in this model, aims to identify the set of enzymes whose knockouts lead to a steady state of the metabolic pathway that is as close to a user supplied goal state as possible.
**Contribution:** We develop two algorithms to find the enzyme set with minimal deviation from the goal state. The first one is a traversal approach that explores possible solutions in a systematic way using a branch and bound method. The second one uses genetic algorithms to derive good solutions from a set of alternative solutions iteratively. Unlike the former one, this one can run for very large pathways. Our experiments show that our algorithms’ results follow those obtained in vitro in the literature from a number of applications. They also show that the traversal method is a good approximation of the exhaustive search algorithm and it is up to 11 times faster than the exhaustive one. This algorithm runs efficiently for pathways with up to 30 enzymes. For large pathways, our genetic algorithm can find good solutions in less than 10 minutes.

**Enzymatic target identification with dynamic states.** The flux of a metabolic network changes from one steady state to another steady state due to presence of external inhibitions. The sequence of intermediate states, called the “dynamic path”, shows the pattern that the underlying network follows to reach the steady state. Understanding this pattern is crucial for metabolic engineering as some of the intermediate states can be undesirable.

**Contribution:** We consider the problem of enzymatic target identification in metabolic networks. Unlike existing strategies, we consider the dynamic behavior of the state changes of the networks. Given a goal pattern for the fluxes of a given network, we aim to find the set of enzyme knockouts that will produce a dynamic state similar to that goal pattern. We consider three distance functions to measure the difference between two dynamic states. These are the Euclidean distance, timewarping distance and pattern distance. Euclidean distance restricts the solution space to the exact goal state. Time-warping distance allows for stretching of the goal pattern in the time domain. Pattern distance allows scaling and shifting of the goal flux in addition to stretching in the time domain. We provide a branch and bound method to solve this problem. We also develop a partitioning strategy to reduce the running time of our method.
This strategy avoids constructing the entire dynamic state by computing a lower bound to the distance between two dynamic states when the entire dynamic state is not available. Our experiments on the Purine metabolism show that our method runs accurately. They also show that our partitioning strategy reduces the number of time intervals computed for dynamic states by a factor of 2 to 6.

**Integrating structural properties of proteins and biological networks for compound selection.** The process of in silico compound selection – finding a new candidate drug from large libraries of compounds by computer aid, plays a significant role in modern drug discovery. One of the popular compound selection methods is to screen libraries of compounds for their ability to bind to biological targets such as receptors and enzymes. This process is also known as “docking”. Recent validation studies show that docking methods have a poor performance in compound selection.

**Contribution:** We develop two novel computational methods that rank a given set of compounds for a given target protein or enzyme. The major difference between our first method and traditional in-silico screening methods is that we consider additional proteins and enzymes while ranking compounds whereas existing strategies often focus only on the target protein alone. A drug compound can alter the state of the metabolic network. Our second method considers the impact of the drug compounds on the metabolic network by integrating the interactions among proteins in metabolic networks with the docking results. Experiments on the pharmacologic chaperones of misfolded rhodopsin show that our method has better accuracy than the traditional methods that focus only on rhodopsin. Our results are in the top 5.7% of all possible rankings. For the same dataset, the traditional method’s results are in the top 81% of all possible rankings.

### 1.4 Outline of the Thesis

An introduction to sketches and methods to metabolic engineering is presented in Chapter 1. Chapter 2 provides heuristic methods for enzymatic target identification by Boolean model.
Chapter 3 discusses enzymatic target identification on multiple enzymes association by linear model. Chapter 4 designs methods for enzymatic target identification by manipulating the steady state. Chapter 5 discusses the enzymatic identification problem with dynamic states. Chapter 6 presents a compound selection method integrating structural properties of proteins and biological networks. Chapter 7 contains the conclusions.

Figure 1-1. A structure of this thesis
Figure 1-2. Graph representation of a metabolic pathway with four reactions $R_1$, $R_2$, $R_3$ and $R_4$, three enzymes $E_1$, $E_2$ and $E_3$, and five compounds $C_1, \cdots, C_5$. 
CHAPTER 2
ENZYMATIC TARGET IDENTIFICATION USING BOOLEAN MODEL BY HEURISTIC METHODS

Drug discovery aims finding molecules that manipulate enzymes in order to increase or decrease the production of desired compounds while incurring minimum side-effects. An important part of this problem is the identification of the target enzymes, i.e., the enzymes that will be inhibited by the drug molecules. Finding the right set of target enzymes is essential for developing a successful drug. The relationship between enzymes and compounds through reactions is defined using metabolic networks. Finding the best set of target enzymes requires a careful analysis of the underlying metabolic network.

This chapter presents the problem of finding the set of enzymes, whose inhibition stops the production of a given set of target compounds, while eliminating minimal number of non-target compounds. Here, target compounds are the compounds whose presence cause the underlying disorder. The non-target compounds are all the remaining compounds. We call this problem “Target Identification by Enzymes (TIE)”. An exhaustive evaluation of all possible enzyme combinations in the metabolic network to find the optimal solution is computationally infeasible for very large metabolic networks. We developed a scalable iterative method which computes a sub-optimal solution within reasonable time-bounds. The method consists of two phases: Iteration and Fusion Phases. The Iteration Phase is based on the intuition that a good solution can be found by tracing backward from the target compounds. It initially evaluates the immediate precursors of the target compounds and iteratively moves backwards to identify the enzymes whose inhibition incurs less side-effects. This phase converges to a sub-optimal solution after a small number of iterations. The Fusion Phase takes the union of a set of sub-optimal results found at the Iteration Phase. Each set, here, is a potential solution. It then increases this set by inserting a small subset of the remaining enzymes randomly. The size of the final set is bounded by the time allowed for the exhaustive search. The Fusion Phase exhaustively searches the final
set to find the optimal subset of enzymes from this set. It, then, recursively creates a new set by inserting random enzymes to the optimal solution found so far and exhaustively searches this set again until a predefined number of iterations are performed.

The experiments on the *E. coli* metabolic network show that the average accuracy of the Iteration Phase alone deviates from that of the exhaustive search only by 0.02%. The Iteration Phase is highly scalable. It can solve the problem for the entire metabolic network of *Escherichia coli* in less than 10 seconds. The Fusion Phase improves the accuracy of the Iteration Phase by 19.3%.

### 2.1 Motivation and Problem Definition

Traditional drug development approaches have often focused more on the efficacy of drugs than their toxicity (untoward side effects). Lack of predictive models that account for the inter-relationships between the metabolic processes often leads to drug development failures. Toxicity and/or lack of efficacy can result if metabolic network components other than the intended target are affected. This is well-illustrated by the failure of Tolcapone (a new drug developed for Parkinson’s disease) due to observed hepatic toxicity in some patients [19]. Post-genomic drug research focuses more on the identification of specific biological targets (gene products, such as enzymes or proteins) for drugs, which can be manipulated to produce the desired effect (of curing a disease) with minimum disruptive side-effects [92, 100]. The main phases in such a drug development strategy are target identification, validation and lead inhibitor identification [21].

Enzymes catalyze reactions, which produce metabolites (compounds) in the metabolic networks of organisms. Enzyme malfunctions that result in the accumulation of a set of compounds may cause diseases. We term such compounds as the “Target Compounds” and the remaining compounds as the “Non-Target compounds”. For instance, the malfunction of the enzyme *phenylalanine hydroxylase* causes buildup of the amino acid, phenylalanine, resulting in phenylketonuria [99], a disease that causes mental retardation. Hence, it is essential to identify
the optimal enzyme set that can be manipulated by drugs to prevent the excess production of
target compounds while incurring minimal side-effects. We term the side-effects of inhibiting
a given set of enzymes as the “damage” caused to the metabolic network. Formally, we define
“damage” of inhibiting a set of enzymes as the number of non-target compounds whose pro-
ductions are stopped due to the inhibition of those enzymes. Note that, it is trivial to extend this
definition to non-uniform costs for different compounds. We use the uniform damage assumption
for different compounds in this chapter for simplicity.

**Problem statement.** Given a metabolic network and a set of target compounds, the problem of
“Target Identification by Enzymes (TIE)” seeks the set of enzymes whose inhibition eliminates
all the target compounds and inflicts minimum damage on the rest of the network.

Evaluating all enzyme combinations is not feasible for metabolic networks with large
number of enzymes. This is because the number of enzyme combinations, i.e., $2^{|E|} - 1$, increases
exponentially with the number of enzymes. Efficient and precise heuristics are needed to tackle
this problem.

Note that different enzymes and compounds may have varying levels of importance in the
metabolic network. Our model simplistically considers all the enzymes and compounds to be of
equal importance. It can be extended by assigning weights to enzymes and compounds based on
their roles in the network. However, we do not discuss these extensions in this chapter as these
extensions can be solved by making trivial modifications to the algorithm that solves the problem
described in this chapter.

### 2.2 Methods

#### 2.2.1 The Iteration Phase

This section presents the Iteration Phase of our method. Iteration phase quickly produces
a set of suboptimal solutions to the TIE problem. The Iteration Phase is based on the intuition
that we can arrive at a solution close to the optimal one by tracing the metabolic network
backwards starting from the target compounds. We evaluate the immediate precursors of the
target compounds and iteratively move backwards to identify the enzymes whose inhibition stops
the production of the target compounds while incurring minimum damage. This phase consists of
an initialization step followed by iterations, until some convergence criteria is satisfied.

Let $E$, $R$ and $C$ denote the sets of enzymes, reactions and compounds of the metabolic
network respectively. Let $|E|$, $|R|$ and $|C|$ denote the number of enzymes, reactions and com-
ounds respectively. The iteration phase uses three primary data structures, namely an “enzyme
vector” $V_E = [e_1, e_2, \ldots, e_{|E|}]$, a “reaction vector” $V_R = [r_1, r_2, \ldots, r_{|R|}]$, and a “compound
vector” $V_C = [c_1, c_2, \ldots, c_{|C|}]$. Each value, $e_i$, in $V_E$ denotes the damage of inhibition of enzyme,
$E_i \in E$. Each value, $r_i^j$, in $V_R$ denotes the damage incurred by stopping the reaction $R_i \in R$ in
$j$th iteration. Each value, $c_i^j$, in $V_C$ denotes the damage incurred by stopping the production of
the compound $C_i \in C$ in $j$th iteration. The terms $r_i^0$ and $c_i^0$ represent the values $r_i$ and $c_i$ at the
initialization step.

**Initialization:** The initialization step computes the initial values of $V_E$, $V_R$, and $V_C$. It first
computes the vector $V_E$, then $V_R$, and finally $V_C$. It computes each entry in these vectors by
simply eliminating the vertex corresponding to that entry from the metabolic network. For
instance, in order to compute the value of $e_1$, it removes the vertex corresponding to enzyme $E_1$
from the graph. In other words, it does not considering combinations of enzymes at this step. It
only aims to find an initial local solution. Next, we elaborate the initialization of vectors $V_E$, $V_R$, and $V_C$.

**Enzyme vector:** The value of $e_i$, $\forall i, 1 \leq i \leq |E|$, is computed as the damage incurred after
inhibiting $E_i$. This value is computed by doing a breadth-first traversal of the metabolic network
starting from $E_i$. The traversal is performed as follows. All the edges that are traversed are
removed from the graph. If a vertex visited during the traversal corresponds to a reaction, that
vertex is removed as well, and its outgoing edges are inserted into a queue to be traversed later.
If a vertex visited during the traversal corresponds to a compound, that vertex is removed only if it has no other incoming edges. If a compound vertex is removed, its outgoing edges are inserted into a queue to be traversed later. Otherwise, its outgoing edges are not traversed.

The damage $e_i$ associated with every enzyme $E_i \in E$, $1 \leq i \leq |E|$, is calculated separately and stored at position $i$ in the enzyme vector $V_E$.

**Reaction vector:** The damage $r^0_j$ is computed as the minimum of the damages of the enzymes that catalyze $R_j$, $\forall j, 1 \leq j \leq |R|$. Let $E_{\pi_1}, E_{\pi_2}, \ldots, E_{\pi_k}$ be the enzymes that catalyze $R_j$. The damage $r^0_j$ is computed as $r^0_j = \min_{i=1}^k \{e_{\pi_i}\}$. This computation is intuitive since a reaction can be disrupted by inhibiting any of its catalysts. The value of $r^0_j$ associated with every reaction $R_j \in R$, $1 \leq j \leq |R|$ is computed and stored at position $j$ in the reaction vector $V_R$. Let $E^0(R_j)$ denote the set of enzymes that produced the damage $r^0_j$. The set $E^0(R_j)$ is also stored along with $r^0_j$.

**Compound vector:** The damage $c^0_k$, $\forall k, 1 \leq k \leq |C|$, is computed by considering the reactions that produce $C_k$. Let $R_{\pi_1}, R_{\pi_2}, \ldots, R_{\pi_j}$ be the reactions that produce $C_k$. First, the set of enzymes $E^0(C_k)$ for $C_k$ is computed as $E^0(C_k) = E^0(R_{\pi_1}) \cup E^0(R_{\pi_2}) \cup \cdots \cup E^0(R_{\pi_j})$. Next, the damage $c^0_k$ is computed as the damage incurred after the inhibition of all the enzymes in $E^0(C_k)$. This computation is based on the observation that a compound disappears from the system only if all the reactions that produce it stop. The value of $c^0_k$ associated with every compound $C_k \in C$, $1 \leq k \leq |C|$ is calculated and stored at position $k$ in the compound vector $V_C$. The set $E^0(C_k)$ is also stored along with $c^0_k$.

Table 2-1 shows the iteration steps for Figure 2.4. $I_0$ is the initialization step; $I_1$ and $I_2$ are the iterations. $V_R$ and $V_C$ represent the damage values of reactions and compounds respectively computed at each iteration. $V_E = [3, 0, 0]$ in all iterations. $flag() = 1$ indicates that the inhibition of the enzymes corresponding to that vector entry in $E(R)$ or $E(C)$ eliminates all the target compounds. Column $I_0$ in Table 2-1 shows the initialization of the vectors for the
network in Figure 2.4. The damage $e_1$ of $E_1$ is three as inhibiting $E_1$ stops the production of three non-target compounds $C_2$, $C_3$ and $C_4$. Since the disruption of $E_2$ or $E_3$ alone does not stop the production of any non-target compound, their damage values are zero. Hence, $V_E = [3, 0, 0]$. The damage values for reactions are computed as the minimum of their catalyzers ($r_1^0 = r_2^0 = e_1$ and $r_3^0 = r_4^0 = e_2$). Hence, $V_R = [3, 3, 0, 0]$. The damage values for compounds are computed from the reactions that produce them. For instance, $R_1$ and $R_2$ produce $C_2$. $E^0(R_1) = E^0(R_2) = \{E_1\}$. Therefore, $c_2^0 = e_1$. Similarly $c_3^0$ is equal to the damage of inhibiting the set $E^0(R_3) \cup E^0(R_4) = \{E_2, E_3\}$. Thus, $c_3^0 = 1$.

**Iterative steps:** The iterations refine the damage values in vectors $V_R$ and $V_C$ by considering the precursors of the corresponding reaction and compound. Thus, at the $n$th iteration, the vertices from which a reaction or a compound vertex is reachable on a path of length up to $n$ are considered. In this chapter, “the length of a path” on the graph constructed for a metabolic network is defined as the number of reactions on that path.

**Definition 1.** In a given metabolic network, the “length of a path” from an enzyme $E_i$ to a reaction $R_j$ or compound $C_k$ is defined as the number of unique reactions on that path.

The enzyme vector, $V_E$, remains unchanged at the iteration step since the enzymes are not affected by the reactions or the compounds. Next, we describe the actions taken to update $V_R$ and $V_C$ at each iteration. We discuss the stopping criteria for the iterations later.

**Reaction vector:** Let $C_{\pi_1}, C_{\pi_2}, \ldots, C_{\pi_t}$ be the compounds that are consumed by $R_j$. We update the damage $r_j^n$ as

$$r_j^n = \min \{r_j^{n-1}, \min_{i=1}^{t} \{c_{\pi_i}^{n-1}\}\}.$$  

The first term of the “min” function is the damage value calculated for $R_j$ during the previous iteration (i.e., $(n - 1)$th iteration). The second term is the damage of the input compound with the minimum damage found in the previous iteration (i.e., $(n - 1)$th iteration). This computation is intuitive since a reaction can be disrupted by stopping the production of any of its input
compounds. The damage of all the input compounds are already computed in the \((n - 1)\)th iteration. Therefore, at the \(n\)th iteration, the second term of the “min” function considers the impact of the reactions and compounds that are away from \(R_j\) by \(n\) edges in the graph for the metabolic network. Let \(E^n(R_j)\) denote the set that contains the enzymes that produced the new damage \(r^n_j\). Along with \(r^n_j\), we also store \(E^n(R_j)\). We update all \(r^n_j \in V_R\) using the same strategy. Note that the values \(r^n_j\) can be computed in any order, i.e., the result does not depend on the order in which the reactions are considered.

**Compound vector:** The damage \(c^n_k\), \(\forall k, 1 \leq k \leq |C|\), is updated by considering the damage computed for \(C_k\) in the previous iteration and the damages of the reactions that produce \(C_k\). Let \(R_{\pi_1}, R_{\pi_2}, \ldots, R_{\pi_j}\) be the reactions that produce \(C_k\). We first compute a set of enzymes as \(E^{n-1}(R_{\pi_1}) \cup E^{n-1}(R_{\pi_2}) \cup \cdots \cup E^{n-1}(R_{\pi_j})\). Here, \(E^{n-1}(R_{\pi_t}), 1 \leq t \leq j\), is the set of enzymes computed for \(R_t\) after the reaction vector is updated in the current iteration. We update the damage value \(c^n_k\) as

\[
c^n_k = \min\{c^{n-1}_k, \text{damage}(\bigcup_{i=1}^{j} E^{n-1}(R_{\pi_i}))\}.
\]

The first term here denotes the damage value computed for \(C_k\) in the previous iteration. The second term shows the damage computed for all the precursor reactions in the current step. Along with \(c^n_k\), we also store \(E^n(C_k)\), the set of enzymes which provides the current minimum damage \(c^n_k\). Similar to the reaction vector, the entries \(c^n_k\) in the compound vector can be computed in any order as they do not depend on each other.

**Condition for convergence:** At each iteration, each value in \(V_R\) and \(V_C\) either remains the same or decreases by an integer amount. This is because each iteration applies a “min” function to update each value as the minimum of the current value and a function of its precursors. Therefore, the values of \(V_R\) and \(V_C\) do not increase. Furthermore, a damage value is always a nonnegative integer since it denotes the number of deleted non-target compounds. Iterative
refinement steps stop when the vectors \( V_R \) and \( V_C \) do not change in two consecutive iterations. This is justified, because, if these two vectors remain the same after an iteration, it implies that the damage values in \( V_R \) and \( V_C \) cannot be reduced any more using our refinement strategy.

Columns \( I_1 \) and \( I_2 \) in Table 2-1 show the iterative steps to update the values of the vectors \( V_R \) and \( V_C \). In \( I_1 \), we compute the damage \( r_1 \) for \( R_1 \) as the minimum of its current damage (three) and the damage of its precursor compound, \( c_3 = 1 \). Hence, \( r_1 \) is updated to 1 and its associated enzyme set is changed to \( \{E_2, E_3\} \). The other values in \( V_R \) remain the same. When we compute the values for \( V_C \), \( c_1 \) is updated to 1, as its new associated enzyme set is \( \{E_2, E_3\} \) and the damage of inhibiting both \( E_2 \) and \( E_3 \) together is 1. Hence, \( V_R = [1, 3, 0, 0] \) and \( V_C = [1, 3, 3, 3, 1] \). In \( I_2 \), we find that the values in \( V_R \) and \( V_C \) do not change anymore. Hence, we stop our iterative refinement and report the enzyme combination \( E_2, E_3 \) as the best solution observed in the iterative algorithm.

An interesting observation that follows the iterative algorithm is that it can produce multiple solutions to the TIE problem. This can be explained as follows. Each iteration produces at least one feasible solution (i.e., an enzyme set that eliminates all the targets). This is because each entry in the vector \( E^n(C) \) denotes a set of enzymes whose inhibition eliminates its corresponding compound, where \( n \) is the iteration number. The union of all the entries of \( E^n(C) \) corresponding to the target compounds is a feasible solution. In addition to this, each set \( E^n(R) \) or \( E^n(C) \) is a feasible solution if the inhibition of the enzymes in that set eliminates all the target compounds.

**Theorem 1.** Let \( V_R = [r_1, r_2, \ldots, r_{|R|}] \) and \( V_C = [c_1, c_2, \ldots, c_{|C|}] \) be the reaction and compound vectors respectively (see Section 2.2.1). Let \( n \) be the length of the preceding path (see Definitions 1) of reaction \( R_j \) (or compound \( C_k \)). The value \( r_j \) (or \( c_k \)) remains constant after at most \( n \) iterations. \( \square \)

Proof: See Song et al’s paper [94].

Complexity analysis is in the following.
Space Complexity: The number of elements in the reaction and compound vectors is $(|R| + |C|)$. For each element, we store an associated set of enzymes. Hence, the space complexity is $O((|R| + |C|) \times |E|)$.

Time Complexity: From Theorem 1, the number of iterations of the algorithm is $O(|R|)$. The computation time per iteration is $O(G \times (|R| + |C|))$, where $G$ is the size of the graph, $G = \text{number of vertices} + \text{number of edges}$. Hence, the time complexity is $O(|R| \times G \times (|R| + |C|))$.

2.2.2 The Fusion Phase

The Iteration Phase computes a set of feasible solutions to the TIE problem quickly. However, these results are not necessarily optimal. In other words, the enzyme set with minimum damage is not guaranteed to be in the solution set. Usually, it is desirable to spend more time if better results can be found. This section presents the Fusion Phase which optimizes the results of the Iteration Phase.

The Fusion Phase creates a subset of enzymes by taking the union of a number of sub-optimal results found at the Iteration Phase. Recall that each sub-optimal solution is a set of enzymes whose inhibition eliminates all the target compounds. It then grows this set by inserting a small number of the remaining enzymes. The Fusion Phase exhaustively searches the enzymes in this set to find the subset of enzymes from this set whose inhibition stops the production of all the target compounds and has the smallest damage. It, then, recursively creates a new set by inserting random enzymes from the upstream subnetwork of the target compounds to the optimal solution found so far and exhaustively searches this set again until a predefined number of iterations are performed. The complexity of the exhaustive search is exponential in the number of enzymes. Therefore, the size of the enzyme set at each step is determined by the amount of time the user is willing to spend on the exhaustive search.

Algorithm 2.1 presents a brief overview of the Fusion Phase. First, we select a set of sub-optimal results found in the Iteration Phase. Recall that, in the Iteration Phase, we record the
Algorithm 2.1 The Fusion Phase

**INPUT:**
- A metabolic network.
- $E$, $R$, $C$ the sets of enzymes, reactions and compounds in the metabolic network respectively.
- $\text{min}$ the minimum number of enzymes selected from the Iteration Phase
- $\text{max}$ the maximum number of enzymes that can be exhaustively searched
- $T$ a set of target compounds.

**OUTPUT:** A subset of enzymes, whose inhibition eliminates all the target compounds and inflicts minimum damage on the network.

Select the best sub-optimal results found in the Iteration Phase so that their union contains $K$ enzymes, where $\text{min} \leq K \leq \text{max}$. Store the union of these enzymes as the potential solution set $E'$.

For $i = 1$ to $m$ do

1. Grow the potential solution set $E'$ by inserting enzymes in $E'$ randomly from the upstream subnetwork of the target compounds until $E'$ contains $\text{max}$ enzymes.
2. Exhaustively search the enzymes in $E'$ to find the subset of enzymes in $E'$ that eliminates all the non-target compounds and has the smallest damage.
3. Update the potential solution set $E'$ as the set of enzymes in the result of the exhaustive search.

End for

Report $E'$ as the result.

---

candidate enzyme sets $E^n(R_k)$ and $E^n(C_k)$ in each iteration. Only the candidate sets that can inhibit all the target compounds are potential solutions. We compute a boolean value $\text{flag}^n(R_k)$ and $\text{flag}^n(C_k)$ to distinguish enzyme sets that are potential solutions from the ones that are not. In detail, if the inhibition of $E^n(R_k)$ eliminates all the target compounds, $\text{flag}^n(R_k) = 1$. Otherwise, $\text{flag}^n(R_k) = 0$. Similarly $\text{flag}^n(C_k) = 1$ only if $E^n(C_k)$ stops the production of all the target compounds.

We maintain a list of top candidate sets with the smallest damage through the iterations.

This list is empty prior to the Iteration Phase. At each iteration, we check the damage values in vectors $V_R$ and $V_C$ as well as the corresponding $\text{flag}^n(R_k)$ and $\text{flag}^n(C_k)$ values. These vectors produce candidate solutions in two different ways.
Case 1: Each entry $\text{flag}^n(R_k) = 1$ (or $\text{flag}^n(C_k) = 1$) indicates that $E^n(R_k)$ (or $E^n(C_k)$) is a candidate solution.

Case 2: The union of all the sets $E^n(C_k)$, where $C_k$ is a target compound, is a candidate solution even if $\text{flag}^n(C_k) = 0$ for some of the $C_k$.

The former case is correct since the flag indicates that the corresponding enzyme set eliminates all the targets. The latter case holds since each enzyme set $E(C_k)$ is guaranteed to eliminate compound $C_k$. Thus, their union is guaranteed to eliminate all the target compounds. At each iteration, we evaluate all the candidate solutions obtained at that iteration and insert the top solutions if there are fewer than $\min$ enzymes in the top candidates list. Otherwise, we replace the new solution with the existing solution in the list that has the largest damage if the damage of the new solution is less than the existing one. Table 2-1 shows the $\text{flag}()$ and the union of the top candidates in each iteration for Figure 2.4. $V_R$, $V_C$, $E(R)$ and $E(C)$ are the same as Table 2-1. In this example, $\min = 3$. In iteration $I_0$, the top solution is $E^0(C_5) = \{E_2, E_3\}$ since $\text{flag}^0(C_5) = 1$ and the damage value of $C_5$, $c^0_5$ is the smallest damage observed so far. Therefore, we initialize the top candidate list to the enzymes in $E^0(C_5)$. There are less than $\min$ enzymes in this list. Therefore, we grow this list by inserting the next best solution in the solutions found so far until we have at least $\min$ enzymes or not other solution exists. All the remaining solutions at this iteration have the same damage value (i.e., damage = 3). Thus, we pick up one of them arbitrarily, say $E^0(C_1)$, and insert the enzymes in it into top candidate list.

Then we improve the current solution as follows. We grow the set $E'$ by inserting random enzymes from the upstream subnetwork of the target compounds until $E'$ contains $\max$ enzymes. The reason is that the optimal enzyme set must reside in the upstream subnetwork of the target compounds. Therefore, we only consider the enzymes in this subnetwork. Step 4 exhaustively searches the subsets of enzymes in the enzyme set $E'$ to find the optimal solution in $E'$. The algorithm repeats a prespecified number of times, and report the best result.
2.3 Results

Experimental setup. We extracted the metabolic network of *Escherichia coli* (*E. coli*) from KEGG [47] (ftp://ftp.genome.jp/pub/kegg/pathways/eco/). The metabolic network in KEGG has been hierarchically classified into smaller networks according to their functionalities. We performed experiments at different levels of hierarchy of the metabolic network including the entire metabolic network, that is an aggregation of all the functional subnetworks. We devised a uniform labeling scheme for the networks based on the number of enzymes they contain. According to this scheme, a network label begins with ‘N’ and is followed by the number of enzymes in the network. For instance, ‘N20’ indicates a network with 20 enzymes. Table 4-3 shows the metabolic networks used in our experiments along with their identifiers and the number of compounds (C), reactions (R) and edges (Ed). The networks are downloaded from the KEGG database. Id, C, R and Ed denote the network identifier used in this chapter, the number of compounds, reactions and edges (interactions) respectively. The edges represent the interactions in the network. For each network, we constructed three query sets. Each query in these sets consists of one, two and four target compounds respectively. For each network in our dataset, we selected target compounds in the query sets randomly among the compounds in that network. Each query set contains 10 queries each.

We implemented Iteration Phase, the Fusion Phase and an exhaustive search algorithm [96] which determines the optimal enzyme combination. We implemented the Iteration Phase and the exhaustive search algorithms in Java. We implemented the Fusion Phase in C. We ran our experiments on an Intel Pentium 4 processor with 2.8 GHz clock speed and 1-GB main memory, running Linux operating system.

2.3.1 Evaluation of the Damage Model on Real Drugs

We first evaluate how well the proposed cost model reflects the biological process. We do this by querying well studies drugs in the literature using double iterative optimization.
KEGG contains a database of known drug molecules along with the enzymes they inhibit and their therapeutic category. We use the drugs at this database as our benchmarks. Due to space limitation, we report only four of them. The value in parenthesis that starts with letter “D”, “C”, or “E” (e.g., D02562) is the unique identifier assigned to the corresponding drug, compound, or enzyme respectively in KEGG.

Benoxaprofen (D03080). This drug inhibits arachidonate 5-lipoxygenase (E1.13.11.34) which appears in several networks including arachidonic acid metabolism network (hsa00590). In Pharmacology, 5-lipoxygenase inhibitors will decrease the biosynthesis of LTB4 (C02165), cysteinylcontaining leukotrienes LTC4 (C02166), LTD4 (C05951) and LTE4 (C05952). According to our graph model, the removal of 5-lipoxygenase eliminates three of these compounds LTB4, LTC4 and LTD4 in arachidonic acid metabolism network. Inhibition of this enzyme also eliminates five more compounds, namely 5(S)-HPETE (C05356), 5-HETE (C04805), LTA4 (C00909) and 20-OH-LTB4 (C04853). These compounds can be considered as damage in our model.

Running double iterative optimization with LTB4, LTC4, LTD4 and LTE4 as the target compound finds LTA4H (E3.3.2.6) and LTC4 synthase (E4.4.1.20) as the optimal enzyme set. The inhibition of these enzymes eliminates only one non-target compound, 20-OH-LTB4 (C04853). Double iterative optimization potentially finds a better solution in this experiment than the existing drug as the same compound is eliminated by the existing drug in addition to four other compounds. Indeed, recent research supports our model since the anti-inflammatory effect of the levels of LTA4H [81] and LTC4 [101] have been observed.

Rasagiline (D02562). This is an antiparkinsonian drug. It inhibits amine oxidase (E.1.4.3.4) which appears in several metabolic networks. In the histidine metabolism network (hsa00340), the removal of amine oxidase eliminates the compounds Methylimidazole acetaldehyde (C05827) Methylimidazoleacetic acid (C05828) according to our graph model. Levels of
pros-methylimidazoleacetic acid has correlation with severity of Parkinson’s disease in patients [5, 76]. This demonstrates that, our model can predict the intended target well. When double iterative optimization is run on the same network with methylimidazoleacetic acid and the methylimidazole acetaldehyde as the target compounds it finds amine oxidase as the optimal target. This implies that Rasgiline is targeting the optimal enzyme according to our model.

For Ozagrel (D01683) and Erythromycin acistrate (D02523), running double iterative optimization can find the same target enzyme as the actual drug. (details omitted)

2.3.2 Evaluation of the Iteration Phase

Evaluation of accuracy: In order to evaluate the accuracy of the Fusion Phase, we compared the damage value of the best result found at this phase with that of the exhaustive algorithm. Note that the exhaustive algorithm guarantees to find the optimal result since it considers all possibilities. Table 2-3 shows the results. We present the results only up to 32 enzyme networks (i.e., $N_{32}$), for the exhaustive search algorithm took longer than one day to finish even for larger networks. We can see that the damage values of our method exactly match the damage values of the exhaustive search for all the networks except $N_{24}$. For $N_{24}$, the average damage differs from that of the exhaustive solution by only 0.02%. This shows that Iteration Phase is a good approximation of the exhaustive search algorithm which computes an optimal solution. The slight deviation in damage is the tradeoff for achieving the scalability of the Iteration Phase (described next).

Evaluation of scalability: Figure 2-2 (a) plots the average execution time of our Iteration Phase for increasing sizes of metabolic networks. The running time increases slowly with the network size. As the number of enzymes increases from 8 to 537, the running time increases from roughly 1 to 10 seconds. The largest network, $N_{537}$, consists of 537 enzymes, and hence, an exhaustive evaluation inspects $2^{537} - 1$ combinations (which is computationally infeasible). Thus, our results show that the Iteration Phase scales well for networks of increasing sizes. This
property makes our method an important tool for identifying the right enzyme combination for eliminating target compounds, especially for those networks for which an exhaustive search is not feasible.

Figure 2-2 (b) shows the plot of the average number of iterations for increasing sizes of metabolic networks. The Iteration Phase reaches to a steady state within 10 iterations in all cases. The various parameters (see Table 4-3) that influence the number of iterations are the number of enzymes, compounds, reactions and especially the number of interactions in the network (represented by edges in the network graph). Larger number of interactions increase the number of iterations considerably, as can be seen for networks $N_{22}$, $N_{48}$, $N_{96}$, $N_{537}$, where the number of interactions is greater than 5. This shows that, in addition to the number of enzymes, the number of compounds and reactions in the network and their interactions also play a significant role in determining the number of iterations. Our results show that the Iteration Phase can reliably reach a steady state and terminate, for networks as large as the entire metabolic network of *E. coli*.

### 2.3.3 Evaluation of the Fusion Phase

This section evaluates how the accuracy and the running time of the proposed method changes when the Fusion Phase is run after the Iteration Phase. We use the entire network, $N_{537}$, in this experiment. We ran 83 queries, where each query contains one, two, or four randomly chosen compounds as the target compounds. We limited the number of iterations of the Fusion Phase to six since we did not observe any improvement in accuracy after that point in our experiments. The number of enzymes taken from the Iteration Phase is no more than $\min = 25$. We use $\max = 32$ as the maximum number enzymes to be searched by the exhaustive search.

**Evaluation of accuracy.** In 16 out of 83 queries, the Fusion Phase improved the result found by the Iteration Phase. The results remained unchanged for the remaining queries. Table 2-4 lists the queries for which the results have improved. Here, the identifier for a compound is the unique identifier assigned to that compound in the KEGG database. According to the definition
of TIE, the smaller damage, the better result. For example, when the target compounds were C00048 (Glyoxylate) and C00010 (CoA), the Iteration Phase could find an enzyme subset whose damage value is 20, while the Fusion Phase improved this solution to the damage value of 16. In addition, if the target compounds are C00255 (Riboflavin) and C00536 (Triphosphate), the enzyme subset that the Fusion Phase gets is \{E2.5.1.9, E2.7.1.26, E2.7.4.1\}. According to our graph model, the removal of these there enzymes eliminates six non-target compounds, C00013, C00009, C04332, C04732, C00061 and C00016. However, the Iteration Phase finds an enzyme subset \{E2.5.1.9, E2.7.4.1, E3.1.3.2\} which eliminates one more non-target compound, namely C00147 (Adenine), besides the above six compounds. Adenine is a purine with a variety of roles in biochemistry including cellular respiration. Thus, if we eliminate Adenine, some important biological functions would be influenced. Therefore, the Fusion Phase could find better results than the Iterative Phase. On the average, the Fusion Phase improved the accuracy of the Iteration Phase by \(\frac{16}{83} = 19.3\%\).

**Evaluation of the running time.** The average execution time of the Fusion Phase is 432 seconds which is longer than that of the Iteration Phase (10 seconds). This is because the Fusion Phase uses the Iteration Phase as a subroutine and performs the exhaustive search on a small set of enzymes. However, 432 seconds is reasonable small for biological computation. Biologists focus more on accurate results than execution time. This means that it is very useful and beneficial to employ the Fusion Phase as the tool to address the TIE problem on large sets of metabolic networks.

### 2.4 Discussion

Efficient computational strategies are needed to identify the enzymes (i.e., drug targets), whose inhibition will achieve the required effect of eliminating a given set of target compounds while incurring minimal side-effects. An exhaustive evaluation of all possible enzyme combinations to find the optimal subset is computationally infeasible for large metabolic networks. We
proposed a double iterative optimization method with two phases, the Iteration Phase and the Fusion Phase. The Iteration Phase is based on the intuition that a good solution can be found by tracing backward from the target compounds. It initially evaluates the immediate precursors of the target compounds and iteratively moves backwards to identify the enzymes whose inhibition incurs less side-effects. This phase converges to a sub-optimal solution after a small number of iterations. The Fusion Phase takes a set of sub-optimal results based on the Iteration Phase as the potential solution set. Then, we extend the potential solution set by randomly inserting some remaining enzymes and apply an exhaustive search on it recursively until a predefined number of iterations are performed. In our experiments on the *E. coli* metabolic network, the accuracy of a solution computed by the Iteration Phase deviated from that found by an exhaustive search only by 0.02%. Our Iteration Phase is highly scalable. It solved the problem for even the entire metabolic network of *E. coli* in less than 10 seconds. The Fusion Phase improved the damage value of the Iteration Phase by 19.3% spending only 432 seconds.

The problem addressed in this chapter introduces a number of exciting computational problems. First, the damage model can be improved to a continuous function where the quantity of each target and non-target compound is taken into consideratation after inhibition of the target enzymes. Second, the goal of the drug can include increasing some of the target compounds. Computable objective functions representing these problems need to be defined.

![A graph constructed for a metabolic network with four reactions $R_1, \cdots, R_4$, three enzymes $E_1, E_2$ and $E_3$, and five compounds $C_1, \cdots, C_5$. $C_1$ is the target compound.](image-url)
Figure 2-2. Evaluation of the Iteration Phase. (a) Average execution time in milliseconds. (b) Average number of iterations
Table 2-1. Iterative Steps for Figure iterEg.

<table>
<thead>
<tr>
<th></th>
<th>$I_0$</th>
<th>$I_1$</th>
<th>$I_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_E$</td>
<td>[3, 0, 0]</td>
<td>[3, 0, 0]</td>
<td>[3, 0, 0]</td>
</tr>
<tr>
<td>$V_R$</td>
<td>[3, 3, 0, 0]</td>
<td>[1, 3, 0, 0]</td>
<td>[1, 3, 0, 0]</td>
</tr>
<tr>
<td>$V_C$</td>
<td>[3, 3, 3, 3]</td>
<td>[1, 3, 3, 3]</td>
<td>[1, 3, 3, 3]</td>
</tr>
<tr>
<td>$E(R)$</td>
<td>${E_1}$, ${E_2}$, ${E_3}$</td>
<td>${E_1}$, ${E_2}$, ${E_3}$</td>
<td>${E_1}$, ${E_2}$, ${E_3}$</td>
</tr>
<tr>
<td>$E(C)$</td>
<td>${E_1}$, ${E_2}$, ${E_3}$</td>
<td>${E_1}$, ${E_2}$, ${E_3}$</td>
<td>${E_1}$, ${E_2}$, ${E_3}$</td>
</tr>
<tr>
<td>$flag(R)$</td>
<td>[1, 1, 0, 0]</td>
<td>[1, 1, 0, 0]</td>
<td>[1, 1, 0, 0]</td>
</tr>
<tr>
<td>$flag(C)$</td>
<td>[1, 1, 1, 1]</td>
<td>[1, 1, 1, 1]</td>
<td>[1, 1, 1, 1]</td>
</tr>
<tr>
<td>$E'$</td>
<td>${E_1, E_2, E_3}$</td>
<td>${E_1, E_2, E_3}$</td>
<td>${E_1, E_2, E_3}$</td>
</tr>
</tbody>
</table>

Table 2-2. Metabolic networks used in our experiments.

<table>
<thead>
<tr>
<th>Id</th>
<th>Metabolic Network</th>
<th>C</th>
<th>R</th>
<th>Ed</th>
</tr>
</thead>
<tbody>
<tr>
<td>N08</td>
<td>Polyketide biosynthesis</td>
<td>11</td>
<td>11</td>
<td>33</td>
</tr>
<tr>
<td>N13</td>
<td>Xenobiotics biodegradation</td>
<td>47</td>
<td>58</td>
<td>187</td>
</tr>
<tr>
<td>N14</td>
<td>Citrate or TCA cycle</td>
<td>21</td>
<td>35</td>
<td>125</td>
</tr>
<tr>
<td>N17</td>
<td>Galactose</td>
<td>38</td>
<td>50</td>
<td>172</td>
</tr>
<tr>
<td>N20</td>
<td>Pentose phosphate</td>
<td>26</td>
<td>37</td>
<td>129</td>
</tr>
<tr>
<td>N22</td>
<td>Glycan Biosynthesis</td>
<td>54</td>
<td>51</td>
<td>171</td>
</tr>
<tr>
<td>N24</td>
<td>Glycerolipid</td>
<td>32</td>
<td>49</td>
<td>160</td>
</tr>
<tr>
<td>N28</td>
<td>Glycine, serine and threonine</td>
<td>36</td>
<td>46</td>
<td>151</td>
</tr>
<tr>
<td>N32</td>
<td>Pyruvate</td>
<td>21</td>
<td>51</td>
<td>163</td>
</tr>
<tr>
<td>N42</td>
<td>Other amino acid</td>
<td>69</td>
<td>63</td>
<td>208</td>
</tr>
<tr>
<td>N48</td>
<td>Lipid</td>
<td>134</td>
<td>196</td>
<td>654</td>
</tr>
<tr>
<td>N52</td>
<td>Purine</td>
<td>67</td>
<td>128</td>
<td>404</td>
</tr>
<tr>
<td>N59</td>
<td>Energy</td>
<td>72</td>
<td>82</td>
<td>268</td>
</tr>
<tr>
<td>N71</td>
<td>Nucleotide</td>
<td>102</td>
<td>217</td>
<td>684</td>
</tr>
<tr>
<td>N96</td>
<td>Vitamins and Cofactors</td>
<td>145</td>
<td>175</td>
<td>550</td>
</tr>
<tr>
<td>N170</td>
<td>Amino acid</td>
<td>54</td>
<td>378</td>
<td>1210</td>
</tr>
<tr>
<td>N180</td>
<td>Carbohydrate</td>
<td>247</td>
<td>501</td>
<td>1659</td>
</tr>
<tr>
<td>N537</td>
<td>Entire Network</td>
<td>988</td>
<td>1790</td>
<td>5833</td>
</tr>
</tbody>
</table>

Table 2-3. Comparison of the average damage values of solutions determined by the Iteration Phase versus that determined by the exhaustive search algorithm.

<table>
<thead>
<tr>
<th>Pathway Id</th>
<th>N14</th>
<th>N17</th>
<th>N20</th>
<th>N24</th>
<th>N28</th>
<th>N32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative Damage</td>
<td>2.51</td>
<td>8.73</td>
<td>1.63</td>
<td>3.39</td>
<td>1.47</td>
<td>0.59</td>
</tr>
<tr>
<td>Exhaustive Damage</td>
<td>2.51</td>
<td>8.73</td>
<td>1.63</td>
<td>3.17</td>
<td>1.47</td>
<td>0.59</td>
</tr>
</tbody>
</table>
Table 2-4. Comparison of the damage values found by the Iteration Phase and the Fusion Phase focusing on the queries in which the Fusion Phase improved the results of the Iteration Phase.

<table>
<thead>
<tr>
<th>Target</th>
<th>Damage</th>
<th>Iteration Phase</th>
<th>Fusion Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00033, C00052</td>
<td>22</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>C00051, C00334</td>
<td>25</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>C00149, C00010</td>
<td>18</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>C00255, C00536</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>C00048, C00010</td>
<td>20</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>C00221, C00334</td>
<td>22</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>C00075, C00058</td>
<td>9</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>C00255, C00058</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>C04184, C00013, C00010, C00052</td>
<td>28</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>C00624, C00075, C00080, C01024</td>
<td>15</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>C00158, C00227, C00334, C00311</td>
<td>21</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>C00334, C00147, C00005, C00267</td>
<td>35</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>C00647, C00624, C04184, C00334</td>
<td>26</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>C00010, C00900, C00052, C05993</td>
<td>26</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>C00018, C00024, C01228, C00044</td>
<td>51</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>C00627, C00147, C00005, C00267</td>
<td>23</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td><strong>AVERAGE</strong></td>
<td><strong>22.25</strong></td>
<td><strong>20.06</strong></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 3
ENZYMATIC TARGET IDENTIFICATION USING LINEAR MODEL FOR MULTIPLE
ENZYMES CATALYZE THE SAME REACTION

Though there exist a lot of articles to study the enzyme knockout strategy, there are few of
these papers considering the enzyme association. This chapter presents the enzyme knockout
strategies on FBA which can deal with the situation that a reaction is catalyzed by multiple
enzymes. Considering the enzyme association, that is, “AND”, “OR” or a combination of them,
we provide a binary method and continuous method for each association. OptKnock is a popular
framework for the enzyme knockout strategy. We prove that finding the enzyme knockout strat-
egy by OptKnock framework is NP-hard. It is consistent with the experiment results that when
the network size increases, the running time of OptKnock framework increases exponentially.
Our experiment section shows that the enzyme association influence the performance of linear
programming method very much. We observe that our binary method runs much faster than con-
tinuous method. For the pathways of *H.sapiens* from KEGG, our binary method runs less than
one second for the whole metabolism. Therefore, our binary method is useful for the biological
application.

3.1 Motivation and Problem Definition

Enzymes play a significant role in metabolism. Enzymes catalyze the chemical reactions.
Reactions transform a set of substrates into products. In fact, multiple enzymes may carry out
the same reaction. For example, substitute enzymes denote that only one of these enzymes need
to be present for the reaction to occur. There exists “OR” associations among these enzymes.
Collaborate enzymes denote that all these enzymes have to be expressed for the reaction to occur.
There exists “AND” associations among these enzymes. Moreover, the reaction is catalyzed
by a set of enzymes whose association is a combination of “OR” and “AND”. Reed et al [82]
presented examples to describe the enzyme and reaction association, which shows in Figure 3-1.
For D-Xylose ABC Transporter, enzymes XylF, XylG and XylH catalyze the reaction XYLabc
with “AND” association, meaning that all have to be expressed for the reaction to occur. For Glyceraldehyde 3-Phosphate Dehydrogenase, enzymes GapC and GapA catalyze the reaction GAPD with “OR” association, meaning that only one of these enzymes need to be present for the reaction to occur.

Increasing or reducing the production of products in a metabolism is essential for many industrial applications such as cosmetics and food industry. One candidate method is to knockout a set of enzymes. When a set of enzymes are deleted, some reactions may not be active. As a result, the final products may be influenced.

In order to find the enzyme knockout strategy in silico, it is necessary to build the computational models for metabolism. Boolean network model is a simple way to describe the metabolism [94–96]. Each node in this model, only has two status, present or not. Flux balance analysis (FBA) is a popular model for metabolism [7, 29, 48] which bases on the linear programming. S-systems [86, 107] and Generalized Mass Action (GMA) [73, 107] models are another way to describe the metabolism which bases on the power-function representation.

Currently, the existing methods of finding the enzyme knockout strategy do not focus on multiple enzymes. Sridhar et al. and Song et al. considered the algorithms for finding the enzyme knockout strategy on boolean network models [94–96]. They used branch-and-bound and iterative techniques to obtain the enzyme knockout strategy. However, their methods can only be applied to the situation that each reaction is catalyzed by a single enzyme. OptKnock is a popular method to find the enzyme knockout strategy based on FBA model [9]. In fact, OptKnock provided a bilevel programming strategy. The first level optimized the bioengineering objective that is, which enzymes should be knocked out. The second level optimized the cellular objective, that is, how to distribute the flux in order to obtain the maximal chemical production targets. Similarly, OptKnock can not deal with the situation that the reaction is catalyzed by multiple enzymes not only one. For S-systems [86, 107] and Generalized Mass Action (GMA) [73, 107]
models, there are no corresponding enzyme knockout strategy presented so far. Therefore, it is necessary to study the enzyme knockout strategy when a reaction is catalyzed by multiple enzymes not only one.

**Contribution:** In this chapter, we first prove that finding the enzyme knockout strategy by OptKnock framework is NP-hard. It is consistent with the experiment results that when the network size increases, the running time of OptKnock framework increases exponentially.

Second, we overcome the limitation of OptKnock. We present an enzyme knockout strategy on FBA which can deal with the situation that a reaction is catalyzed by multiple enzymes. Considering the enzyme association, that is, “AND”, “OR” or a combination of them, we provide a binary method and continuous method for each association. For binary method, we introduce new binary variables to deal with the association. For continuous method, we do not add new binary variables but real variable for the association.

Our experiments using the synthetical and biological dataset validate that the enzyme topology influences the performance of the enzyme knockout strategy very much. The topology that a reaction may be catalyzed by multiple enzymes costs much time than the one that a reaction is catalyzed by a single enzyme. We observe that our binary method runs 60% to 1100% faster than continuous method.

The rest of this chapter is organized as follows. Section 3.2 proves that finding the enzyme knockout strategy by OptKnock framework is NP-hard. Section 3.3 describes the proposed methods for the enzyme topology that a reaction may be catalyzed by multiple enzymes. Section 5.5 discusses experimental results. Section 6.4 concludes this chapter.

### 3.2 Computational Complexity

OptKnock is a bilevel framework for finding the enzyme knockout strategy. It considers maximizing a bioengineering objective given a maximized cellular objective subject to constraints on the fluxes for a steady state metabolic network comprising a set $N = 1, \ldots, N$ of
metabolites, and a set $M = 1, \ldots, M$ of metabolic reactions fueled by a glucose substrate. This formulation is provided by Burgard et al [9].

The decision variables of OptKnock are given as follows:

- $v_j$: the flux of reaction $j$;
- $y_j$: binary variable which is equal to 0 if an enzyme is knocked out, and 1 else;
- $v_{chemical}$: yield corresponding to chemical;
- $v_{biomass}$: yield corresponding to biomass;
- $v_{pts}$: the uptake of glucose through the phosphotransferase system;
- $v_{glk} $: glucokinase;
- $v_{glk\_uptake}$: the basis glucose uptake scenario;
- $v_{atp}$: flux corresponding to ATP.
- $v_{atpm\_main}$: the non growth-associated ATP maintenance requirement.

Other relevant parameters used in this problem are:

- $S_{ij}$: stoichiometric matrix.
- $\mu_{target\_biomass}$: a minimum level of biomass production;
- $\mu_{\text{min}}^j$: minimum possible flow corresponding to flux $j$.
- $\mu_{\text{max}}^j$: maximum possible flow corresponding to flux $j$.
- $h_j$: cost of blocking enzyme $j$ corresponding to reaction $j$.
- $w_j$: weight corresponding to the value of flux $j$.

$\mu_{\text{min}}^j$ and $\mu_{\text{max}}^j$ are identified by minimizing and maximizing every reaction flux subject to the constraints from the OptKnock framework given below.

With these variables and parameters, the integer programming formulation for OptKnock with costs on the enzymes follows.

\[
\text{Maximize} \quad \sum_{j \in M} w_j v_j - \sum_{j \in M} h_j y_j \quad \text{[OptKnock]} \quad (3-1)
\]
subject to: 
\[ \sum_{j=0}^{M} S_{ij} v_j = 0, \quad i \in N \]  
\[ v_{pts} + v_{glk} = v_{uptake} \]  
\[ v_{atp} \geq v_{atp_{main}} \]  
\[ v_{biomass} = OPT_{Primal} \]  
\[ \mu_j^{\min} y_j \leq v_j \leq \mu_j^{\max} y_j \quad j \in M \]  
\[ \sum_{j \in M} y_j \leq K, \quad \forall j \in M \]  
\[ y_j \in \{0,1\} \quad \forall j \in M. \]  

To prove that finding the enzyme knockout strategy by OptKnock with fixed costs is NP-hard, we show that the uncapacitated fixed charge network flow problem, which is NP-hard, is a special case of the OptKnock with fixed costs.

Let \( G = (V,A) \) be a directed graph, where \( V \) is the set of nodes and \( A \) is the set of arcs. Each arc \((i,j)\) is associated with two costs: a fixed cost \( f_{ij} \) for selecting \((i,j)\) and a variable cost \( c_{ij} \) for routing flow on \((i,j)\). Then the uncapacitated fixed charge network flow problem (UFNF) is to find a set of arcs that allow a supply node to send resources to a set of demand nodes, such that the sum of fixed and variable costs are minimized. UFNF can be formulated by the following mixed-integer program:

\[
\min \sum_{(i,j) \in A} c_{ij} X_{ij} + \sum_{(i,j) \in F} f_{ij} Z_{ij} \tag{3-9}
\]

subject to: 
\[
\sum_{(i,k) \in A} X_{ik} - \sum_{(k,j) \in A} X_{kj} = \begin{cases} 
-\sum_{t \in T} d_t & \text{if } k=s; \\
d_k & \text{if } k \in T; \\
0 & \text{if } k \in V \setminus \{T \cup s\}. 
\end{cases} \tag{3-10}
\]
The objective function (3–15) minimizes the sum of the fixed costs associated with selecting arc \((i, j)\) and variable costs for sending flow through \((i, j)\). Constraints (3–10) are classical flow conservation constraints, while constraints (3–10) ensure that there cannot be any flow if \(Z_{ij}\) is 0, and the maximum flow can be at most \(\lambda\) is \(Z_{ij}\) is 1. Constraints (3–18) and (3–19).

**THEOREM:** Finding the enzyme knockout strategy by OptKnock is NP-Hard.

**PROOF:** Let’s rewrite Optknock by just considering the basic constraints given by (3–2), (3–6) and (3–8), i.e. by ignoring other constraints we investigate a special case of Optknock with costs on the enzymes in the objective function. In our metabolic pathway network graph \(G'\), we can define each metabolite \(i \in N\) as a node, and each reaction \(k \in M\) as an arc using metabolite \(i\) to produce metabolite \(j\), and Therefore in our graph \(G'\) we have \(M\) arcs corresponding to each reaction, and \(N\) nodes corresponding to each metabolite. Define a new variable \(x_{ij}\) as the flux corresponding to reaction \(k\) which uses metabolite \(i\) to produce metabolite \(j\). Let \(S\) be a set of source nodes (external metabolites that are imposed to the pathway), and \(T\) be the set of sink nodes (metabolites that is not going to be used within the pathway after produced). We also specialize Optknock more by considering that the flux for the source and sink metabolites in the metabolic pathway is given as a constant \(b_i\).

Thus we can write constraint (3–2) as

\[
\sum_{(i,j) \in M} S_{ij} x_{ij} = \begin{cases} 
-b_i & \text{if } i \in S; \\
0 & \text{if } i \in V \setminus (T \cup s). 
\end{cases}
\]
for each \( i \in N \).

Then we redefine the stoichiometric matrix \( S_{ij} \), say \( S'_{ij} \) as follows:

\[
S'_{ij} = \begin{cases} 
1 & \text{if } (i, j) \in M; \\
-1 & \text{if } (j, i) \in M; \\
0 & \text{otherwise}.
\end{cases}
\]

Note that \( S'_{ij} \) has entries 1, \(-1\), and 0, and thus is a special case of \( S_{ij} \). By using the stoichiometric matrix \( S'_{ij} \), the constraint (3–2) can be written as

\[
\sum_{(i,k) \in M} x_{ik} - \sum_{(k,j) \in M} x_{kj} = \begin{cases} 
-b_i & \text{if } i \in S; \\
b_i & \text{if } i \in T; \\
0 & \text{if } i \in V \setminus \{T \cup s\}.
\end{cases}
\]

for each \( i \in N \).

Define a variable \( z_{ij} \) for each variable \( y_k \), and costs \( c_{ij} \) and \( f_{ij} \) such that \( c_{ij} = -w_k \), \( f_{ij} = h_k \), and a constant \( \lambda \) as \( \lambda = \mu_{k}^{max} \), and set \( \mu_{k}^{min} = 0 \) for each reaction \( k \in M \), which is defined by the arc \((i,j)\). Then, (3–6) can be rewritten by

\[
0 \leq x_{ij} \leq \lambda z_{ij} \quad \forall (i, j) \in M
\]

Then, the following formulation is a special case of Optknock:

\[
\min \sum_{(i,j) \in M} c_{ij}x_{ij} + \sum_{(i,j) \in M} f_{ij}z_{ij} \quad (3–15)
\]
subject to: \[
\sum_{(i,k)\in M} x_{ik} - \sum_{(k,j)\in M} x_{kj} = \begin{cases}
-b_i & \text{if } i \in S; \\
b_i & \text{if } i \in T; \\
0 & \text{if } i \in V \setminus \{T \cup s\}
\end{cases} \quad i \in N \quad (3-16)
\]
\[
x_{ij} \leq \lambda z_{ij} \quad \forall (i,j) \in M \quad (3-17)
\]
\[
x_{ij} \geq 0 \quad \forall (i,j) \in M \quad (3-18)
\]
\[
z_{ij} \in \{0, 1\} \quad \forall (i,j) \in M. \quad (3-19)
\]

A special case of Optknock given above is a UFNF. Since UFNF is NP-Hard [67], Opt-Knock with fixed charges is also NP-Hard.

### 3.3 Methods

Based on stoichiometric model, the mass balance constraints are formulated as,

\[
S \cdot \overline{v} = 0. \quad (3-20)
\]

where \(S\) is the stoichiometric matrix and \(\overline{v}\) is a vector of flux reaction rates. The constraints are formulated based on the enzyme type.

#### 3.3.1 Single Enzyme

The reaction is catalyzed by a single enzyme. In this case there might be a one-to-one correspondence between enzymes and reactions as well as one enzyme might catalyze several reactions. In the Optknock formulation, it is assumed that each reaction is catalyzed by only one enzyme, and we enforce bound restrictions on each flux as a function of the enzymes, i.e. if an enzyme corresponding to a reaction is knocked out then that reaction cannot result any flux. On the other hand, lower bounds enforce that if there is a flux, then there should be an enzyme catalyzing the reaction and forcing a lower bound on the amount of flux produced by the related reaction. Note that in the Optknock enzymes are independent from each other.
The constraints enforcing bounds on the flux is given as follows:

\[ \mu_i^{\text{min}} T_i \leq v_i \leq \mu_i^{\text{max}} T_i. \]  \hspace{1cm} (3–21)

where \( T_i \) is the binary variable takes value 1 if an enzyme is knocked out, and 0 else.

### 3.3.2 Multiple Enzymes

In the presence of the multiple enzymes, one reaction might be catalyzed at least by one or more enzymes. In the following subsections, we investigate the correlation between enzymes catalyzing the reactions. The relation of the enzymes might be substitute, collaborative, both substitute and collaborative or independent from each other. We define a function \( F_i \) which represents the relationship between the enzymes. Then we can rewrite the bounding constraints on the fluxes as follows:

\[ \mu_i^{\text{min}} F_i(T_i) \leq v_i \leq \mu_i^{\text{max}} F_i(T_i). \]  \hspace{1cm} (3–22)

#### 3.3.2.1 Substitute enzyme

Let \( S(v_i) \) denote a set of enzymes corresponding to the reaction \( i \), which produces flux \( v_i \), and an enzyme, \( E_{ij} \) belongs to the set \( S(v_i) \), where \( j \) represents the number of the enzyme among all enzymes catalyzing reaction \( i \), and \(|S(v_i)| = n\). When the reaction \( i \) is catalyzed by a set of substitute enzymes in \( S(v_i) \), only one of these enzymes is necessary for the reaction to occur. Let \( F_i \) represent the relationship between the enzymes that can substitute each other. Then we can write \( F_i \) as follows:

\[ F_i = \max_{E_{ij} \in S(v_i)} \{E_{ij}\} = OR_{E_{ij} \in S(v_i)}(E_{ij}) \]

Then for the multiple enzyme case in which enzymes can substitute each other, we rewrite the constraint (3–22) as follows:

\[ \mu_i^{\text{min}} \max_{E_{ij} \in S(v_i)} \{E_{ij}\} \leq v_i \leq \mu_i^{\text{max}} \max_{E_{ij} \in S(v_i)} \{E_{ij}\}. \]  \hspace{1cm} (3–23)
Constraint (3–23) imposes nonlinearity to our integer program. Here we can resolve nonlinearity in our problem by performing a variable transformation, and obtain a completely linear integer program, which can be solved by commercial solvers such as CPLEX and AMPL. Our linearization technique considers lower and upper bounds separately.

We linearize lower bounding constraints given by the inequality $\mu_i^{\text{min}} \max_{E_{ij} \in S(v_i)} \{E_{ij}\} \leq v_i$ as follows,

$$\mu_i^{\text{min}} E_{ij} \leq v_i \quad \forall E_{ij} \in S(v_i) \quad (3–24)$$

Linearization of the upper bounding constraints given by the inequality $v_i \leq \mu_i^{\text{max}} \max_{E_{ij} \in S(v_i)} \{E_{ij}\}$ is more complicated compared to the lower bound. For the linearization, we consider two methods. In the first method we define $F_i$ as binary variables. Denoting this method as binary, we provide the following linear constraints representing the nonlinear constraint $v_i \leq \mu_i^{\text{max}} \max_{E_{ij} \in S(v_i)} \{E_{ij}\}$ as follows:

$$F_i \geq \frac{\sum_{j=1}^{n} E_{ij}}{n} \quad \forall i \quad (3–25a)$$

$$F_i \leq \sum E_{ij} \quad \forall i \quad (3–25b)$$

$$F_i \in \{0, 1\} \quad \forall i \quad (3–25c)$$

In the second method, we define $F_i$ as continuous variables, and denote this approach as continuous method. Then the upper bounding constraint $v_i \leq \mu_i^{\text{max}} \max_{E_{ij} \in S(v_i)} \{E_{ij}\}$ is represented by the following linear constraints:

$$F_i \leq \sum_{j=1}^{n} E_{ij} \quad \forall i \quad (3–26a)$$

$$F_i \leq 1 \quad \forall i \quad (3–26b)$$

$$F_i \geq E_{ij} \quad \forall i \quad (3–26c)$$
3.3.2.2 Collaborate enzyme

In the collaboration case, a reaction $i$ is catalyzed by a set of collaborate enzymes in $S(v_i)$, then all of these enzymes are necessary for the reaction to occur. Let $C_i$ represent the relationship between the enzymes that collaborate. Then we can write $C_i$ as follows:

$$C_i = \min_{E_{ij} \in S(v_i)} \{E_{ij}\} = \text{AND}_{E_{ij} \in S(v_i)}(E_{ij})$$

For the collaborate enzyme case, we can rewrite the constraint (3–22) as follows:

$$\mu_{i}^{\min} \min_{E_{ij} \in S(v_i)} \{E_{ij}\} \leq v_i \leq \mu_{i}^{\min} \min_{E_{ij} \in S(v_i)} \{E_{ij}\}.$$  \hspace{1cm} (3–27)

As we discussed in the substitute enzyme case, constraint (3–27) forces nonlinearity to our problem. Here we also consider a linearization technique in which we define new variables and provide us a linear integer program. As in the previous case, we consider lower and upper bounds separately.

We linearize upper bounding constraints given by the inequality $v_i \leq \mu_{i}^{\min} \max_{E_{ij} \in S(v_i)} \{E_{ij}\}$ as follows,

$$v_i \leq \mu_{i}^{\min} E_{ij} \quad \forall E_{ij} \in S(v_i)$$ \hspace{1cm} (3–28)

Now we consider the linearization of the lower bounding constraints given by the inequality $\mu_{i}^{\min} \min_{E_{ij} \in S(v_i)} \{E_{ij}\} \leq v_i$. Analogous to the substitute enzyme case, we consider both binary and continuous methods, where we define $F_i$ as a binary variable for the first and a continuous variable for the second method. For the binary method, we provide the following linear constraints representing the nonlinear constraint $\mu_{i}^{\min} \min_{E_{ij} \in S(v_i)} \{E_{ij}\} \leq v_i$ as follows:

$$F_i \leq \sum_{j=1}^{n} E_{ij} \quad \forall i$$ \hspace{1cm} (3–29a)

$$F_i \geq \sum_{j=1}^{n} E_{ij} - 1 \quad \forall i$$ \hspace{1cm} (3–29b)

$$F_i \in \{0, 1\} \quad \forall i$$ \hspace{1cm} (3–29c)
In the continuous method the lower bounding constraint \( \mu_i^{\min} \min_{E_{ij} \in S(v_i)} \{E_{ij}\} \leq v_i \) is represented by the following linear constraints:

\[
\begin{align*}
F_i & \leq E_{ij} \quad \forall i \quad (3-30a) \\
F_i & \geq \sum_{j=1}^{n} E_{ij} - (n - 1) \quad \forall i \quad (3-30b) \\
F_i & \geq 0 \quad \forall i \quad (3-30c)
\end{align*}
\]

### 3.3.2.3 Enzyme complex

In this case, the reaction is associated with a Boolean equation representing its dependency on the presence of one or multiple enzymes. For example, \( F_i = (E_{i1} \text{ AND } E_{i2}) \text{ OR } (E_{i3} \text{ AND } E_{i4}) \). In fact, all the boolean equations can transform into the following formation.

\[
F = (E_{11} \text{ AND } E_{12} \text{ AND } ... \text{ AND } E_{1i_1}) \text{ OR } (E_{21} \text{ AND } E_{22} \text{ AND } ... \text{ AND } E_{2i_1}) \text{ OR } \\
... \ (E_{n1} \text{ AND } E_{n2} \text{ AND } ... \text{ AND } E_{ni_1})
\]

Defining a new variable \( Z_i \) corresponding to each OR clause, we rewrite \( F \) as follows:

\[
F = Z_1 \text{ OR } Z_2 \text{ OR } ... \text{ Z}_3 \text{ ... } Z_n \quad (3-31)
\]

where

\[
Z_j = E_{j1} \text{ AND } E_{j2} \text{ AND } ... \text{ E}_{j2} ... \text{ E}_{jn} \quad (3-32)
\]

We linearize \( F = Z_1 \text{ OR } Z_2 \text{ OR } ... \text{ OR } Z_n \) and \( Z_j = E_{j1} \text{ AND } E_{j2} \text{ AND } ... \text{ AND } E_{jiz} \) by using the methods described in Section 3.3.2.1 and Section 3.3.2.2, respectively.
3.4 Results

In this section, we evaluate our algorithms on synthetical and biological datasets. We evaluate their performance quantitatively by using the running time in seconds.

3.4.1 Datasets

Synthetical data: We build our synthetical data to evaluate the performance of our methods by the following ways. We randomly generate ten networks for each test case. For example, for test case (25, 100), that is, the number of compounds is 25 and the number of reaction is 100. We create these networks which satisfy the power law distribution. That means, the probability of the number of reactions each compound joins decreases exponentially with the number of reactions. Similarly, the probability of the number of enzymes catalyzing the same reaction decreases exponentially with the number of enzymes. We evaluate the performance of our methods by running time.

Biological data: We use the metabolic pathway information of \textit{H.sapiens} from KEGG as the input dataset. We combine all the metabolic pathways as the whole metabolism. We apply our binary method to this metabolism. We evaluate the performance of our binary method by the running time in seconds.

3.4.2 Results

In this section, we evaluate our methods based on linear programming on synthetical and biological network. Section 3.4.2.1 describes the influence of the enzyme topology on the linear programming method. We compare our binary method with continuous method on the synthetical network in Section 3.4.2.2. Section 3.4.2.3 shows the biology application of our method.

3.4.2.1 Effect of single and multiple enzymes

This section, we evaluate the influence of the enzyme topology on the linear programming performance. We create a set of synthetical networks described in Section 3.4.1. These synthetical networks are separate into groups. For the first group, each reaction is catalyzed by only one
enzyme. For the second group, the topology of reactions and compounds are the exactly same as the former group of networks. The only difference between these two groups is that there are 40% of reactions are catalyzed by multiple enzymes in this new network. For multiple enzymes, the number of enzymes satisfies the power law distribution. That is, the probability of the number of enzymes catalyzing the same reaction decreases exponentially with the number of enzymes. The association of enzymes exists two cases. One is the substitute case, that is, there exists “OR” relationship among all these enzymes. One is the collaborate case, that is, these exists “AND” relationship among all these enzymes.

Figure 4-7 shows the running time between these two networks for binary method. The result shows that the multiple enzyme topology cost much more time than the single enzyme topology. Thus, the enzyme topology influences the performance of the linear programming algorithm very much. The running time of the multiple enzymes is 2 to 16 times that of the single enzyme. With the size of network increases, the running time increases exponentially. Therefore, the method based on linear programming is not practical for large networks.

3.4.2.2 Evaluation of binary method and continuous method

Section 3.3 provides two methods for the multiple enzyme knockout problem. In this section, we evaluate the performance of these two methods, binary method and continuous method by running time. Figure 3-3 shows the average running time for binary method and continuous method for the networks whose multiple enzymes are substitute ones. Binary method runs 60% to 1100% faster than continuous method. Similarly, in Figure 3-4, all the multiple enzymes are collaborate association. The binary method still runs much faster than continuous method.

3.4.2.3 Biological application

We use the metabolic pathway information of *H.sapiens* from KEGG as the input dataset. We combine all the metabolic pathways as the whole metabolism. Table 4-3 shows the running
time in seconds for the whole metabolism. #E, #R and #C denote the number of enzymes, reactions and compounds respectively in the metabolism. ‘AND’ means the association of all the multiple enzymes is AND. ‘OR’ means that the association of all the multiple enzymes is OR. From Table 4-3, our binary method runs less than one second for the whole metabolism. Therefore, our binary method is useful for the biological application.

### 3.5 Discussion

Though there exist a lot of articles to study the enzyme knockout strategy, there are few of these papers considering the enzyme association. This chapter presents the enzyme knockout strategies on FBA which can deal with the situation that a reaction is catalyzed by multiple enzymes. Considering the enzyme association, that is, “AND”, “OR” or a combination of them, we provide a binary method and continuous method for each association.

OptKnock is a popular framework for the enzyme knockout strategy. We prove that finding the enzyme knockout strategy by OptKnock framework is NP-hard. It is consistent with the experiment results that when the network size increases, the running time of OptKnock framework increases exponentially.

Our experiment section shows that the enzyme association influence the performance of linear programming method very much. We observe that our binary method runs much faster than continuous method. For the pathways of *H.sapiens* from KEGG, our binary method runs less than one second for the whole metabolism. Therefore, our binary method is useful for the biological application.

<table>
<thead>
<tr>
<th>KEGG</th>
<th>#E</th>
<th>#R</th>
<th>#C</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>whole metabolism</td>
<td>640</td>
<td>1176</td>
<td>1067</td>
<td>0.41</td>
<td>0.14</td>
</tr>
</tbody>
</table>
Figure 3-1. Redraw Protein-reaction associations by Reed et al.

Figure 3-2. The average running time (in seconds) for the networks with single enzyme and multiple enzymes (substitute and collaborate).
Figure 3-3. The average running time (in seconds) of binary and continuous methods for the networks whose multiple enzymes are substitute ones.

Figure 3-4. The average running time (in seconds) of binary and continuous methods for the networks whose multiple enzymes are collaborate.
Metabolic pathways show the complex interactions among enzymes that transform chemical compounds. The state of a metabolic pathway can be expressed as a vector, which denotes the yield of the compounds or the flux in that pathway at a given time. The steady state is a state that remains unchanged over time. Altering the state of the metabolism is very important for many applications such as biomedicine, bio-fuels, food industry and cosmetics. The goal of the enzymatic target identification problem is to identify the set of enzymes whose knockouts lead the metabolism to a state that is close to a given goal state. Given that the size of the search space is exponential in the number of enzymes, the target identification problem is very computationally intensive.

In this chapter, we develop efficient algorithms to solve the enzymatic target identification problem in this chapter. Unlike existing algorithms, our method works for a broad set of metabolic network models. We measure the effect of the knockouts of a set of enzymes as a function of the deviation of the steady state of the pathway after their knockouts from the goal state. We develop two algorithms to find the enzyme set with minimal deviation from the goal state. The first one is a traversal approach that explores possible solutions in a systematic way using a branch and bound method. The second one uses genetic algorithms to derive good solutions from a set of alternative solutions iteratively. Unlike the former one, this one can run for very large pathways.

Our experiments show that our algorithms’ results follow those obtained in vitro in the literature from a number of applications. They also show that the traversal method is a good approximation of the exhaustive search algorithm and it is up to 11 times faster than the exhaustive one. This algorithm runs efficiently for pathways with up to 30 enzymes. For large pathways, our genetic algorithm can find good solutions in less than 10 minutes.
4.1 Motivation and Problem Definition

Metabolic pathways are one of the most important data resources in biology. A metabolic pathway is a complex network of chemical reactions occurring within a cell. Enzymes catalyze these reactions. Reactions transform a set of compounds into another set of compounds. Note that, the term “network” is also used in the literature to denote the union of all pathways of an organism or large pathways. To keep the notation consistent, we will use the term pathway instead of network regardless of the pathway size in this chapter. The “state” of a metabolic pathway can be expressed as a vector, which denotes the yield of the compounds [106] or the flux [71] in the pathway at a given time. Yield of a compound is the amount of product obtained in the chemical reaction [106]. The flux of a reaction is the rate at which each compound is produced or consumed by that reaction [71]. “Steady state” is the state that remains unchanged over time.

Many applications require altering the steady state of a given pathway. For example, external factors or genetic mutations can change the production rate of a set of enzymes. They can even modify the structure of produced enzymes. Low or missing activity of an enzyme may result in the blockage of the pathway. Furthermore, this can propagate to other parts of the pathway that need the compounds produced in the blocked part of the pathway. As a result, the production of compounds may increase or decrease. Such aberrations in the state of the metabolism can lead to severe diseases. Examples include mental retardation, seizures, decreased muscle tone, organ failure and blindness [27, 89]. Thus, changing the state of the metabolism back to a desired level is often needed.

Increasing or reducing the production of certain compounds in a metabolism is essential for many industrial applications such as cosmetics and food industry. For example, fatty acid biosynthesis pathway converts fatty acids that are used in the cosmetic industry in creams and lotions [105, 115]. Butanoate metabolism produces poly-β-hydroxybutyrate which is essential for producing plastics [11]. Mevalonic acid pathway and MEP/DOXP pathway produce carotenoid
that are often used as anti-oxidant in food industry [83]. The metabolism of many organisms, such as bacteria, algae and plants naturally produce these compounds. A common practice is to extract such compounds from these organisms. By manipulating the pathways of these organisms, the production of these compounds can be increased significantly.

One way to change the state of a pathway close to the desired one is to knock out a set of enzymes. When an enzyme is knocked out, it cannot catalyze the reactions it is responsible from. As a result, some entries in the steady state of the pathway may increase and some may decrease. The “Enzymatic Target Identification Problem”, the problem addressed in this chapter, aims to identify the set of enzymes whose knockouts lead to a steady state of the metabolic pathway that is as close to a user supplied goal state as possible. In order to improve the generality of this problem, we allow the goal state to be transient as well as steady. In other words, it may be impossible to reach to the goal state exactly. However, there can still be steady states that are close to the goal state.

The size and the complex structure of the metabolic pathways along with the large size of the solution state space makes the enzymatic target identification problem computationally challenging. We can compute the steady state of a metabolic pathway after knocking out a given set of enzymes in polynomial time [107] (see the later section for a discussion of the steady state computation). However, enzymatic target identification aims to solve the inverse problem of finding the set of enzymes to achieve a steady state that is close to a given goal state. We have shown that a simplified version of the enzyme identification problem is an NP-complete problem [44]. Thus, exhaustive search is impractical for the typical pathways that contain hundreds of enzymes, thousands of compounds and reactions. Assuming that finding a steady state of a pathway takes on the order of 10 milliseconds, it will require nearly a year of computational time to test all solutions with up to four enzyme combinations on a metabolic pathway with 500 enzymes.
There are several algorithms in the literature which aim to address the enzymatic target identification problem. These algorithms however make strong assumptions about the metabolism or the problem domain. As a result, they work only for special settings and fail to address this problem when these assumptions do not hold.

Limitations of integer linear programming-based methods. One class of algorithms use integer linear programming to tackle the enzymatic target identification problem. OptKnock is one example to the algorithms in this class [9]. These strategies simulate the metabolism using Flux Balance Analysis, (FBA) [7, 29, 48]. At a high level, they represent each flux as a variable and solve a linear equation with linear constraints on these variables as follows.

Maximize (or minimize) Objective function

Subject to steady state constraints

This formulation represents the metabolism using a stoichiometric matrix $S$ where the rows and the columns correspond to compounds and fluxes respectively. Assume that $x = [x_1, x_2, \cdots, x_n]^t$ denotes the flux vector for a network with $n$ fluxes. The objective function is typically to maximize a variable or a linear combination of a set of variables. Thus an objective function is typically $\sum_i c_i x_i$ where $c_i$ are given constants. The constraints define the steady state using stoichiometric model. The solution to the equation $Sx = 0$ is the set of all steady states in this model, and this equation defines the constraints of the integer linear programming problem.

Integer linear programming-based methods have serious drawbacks. First, they are limited to the linear models. In other words they require that both the objective function and constrains are represented as linear equations (or inequalities). However, many models for metabolic networks do not satisfy this requirement. For example, S-systems [86, 107] defines the steady state as the solution to the equation system

$$\dot{X}_i = \alpha_i \prod_j X_i^{g_{ij}} - \beta_i \prod_j X_i^{h_{ij}} = 0, \forall i.$$
Here, the variable $X_i$ represents the concentration of the $i$th molecule. $\dot{X}_i$ is the derivative of $X_i$. The constants $\alpha_i$, $\beta_i$ and $g_{ij}$, $h_{ij}$ denote the rate of the reaction and the rate at which each molecule contributes to a reaction. Clearly, the constraints are non-linear in the S-systems of equations. Taking the logarithm of the constraints linearizes the constraints as follows. Define $Y_i = \log X_i$. The constraints become

$$\log \alpha_i + \sum_j Y_j g_{ij} - (\log \beta_i + \sum_j Y_j h_{ij}) = 0, \forall i.$$ 

However, the objective function transforms into the nonlinear form $\sum_i c_i e^{Y_i}$. Therefore, integer linear programming fails to solve the enzymatic target identification problem.

GMA model [73, 107] is even more problematic for integer linear programming-based methods. This model considers each reaction that a compound is a part of separately and represents the steady state using the following equations.

$$\dot{X}_i = \sum_h \gamma_{ih} \prod_j X_j^{f_{ij}} = 0, \forall i.$$ 

Here, the constants $\gamma_{ih}$ and $f_{ij}$ denote the rate of the reaction and the rate at which each molecule contributes to a reaction. In this model, not only the constraints are non-linear, but also the summation of the multiplicative terms make it impossible to take the logarithm of the constraints.

Limitations of the methods for boolean network models. Sridhar et al. and Song et al. considered a simplified version of the enzymatic target identification problem [94–96]. In their version, each entry of the state denotes whether the corresponding compound is present or not. For this simplified version, the goal, then, is to identify the set of enzymes whose knockouts eliminate all the “targeted compounds” while incurring minimum “damage”. Targeted compounds are the ones whose productions need to be stopped. They have defined “damage” as the number of non-targeted compounds that are eliminated because of knockouts. Minimum damage is the minimum number of non-targeted compounds eliminated from the metabolism.
while eliminating the targeted compounds among all possible ways of eliminating the targeted compounds. Sridhar et al., used a branch and bound strategy to solve this problem for pathways with up to 32 enzymes in less than an hour [96]. Sridhar et al. and Song et al. developed an iterative heuristic method to scale their solutions to large pathways [94, 95]. The binary model described in these works, however, is limited. It cannot address partial production of the compounds. In addition, it ignores the change in flux or yield due to the knockouts of a set of enzymes.

Figure 4-1 illustrates the limitations of the binary model used by Sridhar et al. and Song et al. [95, 96]. Inhibiting $E_1$ knocks out compounds $C_2$, $C_4$ and $C_5$. Suppose $C_4$ is the targeted compound. Inhibiting $E_1$ stops two non-targeted compounds ($C_2$ and $C_5$). The damage is two using the binary model. There are several drawbacks. One is that the knockout of $E_1$ accumulates $C_1$. The binary model, however, ignores this. Furthermore, although the knockout of $E_1$ does not fully eliminate $C_7$, it influences the production of $C_7$. The binary model disregards this influence as well.

Klamt et al. introduced a minimal cut set problem, which aimed to find a minimal set of reactions whose deletion leads to no feasible balanced flux distribution in the objective reaction [54]. The authors described several algorithms to solve the minimal cut set problem. However, the minimal cut set model has the drawbacks similar to the one used by Sridhar et al. and Song et al. The aim of this model is to block the objective reaction function which results in the removal of the objective metabolite synthesis. It can not be used when the aim is to partly decrease or increase the objective metabolites.

Limitations of randomized methods. “Extreme Pathway Analysis” [78] uses FBA to find the path in a pathway that maximizes or minimizes the production of a given compound. This problem is similar to a special case of the enzyme target identification problem considered in this chapter. De et al. for example, consider the extreme pathway analysis problem [18]. In order to
reduce the yield of a compound in a pathway, De et al. use FBA to compute the optimal pathway so that the yield of the target metabolite is minimum. They, then, change the concentration of the enzymes in other paths so that these paths are inactive except that optimal one. This method has two major drawbacks. First, it requires changing the concentration of many enzymes. In practice, changing the enzyme concentration is a costly process. Therefore, the number of enzymes whose concentrations are altered should be kept low. Second, the alterations that change the production of a compound can affect the production of other compounds in that pathway. Thus, the solution found by this method can have significant side-effects. In addition to these drawbacks, extreme pathway analysis cannot solve the enzymatic target identification problem considered in this chapter. Here, we develop methods to overcome the above-mentioned disadvantages and solve a more generic problem.

Limitations of evolutionary programming methods. Patil et al presented an evolutionary programming method for finding optimal gene deletion strategies [72]. Their method generates a population of random solutions and use genetic algorithm to improve this population. Their method can be applied to non-linear models as well as linear models. However, it has several drawbacks. The enzymatic target identification problem looks for a set of enzymes that are connected over a complex network and interact through reactions over compounds. Patil et al’s method ignores these interactions while constructing the population of solutions as well as creating new generation of solutions using crossover. They instead create these solutions randomly. The search space of the enzyme target identification problem is exponential in the number of enzymes. As a result, their method fails to converge to a good solution. Furthermore, the solutions found by their method suggests knocking out unnecessarily large number of enzymes. We experimentally evaluate their method in Section 4.4.2.3 and defer further discussion to that section.
All of above, we conclude that, existing algorithms work only under a limited specific set of assumptions about the metabolism. Since, these assumptions do not hold in practice most of the time, more generic methods that can address the enzymatic target identification problem for realistic metabolic network models are needed.

4.2 Computing the Steady State

The developed algorithms require computing the steady state of the metabolic pathway for each enzyme vector (i.e., candidate solution) that is considered during the search. Since our algorithms evaluate many enzyme vectors, we need to answer the following question: Can we compute the steady state of the pathway efficiently? This question has been studied thoroughly in the literature. Here, we briefly discuss two alternative ways to compute the steady state based on two alternative, yet similar, steady state definitions (Section 4.2.1 and 4.2.2).

4.2.1 Computing the Steady State Using Flux

The flux of a reaction shows the speed at which each compound is produced or consumed by that reaction. As the reactions progress, each flux can increase or decrease other fluxes. The pathway reaches to a steady state when all the flux values remain unchanged over time.

Figure 4-2 shows a hypothetical pathway along with fluxes that operate on it. Let \( v_1, v_2, v_3 \) denote the internal flux for reaction \( R_1, R_2 \) and \( R_3 \). \( b_1, b_2, \ldots, b_8 \) are the external flux. The state of the pathway shows these current internal and external flux which relates to the other pathways or compounds. Assume the reactions in the pathway are as follows.

\[
\begin{align*}
R_1 & : C_1 \rightarrow C_2 + 2C_7 \\
R_2 & : 2C_2 + C_3 \rightarrow C_4 + C_5 \\
R_3 & : C_6 \rightarrow C_7 + C_8 + C_9
\end{align*}
\]

One way to compute the steady state of this pathway is to employ Flux Balance Analysis (FBA) [7, 29, 48]. FBA creates a matrix, \( A \), that shows how each flux operates on each compound. Each row of this matrix corresponds to a compound and each column corresponds
to a flux. For example, for the pathway in Figure 4-2, $A$ has nine rows and 11 columns since there are nine compounds and 11 fluxes. Consider the flux $v_1$ which corresponds to reaction $R_1$ above. Assume that the first column of $A$ corresponds to $v_1$. The values of $A$ in this column are $A[1,1] = -1$, $A[2,1] = 1$, $A[7,1] = 2$ and all others are zero. This is because $v_1$ consumes one unit of $C_1$ to produce one unit of $C_2$ and two units of $C_7$. Let $v$ denote the flux vector (i.e., each entry of this vector corresponds to a flux). The product $Av$, then shows the amount of change in the concentration of each of the compounds. In order to compute the steady state, FBA makes several assumptions. One of them is that the total influx of each compound is equal to the total out flux of that compound. Thus, FBA computes the set of all possible steady states as the solution space of $v$ to the equation

$$\frac{dA}{dt} = Av = 0.$$ 

Typically, the number of variables in this equation is more than the number of constraints. As a result there are infinite possibilities for $x$. There are many ways to narrow down the solution space. One of them is that a flux can not be a negative value (i.e., $v_i \geq 0$ for all $i$). Another assumption is that the maximal rate of incoming flux (the flux enters the metabolism from external sources) is limited, e.g. $\leq 1$. Another way is to include an objective function, such as maximizing the biomass or the production of a specific compound or energy.

Example A.1 Consider the pathway in Figure 4-2. Assume that all the enzymes in this pathway are present in the metabolism and they are not knocked out. Also, assume that as the objective function, we maximize the total output flux. In other words, we want to find the steady state that maximizes $b_4 + b_5 + b_6 + b_7 + b_8$.

Thus, the problem of solving $Av = 0$ translates into the following one: Maximize $b_4 + b_5 + b_6 + b_7 + b_8$ subject to the constraints

$$b_1 - v_1 = 0$$
$$v_1 - 2v_2 = 0$$

78
\[ b_3 - v_2 = 0 \]
\[ v_2 - b_4 = 0 \]
\[ v_2 - b_5 = 0 \]
\[ b_2 - v_3 = 0 \]
\[ 2v_1 + v_3 - b_6 = 0 \]
\[ v_3 - b_7 = 0 \]
\[ v_3 - b_8 = 0 \]
\[ b_1, b_2, b_3 \leq 1 \]
\[ v_1, v_2, v_3, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8 \geq 0 \]

The solution is, \( v_1 = 1.0, v_2 = 0.5, v_3 = 1.0, b_1 = 1.0, b_2 = 1.0, b_3 = 0.5, b_4 = 0.5, b_5 = 0.5, b_6 = 3.0, b_7 = 1.0, b_8 = 1.0 \). We define the steady state as all the flux in the pathway. That is, \([v_1, v_2, v_3, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8] = [1, 0.5, 1, 1, 0.5, 0.5, 0.5, 3, 1, 1] \). □

Example A.II Now assume that \( E_1 \) is knocked out in the pathway used in Example I.A. Since \( E_1 \) catalyzes the reaction \( R_1 \), \( R_1 \) becomes inactive. Then \( v_1 \) become zero in the flux computation of Example A.I. Therefore, we add one more limitation \( v_1 = 0 \) to the above flux computation. The steady state under these constraints is \([0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1]\). □

4.2.2 Theoretical Yield Computation

FBA computes the flux at the steady state. It, however, does not describe the concentration of each compound when the metabolism reaches to a steady state. The amount of each compound at this state is a crucial information for many applications. For example, when dopamine concentration in the brain reduces below a certain level, the motor system nerves become unable to control movement and coordination.

We can compute the amount of each compound in the metabolism as its theoretical yield. The theoretical yield of a compound is the amount of that compound produced by the underlying reactions [106]. Compared to the flux computation, which describes the process of the reactions,
the yield computation only depicts the outcomes of the reactions. We can compute the yield of each compound in a pathway for a given initial state using the stoichiometry of that pathway. We use the same matrix $A$ that is used in Section 4.2.1 of Appendix to denote the stoichiometry. Recall that the stoichiometric coefficients of a reaction show the degree to which a chemical species participates in a reaction.

In order to compute the yield of each compound in the steady state, we need the initial state of the pathway. For simplicity, we assume that the initial yields of the compounds which are not produced in that pathway are one unit (e.g. one mol). For all other compounds, we set them to zero. Note that the process of computing the steady state is orthogonal to that of the initial state. Thus, one can replace our strategy of selecting the initial state with a different one. We, then, simulate the reactions in the pathway. We assume that all the reactions take place simultaneously given that the enzymes and input compounds required for them are present in the metabolism. For simplicity, we will assume that all reactions take the same amount of time in our discussion. Computing the yield for varying reaction speeds is similar.

Example B.I Assume the reactions in the pathway are the same as that in Section 4.2.1 of Appendix. Also, assume that all the enzymes are present in the metabolism, similar to Example A.I. In Figure 4-2, compounds $C_1$, $C_3$ and $C_6$ are not produced in the pathway. They are external inputs to the pathway. Thus, the initial state of the pathway is $[1, 0, 1, 0, 1, 0, 0, 0]$. In other words, there is one unit of $C_1$, $C_3$ and $C_6$ and none of the other compounds exist initially.

Initially, the reactions, $R_1$ and $R_3$, are active. This is because the yields of $C_1$ and $C_6$ are non-zero in the initial state. However, the reaction $R_2$ is inactive because of the lack of $C_2$. $R_1$ consumes one mol of $C_1$, and generates one mol of $C_2$ and two mol of $C_7$. $R_3$ consumes one mol of $C_6$ and produces one mol of $C_7$, one mol of $C_8$ and one mol of $C_9$. The current state after these reactions take place is $[0, 1, 1, 0, 0, 3, 1, 1]$. At this moment, $R_2$ is active for both yields of $C_2$ and $C_3$ are non-zero. $R_1$ and $R_2$ are inactive for the yields of $C_1$ and $C_6$ are zero. Then, $R_2$
consumes one mol of \( C_2 \) and \( \frac{1}{2} \) mol of \( C_3 \), and generates \( \frac{1}{2} \) mol of \( C_4 \) and \( \frac{1}{2} \) mol of \( C_5 \). Then, \( \frac{1}{2} \) mol of \( C_3 \) is left. Thus, the state of the pathway is \([0, 0, 0.5, 0.5, 0, 3, 1, 1]\). Since the yields of \( C_1, C_2 \) and \( C_6 \) become zero, \( R_1, R_2 \) and \( R_3 \) are inactive. A steady state is reached at this point as the yield does not change.

Example B.II Assume that \( E_1 \) is knocked out in the pathway used in Example A.II. This makes \( R_1 \) inactive as \( E_1 \) catalyzes \( R_1 \). For consistency, assume the initial state of the pathway is also \([1, 0, 1, 0, 0, 1, 0, 0, 0]\) (i.e., same as that in Example B.I). Then, only \( R_3 \) is active for both the input compound \( C_6 \) and the enzyme \( E_3 \) are present. \( R_3 \) consumes one mol of \( C_6 \) and generates one mol of \( C_7, C_8 \) and \( C_9 \). The current state is \([1, 0, 1, 0, 0, 1, 1, 1, 1]\). At this moment, all the reactions become inactive. Therefore, the metabolism comes to a steady state. As compared to the steady state when all the enzymes are present, the yield of \( C_1 \) increases and the yield of \( C_7 \) decreases.

There are four important observations that follow from the discussion of steady state for the theoretical yield computation.

We assume that all reactions take place simultaneously as long as all the input compounds and the enzymes are present in the metabolism. We also assume that each reaction takes place until there are no sufficient compounds that it can consume.

Theoretically, the state transition of a pathway may lead to an infinite loop of a sequence of more than one state. For example, the translations \( a \rightarrow b \rightarrow c \rightarrow a \) show a loop of three states \( a, b \) and \( c \). Such loops are considered a sequence of steady states. One can use average, max, min or other functions to summarize such states in a single state. We do not discuss this in detail as it is beyond the scope of this chapter.

The steady state depends on the initial state. One can show that the metabolism is guaranteed to reach to a steady state (or a sequence of steady states) and that the steady state depends on the initial state. The proof of the former follows from the synchronous transition model. The state
transition moves each state to a unique next state. Thus, a transition either creates a new state, remains constant or creates one of the previously visited states. The latter two cases correspond to steady states. The proof of the dependance of the steady state on the initial state can be done by simply setting the value of $C_1$ to zero in the initial state of Example II.A. When all the enzymes are present, the steady state after this modification is [0, 0, 1, 0, 0, 1, 1, 1, 1], which is different than the one in that example.

There is a strong correlation between the steady state of the flux and the steady state of the yield. As the flux reaches to a steady state, the total influx and outflux of each compound remains unchanged. Thus the derivative of the yield of each compound remains unchanged.

The yield computation is a good approximation of the state of the pathway and this value can be measured at lab. It is computationally desirable since it does not require the functional or kinetic information unlike concentration or flux. These information is not available for majority of the existing pathway databases. We use yield to denote state in this chapter. Furthermore, this chapter is orthogonal to the state computation.

4.2.3 Concentration Computation

The following equations show the relationship between the concentration of the molecules and their rates of change for Figure 4-2. For example, $\frac{d}{dt} C_1$ means the concentration change of compound $C_1$ per time unit. It equals that the incoming flux $b_1$ minus the outgoing flux $v_1$.

\[
\begin{align*}
\frac{d}{dt} C_1 &= b_1 - v_1 \\
\frac{d}{dt} C_2 &= v_1 - 2v_2 \\
\frac{d}{dt} C_3 &= b_3 - v_2 \\
\frac{d}{dt} C_4 &= v_2 - b_4 \\
\frac{d}{dt} C_5 &= v_2 - b_5 \\
\frac{d}{dt} C_6 &= b_2 - v_3 \\
\frac{d}{dt} C_7 &= 2v_1 + v_3 - b_6
\end{align*}
\]
\[
\frac{d}{dt} C_8 = v_3 - b_7 \\
\frac{d}{dt} C_9 = v_3 - b_8
\]

With more biology information, e.g. enzyme kinetics, metabolic control analysis, the above equations can compute the temporal evolution of the concentrations starting from arbitrarily given values. For example, assume \( b_1 = 0 \) and \( v_1 = k \). Then, \( \frac{d}{dt} C_1 = 0 - k \cdot C_1(t) = C_1(0) - kt \).

Then, we can get the concentration for each compound when time \( \mapsto \infty \). We define the steady state as all the concentrations when time \( \mapsto \infty \) in the pathway.

If enzyme \( E_1 \) is inhibited, the reaction of \( R_1 \) will be changed and the effective substrate concentration will also be altered. Based on the biochemical knowledge, we can add more equations to the above concentration computation. By this way, we can compute the concentration of each compound when \( E_1 \) is inhibited. The steady state of concentration comes out.

4.3 Methods

In this section, we present in silico methods for the enzymatic target identification problem. We first provide a measure to compute the distance between the current steady state and the goal state (Section 4.3.1). We, then, describe two algorithms, traversal and genetic algorithms (Section 4.3.2 and 4.3.3), to solve the enzymatic target identification problem.

4.3.1 State-Distance

The first task that needs to be addressed is to measure the distance between a given goal state and the steady state of the pathway after knocking out a set of enzymes. In order to achieve this, we first compute the steady state of the pathway after a set of enzymes are knocked out. We then measure the distance between this state and the goal state. We call this measure the “State-Distance” (SD).

The “state” of a metabolic pathway is a vector that indicates its current status. There are alternative ways to define and compute the state of a given pathway in the literature. These alternatives are similar in spirit. Each entry of the state vector denotes the yield of a compound or a
flux in the pathway. Yield of a compound is the amount of product obtained in the chemical reaction [106]. The flux of a reaction is the rate at which each compound is produced or consumed by that reaction [71]. We compute the yield of each compound in the steady state using the reaction parameters such as the rate at which it is consumed or produced by each reaction.

We apply FBA [7, 29, 48] or solve S-systems of equations to get the flux of the pathway in the steady state. Usually, FBA produces a space of steady states that contains infinitely many possible steady states. To select a unique steady state, FBA enforces optimizing an objective function in the solution space. The objective function of FBA often maximizes biomass [24] or the production of ATP [80]. Since the literature contains detailed discussion on the steady state computation, we defer the discussion of the steady state computation to Appendix. In the rest of this chapter, each entry of the steady state denotes the yield of a compound unless otherwise stated. Thus, in our notation, the steady state vector for a pathway has as many entries as the number of compounds in that pathway.

Given a goal state for the underlying pathway, next, we discuss how we compute the distance, SD, between the goal state and the current steady state. Note that SD is a generic measure that can be used to represent a broad set of objective functions including the ones used in the literature.

We first present the notation to formally define SD. Assume that the number of compounds in the pathway is $m$. We denote the goal state as $V_G = [g_1, g_2, \ldots, g_m]$, where $g_i = \text{ideal value}$ for the $i$th entry of the steady state. Let $N$ denote the number of enzymes. The “enzyme vector” shows the knockout status of the enzymes. We denote it with $V_E = [e_1, e_2, \ldots, e_N]$, where $e_i = \{0, 1\}$ ($e_i = 1$ if $E_i$ is knocked out, otherwise $e_i = 0$). Let $[r_1, r_2, \ldots, r_m]$ be the steady state of that pathway based on the enzyme vector $V_E$. Let $\omega_i$ be a real number that shows the importance of the $i$th entry of the steady state. We discuss the parameter $\omega_i$ later in this section.
We will use the variable $d_i$ to represent the distance contribution of the $i$th entry of the steady state. We develop two alternative definitions for $d_i$, namely “exact” and “fuzzy” distance.

**Exact distance:** This measure is useful when the exact value of the entry in the goal state is known. For the $i$th entry, we want to approach the goal state as close as possible. We compute the distance as $d_i = \omega_i |g_i - r_i|$.

**Fuzzy distance:** This measure is useful for extreme pathway analysis or when we do not know the exact values for some entries in the goal state. In this case, we, however, know a lower or upper bound for such entries. In other words, we want to increase (or decrease) the value of that entry to at least (or at most) a given value. Thus, we have two possibilities.

Case 1. (decrease the production of a compound) We want to minimize the $i$th value of the steady state with a threshold of $g_i$. In other words, $r_i$ should be smaller than $g_i$. The smaller the better:

$$d_i = \begin{cases} \infty & \text{if } g_i \leq r_i \\ \omega_i/(g_i - r_i) & \text{for } g_i > r_i \end{cases}$$

Case 2. (increase the production of a compound) We want to maximize the $i$th value of the steady state with a threshold of $g_i$. In other words, $r_i$ should be bigger than $g_i$. The bigger the better.

$$d_i = \begin{cases} \infty & \text{if } g_i \geq r_i \\ \omega_i/(r_i - g_i) & \text{for } g_i < r_i \end{cases}$$

The choice of exact and fuzzy distance depends on the underlying application. Our search algorithm in this chapter works for both of them. We use the exact distance measure to compute $d_i$ in the rest of this chapter unless otherwise stated.

The weights, $\omega_i$, show the importance of the $i$th entry of the state vector for the organism. Large $\omega_i$ indicates that the $i$th entry is important. This can be set based on expert input or topological features of the compounds in the pathway. For simplicity and without loss of generality, we set $\omega_i = 1$ for all $i$, for the experiment results that are presented in this chapter.
We compute SD as the largest of the distance of all the entries of the state vector, i.e., $SD = \max_i \{d_i\} = ||d_i||_1$. Note that one can also define it as $SD = \Sigma d_i$. Other combinations of distance measures discussed above are orthogonal with the rest of this chapter. Therefore, we do not discuss them further.

**Example 1** Consider the pathway in Figure 4-1. Assume that the goal state is $V_G = [0, 0, 1, 0, 0, 1, 1, 1]$. That is, we want one unit molecule of each of the compounds $C_3$, $C_7$, $C_8$ and $C_9$, and none of the remaining compounds. Also, assume that none of the enzymes are knocked out in this pathway (i.e., $V_E = [0, 0, 0]$). The steady state of this pathway is $s_0 = [0, 0, 0.5, 0.5, 0.5, 0, 3, 1, 1]$ (See Example B.I of Appendix). The state distance under this condition is $SD(V_E) = ||s_0 - V_G||_1 = 2$.

Now assume that $E_1$ is knocked out (i.e., $V_E = [1, 0, 0]$). We compute the steady state after knocking out $E_1$ as $s_1 = [1, 0, 1, 0, 0, 1, 1, 1]$ (See Example B.II of Appendix). Applying the state distance, we get $SD(V_E) = ||s_0 - V_G||_1 = 1$. Thus, knocking out $E_1$ brings the steady state closer to the goal state. □

**Example 2** Consider the pathway in Figure 4-1 when no enzyme is knocked out. Assume that the goal state is the same as that in Example 1. With the difference that we want to maximize the yield of the compound $C_7$ and obtain a yield of at least one unit of it. In this case, we use fuzzy distance for $C_7$ with a minimum goal = 1. The steady state of the pathway is the same as that in Example 1. Thus, the state distance is $\max \{0, 0, 0.5, 0.5, 0.5, 0, 0.5, 0, 0\} = 0.5$. □

4.3.2 Traversal Method

Given a metabolic pathway and a goal state, we aim to find the set of enzymes whose knockouts lead to a steady state with lowest value of SD. One way to solve this problem is to exhaustively examine the SD value after knockouts of all possible subsets of enzymes. However, this is not feasible because the number of subsets is exponential in the number of enzymes. In this section, we develop a traversal algorithm and two optimization strategies to accelerate it.
We traverse the search space using a branch-and-bound strategy. We consider the search space as a binary tree. Each node of this tree corresponds to a potential solution. Each node records four items: (i) the set of enzymes that are knocked out, (ii) the set of enzymes that are not knocked out, (iii) the set of enzymes that have not been considered so far, and (iv) the SD value of the pathways after all the enzymes in the first set are knocked out. In the root node, the first and the second sets are empty. Therefore, all the enzymes of the pathway are in the third set. We recursively visit the nodes using an in-order traversal method [37]. After visiting the current node, we visit the left and right child. Moving from a parent to a child node means considering a new enzyme on top of the enzymes considered in the parent node. The left child denotes that the new enzyme is knocked out and the right child denotes that it is not knocked out. We, then, compute the current SD. If the current SD is less than the best result seen so far, we update the value of best result with the new one. We propose to improve the performance of this algorithm through two different optimizations:

**Optimization 1.** In many cases, the knockouts of some uninspected enzymes cannot improve the SD. We set the values in the enzyme vector for such enzymes to zero (i.e. not knocked out). This process, called “Filtering”, can improve the performance of the algorithm as it skips many levels of the search tree.

**Optimization 2.** The selection of the enzyme when we move from a parent to a child impacts the performance of the algorithm. This is because if the nodes of the top levels in the search tree have small SD, the chance of filtering the nodes in its subtree becomes large. We call this the “Prioritization” strategy.

We discuss these two optimizations later in this section. In order to implement these two optimizations we, first, discuss how we quickly predict SD incrementally when a new enzyme is knocked out.
4.3.2.1 Predicting the value of SD

Computing the SD value of a given enzyme vector requires computing the steady state of the pathway. An effective prediction strategy can help in avoiding computation of the steady states of a large number of nodes if their SD values are greater than the current best.

We conjecture that the steady state after knockouts of enzymes $E_i$ and $E_j$ simultaneously is close to the average of that after knockouts of $E_i$ and $E_j$ separately. Note that similar conjectures have been made in the literature [13, 61]. This is intuitive, because the influence of knockouts of two enzymes should be correlated with the total influence of knockouts of the individual enzymes. We tested this conjecture by randomly sampling 50 enzyme pairs in each of ten randomly selected metabolic pathways of Homo sapiens ($H.sapiens$). The average correlation coefficient [28] of the steady state between actual and predicted values of these 500 random samples was 0.91.

It is worth mentioning the our conjecture above may not hold if $E_i$ and $E_j$ are dependent. The dependency can be in several ways. For example, multiple enzymes may carry out the same reaction. In some cases, the presence of only one of these enzymes suffices for the reaction to occur while in some cases all these enzymes have to be expressed for the reaction to occur. These dependencies may cause over or under predictions if the two enzymes create antagonism or synergism respectively. Then, how much can we trust in this conjecture? In order to answer this question, we performed another experiment as follows. We selected a more than 100 pairs of dependent enzymes. Each enzyme pair in this set are within three interactions of each other. We then measure the estimated and actual steady state values after deleting them. The average correlation coefficient between was still 0.91. This high correlation supports our conjecture.

4.3.2.2 Filtering strategy

We filter some uninspected nodes as follows. If the predicted SD values of these nodes are bigger than the current minimum SD, we filter these nodes. For a given node in the search tree,
let $A$, $B$ and $C$ denote the set of enzymes that are knocked out, the set of enzymes that are not knocked out and the set of enzymes that are not yet considered respectively. For each enzyme in set $C$ we predict the SD value after that enzyme is knocked out in addition to the enzymes in $A$.

If the predicted value for an enzyme in $C$ is worse than the best SD value found so far, we move that enzyme to set $B$. Moving $h$ enzymes from set $B$ to set $C$ is equivalent to filtering $h$ levels of the search subtree rooted at the current node. We predict the steady state for a single enzyme during filtering in time proportional to the size of the state vector by precomputing the steady state after knockout of each enzyme alone.

It is worth noting that we are using each enzyme independently to predict the steady state using a combination of enzymes. This process is error prone and can lead to pruning of potentially useful enzymes when using the filtering strategy. We address this problem by giving each enzyme several chances before we filter it. Let $K$ denote the number of chances, where $K$ is a positive integer. We call this “$K$-chance strategy”. To incorporate this strategy, we keep a vector, where each entry of this vector denotes the number of times that an enzyme is tested positively for a filter. We filter an enzyme only if that enzyme uses all of its $K$ chances.

### 4.3.2.3 Prioritization strategy:

We select the most promising enzyme in set $C$ to consider for knockout. An enzyme is promising if its knockout in addition to the enzymes in set $A$ has a small SD with high probability. This increases the chance of filtering the nodes in its subtree. Our method works as follows. For each of the uninspected enzymes, we predict the steady state after knockout of that enzyme in addition to already deleted enzymes.

We then compute the SD between that state and the goal state. We move the enzyme with the smallest predicted SD to set $A$ to create the next child node.
4.3.2.4 Multiple optimal solutions:

Our traversal algorithm can find multiple solutions (say $t$ solutions) as follows. We store the top $t$ solutions (i.e., the $t$ solutions with smallest SD values) found so far. As we traverse the search space, if we find a solution better than the $t$th best solution so far, we replace the worst solution in our list with the new one. Also, when filtering the search space, we use the SD value of the $t$th best solution so far rather than that of the top solution.

4.3.3 Genetic Algorithm

The time complexity of the traversal method remains exponential, though the filtering and prioritization strategies reduce the search space significantly. From experiments, the method described in the previous section is only useful for 30-35 enzymes. In this section, we propose a genetic algorithm to solve the target identification problem for large pathways. The genetic algorithm exploits the traversal method as a building block. The main idea of the genetic algorithm is to generate a population of candidate solutions and improve these solutions through crossover and mutation operations. The algorithm stops after a predefined number of “epochs” (or iterations). Next, we describe the genetic algorithm in detail. The genetic algorithm uses the following data structure.

Population: The population $P$ is a set of candidate solutions $\{S_1, S_2, \ldots, S_{\text{num seed}}\}$. Here, each solution, $S_i$ is a vector that shows which enzymes are knocked out and which enzymes are not. Let $N$ be the number of enzymes in the underlying pathway. We represent a solution with $S_i = [s_i^1, s_i^2, \ldots, s_i^N]$, where $s_i^j = 0$ means that the enzyme $E_j$ is not knocked out. Similarly, $s_i^j = 1$ means that $E_j$ is knocked out.

Algorithm 4.1 summarizes our solution. We discuss the details of this algorithm next.

Initialize population This step generates the initial population, $P$, which contains candidate solutions. Ideally, a good candidate resembles the optimal solution in terms of both the number and the selection of enzymes that are knocked out by it. However, we do not know the optimal
solution at this step. To address this problem, we need to answer two questions. (i) How many enzymes are knocked out in each candidate? (ii) How do we decide which enzymes are knocked out in each candidate? To address the first problem, we employ our traversal algorithm in Section 4.3.2 as follows. We estimate the number of removed enzymes in good solutions (i.e., solutions with low SD). Let \( \lambda \) denote this estimate. We run the traversal method for the top several levels of the search space. We search 10 levels to limit this traversal time. We, then, select a set of solutions with smallest SD (say 20 best solutions). We estimate \( \lambda \) as the average number of removed enzymes in these solutions. The filtering and prioritization strategies push the results with small SD to the top levels of the search tree with high probability. Thus, limiting the search to only a few levels of the tree does not degrade the accuracy of \( \lambda \) greatly.

Once we compute \( \lambda \), the next problem is to decide which enzymes will be knocked out. We use binomial distribution to compute the knockout probability of each enzyme. Ideally, the probability of knocking out an enzyme should be high if that enzyme has a high potential to contribute to a good solution. We predict this potential of each enzyme by considering the SD value obtained after knocking out that enzyme. Suppose that \( SD(E_i) \) denotes the value of
SD when only enzyme $E_i$ is knocked out. We conjecture that if $SD(E_i)$ is small, then $E_i$ has a high probability to contribute to a good solution. Let $p_i$ denote the probability of knocking out enzyme $E_i$ in a good solution. There is a negative correlation between $p_i$ and $SD(E_i)$. We use the following equations to estimate $p_i$.

$$p_i = \frac{\beta}{1 + SD(E_i)}.$$ \hfill (4–1)

In this equation, the parameter $\beta$ is a normalizing constant. We compute the value of $\beta$ from the observation that the expected value of the total number of removed enzymes is $\lambda$. We do this using the following equation.

$$\lambda = \sum_i p_i = \beta \sum_i \frac{1}{1 + SD(E_i)}.$$ \hfill (4–2)

From (4–1) and (4–2), we compute the probability of knocking out enzyme $E_i$ as:

$$p_i = \frac{\lambda}{(1 + SD(E_i)) \sum_i \frac{1}{1 + SD(E_i)}}.$$ \hfill (4–3)

Now, we are ready to generate candidate solutions. We create each candidate by knocking out enzyme $E_i$ with probability $p_i$, $\forall j$. In practice, we generate 100 candidate solutions to create the initial population.

**Generate children using crossover.** This step computes a child population from the current population. It aims to combine two existing solutions to create a better one. Generating a child solution involves the following steps: selection of two parent solutions from the existing population and crossover of these two parents.

We first discuss how we pick two parent solutions from the current population. We conjecture that good parents can generate good children with high probabilities. We, thus, randomly choose each parent using a biased distribution that prefers parents with small SD. Suppose the probability of choosing the solution $S_i$ as a parent is $x_i$. Based on our conjecture, there is an
inverse relation between \(x_i\) and \(SD(S_i)\). Assume that \(\gamma\) is the coefficient of that correlation. We can write \(x_i\) as:

\[
x_i = \frac{\gamma}{1 + SD(S_i)} \quad (4-4)
\]

We need the parameter \(\gamma\) to compute the probability \(x_i\). We can compute \(\gamma\) from the observation that the selection probabilities of all the solutions in the current population add up to one. Formally, \(\sum_i x_i = 1\). Thus, we have

\[
1 = \sum_i x_i = \gamma \sum_i \frac{1}{1 + SD(S_i)} \quad (4-5)
\]

We get the value of \(\gamma\) from (4–5) and use it in (4–4) to compute \(x_i\) as:

\[
x_i = \frac{1}{(1 + SD(S_i)) \sum_i \frac{1}{1 + SD(S_i)}} \quad (4-6)
\]

So far, we have discussed the selection of parents. Next, we discuss the creation of a single child solution from two parent solutions. We denote the parent solutions with \(F\) and \(M\) \((F, M \in P)\), where \(F = [f_1, f_2, \ldots, f_N]\) and \(M = [m_1, m_2, \ldots, m_N]\). We denote the child of \(F\) and \(M\) with \(Ch = [Ch_1, \ldots, Ch_N]\). We use the following crossover method to produce the child. We first set the enzymes in the child vector for which both parents have the same value. Formally, if \(f_j = m_j\), then we set \(ch_j = f_j = m_j\). We decide the values of the remaining enzymes using our traversal method in Section 4.3.2. This strategy favors children that have small SD values as the traversal strategies seeks solutions with small SD.

Table 4-1 demonstrates this process on an example. In this example, \(f_2 = m_2 = 1\), so \(ch_2 = 1\). We set the values \(ch_5, ch_6\) and \(ch_9\) similarly. The parents do not agree for the values of \(ch_1, ch_3, ch_4, ch_7\) and \(ch_8\). We use the traversal method to find their values. To do this, we initialize the root of the search tree as follows; the set of removed enzymes is \(\{E_2, E_5\}\), the set of enzymes that are not knocked out is \(\{E_6, E_9\}\), rest of the enzymes are undecided. The traversal algorithm, then, traverses the search space defined by the undecided enzymes to determine their values that minimizes SD.
We repeatedly select two parents and create a child until the number of children is equal to the total number solutions in the initial population.

**Perform Selection.** At the end of previous step, we have a set of already existing solutions, \( P \), and a set of child solutions. Thus the total number of solutions is double the size of \( P \) (i.e., it is \( 2 \cdot \text{num\_seed} \)). In this step, we update \( P \) as the \( \text{num\_seed} \) solutions with the smallest SD among the union of \( P \) and the child solutions.

**Perform mutation.** The previous steps repeatedly improve the solutions in the initial population \( P \). There is, however, the risk that these steps get stuck in a local minima or a plateau. This step aims to avoid such local minima or plateaus. To do this, we alter the solutions in \( P \) by mutation (i.e., changing the knockout status of some of the enzymes). We mutate each solution in \( P \) except the one with the minimum value of SD. For a given mutation rate \( \delta \) (we used a rate of 0.04), we change the value of each \( s^i_j \) to \( 1 - s^i_j \) with probability \( \delta \).

**Shrink each solution to minimal subset.** In practical applications, solutions that knock out fewer enzymes are desirable. This is because knocking out an enzyme is a costly task. One way to do this is to knock out fewer enzymes than given in each solution without changing the SD value of that solution. This step improves the solutions in the population by shrinking the set of removed enzymes. In details, we iteratively test each removed enzyme of a solution. For each such enzyme, we update its state to unremoved. If the SD of the solution does not increase after this modification, we remove this enzyme from the set of removed enzymes permanently. Otherwise we keep it. We repeat this iteration until we test all the enzymes. In fact, there can be multiple minimal subsets with the same SD values. We however do not see how to compute the SD value of a reduced set of enzymes without actually computing the steady state after reducing the enzyme set. Thus, the number of reduced sets tried while shrinking becomes the bottleneck of this step. Selecting one enzyme at a time for removal guarantees a small upper bound to the running time of the shrinking step. Therefore, we use the above strategy. Note that, one can
randomly shuffle the enzymes and try to find alternative reduced sets using different enzymes as seeds. However, to keep the focus of the chapter, we do not provide experiments with alternative implementations of the shrinking step.

**Finding multiple solutions:** An obvious question that follows the genetic algorithm is whether it can find multiple alternative solutions rather than a single solution. Our genetic algorithm can return alternative solutions in two different ways. First, recall that each generation contains a population of solutions. Thus, each member of each population is a solution itself. Keeping the top \( t \) solutions with smallest SD values will return \( t \) alternative solutions. Second, the shrinking phase of our genetic algorithm can be altered to return alternative shrink sets of enzymes with the same SD values. This can be done by shrinking the enzymes in random order multiple times as discussed above. Each random ordering can produce another alternative minimal subset of enzymes that has the same SD value if such alternative minimal subsets exist.

### 4.3.4 Use of Traversal Algorithm in Crossover and Performance Hiccups

The crossover step (Step 3) of our genetic algorithm uses our traversal algorithm to create child solutions with small SD values. We, however, advocated the use of our genetic algorithm over our traversal algorithm for large pathways because the traversal algorithm is not scalable. Thus, we need to show why our genetic algorithm is scalable despite it uses the traversal algorithm for each child at each iteration.

The scalability of the traversal algorithm used in the genetic algorithm follows from the observation that it is used only for the undecided enzymes (i.e., the enzymes for which the two parents disagree). Using the notation we defined in this section, we can compute the probability that enzyme \( E_i \) is undecided as \( 2p_i(1 - p_i) \). This is because one parent knocks out \( E_i \) while the other does not. Since the knockout status of each enzyme in a parent is decided independently, the expected number of undecided enzymes is \( 2\sum_i p_i(1 - p_i) \), or simply \( 2\lambda - 2\sum_i p_i^2 \). The actual value of this expectation depends on the probability values \( p_i \). Table 4-2 lists the expected
number of undecided enzymes for four pathways of varying sizes. It shows that the expected value is small, and thus, the traversal algorithm can be used without affecting the scalability of the genetic algorithm in practice.

One can argue that the pathways in Table 4-2 might have a small expected number of undecided enzymes due to their topology or size, and thus, the search space may be large for larger pathways or pathways with different topologies. To complete our discussion on scalability, we need to compute the expected number of undecided enzymes in the worst possible distribution of probabilities $p_i$. The following theorem computes the worst expectation and the corresponding standard deviation.

**Theorem 2.** Let $N$ denote the number of enzymes in a given pathway. Let $\lambda$ be the expected number of removed enzymes in a solution of the population of solution generated for that pathway by our genetic algorithm. The expected number of undecided enzymes in a child solution is at most $2\lambda - \frac{2\lambda^2}{N}$. The standard deviation for the worst scenario is $\sqrt{\frac{(2\lambda^2 - 2\lambda N + N^2)(N^2 - 1)N}{N^2}}$.

**Proof:** We denote the expected value of a random variable with $E[.]$. We will first prove the mean in the first part of the proof. In the second part, we will prove the standard deviation.

Part 1 of proof: The worst scenario analysis of the expected number of undecided enzymes. Let $F = [f_1, f_2, \cdots, f_N]$ and $M = [m_1, m_2, \cdots, m_N]$, denote two arbitrary solutions taken from the population of solutions. In order to compute the expected number of undecided enzymes during the crossover of $F$ and $M$, we first estimate the number of decided enzymes, that is, the number of enzymes for which, $f_i = m_i$. We define the random variable $X_i$ as:

$$X_i = \begin{cases} 
1 & \text{if } f_i = m_i \\
0 & \text{otherwise}
\end{cases}$$

$X_i = 1$ in two cases: $f_i = m_i = 1$ and $f_i = m_i = 0$. Let $p_i$ be the probability that the $i$th enzyme in a solution is knocked out. Then, the expected value of $X_i$ is
\[ E[X_i] = p_i \cdot p_i + (1 - p_i) \cdot (1 - p_i) \]
\[ = 2p_i^2 - 2p_i + 1 \]

We would like to minimize \( E[\sum X_i] \) subject to the constraint that \( \sum p_i = \lambda \). The random variables \( X_i \) and \( X_j \) are independent for \( i \neq j \). Therefore, we compute the expected number of decided enzymes as

\[
E[\sum_i X_i] = \sum_i E[X_i] = \sum_i (2p_i^2 - 2p_i + 1) = 2 \sum_i p_i^2 - 2 \sum_i p_i + N
\]

From the definition of the probabilities \( p_i \) in Equation (2) we have \( \sum_i p_i = \lambda \).

\[
\lambda = \sum_i p_i \]
\[\iff\]  \[
\frac{\lambda}{N} = \frac{\sum_i p_i}{N} \]
\[\implies\]  \[
\frac{\lambda}{N} \leq \sqrt{\frac{\sum_i p_i^2}{N}} \quad \text{(Follows from power mean inequality)}
\]
\[\implies\]  \[
\frac{\lambda^2}{N} \leq \sum_i p_i^2
\]

Using the inequality we obtained above, we get

\[
E[\sum_i X_i] = 2 \sum p_i^2 - 2\lambda + N \geq 2 \frac{\lambda^2}{N} - 2\lambda + N
\]
By definition of power mean inequality, we minimize the value of the expectation (i.e., equality case) when all $p_i = p_j$ for all $i, j$. Thus, we conclude that $p_i = \frac{\lambda}{N}$ for all $i$. Using this value for $p_i$, we compute the minimum value of $E[\sum X_i]$ as

$$E[\sum X_i] = \frac{2\lambda^2}{N} - 2\lambda + N.$$  

The expected number of undecided enzymes is $N - E[\sum X_i]$. Therefore, the maximum number of undecided enzymes is

$$N - (\frac{2\lambda^2}{N} - 2\lambda + N) = 2\lambda - \frac{2\lambda^2}{N}.$$  

Part 2 of proof: The standard deviation of the worst scenario of the expected number of undecided enzymes. Here, we analyze the standard deviation $\sigma$ for the undecided enzymes in the worst scenario for the mean.

Assume that,

$$X = X_1 + X_2 + \ldots + X_n.$$  

From the definition of the standard deviation, $\sigma$, is

$$\sigma = \sqrt{E[(X - E(X))^2]} = \sqrt{E[X^2] - (E[X])^2}.$$  

We proved in Part 1 of our proof of Theorem 1 that $E[X] = \frac{2\lambda^2}{N} - 2\lambda + N$ in the worst scenario, and that this scenario happens when, $p_i = \frac{\lambda}{N}$. Let $q$ be the probability of $X_i = 1$ in the worst scenario. Then,

$$q = p_i \cdot p_i + (1 - p_i) \cdot (1 - p_i) = (\frac{\lambda}{N})^2 + (1 - \frac{\lambda}{N})^2 = 1 - 2 \cdot \frac{\lambda}{N} + 2 \cdot \frac{\lambda^2}{N^2}.$$  

Before we compute $E[X^2]$, we consider $E[X_i^2]$. The values of $X_i$ is either 0 or 1. Therefore,

$$E[X_i] = Pr(X_i = 1) = Pr(X_i^2 = 1) = E[X_i^2].$$  

98
The multiplication $X_i X_j (i \neq j)$ evaluates to one only when $X_i = X_j = 1$. Assume that $k$ of the $X_i$s have value of one and the remaining $N - k$ of the $X_i$s have value of zero. Then, exactly $k(k - 1)$ of the $X_i X_j$ multiplications evaluate to one. Let $Z_k = k(k - 1)$ denote this. Now, we are ready to consider $E[X^2]$.

$$E[X^2] = E[(X_1 + X_2 + \ldots + X_n)^2]$$

$$= E[\sum_i X_i^2 + \sum_{i \neq j} X_i X_j]$$

$$= E[\sum_i X_i^2] + E[\sum_{i \neq j} X_i X_j]$$

$$= E[\sum_i X_i] + E[\sum_{i \neq j} X_i X_j]$$ (by $E[X_i] = E[X_i^2]$)

$$= E[\sum_i X_i] + \sum_{i \neq j} Pr(X_i = 1, X_j = 1)$$

$$= E[\sum_i X_i] + \sum_{k \geq 2} Pr(X = k) Z_k^2$$

$$= E[\sum_i X_i] + \sum_{k \geq 2} \left( \frac{N!}{k!(N-k)!} \right) \cdot q^k (1 - q)^{N-k} \cdot Z_k^2$$

$$= E[\sum_i X_i] + q^2 \sum_{k \geq 2} \left( \frac{N!}{k!(N-k)!} \right) \cdot q^k (1 - q)^{N-k} k(k - 1) Z_k^2$$

$$= E[\sum_i X_i] + \sum_{k \geq 2} \left( \frac{N!}{(k-2)!(N-k)!} q^k (1 - q)^{N-k} \right)$$
\[
\begin{align*}
= & \mathbb{E}[\sum X_i] + N(N-1)q^2 \sum_{k \geq 2} \binom{N-2}{k-2} \cdot q^{k-2}(1-q)^{N-k} \\
= & \mathbb{E}[\sum X_i] + N(N-1)q^2(q + 1 - q)^{N-2} \\
= & \mathbb{E}[\sum X_i] + N(N-1)q^2 \\
= & \frac{2\lambda^2}{N} - 2\lambda + N + N(N-1)(1 - 2 \cdot \frac{\lambda}{N} + 2 \cdot \frac{\lambda^2}{N^2})^2 \quad \text{(by } q = 1 - 2 \cdot \frac{\lambda}{N} + 2 \cdot \frac{\lambda^2}{N^2} \text{) }
\end{align*}
\]

Finally, we compute the standard deviation.

\[
\begin{align*}
\sigma &= \sqrt{\mathbb{E}[X^2] - (\mathbb{E}[X])^2} \\
&= \sqrt{\frac{2\lambda^2}{N} - 2\lambda + N + N(N-1)(1 - 2 \cdot \frac{\lambda}{N} + 2 \cdot \frac{\lambda^2}{N^2})^2 - (\frac{2\lambda^2}{N} - 2\lambda + N)^2} \\
&= \sqrt{(2\lambda^2 - 2\lambda N + N^2)(N^2 - 1)N} \\
&= \frac{\sqrt{2\lambda^2 - 2\lambda N + N^2}(N^2 - 1)N}{N^2}
\end{align*}
\]

Figure 4-3 plots the expected number of undecided enzymes in the worst case for varying pathway sizes and $\lambda$. The vertical lines show the standard deviation in each direction. We plot the expectation for up to $\lambda = 10$, since we observed $\lambda < 10$ in practice. The figure shows that for practical values of $\lambda$, the expected number of undecided enzymes remain small enough to make the genetic algorithm efficient. Furthermore, as the pathway size grows by eight fold, the upper bound for the expectation grows by only a few enzymes. Thus, we conclude that our algorithm is scalable to large pathways.
4.4 Results

In this section, we evaluate our algorithms on real datasets. We evaluate their biological significance on real applications (Section 4.4.1). We also evaluate their performance quantitatively (Section 4.4.2) using the following two measures:

**SD:** The SD value is the distance from the goal state. A small SD value indicates a better result.

**Execution time:** This indicates the total time (in seconds) taken by our algorithms.

We use the metabolic pathway information of *H. sapiens* and *E. coli* from KEGG as the input dataset. We present results for eight pathways of varying sizes in our experiments. Details of these datasets are shown in Table 4-3. KEGG Id is the unique identifier of each pathway in the KEGG database. #E, #R and #C denote the number of enzymes, reactions and compounds respectively in the pathway. The symbol “-” means that no KEGG Id exists for this metabolism.

We implemented the developed algorithms in C++. In our experiments, we set the default values of the parameters as follows. In the computation of SD, $\omega_i = 1$, for all the entries for simplicity. From our experience, the filtering strategy does a good job with two chances. The genetic algorithm works well when the mutation rate, $\delta$, is 0.04. To estimate $\lambda$, we search the first 10 levels of the search space to get top 20 best solutions for the balance of efficiency and accuracy. We ran our experiments on a system with dual 2.2 GHz AMD Opteron Processors, 4 gigabytes of RAM, and a Linux operating system.

We use the following strategy to set the initial state of the metabolic networks in our experiments. Some of the compounds used in the metabolic pathway are produced within the same pathway while some are supplied from external sources. This information is available in the existing metabolic pathway databases. In our experiments, we assume the same amount (one mol) of each compound that is externally supplied. This is because we do not want to artificially put any bias towards any externally supplied compound. We also assume that the internally produced compounds initially do not exist in the system. This is because, we it is realistic to let
the reactions decide which one is produced more rather than artificially setting some values. That said, it is important to realize that our algorithms can work for any initial state as they do not benefit from these assumptions.

### 4.4.1 Evaluation of the Biological Significance

We first evaluate the biological significance of our algorithms using flux or yield. We do this by comparing our results to known results in the literature.

#### 4.4.1.1 Metabolic engineering of Glycolysis/Gluconeogenesis pathway

The Glycolysis pathway of *T.pallidum* produces Phosphoenolpyruvate. De et al., [18] studied the path that maximizes the production of this compound. *T.pallidum*, however, has a simple pathway that contains only four enzymes. We considered the Glycolysis/Gluconeogenesis pathway of *E.coli*. This organism has significantly larger and more complex pathway, that contains all the enzymes of *T.pallidum* and many more. We ran our algorithm by setting the goal state to maximization of Phosphoenolpyruvate (i.e., goal > 0 for Phosphoenolpyruvate). Our results suggest knocking out the set of enzymes \{phosphoglucomutase, aldose 1-epimerase, fructose-1,6-bisphosphatase II, phosphoglycerate kinase, pyruvate kinase\}, (see red, crossed out enzymes in Figure 4-4). When all these enzymes are knocked out, the pathway from alpha-D-Glucose 1-phosphate to Phosphoenolpyruvate is, alpha-D-Glucose 1-phosphate $\rightarrow$ alpha-D-Glucose $\rightarrow$ alpha-D-Glucose 6-phosphate $\leftrightarrow$ (beta-D-Glucose-6P $\leftrightarrow$) beta-D-Fructose-6P $\rightarrow$ beta-D-Fructose-1,6P2 $\leftrightarrow$ (Glycerone-P $\leftrightarrow$) Glyceraldehyde-3P $\leftrightarrow$ Glycerate-1,3P2 $\rightarrow$ Glycerate-3P $\leftrightarrow$ Glycerate-2P $\leftrightarrow$ Phosphoenolpyruvate. This is the same path found for *T. pallidum* [18]. This can have several explanations. The enzyme fructose-bisphosphatase takes part in several pathways of *E.coli* such as the Fructose and mannose metabolism. This enzyme however slows down the production of beta-D-Fructose-1,6P2 which is needed for production of Phosphoenolpyruvate. Our results suggests that knocking this enzyme out will accelerate the production of Phosphoenolpyruvate. This enzyme however does not appear in *T. pallidum*. 

102
Therefore, the production of Phosphoenolpyruvate goes through the same path in *T. pallidum* without interruption of other enzymes. Similarly, pyruvate kinase converts Phosphoenolpyruvate to pyruvate in *E.coli*, but not in *T. pallidum*. Thus it reduces the amount of Phosphoenolpyruvate in *E.coli*. This example shows that we can successfully identify these factors and optimize the production of the desired compounds.

### 4.4.1.2 Application on the production of Phenylalanine

L-Phenylalanine is widely used in the manufacture of aspartame and in parenteral nutrition [1]. Industrial application need an efficient process to generate L-Phenylalanine. Considering the commercial application, Backman et al. selected *E. coli* as the production organism [1]. Thus, we studied the Phenylalanine, tyrosine and tryptophan biosynthesis pathway of *E.coli*. We ran our algorithm by setting the goal state to maximization of Phenylalanine. Our results suggest knocking out the set of enzymes \{phenylalanine translase, chorismate lyase, aspartate aminotransferase, tyrosyl-tRNA synthetase\}, (see the red, crossed out enzymes in Figure 4-5). Usually, phenylalanine is synthesized from glucose and ammonia in common bacterial systems. Inhibiting phenylalanine translase stops the Phe-tRNA synthesis from phenylalanine. Thus, the concentration of pheylalanine is accumulated. Inhibiting chorismate lyase cuts the flux to other biosynthesis. It, then, reduces the by-products and efficiently uses the raw materials.

### 4.4.1.3 Increasing the production of cGMP

The compound 3’-5’-Cyclic GMP (cGMP) plays an important role in the heart failure. One treatment is to increase cGMP [12, 42, 74]. cGMP appears in the Purine metabolism. We consider this pathway in *H.sapiens* in this experiment. We design the experiment as follows. We first compute the steady state when all the enzymes are active. We consider this steady state as the goal state except the cGMP entry. For the cGMP entry, we set our goal to maximize its production. Our algorithms on the pathway suggests knocking out the enzymes PDE and guanase. The literature supports this result. PDE inhibitor is a kind of drug that blocks the
subtypes of PDE, then it increases the concentration of cAMP or cGMP or both. It can treat heart failure [12, 32]. We predict that the side effect may be less if PDE inhibitor and guanase inhibitor are used together rather than knocking out PDE alone.

4.4.1.4 Metabolic engineering of the Butanoate metabolism.

Poly-$\beta$-hydroxybutyrate is an essential compound for producing plastics. Thus, increasing its production is critical for many industrial applications. Butanoate metabolism produces this compound. We run our algorithm on the Butanoate pathway of *E.coli* with the goal of maximizing this compound. Our results predict that the knockouts of phosphotransbutyrylase and BHBD increase the entry value of poly-$\beta$-hydroxybutyrate while incurring minimum damage to the rest of the metabolism. In fact, Vazque et al. shows the evidence of an association between poly-$\beta$-hydroxybutyrate and phosphotransbutyrylase [103].

4.4.1.5 Acetate reduction in E.coli

In this experiment, we show how our method performs when it is applied on a metabolism using the linear constraints of the classical flux balance analysis.

Yang et al. shows several strategies for acetate reduction in the central metabolic pathways of *E. coli* [113]. One method is to directly influence the formation of acetate. When we run our algorithms on *E.coli* Glycolysis/Gluconeogenesis pathway by setting the goal to minimization of the yield of acetate, the result enzyme is acyl-activating enzyme. This result is consistent with the biological discovery. Inhibition of acyl-activating enzyme cuts down the acetyl-CoA - acetate pathway. The destruction of this pathway results in the low level of acetate [113]. The results found by our algorithms suggests to knockout dihydrolipoamide acetyltransferase and pyruvate decarboxylase, which destroy the pathway from Pyruvate to Acetyl-CoA. These results are consistent with the metabolic engineering methods, e.g., the destruction or complete elimination of the portion of the reaction pathway at the acetyl-CoA node [113].
4.4.1.6 Purine metabolism on Generalized Mass Action (GMA) model

In this example, we demonstrate the results found by our method when it is applied on a metabolism using a non-linear constraints. One of the significant productions in Purine metabolism is Uric acid. When the concentration of Uric acid in human body is above the normal range, it results in some disease, such as, hyperuricemia and gout. We apply our algorithms to Purine metabolism on Generalized Mass Action (GMA) model. GMA model considers the concentration of each compound and it is non-linear system (see Section 6.1). We utilize the GMA model on Purine metabolism in Chapter 10 in Voit’s book [108]. We run our genetic algorithm on this metabolism by setting the goal to minimize the concentration of uric acid and keep the concentration of other compounds unchanged. The resulting enzyme set is \{xanthine dehydrogenase, adenosine deaminase, Oxidoreductases\}. From the literature, we know that inhibiting xanthine dehydrogenase or oxidase is widely used for treatment of hyperuricemia and gout [70]. When uric acid production is blocked by inhibiting of xanthine oxidase, it results in an increase in hypoxanthine and xanthine. Thus, inhibiting adenosine deaminase can decease the concentration of hypoxanthine and xanthine.

4.4.2 Quantitative Analysis of the Proposed Methods

In this section we quantitatively evaluate the performance of our traversal and genetic algorithms.

4.4.2.1 Evaluation of the traversal method

The goal of this experiment is to evaluate the performance of the traversal method. For each pathway, we construct the goal state as the steady state of that pathway when no enzyme is knocked out. We, then, modify the pathway by eliminating an enzyme from that pathway and search the resulting pathway to find the solution that is closest to the goal state. For each pathway, we create one query for each enzyme. Thus, we have as many queries as the number of enzymes. We report the average values over all queries for each pathway.
The experiment described above is based on the following biological intuition. An organism is, often, healthy if all of its enzymes function well. This corresponds to the case when no enzymes are knocked out. Thus, the corresponding steady state is the goal state. An organism may suffer from a disease if an enzyme malfunctions. In this case the query will correspond to a pathway that has malfunctioning enzymes. We aim to find an enzyme set whose knockout leads it back to the healthy state as close as possible.

We did not compare the traversal algorithm to exhaustive search for bigger pathways as the exhaustive search requires weeks to months even for a single query. For example, the exhaustive method will cost days when the number of enzymes are more than 30. In Figure 4-6 and Figures 4-7, ‘E’ followed by a number shows the number of enzymes in the pathway. Figure 4-6 shows the average SD of the solutions computed by the traversal method and those of the exhaustive search algorithm. The results show that the SD values of the traversal method and those of the exhaustive solution are almost identical. Thus, the traversal method is a good approximation to the optimal solution. Figures 4-7 presents the average running time of the traversal method as compared to the exhaustive search algorithm. The running time of exhaustive search is 1.2 to 11 times that of the traversal method. However, it is also clear that the running time of the traversal method increases exponentially with the number of enzymes. For example, in Glycine, serine and threonine metabolism (the number of enzymes is 32), the running time of the traversal method is more than two days. Therefore, it is impractical for very large pathways, although it can be used to solve larger sized problems than the exhaustive search.

Our final experiment in this section evaluates the variation of the number of enzymes knocked-out in the top results found by our traversal algorithm. We measure this as follows. We use the Urea cycle and metabolism of amino groups in this experiment. We run a query after deleting each of the 21 enzymes in this network. We find the top ten solutions for each of the 21 queries (i.e., totally 210 solution). Figure 4-8 shows the distribution of the number of enzymes knocked-out.
knocked out in these solutions. The results show that the number of enzymes deleted can vary. However, they cluster around several values (in this case two values). This resembles a mixed Gaussian distribution.

4.4.2.2 Evaluation of the genetic algorithm

This experiment evaluates the performance of our genetic algorithm. Similar to the evaluation of the traversal method, we design the experiment with pathways up to 84 enzymes. We obtain the pathways that have more than 52 enzymes by combining multiple pathways. For example, Pathway 00230+00790 denotes the pathway obtained by combining 00230 and 00790. For each of these pathways, we created one query for each enzyme similar to the previous section. We report the average values over all queries for each pathway.

The traversal algorithm is impractical for such large pathways due to the exponential time complexity. We, therefore, implement a truncated version of the traversal algorithm. This version truncates all the nodes deeper than $L$ levels, where $L$ is a given parameter. Choosing an appropriate value for $L$ makes the execution time of the truncated method suitable for pathways with a large number of enzymes. For example, the number of enzymes in Glycolysis / Gluconeogenesis pathway (00010) is 27. In the worst case it generates $2^{27}$ nodes requiring an execution time of 15+ hours. In order to bound the running time of our traversal algorithm for large pathways, we truncated the depth of the search after a fixed depth $L$. We chose $L$ to be 20 or 23 as the running time quickly becomes impractical for larger $L$.

Table 4-4 presents the SD value and the running time of the traversal method and the genetic algorithm for different pathways. $D_{20}$ and $T_{20}$ denote the average SD of the best solution and the average time requirement for multiple queries using the truncated method. $D_{tral}$ and $T_{tral}$ denote the average SD and the average running time for the traversal method. $D_{ga}$ and $T_{ga}$ denote the average SD and the average running time for the genetic algorithm respectively. All the time is in seconds. $D_0$ denotes the average SD between the steady state of the query pathway.
and the goal state. $\infty$ denotes that the running time is more than one day. ? denotes that no SD values can be computed within one day. Figure 4-6 had already demonstrated that our traversal algorithm finds near optimal solutions for small sized pathways. This results in this table shows that the performance of the genetic algorithm is comparable to the traversal method. Thus, it finds accurate results. For example, in Metabolism of xenobiotics by cytochrome P450 (00980), SD values and running time are similar for both methods. In Urea cycle and metabolism of amino groups (00220), the genetic algorithm runs much faster while it obtains worse SD values than the traversal method. When the size of pathways increases, the performance of the genetic algorithm is much better than the traversal method.

In Table 4-4, the difference between the initial damage $D_0$ and $D_{20}$ shows how much the truncated traversal algorithm reduces the distance between the initial and the goal state. The difference between $D_{20}$ and $D_{ga}$ show the amount of improvement obtained by the genetic algorithm over the truncated traversal method. These results show that the genetic algorithm generates significantly better solutions (lower damage values) as compared to the truncated traversal method for all the cases. The genetic algorithm found solutions that have SD values that are 2% to 39% lower than that found by the truncated traversal algorithm. In addition, the time requirements of the genetic algorithm were comparable or better than the truncated traversal method. For pathway 00010, the genetic algorithm generated on an average 8.06% improvement over the traversal method. The maximum improvement in these experiments was 39%. We have similar comparative gains for other pathways.

One can argue that the effectiveness of the traversal method can be limited due to the limited number of levels. To evaluate the trade off between the accuracy and running time of the truncated algorithm better, we increase the depth of the traversal algorithm until it spends at least as much time as the genetic algorithm for all the pathways. For this purpose, we tested the truncated algorithm for 23 levels. Table 4-5 shows the comparison between the genetic
algorithm and the traversal method with 23 levels for the largest pathways. From Table 4-5, the running time of truncated algorithm with the additional 3 levels is up to an order of magnitude higher. However, the accuracy only improved by a small amount and is much lower than the genetic algorithm. It is worth noting that the genetic algorithm required considerably less time for most of these cases. Therefore, the genetic algorithm is a good choice for the enzymatic target identification problem for very large pathways.

We apply our genetic algorithm to the large metabolism. For the energy metabolism and amino acid metabolism, our genetic algorithm costs less than one hour. Even for the whole metabolism, our genetic algorithm runs less than twelve hours. Details are in Table 4-6.

In genetic algorithm, there exist several parameters. First, we test the impact of mutation rate on the SD values. Table 4-7 presents the average SD values with different mutation rates in several pathways. In Table 4-7, the average SD values do not vary much with different mutation rate. Thus, we select \( \delta = 0.04 \) as we observed slightly better SD values with this choice in our experiments.

In Step 1 of the genetic algorithm, to estimate \( \lambda \), we search the first 10 levels of the search space to get top 20 best solutions. Table 4-8 presents the estimated \( \lambda \) values and the cost time for different levels for the Glycolysis/Gluconeogenesis pathways. As we increase the number of levels the value of \( \lambda \) increases monotonically. This is because each level suggests knocking out new enzymes on top of already knocked out ones. However, although the number of solutions increase exponentially as we look at deeper levels, the estimated value of \( \lambda \) grows slowly. Between levels 8 and 10 we estimate the actual value of \( \lambda \) correctly.

Table 4-8 also shows that traversing more levels increases the running time exponentially. In practice, we have observed that there is no improvement in the SD value found by our genetic algorithm as we go traverse more levels to estimate \( \lambda \). Also the running time grows exponentially
with the number of levels. Therefore, in order to balance efficiency and accuracy, we traverse only 10 levels for these practical purposes.

4.4.2.3 Comparison to an existing genetic algorithm method

Our last experiment compares the performance of our genetic algorithm to a recent work. Patil et al. presented an evolutionary programming method for finding optimal gene deletion strategies [72]. Their method uses genetic algorithm to generate a population of random solutions. This method can be applied to our enzymatic target identification problem directly. We implement Patil’s method in C++. We compare our genetic algorithm with Patil’s method in accuracy. Table 4-9 shows the SD value of Patil’s method and our genetic algorithm for seven pathways and large metabolism. $D_{Pat}$ and $D_{ga}$ denote the average SD for Patil’s and our method respectively. Similarly, $NE_{Pat}$ and $NE_{ga}$ denote the average number of enzymes knocked out using Patil’s method and our genetic algorithm respectively.

Our method consistently outperforms Patil et al.’s method in all test cases. The SD value of Patil et al.’s method is 40% to 3000% more than that of our method. One obvious question is whether our method finds better SD values at the expense of knocking out more enzymes than Patil et al.’s method. The last two columns of Table 4-9 show that Patil et al.’s method is knocking out up to 20.9 times more enzymes than our algorithm on the average. We conclude that our algorithm is superior both in terms of finding a solution close to the goal state and the cost in doing that.

The reasons behind the success of our method over Patil et al.’s is that our method randomly knocks out each enzyme using a different probability distribution. This distribution depends on the likelihood that the each enzyme is a part of a good solution. Furthermore, unlike Patil et al.’s method, our crossover strategy optimizes the child solution created at each generation.
4.5 Discussion

The goal of enzymatic target identification problem is to identify the set of enzymes whose knockouts lead to a steady state of the metabolic pathway that is close to a goal or desired state. We develop a novel distance measure, State-distance (SD) that measures the damage of the knockouts of a set of enzymes as a function of the deviation of the entry in the steady state after their knockouts from that in the goal state. Using this measure, we develop two algorithms that are based on search space traversal and genetic algorithms.

Experiments using the metabolic pathways of *H. sapiens* and *E.coli* from KEGG show that our algorithms can be useful for numerous application including metabolic engineering and biomedicine. Our traversal method is effective for pathways with up to 30-35 enzymes. Our genetic algorithm is effective for arbitrarily large pathways.

In Section 4.3, we discuss the enzyme deletion strategies for the enzymatic target identification problem. However, genetic manipulations can lead to partial reduction of the enzyme concentrations rather than knocking it out entirely. One way to deal with this is to represent an enzyme status by an integer variable, \{0, 1, 2, \ldots, m\} that shows the level of enzyme activity. Here, the enzyme can exist in \(m + 1\) status. 0 means that the enzyme is inactive. 1 to \(m\) presents the activity level of the enzyme with \(m\) being the highest activity. In this situation, our methods, traversal method and genetic algorithm can still work. Genetic algorithm can be used to this situation directly. For traversal method, the search space is no longer a binary tree. For each node, it has \(m + 1\) children which denotes the \(m + 1\) enzyme status. The filtering and prioritization strategy can also be applied to this situation. We expect that this will hurt the running time of the traversal method significantly.
Figure 4-1. Graph representation of a metabolic pathway with three reactions $R_1$, $R_2$, $R_3$, two enzymes $E_1$ and $E_2$, and nine compounds $C_1, \cdots, C_9$. Dashed lines show the impact of knocking out enzyme $E_1$.

Figure 4-2. Flux distribution of a hypothetical pathway.

Table 4-1. An example showing the generation of a child $Ch$ from two hypothetical parents $F$ and $M$ for a pathway that contains nine enzymes. $?$ denotes an undecided value.
Figure 4-3. Upper bound to the expected number of undecided enzymes for different $\lambda$ and pathway sizes ($N = |E|$). The vertical bars show the standard deviation in each direction.

Table 4-2. The expected number of undecided enzymes for different pathways. #E denotes the number of enzymes.

<table>
<thead>
<tr>
<th>Metabolic pathway</th>
<th>#E</th>
<th>Exp. num.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valine, leucine and isoleucine degradation</td>
<td>24</td>
<td>3.40</td>
</tr>
<tr>
<td>Glycolysis/Gluconeogenesis</td>
<td>27</td>
<td>2.69</td>
</tr>
<tr>
<td>Glycine, serine and threonine metabolism</td>
<td>32</td>
<td>7.15</td>
</tr>
<tr>
<td>Purine metabolism</td>
<td>52</td>
<td>1.72</td>
</tr>
</tbody>
</table>

Table 4-3. Metabolic pathways from KEGG that are used in our experiments in this chapter.

<table>
<thead>
<tr>
<th>KEGG Id</th>
<th>Metabolic pathway</th>
<th>#E</th>
<th>#R</th>
<th>#C</th>
</tr>
</thead>
<tbody>
<tr>
<td>00980</td>
<td>Metabolism of xenobiotics by cytochrome P450</td>
<td>7</td>
<td>49</td>
<td>57</td>
</tr>
<tr>
<td>00670</td>
<td>One carbon pool by folate</td>
<td>17</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td>00220</td>
<td>Urea cycle and metabolism of amino groups</td>
<td>21</td>
<td>22</td>
<td>27</td>
</tr>
<tr>
<td>00310</td>
<td>Lysine degradation</td>
<td>21</td>
<td>25</td>
<td>28</td>
</tr>
<tr>
<td>00280</td>
<td>Valine, leucine and isoleucine degradation</td>
<td>24</td>
<td>33</td>
<td>32</td>
</tr>
<tr>
<td>00010</td>
<td>Glycolysis/Gluconeogenesis</td>
<td>27</td>
<td>31</td>
<td>25</td>
</tr>
<tr>
<td>00260</td>
<td>Glycine, serine and threonine metabolism</td>
<td>32</td>
<td>33</td>
<td>37</td>
</tr>
<tr>
<td>00230</td>
<td>Purine metabolism</td>
<td>52</td>
<td>92</td>
<td>65</td>
</tr>
<tr>
<td>-</td>
<td>Energy Metabolism</td>
<td>50</td>
<td>46</td>
<td>60</td>
</tr>
<tr>
<td>-</td>
<td>Amino Acid Metabolism</td>
<td>195</td>
<td>317</td>
<td>305</td>
</tr>
<tr>
<td>-</td>
<td>whole metabolism</td>
<td>640</td>
<td>1176</td>
<td>1067</td>
</tr>
</tbody>
</table>
Figure 4-4. Glycolysis/Gluconeogenesis for *E.coli*. The enzymes highlighted in green are the enzymes that exist in *E.coli.*
Figure 4-5. Phenylalanine, tyrosine and tryptophan biosynthesis for *E.coli*. The enzymes highlighted in green are the enzymes that exist in *E.coli*.

Figure 4-6. Average SD values of the traversal method and exhaustive search for different pathways over multiple queries.
Figure 4-7. The average running time (in seconds) of the traversal method and exhaustive search over multiple queries for different pathways.

Figure 4-8. Distribution of the number of enzymes for Urea cycle and metabolism of amino groups.

Table 4-4. Comparison of the truncated traversal method (maximum number of levels = 20) and the genetic algorithm for different pathways. ? denotes that no SD values can be computed within one day.

<table>
<thead>
<tr>
<th>Pathways</th>
<th>SD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D₀</td>
<td>D₂₀</td>
</tr>
<tr>
<td>00980 (#E 7)</td>
<td>0.808</td>
<td>0.75</td>
</tr>
<tr>
<td>00220 (#E 21)</td>
<td>0.970</td>
<td>0.882</td>
</tr>
<tr>
<td>00280 (#E 24)</td>
<td>2.611</td>
<td>2.596</td>
</tr>
<tr>
<td>00010 (#E 27)</td>
<td>2.687</td>
<td>2.556</td>
</tr>
<tr>
<td>00260 (#E 32)</td>
<td>0.837</td>
<td>0.819</td>
</tr>
<tr>
<td>00230 (#E 52)</td>
<td>11.218</td>
<td>1.160</td>
</tr>
<tr>
<td>00230+00790 (#E 61)</td>
<td>6.848</td>
<td>1.497</td>
</tr>
<tr>
<td>00230+00030 (#E 67)</td>
<td>8.590</td>
<td>3.682</td>
</tr>
<tr>
<td>00230+00340 (#E 67)</td>
<td>5.504</td>
<td>1.274</td>
</tr>
<tr>
<td>00230+00260 (#E 84)</td>
<td>6.342</td>
<td>1.322</td>
</tr>
</tbody>
</table>
Table 4-5. The average SD value and the running time of the truncated traversal algorithm (with maximum number of levels = 23) and the Genetic Algorithm. The running time is reported in seconds.

<table>
<thead>
<tr>
<th>Pathways</th>
<th>SD $D_{23}$</th>
<th>$D_{ga}$</th>
<th>Time $T_{23}$</th>
<th>$T_{ga}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00280 (#E 24)</td>
<td>2.581</td>
<td>2.581</td>
<td>144</td>
<td>2</td>
</tr>
<tr>
<td>00010 (#E 27)</td>
<td>2.525</td>
<td>2.370</td>
<td>987</td>
<td>52</td>
</tr>
<tr>
<td>00260 (#E 32)</td>
<td>0.819</td>
<td>0.797</td>
<td>1042</td>
<td>38</td>
</tr>
<tr>
<td>00230 (#E 52)</td>
<td>1.568</td>
<td>1.084</td>
<td>1366</td>
<td>153</td>
</tr>
<tr>
<td>00230+00790 (#E 61)</td>
<td>1.368</td>
<td>0.920</td>
<td>335</td>
<td>573</td>
</tr>
<tr>
<td>00230+00030 (#E 67)</td>
<td>3.552</td>
<td>3.159</td>
<td>3310</td>
<td>410</td>
</tr>
<tr>
<td>00230+00340 (#E 67)</td>
<td>1.242</td>
<td>1.064</td>
<td>828</td>
<td>168</td>
</tr>
<tr>
<td>00230+00260 (#E 84)</td>
<td>1.271</td>
<td>0.996</td>
<td>974</td>
<td>494</td>
</tr>
</tbody>
</table>

Table 4-6. The average running time of our Genetic Algorithm for large metabolism. The running time is reported in seconds.

<table>
<thead>
<tr>
<th>Metabolism</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy Metabolism (#E 50)</td>
<td>90</td>
</tr>
<tr>
<td>Amino Acid Metabolism (#E 195)</td>
<td>2794</td>
</tr>
<tr>
<td>whole metabolism (#E 640)</td>
<td>43262</td>
</tr>
</tbody>
</table>

Table 4-7. The average SD values with different mutation rates $\delta$ in several pathways.

<table>
<thead>
<tr>
<th>Pathways</th>
<th>$\delta = 0.02$</th>
<th>$\delta = 0.04$</th>
<th>$\delta = 0.06$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010</td>
<td>2.370</td>
<td>2.370</td>
<td>2.370</td>
</tr>
<tr>
<td>00230</td>
<td>1.131</td>
<td>1.084</td>
<td>1.102</td>
</tr>
</tbody>
</table>

Table 4-8. The estimated $\lambda$ values and the cost time for different levels for the Glycolysis / Gluconeogenesis pathway. The average number of knocked out enzymes in the optimal solution of this pathway is 2.2.

<table>
<thead>
<tr>
<th>levels</th>
<th>estimated $\lambda$ value</th>
<th>time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1.55</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>2.19</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>2.52</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>3.08</td>
<td>64</td>
</tr>
<tr>
<td>14</td>
<td>3.53</td>
<td>119</td>
</tr>
</tbody>
</table>
Table 4-9. The average SD value and the running time of Patil’s method and our Genetic Algorithm. The running time is reported in seconds.

<table>
<thead>
<tr>
<th>Pathways</th>
<th>SD</th>
<th>NE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(D_{Pat})</td>
<td>(D_{ga})</td>
</tr>
<tr>
<td>00280 (#E 24)</td>
<td>3.572</td>
<td>2.545</td>
</tr>
<tr>
<td>00010 (#E 27)</td>
<td>4.207</td>
<td>2.370</td>
</tr>
<tr>
<td>00260 (#E 32)</td>
<td>1.039</td>
<td>0.797</td>
</tr>
<tr>
<td>00230 (#E 52)</td>
<td>5.968</td>
<td>1.084</td>
</tr>
<tr>
<td>00230+00030 (#E 67)</td>
<td>13.996</td>
<td>3.159</td>
</tr>
<tr>
<td>00230+00340 (#E 67)</td>
<td>4.698</td>
<td>1.064</td>
</tr>
<tr>
<td>00230+00260 (#E 84)</td>
<td>5.754</td>
<td>0.996</td>
</tr>
<tr>
<td>Amino Acid Metabolism</td>
<td>2.408</td>
<td>0.737</td>
</tr>
<tr>
<td>whole metabolism</td>
<td>60.904</td>
<td>1.962</td>
</tr>
</tbody>
</table>
CHAPTER 5
ENZYMATIC TARGET IDENTIFICATION WITH DYNAMIC STATES

In Chapter 2, Chapter 3 and Chapter 4, we discuss the strain optimization based on the steady state. That is, we discuss the problem to identify the enzyme set whose knockouts lead the final steady state to satisfy the optimal constrains. In details, in chapter 4, we discuss the problem identify the set of enzymes whose knockouts lead the metabolism to a state that is close to a given goal state. In chapter 3, we discuss the problem to identify the enzyme set whose knockouts lead the final state to obtain the optimal objective function values. In chapter 2, we discuss the problem to find the set of enzymes, whose knockouts lead the final state to stop the production of a given set of target compounds, while eliminating minimal number of non-target compounds.

However, the biological system needs quite a few time, e.g. several minutes or hours even days from one steady state to another state. The process between two steady states is significant. If there exists two different “pathes” from the start state to the final state, the influence of these two pathes on the whole biological system may be significant different. If we want to increase the blood sugar concentration of a biological system, one path is to gradually enhance the blood sugar concentration. Another way is to aggrandize the blood sugar concentration with a sharp curve then decrease it to the goal concentration. The first method may cost much time to reach the goal however the second method may bring dangerous side-effect. For example, the sugar concentration may reach a serious high extent and lead the organism to die. Thus, it is necessary to consider the transient path when we identify the enzyme set to change the state of the biological system.

5.1 Motivation and Problem Definition

Metabolic networks show how enzymes and compounds interact through reactions. Enzymes catalyze reactions that transform a set of compounds into another set of compounds. The “state” of a metabolic pathway can be expressed as a vector, where each entry denotes the
concentration of a compound [106] or a flux [71] in the network at a given time. “Steady state” is the state that remains unchanged over time. A number of methods have been developed to compute the steady state of a given metabolism for different network models (See Voit’s book for an overview of these models and methods [107]).

When an enzyme is inhibited, it cannot catalyze the reactions it is responsible from. As a result, the production of a set of compounds can change. Following from this observation, the “enzymatic target identification problem” aims to identify the set of enzymes whose knockouts lead the steady state of the metabolism close to a given goal state [93]. In the literature, this problem has been considered for a number of network models including Boolean [93, 94, 96] and stoichiometric [93] models.

The enzymatic target identification problem above and the existing solutions for this problem have a serious shortcoming. This is because they consider only the steady state of the given metabolic network. However, the biological system reaches to the steady state over a period of time, after a sequence of changes to its current state. In other words, reaching to a steady state is a process that involves observing many other states. Inhibition of two different sets of enzymes can lead to the same steady state in two different ways.

Figure 5-1 illustrates this on a hypothetical example by focusing on the concentration of a single compound. Both manipulations lead to the same steady state in the same time, however they do this through different sequence of states. The area between the two plots show the difference between two dynamic states $P_1$ and $P_2$ of one compound. The pattern of states observed while reaching a steady state is a critical information that is ignored by current enzymatic target identification methods. For instance, if the blood sugar concentration increases (or decreases) too rapidly or if it becomes over (or below) a threshold, this pattern may cause dangerous side-effects. Thus, it is necessary to consider the dynamic process while solving the enzymatic target identification problem.
We use the term “dynamic state” to describe the sequence of states observed over time. We consider the dynamic state as a vector where the \( i \)th entry denotes the state of the network at the \( i \)th time instance. There are several models to describe the dynamic state of the biological system. We discuss them in Section 5.2. However, to the best of our knowledge, there is no published study that considers the dynamic states for the enzyme target identification problem. Following from this observation, we focus the following problem in this paper.

**Problem statement.** (Dynamic enzymatic target identification problem) Given a goal dynamic state of a set of compounds in a metabolic network, we aim to identify the set of enzymes whose inhibition leads to the dynamic state of these compounds as close to the goal dynamic state as possible.

**Contributions:** We address the dynamic enzymatic identification problem in this paper. In order to solve this problem, it is necessary to provide a measure to evaluate the difference between two dynamic states. We consider three alternative measures to cover a broad range of distance functions. The first one measures the area between the curves defined by the two dynamic states. For example, in Figure 5-1, the shaded region corresponds to the distance between dynamic states \( P_1 \) and \( P_2 \). This measure makes the simplifying assumption that the metabolism reaches to the steady states in the same amount of time when different enzyme sets are inhibited. Our second measure eliminates this assumption by allowing the dynamic states stretch along the time dimension by arbitrary amounts.

Figure 5-2 depicts this on a simple pattern representing a hypothetical dynamic state. We build a dynamic programming solution to compute this distance function. Our third measure further generalizes the distance measure by allowing scaling and shifting of dynamic states (see Figure 5-2). This generalization is motivated by the fact that two different patterns can have similar trends while having significantly different values. We build an iterative algorithm that
finds the distance between two dynamic states when one of them is allowed to stretch along the
time domain as well as shift/scale on the dimensions corresponding to the flux values.

We develop a branch and bound strategy that uses these distance measures to solve the
dynamic enzymatic target identification problem. The basic search strategy follows the standard
OPMET algorithm for Boolean networks [96]. It considers the search space as a hierarchical tree
where each node of this tree corresponds to a set of enzymes to be inhibited. The fundamental
difference is that each node now corresponds to a dynamic state. This difference introduces
additional computational cost on the basic OPMET algorithm in two different ways. First,
computing the dynamic state can be significantly harder than just computing the steady state.
Second, the time complexity of computing the distance between two dynamic states can be
quadratic in the number of time instances, while that for the steady states only is constant. We
deal with these two challenges by develop a partitioning strategy as follows. Instead of creating
the entire dynamic state, we quickly create a short prefix of it. We then use this prefix to compute
a lower bound to the distance between the entire dynamic states. This bound helps us to prune
unpromising solutions in the search space. We extend this prefix by computing more values in the
dynamic state if needed. As longer prefix of the dynamic state becomes available, we improve the
lower bound using the new values for further pruning of the search space.

Our experiments demonstrate that our method is 85-100% accurate when a single enzyme
is inhibited. It is 65-75% accurate when two enzymes are inhibited. Our partitioning strategy
improves the running time of our algorithm over the basic OPMET strategy by a factor of 4.4 to
14.

The rest of the chapter is organized as follows. Section 5.2 discusses the related work.
Section 5.3 defines three dynamic distance functions to measure the difference between two
dynamic states. Section 5.4 discusses how we search the set of possible solutions and our
partitioning strategy. Section 5.5 presents experimental results. Section 5.6 concludes the paper.
5.2 Related Work

Existing methods for enzymatic target identification use three models to explain how metabolism works, namely Boolean, linear and non-linear models. A detailed discussion of these methods is available at Song et.al., [93]. We briefly summarize some of them here. Sridhar et. al and Song et al. considered a Boolean model of the enzymatic target identification problem [94–96]. In their version, each entry of the state denotes whether the corresponding enzyme is present or not. Flux Balance Analysis, (FBA) [7, 29, 48, 90] uses a set of linear equations to describe a given metabolic network. Methods, that use this system often use Linear Programming (LP), Integer Linear Programming (ILP) or genetic algorithms to solve the enzymatic target identification problem. OptKnock [9] and the method by Patil et.al., [72] are two examples to the algorithms in this class [9]. Though the linear model works well for some cases, there exist more complex non-linear models to describe the metabolism. These non-linear models can simulate the cell system better than the linear model. For example, S-systems [86, 107] and Generalized Mass Action (GMA) model [73, 107] belong to this category. Most of the existing methods are suited well for linear models. Thus they do not work when these more complex models are used to compute the steady state of the metabolic network. Song et al. proposed a genetic algorithm solution for non-linear models [93]. All the above mentioned methods consider only the steady state of the metabolism. They ignore the sequence of states the underlying network visits while reaching the steady state. As a result, although their solution may be optimal at the steady state, the intermediate states of their solutions can be undesirable.

There are several models that simulate the dynamic state of a given metabolic network. For example, Dynamic Flux Balance Analysis (DFBA) extended the traditional FBA to describe the change rate of the fluxes over a period of time [62, 64]. DFBA incorporates the time parameter which can predict the metabolite concentrations. It considers the entire time period and builds a non-linear programming problem. It separates the time into several intervals. For each interval it
employs a linear-programming method to estimate the flux values during that interval. Integrated dynamic FBA (idFBA) simulates the integrated system including signaling, metabolic and regulatory networks [56]. Similar to DFBA, idFBA separates the time into several intervals. For each interval, it applies FBA to compute the flux values. From these values, it decides which reactions will take place during the next interval. Integrated FBA (iFBA) model builds a dynamic simulation among metabolic, regulatory and signaling networks [16] along the same lines as DFBA and idFBA. It first separates the time to several intervals. It then applies ordinary differential equations (ODEs) and Boolean regulatory model to constrain the FBA linear programming problem. It updates the biomass and external metabolite concentrations for use in subsequent time steps. All these methods aim to find the dynamic state of a given metabolic network. They however do not consider the dynamic enzymatic target identification problem, which is the focus of this paper.

In this paper, we use Generalized Mass Action (GMA) model [73, 107] to model metabolic networks as this is one of the most accurate models. It is a generalization of the S-systems of equations. We employ DFBA to compute the dynamic state of the given metabolic network. It is worth noting that one can replace these with another mathematical model for computing the dynamic state with little or no change to the rest of this paper.

5.3 Calculating the Distance between Transient Paths

In order to solve the dynamic enzymatic target identification problem, the first task is to measure the distance between two dynamic states. We first present the basic notation we use in this paper in Section 5.3.1 We then describe three distance measures. Section 5.3.2 discusses the exact distance. Section 5.3.3 discusses how we can measure the distance when we allow flexibility to the amount of time it takes for the metabolic network to reach the steady state. Section 5.3.4 describes the pattern distance that allows the dynamic state to be scaled of shifted by arbitrary amounts.
5.3.1 Notation

We start by describing the notation that will be used in the rest of the paper. Assume that a given metabolic network contains \(M\) fluxes. Let us denote the fluxes in the given metabolic network with \(X_1, X_2, \ldots, X_M\). Let us denote the amount of instantaneous change in \(X_i\) with \(\dot{X}_i\). Assume that \(X_i\) takes part in \(N\) reactions. The GMA model expresses each flux \(X_i\) using the equation

\[
\dot{X}_i = \sum_{k=1}^{N} \gamma_{i,k} \prod_{j=1}^{M} X_{f_{i,j,k}}.
\]

Here, the constant \(\gamma_{i,k}\) is the speed of the \(k\)th reaction and the constant \(f_{i,j,k}\) is the rate of contribution of the \(j\)th flux to the \(k\)th reaction. For a given set of equations of this form, when the initial values (i.e., the value at time zero) of all the \(X_i\)s are provided, we can compute the value of each \(X_i\) at a given time \(t\) by simulating this process or by solving these equations.

We represent the dynamic state of \(X_i\) using two vectors \(X_{i,T}\) and \(X_{i,V}\). The first one, \(X_{i,T} = [x_{t_{i,1}}, x_{t_{i,2}}, \ldots, x_{t_{i,n}}]\), shows the time points at which we compute the value of \(X_i\) until the metabolic network reaches to the steady state (\(\forall j, x_{t_{i,j}} < x_{t_{i,j+1}}\)). The second one, \(X_{i,V} = [x_{v_{i,1}}, x_{v_{i,2}}, \ldots, x_{v_{i,n}}]\), shows the values of \(X_i\) at these time points. The collection of \(X_{i,T}\) and \(X_{i,V}\) for all \(X_i\)s is the dynamic state of the given metabolic network.

In order to simplify our notation, in the rest of the paper we will focus on a particular \(X_i\) and represent its dynamic state with the vector pair \(X = (X_V, X_T)\), where \(X_V = [x_{v_{1}}, x_{v_{2}}, \ldots, x_{v_{n}}]\) and \(X_T = [x_{t_{1}}, x_{t_{2}}, \ldots, x_{t_{n}}]\) by dropping the subscript \(i\).

We use a similar notation to denote the goal dynamic state for \(X_i\) using a vector pair \(G = (G_V, G_T)\). The first vector \(G_V = [g_{v_{1}}, g_{v_{2}}, \ldots, g_{v_{m}}]\) denotes the goal values for \(X_i\) until it reaches to steady state. The second vector \(G_T = [g_{t_{1}}, g_{t_{2}}, \ldots, g_{t_{m}}]\) stores the ideal time for \(X_i\) to have each of these values.
5.3.2 Exact Distance

Our first distance function measures the area between the two curves defined by the two given dynamic states. We name this measure the “Exact distance”. The area of the shaded region in Figure 5-1 shows the exact distance on a hypothetical example.

Given a goal dynamic state \((G_V, G_T)\) and the dynamic state \((X_V, X_T)\), this measure assumes that \(G_T\) and \(X_T\) has exactly the same entries. Although this is a strong assumption, we ensure that this is satisfied as follows. For all the values in \(G_T\) that are missing in \(X_T\) we insert those values in \(X_T\) and compute the value of \(X_V\) at those new time points using the GMA model as described in Section 5.3.1. For all the values in \(X_T\) that are missing in \(G_T\) we insert those values in \(G_T\). We follow a different procedure to compute the \(G_V\) values for the new time points as there is no metabolic network available for the goal state. We simply use linear interpolation of the existing time points and values in \(G_T\) and \(G_V\) to compute the new entries for \(G_V\).

Once we ensure the equality of the vectors \(G_T\) and \(X_T\), we compute the exact distance between the two dynamic states as:

\[
\sum_i \left( |g_{V_i} - x_{V_i}| + |g_{V_{i+1}} - x_{V_{i+1}}| \right) \times \frac{|x_{T_{i+1}} - x_{T_i}|}{2}.
\]

Geometrically, this equation approximates to the area between the two curves by splitting the region between them into trapezoids between consecutive time intervals. Each summation term in the above formulation corresponds to the area of one trapezoid.

The exact distance measure can be computed efficiently in \(O(n)\) time, where \(n\) is the number of time points. It however is restrictive as it returns a small value only if the two dynamic states have the similar values at similar times. In the following sections, we will discuss how we relax this constraint.
5.3.3 Time-warping Distance

The exact distance function requires the two states to have similar flux values for the same period of time. This restriction, however, will fail to find the similarities between two pathways under the following scenario. It is possible to have two different metabolisms that have similar sets of reactions but the speed at which the reactions take place differ. In such scenarios, the dynamic states of the two metabolisms can have similar values but the time it takes to reach these values will differ. We say that such dynamic states are “stretched” along the time axis. The left-most dynamic states in Figure 5-2 illustrate a pair of dynamic states with one stretched.

Our second distance function uses time-warping distance function to address problems caused by the metabolic manipulations that change the speed of the reactions. Dynamic time warping technique is widely used in data mining [4, 51, 114], gesture recognition [31], robotics [87], speech processing [79]. Here, we apply dynamic time warping technique to evaluate the distance between two dynamic states. Assume that we are given a goal dynamic state $G = (G_V, G_T)$ and the dynamic state $X = (X_V, X_T)$ as described in Section 5.3.1. Time-warping distance aligns the two dynamic states to find a mapping between their time points. Figure 5-3 illustrates the alignment of two hypothetical dynamic states and how the time-warping distance stretches them to bring their similar values close to each other. Once the two dynamic states are aligned, this measure computes the distance as the area between their curves after the dynamic state $(X_V, X_T)$ is stretched along the time dimension to match its time points to those of the goal dynamic state.

We use dynamic programming method to align the two dynamic states as follows. Let us denote the distance between the $i$th value of $G$ and the $j$th value of $X$ with $d(G, i, X, j)$. We discuss computation of $d(G, i, X, j)$ later in this section.

Let us also denote the distance between the first $i$ values of $G$ and the first $j$ values of $X$ after their optimal alignment (i.e., alignment with smallest distance) with $\gamma(i, j)$. We compute
\( \gamma(i, j) \) as
\[
d(G, i, X, j) + \min\{\gamma(i - 1, j - 1), \gamma(i - 1, j), \gamma(i, j - 1)\}.
\]
The first of the three scenarios in the \( \min \) function correspond to the case that the first \( i - 1 \) values of \( G \) is aligned to the first \( j - 1 \) values of \( X \). The second scenario corresponds to the case that the first \( i - 1 \) values of \( G \) is aligned to the first \( j \) values of \( X \). In other words \( X \) is stretched along the time axis to match its \( j \)th entry to both \( i \)th and \((i - 1)\)th entries of \( G \). The last scenario corresponds to the case that the first \( i \) values of \( G \) is aligned to the first \( j - 1 \) values of \( X \). That is \( X \) is contracted along the time axis to match both of its \((j - 1)\)th and \( j \)th entries to the \( i \)th entry of \( G \). We compute the time-warping distance between \( G \) and \( X \) as
\[
\text{Dis}(G, X) = \sqrt{\gamma(m, n)}.
\]
We skipped two important details in computation of the time-warping distance so far. The first one is the distance between two time points of the two dynamic states \( d(G, i, X, j) \). We compute this value as
\[
(gv_i - xv_j)^2 \cdot \frac{(gt_{i+1} - gt_{i-1})}{2(gt_m - gt_1)} + \frac{xt_{j+1} - xt_{j-1}}{2(xt_n - xt_1)} / 2.
\]
Briefly, this function approximates to the area between the two curves when the \( j \)th interval of \( X \) is moved to the \( i \)th interval of \( G \) at that time interval. It approximates this area as the sum of two triangles. The first triangle defines the area in the time interval \([gt_{i-1}, gt_{i+1}]\). The second one defines that in the time interval \([xt_{j-1}, xt_{j+1}]\).

The second detail we omitted is the initialization of the \( \gamma(i, j) \) matrix. We initialize this matrix for the cases when at least one of the two dynamic states contain no values as follows.

Case 1: \( \gamma(0, 0) = 0 \)

Case 2: \( \gamma(i, 0) = \gamma(i - 1, 0) + gv_i^2 \cdot \frac{(gt_{i+1} - gt_{i-1})}{4(gt_m - gt_1)} \)

Case 3: \( \gamma(0, j) = \gamma(0, j - 1) + xv_j^2 \cdot \frac{(xt_{j+1} - xt_{j-1})}{4(xt_n - xt_1)} \)
The first case indicates that both dynamic states have no value. The last two cases indicate that one of the two dynamic states have values. In this case the distance is the area under the nonempty curve.

5.3.4 Pattern Distance

We have discussed how to deal with the differences in the dynamic states due to differences in the reaction speeds using time warping distance. Two similar networks, however, can have different dynamic states even under the time warping distance when the values of one dynamic state is shifted or scaled. The dynamic states in the middle and on the right of Figure 5-2 illustrate this event. This kind of alterations often happens due to external factors, such as increasing or decreasing the concentrations of a set of compounds. It can also be caused by inaccuracies in measurements. Our last distance measure computes the smallest distance between two dynamic states when one of them is shifted, scaled and stretched to make as close to the other as possible. We call this measure the “pattern distance”.

Assume that we are given a goal dynamic state $G = (G_V, G_T)$ and the dynamic state $X = (X_V, X_T)$ as described in Section 5.3.1. Let $\alpha$ and $\beta$ be two real numbers. Scaling the goal state with $\alpha$ moves $G$ to $(\alpha \cdot G_V, G_T)$. Similarly, shifting $G$ by $\beta$ moves $G$ to $(G_V + \beta, G_T)$.

If we are given the optimal scaling and shifting coefficients $\alpha$ and $\beta$, one can find the distance between $G$ and $X$ as the time warping distance between $X$ and $(\alpha \cdot G_V + \beta, G_T)$ using the dynamic programming method in Section 5.3.3. This, however, is not feasible as the values of $\alpha$ and $\beta$ however are not available. Inversely, given an alignment of the time points of $G$ and $X$, through algebraic manipulations, we can compute the scaling and shifting coefficient $\alpha$ and $\beta$ that will minimize the distance between them. We discuss how this can be done later in this section. However, this is not feasible as well as their alignment is unknown.

We develop an iterative algorithm to solve this problem (see Algorithm 1). Each iteration of our algorithm contains two phases. In the first phase, we fix the values of $\alpha$ and $\beta$ and optimize
Algorithm 5.1 *Iterative Algorithm for Pattern Distance*

**Input:** Dynamic states $G$ and $X$

Initialize $\alpha = 1$ and $\beta = 0$.

Phase 1: Use the dynamic programming method of Section 5.3.3 to find the alignment of $G$ and $X$.

Phase 2: Compute the $\alpha$ and $\beta$ values that minimize the distance for the current alignment. Update $G_V$ as $\alpha G_V + \beta$.

Go to phase 1 until a given number iterations or no distance improvement.

In the second phase, we fix the alignment and compute the optimal $\alpha$ and $\beta$ values for that alignment. Each phase guarantees that the resulting distance is less than or equal to that in the previous phase/step. Thus, the distance value our algorithm computes decreases monotonically throughout iterations. Thus, the algorithm is guaranteed to converge to a minimal distance. Next, we focus on Phase 2 of our algorithm and explain how to compute the optimal values of $\alpha$ and $\beta$.

At the end of Phase 1 of Algorithm 5.1, we have an alignment between $G$ and $X$. Recall that, this alignment can map one time point of one dynamic state to several consecutive time points of the other. For instance, Figure 5-3 shows an illustration of two dynamic states. Each square (circle) shows a time point. The time points of one of the dynamic states is labeled with numbers 1 to 6. Those of the other are labeled with letters a to e. One of the dynamic states is shifted down slightly to make it easier to visualize. Dashed lines of top figure show the actual pattern of the dynamic states. Solid lines show the time-warping alignment of the two states. In the bottom figure, the time points that are aligned with multiple time points from the other state are duplicated and stretched to match them along the time axis. Time point “a” is mapped to time points “1” and “2”. To make our notation for the rest of this section simple, we duplicate such time points (see the bottom figure in Figure 5-3). Let us call the resulting dynamic states as $G'' = \ldots$
\((G'_V, G'_T)\) and \(X' = (X'_V, X'_T)\), where \(G'\) and \(X'\) both have same number of \((n)\) time points. In other words, the \(i^{th}\) time point of \(G'\) is aligned to the \(i^{th}\) time point of \(X'\). Formally, we will use the following notation for \(G'\) and \(X'\):

\[
X'_V = [xv'_1, xv'_2, \ldots, xv'_n]
\]

\[
X'_T = [xt'_1, xt'_2, \ldots, xt'_n]
\]

\[
G'_V = [gv'_1, gv'_2, \ldots, gv'_n]
\]

\[
G'_T = [gt'_1, gt'_2, \ldots, gt'_n]
\]

We define two functions over time intervals of \(G'\) and \(X'\) as

\[
\omega(gt'_i) = \frac{gt'_{i+1} - gt'_{i-1}}{2(gt'_n - gt'_1)} \quad \text{and} \quad \omega(xt'_i) = \frac{xt'_{i+1} - xt'_{i-1}}{2(xt'_n - xt'_1)}.
\]

The pattern distance between \(G\) and \(x\) is

\[
Dis(G, X) = \sqrt{\sum_i (\alpha \cdot gv'_i + \beta - rv'_i)^2 \cdot (\omega(gt'_i) + \omega(rt'_i))/2}.
\]

We solve the following equations to get the values of \(\alpha\) and \(\beta\).

\[
\frac{\partial Dis(G, X)}{\partial \alpha} = 0, \quad \frac{\partial Dis(G, X)}{\partial \beta} = 0.
\]

We skip the individual algebraic steps on how we solve these equations and present the final result here. The values of \(\alpha\) and \(\beta\) are

\[
\alpha = \frac{C - B \cdot D}{A - B^2} \quad \text{and} \quad \beta = D - \alpha \cdot B,
\]

where

\[
A = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot (gv'_i)^2,
\]

\[
B = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot gv'_i,
\]

\[
C = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot (gt'_i)^2.
\]
\[ C = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot xv'_i \cdot gv'_i, \text{ and} \]
\[ D = \sum_i (\omega(gt'_i) + \omega(xt'_i)) \cdot xv'_i. \]

5.4 Methods

So far we have discussed how to compute the distance between two dynamic states. The next step is to identify the set of enzymes whose elimination leads to the dynamic state as close to the goal dynamic state as possible with respect to this distance function. One way to solve this problem is to exhaustively traverse all possible subsets of enzymes, examine the distance value and pick the lowest one. However, this is not feasible as the number of subsets is exponential in the number of enzymes.

OPMET solves this problem using a branch and bound algorithm for a simplified Boolean model of metabolic networks [96]. At this paragraph, we take a detour and summarize the OPMET algorithm as we use it in this paper. Unlike the problem considered in this paper, OPMET considers only the steady state (i.e., the last value of the dynamic state) instead of the entire dynamic state. It systematically searches subsets of enzymes. Each node in the branch-and-bound search tree is a candidate solution (i.e., a set of enzymes to be inhibited). The root node contains the empty set. As the search space is traversed, OPMET keeps the current best node with the minimum distance so far as the “current best solution” and the associated distance as the “global cut-off threshold”. If the current node has distance less than the current threshold, it saves that node as the new best solution and updates the global threshold with the current distance value. It then selects another enzyme to insert into the current set of inhibited enzymes if the insertion improves the current solution. Otherwise, it backtrack to the previous solution.

We use the OPMET algorithm to search subsets of enzymes systematically to find the one with the closest dynamic state to the goal state. Using the dynamic state, however, introduces a new challenge that does not exist in OPMET. The computational cost for computing the
dynamic state $X$ of a metabolic network often dominates the time it takes to compute the distance between two dynamic states. The gap between the two depends on the number of equations and variables that explain the fluxes of the metabolic network as well as the dependencies between those variables. Therefore, it is essential that we avoid computing the dynamic state for the nonpromising enzyme subsets. We achieve this for each subset of enzymes as follows. We do not compute the entire dynamic state $X$. Instead, we compute only a short prefix of the values in $X$. Using this prefix, we compute a lower bound to the distance between $G$ and the entire $X$. Computation of the lower bound varies for the distance measures we defined in Section 5.3. We discuss how to compute the lower bound for each of the three distance measures next.

**Exact distance.** Our solution follows from the observation that this distance function is additive over the time domain. That is the exact distance between two dynamic states over a time period is simply the sum of their distances of the shorter time intervals that make that time period.

We split the entire time span till the metabolic network reaches steady state into nonoverlapping intervals. We compute the dynamic state $X$ only for the first time interval. We compute its distance to $G$ inside this time interval by considering only those values of $G$ that are within this interval as described in Section 5.3.2. If the distance is greater than the global cut-off threshold we filter that enzyme set. Otherwise, we compute the values of $X$ for the next time interval and update the distance with these new values. As the distance is additive and nonnegative, it monotonically increases with the number of time intervals. We repeat updating the distance with new intervals until the distance exceeds the threshold or we compute the entire $X$. This way, we can avoid computing the entire $X$ if the distance grows beyond the cutoff quickly.

**Time-warping/pattern distance.** Both time-warping and patten distances use dynamic programming to compute the alignment of two dynamic states. As a result, these distances are not additive. We develop a different strategy for them. Next, we explain only the time-warping distance as both distances use the same dynamic programming method.
Figure 5-4 illustrates our strategy. In the figure, solid arrows show the entries in the dynamic programming matrix that can contribute to the computation of entry \((i, j)\). The dashed arrow shows the path corresponding to the optimal alignment of \(G\) and \(X\). \(d_{\text{min}}\) (i.e., value of the shaded entry) is the minimum value at each row. If the value of \(d_{\text{min}}\) at the end of an interval is more than the cutoff we do not need to compute the rest of the dynamic state \(X\). Similar to the exact distance, we compute the dynamic state \(X\) only for a short prefix of \(X\). Assume that this prefix contains \(m_1\) values we compute the values of \(\lambda(m_1, j)\) for all \(j \in [1, n]\). If the smallest value among \(\lambda(m_1, j), \forall j\) is greater than the global cut-off threshold we filter that enzyme set. Otherwise, we compute the values of \(X\) for the next time interval and continue filling the dynamic programming matrix. We repeat this process until the minimum distance in a row of the dynamic programming matrix exceeds the threshold or we compute the entire \(X\). This way, we can avoid computing the entire \(X\) if the minimum distance grows beyond the cutoff quickly.

5.5 Experimental Results

In this section, we evaluate our algorithms on real datasets.

**Dataset:** We test our three distance functions and the search on the Purine metabolism as all the reaction coefficients needed to compute the dynamic states are available in literature for this network. Purine metabolism is an important metabolic network. It synthesizes and breaks down purines. We use Voit’s computational model for Purine metabolism [107]. We compute the dynamic state of the Purine metabolism by solving the GMA system equations. The kinetic orders and rate constants of the corresponding GMA system equations are available in the literature [107]. Briefly this dataset contains 36 fluxes and 16 ODEs that describe the relationship between different variables.

**Query sets:** We created four query sets, namely \(Q_1, Q_2, Q_3\) and \(Q_4\) from the Purine metabolism dataset [107]. Each query set contains 10 goal dynamic states. We created each query in \(Q_1\) as follows. We randomly selected one enzyme from the Purine metabolism. We then computed the
the dynamic state of that metabolism after inhibiting that enzyme and used it as the goal state. Similarly, we created each query in Q2, Q3 and Q4 by inhibiting two, three and four randomly selected enzymes respectively. This ensures that there is a solution to the dynamic enzymatic target identification problem for each query with zero distance. In order to simulate different kinds of mutations on the goal dynamic state, we also created three more query sets for each of Q1, Q2, Q3 and Q4 as follows.

Stretch. The first one simulates stretching in the time domain. For this, we stretched each query by randomly shifting its time value at each time point by a random amount. We denote these query sets by including the prefix S- to the query set (e.g., S-Q1).

Shift/Scale. This query set simulates scaling and shifting mutations. We obtained this by scaling and shifting the dynamic states in the original query sets by random α and β values. We denote these query sets by including the prefix SS- to the query set (e.g., SS-Q1).

Shift/Scale/Stretch. This query set simulates scaling and shifting mutations as well as stretching in the time domain. We obtained this by scaling and shifting the dynamic states in the stretched query sets by random α and β values. We denote these query sets by including the prefix SSS- to the query set (e.g., SSS-Q1).

Implementation and system details: We implemented the developed algorithms in C++ and MATLAB. In our experiments, we apply the ordinary differential equation functions (e.g. ode23t, ode45) of MATLAB to compute the GMA system equations. Then, we created a C++ shared library from MATLAB M-file for dynamic state computation. Thus, we can use our traversal method to find the results. We ran our experiments on a system with Intel Core i7-920 2.66GHz Processor, 4 gigabytes of RAM, and a 64-bit Windows7 operating system.

Experimental setup: We evaluate the speed and the accuracy of our method for each of the three distance measures in terms of several metrics.
Execution time: This indicates the average total time taken by our algorithms to find the enzyme set whose inhibition leads the metabolic network to the closest dynamic state.

Percentage of time intervals: This indicates the percentage of the dynamic state our method computes after splitting it to smaller intervals. Small percentage is desirable as it often indicates lower running time.

Percentage of success: Each goal dynamic state in our query set has a matching enzyme subset that has the smallest distance. This metric reports the percentage of queries for which our algorithm could identify the optimal result. It is worth noting that this measure is biased against our method. This is because even when our methods does not find the optimal enzyme subset, the result it find can be very close to the optimal one. This metric considers those results as failed results. However, as we present later in this section, our method has high accuracy for this metric despite this bias.

5.5.1 Evaluation of the Performance

Our first experiment evaluates the performance of our method with and without splitting the dynamic states in the time domain. While searching the possible enzyme sets, recall that, our algorithm splits the dynamic state into short intervals and incrementally computes these intervals. The first questions that we need to answer are: (i) What is the typical running time of our algorithm without splitting and (ii) what is the impact of splitting on the performance of our algorithm? To answer these questions, we ran queries without splitting and with splitting when we split the dynamic state into 2, 4 and 8 intervals.

Table 5-1 shows the performance results for the exact and the time-warping distance. It presents the comparison of the average percentage of the intervals our algorithm generates for the exact and time-warping distance with and without splitting the dynamic states for query sets $Q_2$, $Q_3$ and $Q_4$ into different number of intervals. 1 interval indicates that the dynamic state is
not split. $K$ intervals ($K > 1$) indicates that the dynamic state is split into $K$ equal sized non-overlapping intervals along the time axis. When the underlying distance is the exact distance, the percentage of intervals generated tends to decrease as we split the the dynamic state into smaller pieces. This indicates that, our algorithm can often filter an enzyme subset after considering a small prefix of it. Thus by splitting each dynamic state into a larger number of (shorter) intervals we avoid computing a large portion of the dynamic state. We observe the largest improvement after splitting the dynamic states into two intervals. Further splittings gradually improves the percentage of intervals generated. As the number of intervals grows beyond four, we observed that the performance gain is negligible. The largest average running time we observed in our experiments was around two minutes (when the number of intervals is = 1 and we use the $Q_4$ query set). The improvement in the running time as we increase the number of intervals changes greatly from one query to another depending on the complexity of the set of ordinary differential equations and the corresponding time interval. On the average, we observed the best running times for four to six intervals.

The results for the time-warping distance follows a similar pattern. The percentage of intervals generated drops as we split the dynamic state into shorter intervals. As the number of intervals increases to four, we filter enzyme subsets after considering a small fraction (around 27%) of the values of the dynamic state. As we split the dynamic state further into smaller intervals, the amount of additional intervals filtered grows very slowly. Similar to the exact distance, the largest average running time we observed in our experiments was around two minutes (when the number of intervals is = 1 and we use the $Q_4$ query set). Filtering time intervals improved the average running time of our algorithm by a factor of two to four. However, it is worth mentioning that the running time does not necessarily drop linearly along with the number of time intervals as we discussed in the previous paragraph. In summary, we observe that partitioning the dynamic state to a small number of intervals (such as four intervals) improves
the running time. Further splittings do not help significantly. They can even increase the running time.

5.5.2 Evaluation of the Accuracy

This experiment evaluates the accuracy of our method using the three distance functions we defined in Section 5.3 on query sets with different characteristics. We measure the accuracy in terms of the “percentage of success”, i.e., the percentage of queries for which our method finds the optimal result correctly. We used all the four classes of query sets we generated for this purpose. Table 5-2 presents the results.

A set of interesting observations follow from these results.

When the query dynamic state can be achieved without any stretch/shift/scale transformation, our method has similar accuracies for all the three distance measures. This is expected as this is a special case of the transformations (e.g., $\alpha = 1$, and $\beta = 0$).

As the goal states are transformed through stretching along the time domain, the accuracy of the exact distance drops rapidly, while that of the other two distance functions remain identical. This justifies the need for the dynamic programming solution of the time-warping distance. This is because even small perturbations in the network can alter the result of the exact distance greatly.

When the goal state is shifted or scaled, both the exact and the time-warping distance measures fail while the pattern distance remain to have high accuracy. The high accuracy values of the pattern distance suggests that our iterative algorithm for optimizing the pattern distance is highly accurate.

We observe that the best accuracy values we observe in each row of this table can be less than 100% even when the query states are not transformed (i.e., consider the results for $Q_2$). This is caused because of the heuristic filtering strategy of OPMET [96] while searching possible
enzyme subsets. This problem can be alleviated by using a more stringent filtering strategy at the expense of increased running time.

We conclude from these observations that the pattern distance is the most promising distance measure among the three. The exact distance is faster than the pattern distance. However, the gap between their accuracies when goal states are transformed justifies their small performance difference.

5.6 Discussion

The classic enzymatic target identification problem aims to identify the set of enzymes whose knockouts lead the steady state of the metabolism close to a given goal state. This definition is problematic for a biological system reaches to the steady state over a period of time, after a sequence of changes to its current state. These sequence of states, called the dynamic states can be crucial as they can lead to serious side effects. We addressed a new variant of the enzymatic target identification problem, named the dynamic enzymatic identification problem in this paper. Unlike the existing problem we considered the entire trajectory, the given network’s state follows to reach the steady state. To the best of our knowledge, this problem has not been considered in the literature so far.

We considered three alternative distance measures to compute the dissimilarity between two dynamic states, namely exact, time-warping and pattern distance. The first one measures the area between the curves defined by the two dynamic states. The second one allowes the dynamic states stretch along the time dimension by arbitrary amounts. The last measure further generalizes the distance by allowing scaling and shifting of dynamic states. We exploited the OPMET algorithm to develop a branch and bound strategy that uses these distance measures to solve the dynamic enzymatic target identification problem. In order to improve the running time of this algorithm for the dynamic states, developed a partitioning strategy as follows. Instead of creating the entire dynamic state, we quickly created a short prefix of it. We then used this
prefix to compute a lower bound to the distance between the entire dynamic states. If this lower bound exceeds the distance of the best solution found so far we prune that solution without generating the rest of the dynamic state. Our experiments demonstrated that our method is 85-100% accurate when a single enzyme is inhibited. It was 65-75% accurate when two enzymes are inhibited. Furthermore, our partitioning strategy reduced the number of time intervals computed for dynamic states by a factor of 2 to 6.

Figure 5-1. The patterns $P_1$ and $P_2$ show the sequence of concentration of a compound resulting from two alternative manipulations to the metabolic network.

Figure 5-2. Three transformations, stretch, scale and shift applied on a hypothetical dynamic state.
Figure 5-3. An illustration of two dynamic states.

Figure 5-4. An illustration of how partitioning the dynamic state $X$ into shorter intervals improves the running time.
Table 5-1. Comparison of the average percentage of the intervals our algorithm generates for the exact and time-warping distance with and without splitting the dynamic states.

<table>
<thead>
<tr>
<th>Number of intervals</th>
<th>Exact distance</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Q_2$</td>
<td>$Q_3$</td>
<td>$Q_4$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>72.5</td>
<td>60.7</td>
<td>64.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>58.7</td>
<td>43.9</td>
<td>49.0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>58.0</td>
<td>40.7</td>
<td>47.5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>54.9</td>
<td>36.8</td>
<td>43.7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of intervals</th>
<th>Exact distance</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Q_2$</td>
<td>$Q_3$</td>
<td>$Q_4$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>52.3</td>
<td>51.8</td>
<td>52.1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>27.6</td>
<td>27.8</td>
<td>27.7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>18.7</td>
<td>19.4</td>
<td>19.0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>14.9</td>
<td>15.1</td>
<td>15.1</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-2. Accuracy of our algorithm using the three distance measures on datasets with different characteristics. The accuracy values are reported in terms of percentage of success.

<table>
<thead>
<tr>
<th>Query Set</th>
<th>Distance measure</th>
<th>Exact</th>
<th>Time-warping</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>$Q_2$</td>
<td></td>
<td>70</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>$S-Q_1$</td>
<td></td>
<td>10</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>$S-Q_2$</td>
<td></td>
<td>20</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>$SS-Q_1$</td>
<td></td>
<td>5</td>
<td>5</td>
<td>85</td>
</tr>
<tr>
<td>$SS-Q_2$</td>
<td></td>
<td>10</td>
<td>10</td>
<td>75</td>
</tr>
<tr>
<td>$SSS-Q_1$</td>
<td></td>
<td>0</td>
<td>0</td>
<td>85</td>
</tr>
<tr>
<td>$SSS-Q_2$</td>
<td></td>
<td>0</td>
<td>5</td>
<td>65</td>
</tr>
</tbody>
</table>
The process of in silico compound selection – finding a new candidate drug from large libraries of compounds by computer aid, plays a significant role in modern drug discovery. One of the popular compound selection methods is to screen libraries of compounds for their ability to bind to biological targets such as receptors and enzymes. This process is also known as “docking”. Recent validation studies show that docking methods have a poor performance in compound selection. Often, the compounds that have the highest affinity to the targeted protein according to a docking algorithm do not lead to the highest activity levels when they are tested in vitro or in vivo. Thus, there is a great need of accurate in silico compound selection methods. In this chapter, we develop two novel computational methods that rank a given set of compounds for a given target protein or enzyme. The major difference between our first method and traditional in-silico screening methods is that we consider additional proteins and enzymes while ranking compounds whereas existing strategies often focus only on the target protein alone. A drug compound can alter the state of the metabolic network. Our second method considers the impact of the drug compounds on the metabolic network by integrating the interactions among proteins in metabolic networks with the docking results. Experiments on the pharmacologic chaperones of misfolded rhodopsin show that our method has better accuracy than the traditional methods that focus only on rhodopsin. Our results are in the top 5.7% of all possible rankings. For the same dataset, the traditional method’s results are in the top 81% of all possible rankings.

6.1 Motivation and Problem Definition

In silico compound selection – finding a new candidate drug from large libraries of compounds by computer aid, plays a significant role in modern drug discovery. Selecting candidate drugs (i.e., compounds) that satisfy the pharmacological properties, i.e. absorption, distribution,
metabolism, excretion and toxicity, will largely reduce the cost and time of the rest of the drug discovery process, e.g. in vitro and in vivo screens, preclinical testing and clinical testing.

One of the popular compound selection methods is to screen libraries of small compounds for their ability to bind to biological targets such as receptors and enzymes [38]. This process is also known as “docking”. Docking algorithms estimate how two molecules can bind with each other to form a stable complex [59]. DOCK [26], Glide [30] and GOLD [41] are a few examples of existing docking software. These tools predict the binding affinity between each small molecule and the target protein. Once the docking software computes the affinity of each small molecule in a library of molecules, the next step is often to pick the ones that have high predicted affinity values and test them in the lab. This process has been successful in several applications. For example, Lyne et al used FlexX-Pharm [34] docking software to search about 200,000 compounds and identified four novel classes of inhibitor for Chk1 kinase [63]. Kellenberger et al. searched about 44,000 compounds by docking software, GOLD and Surflex, and found novel non-peptide ligands for GPCR CCR5 [50].

Despite some success in compound prediction in several applications, recent validation studies show that docking methods have a poor performance in compound selection [110]. Warren et al evaluated 10 popular docking programs and 37 scoring functions for eight proteins. The results showed that none of the docking programs or scoring functions can predict a useful binding affinity [109]. Often, the compounds that have the highest affinity to the targeted protein do not necessarily show the highest activity levels (the observed results by in vitro or in vivo assays) when they are tested in lab. To verify this, we have ranked the compounds from a real application using docking algorithms. We have also ranked the same compounds according to their actual activities in vivo experiments. The two ranking were significantly different (details in Section 6.3). Furthermore, the ones that produce high activity in vitro are frequently toxic, making them practically useless.
There are many factors that cause the failure of the use of docking programs. A major factor that causes deviation between the predicted affinities and observed activities, which is also the central problem tackled in this chapter, is that existing in silico methods rank compounds solely based on their affinities to the targeted proteins. Clearly, this is not a realistic strategy as the drug molecule can bind to proteins other than the targeted one. This reduces the chance that it will bind to the target protein. Furthermore, the proteins and enzymes do not necessarily work independent of each other. They often interact over a complex biological network. If some of these proteins are enzymes, the reactions catalyzed by them may be influenced. Thus, the concentration of the substrates or products in the metabolism may be altered. As a result, the metabolism may be altered or toxicity may occur.

Another significant factor is the inaccuracy within the docking algorithm. For example, most of the docking softwares lack proper treatment of protein flexibility and solvation. Treating the receptor as too rigid or flexible also results in incorrect binding affinity. In some cases, the incorrect choice of protonation or tautomerisation states contributes to a significant error in scoring [110]. New versions of the existing docking software have already been developed to alleviate these problems.

**Problem definition:** In this chapter, we consider the compound selection problem. We define this problem as follows. Assume that we are given a target protein or enzyme and a library of compounds. Compound selection problem aims to identify the compounds from this library that will bind to the target at a high rate and change the activity level of the target. More specifically, we develop a ranking algorithm that sorts the compounds in the compound library according to their probability of altering the activity of the target protein or enzyme.

**Contributions:** The central hypothesis tested in this chapter is that in-silico compound selection will be more accurate by considering a rich set of proteins and enzymes in the metabolism, in addition to the targeted protein. We develop two novel in silico compound selection methods.
The major difference between our methods and traditional in-silico screening methods is that we consider additional proteins and enzymes for ranking drug compounds while existing strategies often focus only on the target protein alone. We also consider the interactions among proteins by integrating metabolic networks with the docking results.

Given a target protein, first, we select a set of proteins besides the targeted one. This set contains the proteins that are structurally similar to the target protein and the proteins that are close to the target protein in the metabolic network. The former proteins are the ones that have a high chance of binding to the same drug molecule as the target. The latter proteins are the ones whose inhibitions can alter the metabolism greatly if they are inhibited along with the target. We, then, predict the binding affinity values between these selected proteins and candidate compounds using a docking tool.

Our first method uses linear regression on the predicted affinity values and learns the parameters of the ranking function that explains the historical experiment database the best. In order to avoid the overfitting problem, it recursively eliminates the protein that has the lowest correlation with the activity level observed using in vitro or in vivo experiments until all the remaining proteins have high correlation.

Our second method computes the binding probability of the compounds with each of the proteins from the predicted affinities. Based on these probabilities, it uses Monte Carlo simulation to compute the expected “impact” of each compound to the metabolic network. The impact of a compound to a network is the amount of change in the steady state of that network when that compound is included. This strategy applies linear regression on these impact values and learns the parameters that explains the historical experiment database the best.

Figure 6-1 illustrates the three compound selection strategies discussed above. Strategy ‘A’ is the traditional in-silico screening method. This method considers the affinity of the compounds to target proteins only. Here, docking algorithm computes the affinity between a given compound
and a protein. ‘B’ denotes our first ranking method. This method uses affinity of compounds to a superset of the target proteins. ‘C’ denotes our second ranking method. This one estimates the impact of compounds on metabolic networks from their affinities to the enriched set of proteins.

Experiments on the prediction of pharmacologic chaperones of misfolded rhodopsin show that both of our methods have better accuracy than the traditional methods. Our second method has the highest accuracy among the three. Our results are in the top 5.7% of all possible rankings. For the same dataset, the traditional method’s results are in the top 81% of all possible rankings.

In summary, the technical contributions of this chapter are as follows.

We develop two methods that use multiple proteins for ranking compounds. These methods use affinities of the proteins to the compounds and integrate the impact of compounds on metabolic networks.

We develop a Monte Carlo method for computing the expected impact of a compound on a metabolic network.

We experimentally verify our method on a real problem, Retinis Pigmentosa, both in silico and in vitro. The results show that our algorithm ranks compounds significantly better than the traditional method [50, 63, 68].

The rest of the chapter is organized as follows. Section 6.2 describes our in silico compound selection method. Section 6.3 presents the experimental results. Section 6.4 discusses this chapter.

6.2 Methods

In this section, we discuss our in silico compound selection method. Our method employs existing docking software as a building block. It also uses a small database of historical observed activity results (obtained in vitro or in vivo) and in-silico affinity values (obtained by using a docking software). Briefly, our method works as follows.
It selects a set of proteins of interest besides the target one and predicts the binding affinity values between these selected proteins and candidate compounds. (Section 6.2.1)

The basic algorithm ranks the candidate compounds based on these predicted affinity values (Section 6.2.2.1). We incorporate the affinity values and the impact of the compounds on the metabolic networks to the ranking function (Section 6.2.2.2).

6.2.1 Protein Selection

The central hypothesis tested in this chapter is that in-silico compound selection will be more accurate by considering a rich set of proteins and enzymes in the metabolism, in addition to the targeted protein. The obvious first step then would be to compute the affinity between all compound-protein pairs before selecting a compound. This strategy however is not feasible for several reasons.

The size of the protein and the compound databases make it impractical. It takes more than five minutes to compute the affinity of a compound to a single protein. There are 10 million compounds in PubChem. Even if we restrict the compound database to 0.1% of PubChem, it will take more than five thousand years to compare them against all of the 58,000 proteins in the Protein Data Bank (PDB) using a single CPU.

Most of the compound-protein affinities would be useless for compound selection as a significant portion of them will have no affinity.

The datasets that contain actual in vitro experimental results on actual compound-protein bindings are very limited. As a result of this, using the affinity values of many proteins to learn a function that estimates drug activity will result in “overfitting problem”. In other words, the model that can be learned using any machine learning method will explain the observed data well, but it will fail to generalize unobserved compounds.

In order to avoid these problems, we select a small set of relevant proteins. Note that, this selection can be assisted by expert analysis. In order to automate the compound selection
process, we develop a generalized framework that can work without expert guidance using the following two criteria.

**Structural properties.** Clearly, a feasible drug compound needs to be chosen so that it binds to the target protein strongly. This means that there is a high probability that the same compound binds to all the proteins that are structurally similar to the target protein. Thus, we choose all the proteins that are in the same superfamily as the target protein into protein set. We use the SCOP database \[65\] to determine the superfamily of the proteins. Note that one can also use a structural alignment algorithm such as CE \[39\] or Dali \[36\] for this purpose and choose the proteins that align well with the target protein.

**Spatial properties.** Proteins interact with each other over a complex biological network. As a result, when a compound binds to a protein, its effect is not limited to the function of that protein. It can affect a large subset of the metabolism. This kind of influence usually magnifies greatly when the same compound binds to multiple proteins that are close by in the interaction network. To take the impact of the compounds into account better, we choose all the proteins that are in the neighborhood of the targeted protein in the metabolic network. We say that a protein is in the neighborhood of the target protein if it appears in the same network or in a network that shares at least one common compound with a network that the target protein appears.

We use the networks provided in the KEGG database \[47\] to determine whether two proteins are neighbors. Note that one can use other network databases as well for this purpose.

### 6.2.2 Ranking Compounds

Given a target protein or enzyme, so far we have discussed how we select a set of proteins that will help in ranking the compounds in a large library of compounds. We develop two ranking methods in this section.

The first one uses affinity values of all the selected proteins computed by a docking program. (Section 6.2.2.1)
The second one computes the expected “Impact” of each compound to the metabolic network. (Section 6.2.2.2)

Both methods apply linear regression to learn the parameters those explain the historical experiment database the best.

Our algorithms learn the ranking function from a historical query workload (i.e., training dataset). The training dataset is a population of compounds whose actual activity levels on the target protein is measured by in vivo or in vitro experiments. These experiments require many hours for each compound. For example, the pharmacologic chaperone experiments for misfolded rhodopsin need half a month for 24 compounds. Therefore, typically the training dataset contains a small number of compounds. As the size of the training dataset grows, the accuracies of our ranking algorithms improve.

We will use the following notation in the rest of this chapter. We will denote the selected proteins (see Section 6.2.1) with $P_1, P_2, \ldots, P_n$. We will denote the compounds in the training dataset with $C_1, C_2, \ldots, C_m$. We will represent the actual activity level of each training compound $C_i$ with $A_i$. We discuss our ranking methods next.

### 6.2.2.1 Ranking based on affinities

Our first ranking method is based on the affinities of the compounds to the selected proteins. This method follows from the hypothesis that there exists the linear correlation between the affinities and the activity levels.

Using a docking software, we first compute the binding affinity value for each $<\text{training compound}, \text{protein}>$ pair. We will denote the affinity between the compound $C_i$ and protein $P_j$ with $a_{ij}$. Thus, we build an affinity matrix $A = \{a_{ij}\}$, for $1 \leq i \leq m$ and $1 \leq j \leq n$. We use linear regression to learn the ranking function, which shows the relationship between the dependent variable and one or more independent ones. In this problem, the independent variables are the actual activity levels; the dependent variable is the predicted affinity value. Formally, we
compute the linear function

\[ A_i = \beta_0 + \beta_1 a_{i1} + \beta_2 a_{i2} + \ldots + \beta_n a_{in} + \varepsilon_i \]  

(6–1)

that holds for all \( A_i, i \in \{1, 2, \ldots, m\} \). Here, \( \varepsilon_i \) denotes a noise term, which is a random variable. We assume that \( \varepsilon_i \) follows the standard normal distribution, that is, \( \varepsilon_i \sim \text{Normal}(0, 1) \). We compute the parameters of equation (6–1) using least-squares estimation [14].

**Recursive feature elimination.** One potential problem with using linear regression on the affinity matrix is that this matrix typically has a small number of rows. This is because it is very costly to get the experimental data which tells the value of the dependent variable in practice. As a result of this, often only several tens of the experimental results are observed. When the number of selected proteins (i.e., independent variables or columns in the matrix) is large, there is a danger that the linear regression will overfit the data. That is, the independent variables can explain the dependent one perfectly on the observed data, but it will fail on unobserved data. Therefore a computational approach is needed to address this problem.

In order to avoid the overfitting problem we reduce the number of independent variables by recursive feature elimination. This method recursively removes one protein from the protein set; the one that is least significant in determining the dependent variables. We declare an independent variable (say the \( j \)th variable) as insignificant if its absolute coefficient value \( \beta_j \) is close to zero or its P-value is larger than 0.05. If there are more than one such proteins we remove the one with the largest P-value or the smallest \( \beta_j \) value. Once we remove the least significant protein, we apply the linear regression again and eliminate another protein iteratively. This elimination process stops when all the remaining proteins are significant. Algorithm 6.2.2.1 presents the pseudocode for recursive feature elimination.

**Ranking unobserved data.** Once we eliminate all the redundant proteins, the next step is to build a ranking function that predicts the activity of the unobserved compounds from their
Algorithm 6.1 Recursive feature elimination

1. Use linear regression on the affinity matrix \( A \) and the observed activities \([A_1, A_2, \cdots, A_m]\) to compute coefficients \( \beta_j \) and \( \epsilon_i, \forall i, j, 1 \leq i \leq m \) and \( 1 \leq j \leq m \).

2. Among all the features (i.e., proteins) with insignificant coefficient value (i.e., \( |\beta_j| < 0.01 \)) or with P-value larger than 0.05, select the one with the largest P-value.

3. If no protein is selected in the previous step, then all the features are significant. Thus, return the current protein set and stop feature elimination.

4. Otherwise, there is at least one insignificant feature in the feature set. Remove the protein chosen in Step 2 from the protein set. Remove the corresponding column from the affinity matrix \( A \).

5. If the feature set is not empty, go to Step 1.

affinities to the remaining proteins. We build this ranking function using linear regression as well. Assume that the number of non-redundant proteins after feature elimination is \( n' (n' \leq n) \). We then use linear regression to learn the coefficients \( \beta_i (0 \leq i \leq n') \) as discussed at the beginning of this section. The ranking function is then

\[
\text{Rank}([x_1, x_2, \cdots, x_{n'}]) = \beta_0 + \sum_{i=0}^{n'} \beta_i x_i
\]

where \( x_i \) is the affinity between the unobserved compound and the \( i \)th non-redundant protein computed by the underlying docking software. The larger the value of this function, the higher the predicted activity of the compound.

6.2.2.2 Integrating networks in ranking

The ranking method we developed in the previous section assumes that there is a linear correlation between the affinities of the compounds to selected proteins and the activity levels of those compounds. The relationship between the two however may not be a simple linear function. This is because the proteins and enzymes interact with each other over a complex metabolic network. When a compound binds to a protein or enzyme, it may affect the rest of the metabolism even if it does not bind to other proteins or enzymes. For example, the reactions
catalyzed by the binding enzymes may be influenced. Thus, the concentration of substrates or products in the metabolism may be altered. As a result, the function of the metabolism may be altered or toxicity may occur.

In this section, we develop a new ranking function that integrates the metabolic networks with the predicted affinity values. Unlike the previous ranking function, it computes the impact of the compound on the metabolic networks and uses these values to determine a rank for each compound. This method works as follows.

a. Given a compound and a set of target proteins, we first select a richer set of proteins as explained in Section 6.2.1. We then compute the affinity matrix $A$ as discussed in Section 6.2.2.1.

b. Using the affinity values in $A$, we compute a binding matrix $B$. The binding matrix has the same number of dimensions as the affinity matrix. Each entry $b_{i,j}$ ($1 \leq i \leq m$, $1 \leq j \leq n$) of $B$ shows the probability that the $i$th drug compound will bind to the $j$th selected protein.

c. We compute the expected “impact” of each compound on each metabolic network using Monte Carlo simulations.

d. We derive a ranking function based on the expected impact values.

Sections 6.2.1 and 6.2.2.1 discussed the first step above. We elaborate on the remaining steps next.

**Step 2: The binding matrix.** Binding a compound in a cell to a protein or enzyme is a probabilistic event that depends on a number of factors. One important factor is their affinities. In order to compute the impact of a compound on the metabolism, we need to model this probabilistic event. We derive this model using the affinity matrix $A$.

We use the random variable $B_j$ to denote the event that a given compound binds to $j$th protein ($B_j = 1$ if it binds, $B_j = 0$ otherwise). We estimate the probability that $B_j = 1$ for a compound from its affinity value $a_{i,j}$. To do this, we first analyze the distribution of the
affinity values of a large number of randomly selected compounds to a given protein. Figure 6-2 shows the distribution of binding affinity for rhodopsin (PDB:1F88) on the diversity set of NCI/DTP [35]. This set contains 1,990 compounds. The affinity values seem to follow a normal distribution. The mean $\mu$ and the variance $\sigma^2$ of this distribution can be obtained as those of several hundred randomly selected compounds.

Smaller affinity values imply that the compound will bind to that protein with a higher chance. Since the compound is competing with other compounds to bind to a protein, the probability that it binds to that protein also depends on the affinity values of the competing compounds. From these observations, we derive two alternative ways to measure the probability that a compound binds to the $j$th protein.

All compounds are competing. In this case we assume that all compounds compete for the same protein. In this case, the probability becomes the cumulative density function of the distribution we found. More precisely

$$\text{Prob}(B_j = 1 | \text{Affinity} \leq a) = \frac{1}{2} \left( 1 + \frac{2}{\sqrt{\pi}} \int_{-\infty}^{\frac{\mu - a}{\sigma \sqrt{2}}} e^{-t^2} dt \right)$$  \hspace{1cm} (6–2)

All the natural ligands are competing. Proteins often have one or more natural ligands that bind strongly to them. Thus a new compound has to compete with them in order to replace them. Assume that the $j$th protein has $g$ ligands. Also assume that the affinity of the $k$th ligand is $L_k$. Then, the probability that a compound with affinity $a$ wins over the $k$th ligand is

$$p_k = \frac{1}{2} \left( 1 + \frac{2}{\sqrt{\pi}} \int_{-\infty}^{\frac{L_k - a}{\sigma \sqrt{2}}} e^{-t^2} dt \right).$$  \hspace{1cm} (6–3)

Thus, the probability that this compound wins over at least one of the natural ligands is

$$\text{Prob}(B_j = 1 | \text{Affinity} \leq a) = 1 - \prod_{i=1}^{g} (1 - p_i).$$
Now, we are ready to formally define the binding matrix $B$. Each entry of this matrix shows the probability that a compound binds to a protein in the protein set. We compute this value as

$$b_{i,j} = \text{Prob}(B_j = 1 | \text{Affinity} \leq a_{i,j}).$$

**Step 3: The impact of compounds on networks.** Once we compute the binding matrix, we estimate the “impact” of each compound on the metabolic networks. We elaborate on the impact computation later in this section. We consider the metabolic networks that contain at least one of the selected proteins/enzymes. Thus, if the selected set of proteins appear in $k$ different networks, then we compute $k$ impact values for each compound. This is desirable since the number of networks is much smaller than the number of enzymes or proteins. In other words, keeping impact values rather than affinities naturally reduces the number of features, and thus the risk of data overfitting. We discuss the computation of impact on a network next.

When a drug compound binds to an enzyme, it inhibits that enzyme and slows down its reaction. This can affect the rest of the network as other reactions may be consuming the compounds produced by the reaction catalyzed by that enzyme or producing the compounds as the substrate of the reaction catalyzed by that enzyme. We define the impact of inhibiting a set of enzymes of a metabolic network as the change in the steady state of that network.

In order to discuss the concept of “impact”, we first need to understand the “steady state” of a network. The steady state of a network is the flux of all the reactions when the flux does not change anymore. An alternative definition is the yield of all the compounds in that network when the yield does not change any more. One way to compute the steady state of the metabolic network is to use Flux Balance Analysis (FBA) [7, 29, 48]. In our earlier work, we have also defined an algorithm for computing the steady state [93]. We refer the reader to these papers for the details of the steady state computation. Let us denote the steady state of a network before and after inhibiting a given set of enzymes $S$ with vectors $u = [u_1, u_2, \cdots, u_c]$ and
\[ v = [v_1, v_2, \cdots, v_c]. \]

Here, \( u_i \) and \( v_i \) are the yield of the \( i \)th compound in the network, and the number of compounds is \( c \). We compute the impact as

\[
\text{Impact}(S) = \sum_{i=1}^{c} |u_i - v_i|.
\]

Now, we know how to compute the impact when a given set of enzymes are inhibited. We still need to tackle one more problem. The process of binding a compound to a protein or enzyme is a probabilistic event. Therefore, we do not know in advance which enzymes are inhibited. Mathematically, the expected impact of each compound on a network can be computed as follows. Let us denote the set of selected proteins with \( P \). We denote the probability that the \( i \)th compound binds to the set of proteins in \( S \) (\( S \subseteq P \)) with \( \text{Prob}(S|i^{th} \text{ compound}) \). The expected impact of the \( i \)th compound is then

\[
\sum_{S \subseteq P} \text{Impact}(S) \text{Prob}(S|i^{th} \text{ compound}).
\]

Computing this summation is difficult as the number of subsets \( S \) is exponential in the size of \( P \). We therefore compute the expected impact of a given compound using Monte Carlo simulation in Algorithm 6.2.2.2. Once we compute the expected impact values, we use linear regression on these values to learn the ranking function for the actual activity levels similar to Section 6.2.2.1.

### 6.3 Results

We have tested the algorithms described in the previous section for prediction of ongoing experiments on deriving pharmacologic chaperones of misfolded rhodopsin. In this section, we present the performance of our algorithms on the dataset that is available to date.

**Application:** Retinitis pigmentosa (RP) is a prevalent genetic eye disease [6] that is caused by a mutation in rhodopsin among nearly 20% to 25% of patients with autosomal dominant RP [68]. In United States, approximately, one third of such cases results from the P23H mutation [68].
INPUT: Metabolic networks, binding matrix $B$, compound number $i$
OUTPUT: Expected impact

**Algorithm 6.2** Expected impact by Monte Carlo simulation

1. Create a sample state for all the selected proteins as follows. For each protein $j$ in the selected protein set, randomly decide whether compound $i$ binds to protein $j$ using Bernoulli distribution. The binding probability for the $j$th protein is $b_{i,j}$.

2. Remove the proteins/enzymes for which the Bernoulli trial is successful in the sample state. Compute the impact of removing these enzymes on each network as discussed Equation 6–4 and store these values.

3. Repeat steps 1 and 2 many times (a few hundred times is sufficient in practice to dramatically reduce the variance of the estimate). Report the average of all impact values computed in these iterations.

One strategy to deal with RP is to find compounds which act as pharmacologic chaperones binding and stabilizing mutant proteins. Thus, these compounds change the function of mutant proteins and arrest RP potentially [68, 88, 102]. We apply our in silico compound selection method to identify the small molecule pharmacologic chaperones of misfolded rhodopsin.

**SELECTED PROTEINS.** In addition to Rhodopsin (PDB:1F88), we computed the affinity to Rhodopsin kinase (EC:2.7.11.14, PDB:3C4W) because it catalyzes the reaction ATP:rhodopsin phosphotransferase that produces rhodopsin. Rhodopsin comes from retinol metabolism. Thus, the enzymes of retinol metabolism can influence the generation of rhodopsin. KEGG database [47] shows retinol metabolism of Homo sapiens. There are tens of enzymes in retinol metabolism. We limit our focus on only enzymes that have a 3-D structures. These are alcohol dehydrogenase (EC:1.1.1.1, PDB:1U3T), unspecific monooxygenase (EC:1.14.14.1, PDB:2FDU), retinal dehydrogenase (EC:1.2.1.36, PDB:1BI9) and glucuronosyltransferase (EC:2.4.1.17, PDB:2Q6V).
Starch and sucrose metabolism (hsa00500) can also influence role the retinol metabolism. Specially, ketodase (EC:3.2.1.31, PDB:1BHG) catalyzes the reaction which consumes Glu-
curonide. Retinal metabolism influences the generation of Glucuronide. Therefore, when the function of ketodase is inflected, the function of retinol metabolism may be influenced. We also added these to the list.

**Docking Software.** We compute the affinity between a compound and protein based on computing the docking energy. A number of software packages are available for this purpose. Some of the popular ones are DOCK [26], Glide [30] and GOLD [41]. Studies by Cummings et al concluded that Glide provided the most consistent results on their test cases [17]. Therefore, we chose Glide as our docking software in this chapter.

**Selected Metabolic Networks.** As described in the previous section, one of our methods also consider the interactions among proteins by integrating metabolic networks. Rhodopsin works on retinol metabolism and connect to Starch and sucrose metabolism. Thus, we consider these two metabolic networks. Figure 6-3 shows the considered metabolic networks in this chapter.

**In-vitro and In-vivo Data.** We have measured the actual activity levels of 23 compounds as pharmacologic chaperones of misfolded rhodopsin through in vitro experimentation at lab. In order to ensure that our experiments reflect the actual protein activity, we have also carried out in-vivo experiments on Bovine cell culture. In these experiments, the cell culture contains most of the proteins and enzymes of this organism and the corresponding pathways. Thus, we expect that our wet-lab results will be similar those that one can observe on similar organisms.

In this section, our goal is to predict the observed activity level of these compounds using in-silico data collected for docking with Rhodopsin and other proteins. Table 6-1 provides the list of compounds that have been tested so far. “Obs” denotes the observed results by in vivo or in vitro assays. hsa00830 is the retinol metabolism and hsa00500 is the Starch and sucrose
metabolism of Homo sapiens. The values in “Affinity to proteins” columns show the binding affinity between compounds and proteins. The values in “Impact on pathways” columns show the impact values for the compounds on the networks. “Rho” denotes rhodopsin. The total experimental time required for measuring the activity levels of these compounds in the lab was around one man month. The “Obs” column in Table 6-1 shows the observed results by in vitro or in vivo assays.

**Experiment design:** We used five fold cross validation for comparing the different methods. For this we decomposed the dataset in Table 6-1 into five groups randomly such that each of the groups roughly have the same number of compounds. For each fold, we treated one of the groups as a testing data set and the other four groups as a training set. The four methods that were compared are as follows.

**TARGET-ONLY:** This is the traditional screening method used in the literature [50, 63, 68]. This method focuses only on the target protein alone, that is, rhodopsin in our biological application. To replicate this process, we use linear regression to build a linear regression based on the binding affinity of rhodopsin and the observed activity.

**AFFINITY:** This is the ranking method described in Section 6.2.2.1. It uses the binding affinity information of additional proteins.

**RANDOM-AFFINITY:** An interesting question would be “to what extend does the protein selection help in our algorithm?” To answer to this question, we have implemented a randomized version of our affinity algorithm. This algorithm uses the same number of proteins and it works the same as our affinity method. The difference is that it uses randomly selected proteins.

**IMPACT:** This is the ranking method discussed in Section 6.2.2.2. It estimates the impact of compounds on metabolic networks. Preliminary experiments showed that equation 6–2 had a better result than equation 6–3. Thus, we assume that all the compounds are competing and apply equation 6–2 to compute the probability that a compound binds to the protein.
6.3.1 Accuracy in Estimation

We test the accuracy of each method as follows. For each method, we built a model using the training set. We then used this model to predict the activities of the samples in the test. We measured the accuracy of these predictions using three criteria. The first one is the correlation coefficient between the predicted and the observed activity levels. The second and the last ones are the average absolute distance and the average squared distance between the predicted and actual activity levels respectively. We report the average of all the five runs for the five training datasets.

Table 6-2 reports the accuracies of four methods. The results show that our impact method has the best accuracy both in terms of a better fit and lower error in estimation. This indicates that considering the impact of the compounds on the metabolic networks is more promising than considering the affinity between the individual proteins and the compounds. This is because altering the activity level of a set of proteins can affect the activity of the remaining proteins and enzymes in the metabolic network through interactions even when the drug compound has little or no affinity to them. Our Impact method exploits this observation successfully to rank the compounds.

Our affinity method has significantly better correlation coefficient than the Random-Affinity method. This supports our hypothesis that proteins should be carefully selected. Interestingly, the traditional method performs better than the Random-Affinity method. This means that simply using more proteins to select compounds does not help in ranking compounds. They may instead lower the quality of the final ranking as they may introduce noise. Thus, the additional proteins need to be relevant to the target proteins.

Finally, we observe that the prediction accuracy of the affinity method and the traditional method are comparable. Therefore, one might think that these two methods have the same
value in selecting compounds. Next section carries out another experiment that shows that the traditional method performs significantly worse than both of our methods.

6.3.2 Accuracy in Ranking

The eventual goal of a compound selection algorithm is to rank compounds according to their expected activity levels. In this section, we evaluate the ability of each of the algorithms to correctly rank the test set. To do this we compute the “ranking error” of each algorithm as follows. We first rank the test compounds based on our prediction. We then rank the same compounds based on the actual activity levels of the compounds. We then compute the sum of the \[ |\text{predicted rank} - \text{actual rank}| \] of all the compounds in the test set.

We also generated a distribution of the ranking error by randomly permuting the compounds in the test set a large number of times. We used this distribution to derive the expected value of the sum for a random ranking. We also computed the probability of a null hypothesis that a method is generating a random ranking being correct based on the sum derived using that method. For instance, if this probability value is 10% for a given ranking, then it implies that 10% of the random rankings will produce a lesser ranking error than that ranking. A lower value of this probability implies that the chances are high that the method can better predict relative ranking of compounds.

Table 6-3 shows the results. \( R_p \) denotes the rank of the predict result. \( R_a \) denotes the rank of the actual observed result. Percentage denotes the percentage that the value of \( \sum |R_p - R_a| \) by random ranking is less than the value by the method. From the results, the impact method predict the ranks with a smallest distance from the actual ranks. This coincides with our results in Table 6-2. Less than 6% results by random ranking are better than the sum generated by the impact method. Also, just using affinity for the target protein only is not significantly better (or worse) than random ranking. We observe that our affinity method is significantly better than the traditional method in selecting compounds although they have the same prediction error (see
results in Table 6-2). This implies that small alterations in the predicted activity levels can lead to significant changes in the ranking error. Finally, extremely high probability value (81.3%) for the target-only method implies that the traditional methods will fail to distinguish compounds that have high affinity to the target protein. Thus, from Tables 6-2 and 6-3, we conclude that considering metabolic networks improves ranking of compounds significantly.

6.4 Discussion

In this chapter, we develop two novel in silico compound selection methods. The major difference between our methods and traditional in-silico screening methods is that we consider additional proteins and enzymes for ranking drug compounds while existing strategies often focus on target protein alone. We consider the interactions among enzymes by integrating metabolic networks with the docking results. In particular our metabolic network based method uses Monte Carlo simulations to compute the “impact” of each compound to the metabolic network. The impact of a compound to a network is the amount of change in the steady state of the network when that compound is included.

Experiments on the prediction of pharmacologic chaperones of misfolded rhodopsin show that our in silico compound selection method has better performance than the docking method that only uses rhodopsin. Also, just using affinity for Rhodopsin alone is not significantly better (or worse) than random ranking.

Our methods can select compounds based on learning from previous results. We can do this as follows. First, we build a training dataset which is a population of compounds whose results are measured by in vivo or in vitro experiments. As the size of the training dataset grows, the accuracies improves. In fact, the training dataset contains a small number of compounds for each experimental result usually requires many hours. We provide two methods for ranking the compounds, ranking based on affinities and integrating networks in ranking. We estimate the parameters of these two methods based on the training dataset. Thus an active learning process
can be created that continuously refines the prediction as more data is collected. Clearly this has to be used carefully to minimize the potential pitfalls of multihypothesis testing.

**Figure 6-1.** A framework for three compound selection strategies denoted by A, B, and C.

**Figure 6-2.** Distribution of binding affinity for rhodopsin (PDB:1F88) and all the 1,990 compounds in diversity set.
Table 6-1. Dataset information needed for the application to the small molecule pharmacologic chaperones of misfolded rhodopsin.

<table>
<thead>
<tr>
<th>Compound</th>
<th>Impact to proteins</th>
<th>Impact on pathways</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Obs</td>
<td>Rho 1F88</td>
</tr>
<tr>
<td>NSC723</td>
<td>0.98</td>
<td>-8.58</td>
</tr>
<tr>
<td>NSC163936</td>
<td>0.3</td>
<td>-5.47</td>
</tr>
<tr>
<td>β damascone</td>
<td>0.8</td>
<td>-5.13</td>
</tr>
<tr>
<td>Pseudoionone</td>
<td>0</td>
<td>-4.99</td>
</tr>
<tr>
<td>dihydro β ionone</td>
<td>0.7</td>
<td>-4.24</td>
</tr>
<tr>
<td>NSC4883</td>
<td>-0.14</td>
<td>-7.63</td>
</tr>
<tr>
<td>NSC34560</td>
<td>0.9</td>
<td>-5.69</td>
</tr>
<tr>
<td>NSC218</td>
<td>-0.08</td>
<td>-5.3</td>
</tr>
<tr>
<td>NSC26718</td>
<td>0.2</td>
<td>-4.5</td>
</tr>
<tr>
<td>β ionone</td>
<td>1</td>
<td>-5.55</td>
</tr>
<tr>
<td>α ionone</td>
<td>1</td>
<td>-5.71</td>
</tr>
<tr>
<td>β ionol</td>
<td>0.8</td>
<td>-6.12</td>
</tr>
<tr>
<td>NSC45012</td>
<td>0.4</td>
<td>-6.65</td>
</tr>
<tr>
<td>NSC163995</td>
<td>0.5</td>
<td>-4.69</td>
</tr>
<tr>
<td>NSC170691</td>
<td>0.3</td>
<td>-6.48</td>
</tr>
<tr>
<td>NSC4777</td>
<td>0.5</td>
<td>-5.32</td>
</tr>
<tr>
<td>NSC3975</td>
<td>-0.01</td>
<td>-8.4</td>
</tr>
<tr>
<td>NSC49193</td>
<td>0.2</td>
<td>-3.39</td>
</tr>
<tr>
<td>NSC92799</td>
<td>0.1</td>
<td>-6.26</td>
</tr>
<tr>
<td>NSC415</td>
<td>-0.2</td>
<td>-5.26</td>
</tr>
<tr>
<td>NSC74159</td>
<td>0.15</td>
<td>-4.84</td>
</tr>
<tr>
<td>NSC47520</td>
<td>0.45</td>
<td>-6.53</td>
</tr>
<tr>
<td>NSC2280</td>
<td>0.03</td>
<td>-7.86</td>
</tr>
</tbody>
</table>

Table 6-2. The accuracy of the activity level predictions of four strategies. Correlation coefficient shows the correlation between the actual and the predicted activity levels. $Ob_p$ and $Ob_a$ denote the predicted and the actual activity levels.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Correlation coefficient</th>
<th>Average distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target-only</td>
<td>0.65</td>
<td>0.17</td>
</tr>
<tr>
<td>Random-Affinity</td>
<td>0.56</td>
<td>0.26</td>
</tr>
<tr>
<td>Affinity</td>
<td>0.65</td>
<td>0.17</td>
</tr>
<tr>
<td>Impact</td>
<td><strong>0.68</strong></td>
<td><strong>0.16</strong></td>
</tr>
</tbody>
</table>

Table 6-3. The ranking accuracy of three strategies.

| Methods   | $\sum |R_p - R_a|$ | Percentage (%) |
|-----------|--------------|----------------|
| Target-only   | 38           | 81.3           |
| Affinity      | 32           | 40.9           |
| Impact        | **24**       | **5.69**       |
Figure 6-3. Retinol metabolism and Starch and sucrose metabolism. The dark solid circle shows the enzymes in the selected protein set.
CHAPTER 7
CONCLUSION

The focus of this thesis is on the enzymatic target identification and compound selection. The theoretical and practical findings of our work can be summarized as follows.

We developed a scalable iterative method which computes a sub-optimal solution within reasonable time-bounds for the enzymatic target identification problem by boolean network models. The method consisted of two phases: Iteration and Fusion Phases. The experiments on the *E. coli* metabolic network showed that the average accuracy of the Iteration Phase alone deviates from that of the exhaustive search only by 0.02%. The Iteration Phase is highly scalable. It can solve the problem for the entire metabolic network of *Escherichia coli* in less than 10 seconds. The Fusion Phase improves the accuracy of the Iteration Phase by 19.3%.

We proved that finding the enzyme knockout strategy by OptKnock framework is NP-hard. We provided a binary method and continuous method for the enzymatic target identification problem on multiple enzymes association. Experiments showed that the enzyme association influence the performance of linear programming method very much. We observed that our binary method runs much faster than continuous method. For the pathways of *H.sapiens* from KEGG, our binary method runs less than one second for the whole metabolism. Therefore, our binary method is useful for the biological application.

We developed two algorithms for the enzymatic target identification problem by manipulating the steady state. That is, traversal approach and genetic algorithm. The traversal approach explored possible solutions in a systematic way by filtering and prioritization strategies to reduce the search space. The genetic algorithm derived good solutions from a set of alternative solutions iteratively, which can run for very large pathways.

Our experiments showed that our algorithms’ results follow those obtained in vitro in the literature from a number of applications. They also showed that the traversal method is a good approximation of the exhaustive search algorithm and it is up to 11 times faster than the
exhaustive one. This algorithm runs efficiently for pathways with up to 30 enzymes. For large pathways, our genetic algorithm can find good solutions in less than 10 minutes.

We addressed a new variant of the enzymatic target identification problem, named the dynamic enzymatic identification problem. Unlike the existing problem we considered the entire trajectory, the given networks state follows to reach the steady state. We considered three alternative distance measures to compute the dissimilarity between two dynamic states, namely exact, timewarping and pattern distance. We exploited the OPMET algorithm to develop a branch and bound strategy that uses these distance measures to solve the dynamic enzymatic target identification problem. In order to improve the running time of this algorithm for the dynamic states, developed a partitioning strategy.

Our experiments demonstrated that our method is 85-100% accurate when a single enzyme is inhibited. It was 65-75% accurate when two enzymes are inhibited. Furthermore, our partitioning strategy reduced the number of time intervals computed for dynamic states by a factor of 2 to 6.

We developed two novel computational methods that rank a given set of compounds for a given target protein or enzyme for the compound selection problem integrating structural properties of proteins and biological networks. The first method considered additional proteins and enzymes while ranking compounds whereas existing strategies often focus only on the target protein alone. A drug compound can alter the state of the metabolic network. Our second method considered the impact of the drug compounds on the metabolic network by integrating the interactions among proteins in metabolic networks with the docking results. Experiments on the pharmacologic chaperones of misfolded rhodopsin showed that our method has better accuracy than the traditional methods that focus only on rhodopsin. Our results are in the top 5.7% of all possible rankings. For the same dataset, the traditional method’s results are in the top 81% of all possible rankings.
REFERENCES


BIOGRAPHICAL SKETCH

Bin Song was a Ph.D. student in Computer Science at the University of Florida, CISE Department, where she was also a member of the Database Center. Bin Song worked under the supervision of Dr. Tamer Kahveci and Dr. Sanjay Ranka. Bin Song’s research interests were metabolic engineering, compound selection and algorithm.

Bin Song received the BS and MS degree in computer science from the University of Science and Technology of China, Hefei, China in 2002 and 2005. She received a Ph.D. degree in computer science at the University of Florida in 2010.