Investigating Data Representations on Efficacy of Bioinformatics Algorithms on Biological Data

by

Nafiseh Sedaghat

M.Sc., Islamic Azad University, 2010
B.Sc., Ferdowsi University of Mashhad, 2007

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2021

Declaration of Committee

Name: **Nafiseh Sedaghat**

Degree: **Doctor of Philosophy**

Thesis title: **Investigating Data Representations on Efficacy of Bioinformatics Algorithms on Biological Data**

Committee: **Chair:** Nick W. Sumner
Associate Professor, Computing Science

Leonid Chindelevitch
Co-Supervisor
Associate Professor, Computing Science

Maxwell Libbrecht
Co-Supervisor
Assistant Professor, Computing Science

Tamon Stephen
Committee Member
Professor, Mathematics

Martin Ester
Examiner
Professor, Computing Science

Alexandre Bouchard-Côté
External Examiner
Associate Professor, Statistics
University of British Columbia

Abstract

During my PhD studies I have worked on two projects. The first is about speeding up a recursive algorithm, so-called FK-B, that certifies whether two given monotone Boolean functions in the form of Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF) are dual or not and in case of not being dual it returns a conflicting assignment (CA), i.e. an assignment that makes one of the given Boolean functions *True* and the other one *False*. The FK-B algorithm is the core of the dualization procedure where it generates the dual of a given monotone Boolean function. In this regard, we propose six improvements/techniques applicable to the FK-B algorithm as well as the dualization process. Although these improvements/techniques do not reduce the time complexity, they considerably reduce the running time in practice that is important because of a wide range applications of the FK-B algorithm and dualization procedure. Here, to evaluate how effective the proposed improvements are, we apply them to the metabolic network analysis field where we find the minimal cut sets given elementary flux modes. The obtained results show a considerable speed up in comparison with the original dualization procedure.

In the second project, we investigate different data representations to predict drug resistance in Tuberculosis (TB). TB is an airborne disease which mostly affects the lungs. TB is treated using antibiotics, however, it has been revealed that some TB strains have become resistant to the drugs. Drug resistance in TB is usually diagnosed using a time-consuming and expensive laboratory experiment that is not always available. Nowadays, it has been discovered that mutations are mostly responsible for emergence of drug resistance. Considering this, a machine learning model, that is faster and cheaper than laboratory techniques, can be designed to predict drug resistance based on the detected mutations. In our study, we use deep neural networks to predict drug resistance in TB. To this end, we first detect the Single Nucleotide Polymorphisms (SNPs) in TB isolates. Then, we reconstruct gene and protein sequences and two other related data types. We design and experiment several neural networks with different input(s) and settings to get an insight on efficacy of each data type. The results show that protein sequence data as well as SNP data are the most informative data sources for predicting drug resistance. However, it is notable that processing sequence data requires so much computational resources.

Keywords: Dualization, Monotone Boolean function, Conjunctive Normal Form (CNF), Disjunctive Normal Form (DNF), Tuberculosis, Drug resistance, Prediction, Deep neural network, Multi-task neural network, Multi-input neural network.

Dedication

To my wonderful mom and dad, Homa and Esmaeil

&

To my amazing siblings Yasser and Naghmeh

&

To my beloved husband Saeed

&

To my little angel Elissa

Acknowledgements

First of all, I would like to express my sincere gratitude to Dr. Leonid Chindelevitch, my senior supervisor for his support, guidance and patience throughout my PhD studies. He is one of the smartest people I have had the privilege to work with and I was so lucky to get this opportunity. I would also like to say a special thank you to Dr. Maxwell Libbrecht and Dr. Tamon Stephen. I appreciate all their contribution of time, ideas, patience, and understanding. My gratitude also goes to my committee members, Dr. Martin Ester and Dr. Alexandre Bouchard-Côté, for their valuable comments and feedbacks.

I am very grateful to Simon Fraser University and School of Computing Science for providing me with the opportunity to obtain several fellowships/scholarships including Simons Foundation Doctoral Entrance Fellowship, CMPT Graduate Fellowships, ODSI Graduate Scholarship, Brian J. Blaha Memorial Graduate Scholarship (twice) and The Kaltenegger Family Graduate Scholarship.

My friends, lab mates, colleagues and research team were also part of my pleasant time at Simon Fraser University. Especially, I offer enduring gratitude to Faezeh Bayat, Maryam Hayati, Alice Yue, Hooman Zabeti, Amir Hosein Safari, Mohammad Hossein Rezaie and Reza Miraskarshahi all for cherished time spent together in the lab, and in social settings.

My absolute heartfelt gratitude also goes out to my parents and my siblings for their beautiful and loving spirit, encouragement and support all through my life. I am eternally grateful and blessed to have been gifted such a family. Without their pure love, tremendous understanding and encouragement in the past years, it would be impossible for me to complete my studies.

Last but not least, I devote a special thanks to the love of my life, Saeed, for always making me feel special and loved. I deeply appreciate his continued and unfailing love, support and understanding during my studies that made the completion of thesis possible.

Table of Contents

List of Tables

List of Figures

Summary of Contributions

In this thesis we considered two problems: speeding up Fredman-Khachiyan algorithm B (FK-B) to generate dual of a monotone Boolean function and predicting drug resistance in tuberculosis using deep neural network.

For the first problem, we developed several techniques/modifications to speed up the FK-B algorithm. The techniques/modifications include:

- Reducing the number of redundancy tests,
- Finding multiple conflicting assignments,
- Using a hash table,
- Using weighted approach to select the splitting variable,
- Choosing order of settings for a non-μ-frequent splitting variable and
- Shrinking the Boolean functions in the intermediate steps of FK-B.

It is worth mentioning that we are the first ones to propose these techniques/modifications.

For the second problem, we used deep neural network with different input types and different architectures to predict drug resistance in tuberculosis.

- In addition to SNP absence/presence data that is usually used in literature, we used new data types like protein sequence data as well as mutation type data for the first time.
- While we considered common architectures like fully connected, wide-n-deep and residual networks and tried to find the best hyperparameters using Bayesian optimization, we utilized background knowledge like gene-pathway information and binary patterns of resistance labels to modify the architectures.
- We tried data fusion, i.e. integrating multiple data sources, aiming to improve the prediction performance.

Part I

Speeding Up the Structural Analysis of Metabolic Network Models Using the Fredman-Khachiyan Algorithm B

Chapter 1

Introduction

Boolean functions, defined as functions whose input is a vector $x \in \{0, 1\}^n$ and whose output is $f(x) \in \{0, 1\}$, are a powerful modeling tool in a variety of settings. In many applications, such as those described in [1], the relevant Boolean functions have a natural monotone structure, meaning that $x \leq y \implies f(x) \leq f(y)$, with the vector inequality interpreted component-wise. For this reason, they can be fully represented and analyzed in terms of either their minimal true settings or their maximal false settings. The **dualization** problem for monotone Boolean functions, which consists of translating between these two representations, is both a deep theoretical question as well as a practically important challenge. It is closely related to the **generation** problem, which consists of enumerating all the minimal true and maximal false settings of a monotone Boolean function specified as an oracle, i.e. providing the value of $f(x)$ given an input x [2].

The dualization problem has numerous applications in subfields of mathematics such as graph theory (computing the transversal of a hypergraph), combinatorics (finding minimal hitting sets), and machine learning (model-based fault diagnosis), as well as more applied fields, including security, networking, distributed systems and computational biology. Our interest in the problem stems from computational biology, where we seek to perform a complete structural analysis of a metabolic network model by generating its elementary flux modes (EFMs) and minimal cut sets (MCSs), following [3].

The verification version of the dualization problem, also called the **duality** problem in the literature, consists of deciding whether a list of maximal false settings and a list of minimal true settings define the same monotone Boolean function. The computational complexity of the duality problem is not tightly characterized. Fredman and Khachiyan [4] found two novel algorithms for this decision problem that also extend to oracle-based generation of both the main function and its dual [5]. These algorithms, called FK-A and FK-B, either certify the duality or generate a new minimal true or maximal false setting in quasi-polynomial time in the joint size of the two lists. Their behaviour in practice is poorly understood. The only available open-source implementation for oracle-based generation is cl-jointgen [6] for FK-A, though some experiments on FK-A and FK-B are described in [7], and an FK-A based dualization algorithm is also available [8].

In this project, we address some computational challenges of using the FK-B algorithm for dualizing a monotone Boolean function. Our techniques can also be directly applied to the setting of jointly generating the minimal true and maximal false settings of a monotone Boolean function given as an oracle. While motivated by metabolic networks, our techniques are completely general.

A preliminary version of this work [9] includes results on three basic modifications to improve the performance of the FK-dualization procedure. These are producing multiple conflicting assignments in a single iteration, substantially reducing the number of redundancy tests during the execution of FK-B, and using a memoization technique to speed up dualization. We showed that each improvement alone produces a substantial speed-up, and in combination, they result in an order of magnitude speed gain relative to a naive (unoptimized) implementation.

Here we extend that work by introducing three additional improvements that speed up the FK-B dualization algorithm. Briefly, these are choosing the splitting variable based on information learned in the previous stages, choosing whether to first set a variable to *true* or *false* when the the variable is almost equally frequent in both the CNF and DNF, and shrinking the CNF and DNF during dualization. The first two improvements are heuristics that use previous information to make a decision at the current state, while the third one is an exact test, but only applies in a special case. Our results show that these modifications further improve the performance of the FK-B dualization algorithm, speeding it up by up to an additional order of magnitude.

Chapter 2

Definitions

Let $n \in \mathbb{N}$ be fixed. We write $\mathcal{B}$ to denote the set $\{0, 1\}$. A Boolean function $f : \mathcal{B}^n \to \mathcal{B}$ is *monotone* if $f(s) \leq f(t)$ for any two vectors $s \leq t \in \mathcal{B}^n$, where the inequality is interpreted component-wise. In other words, replacing a 0 with a 1 in the input cannot decrease f's value. Monotone functions are precisely those that can be constructed using the OR and AND operations, without using any NOTs (negations). We denote the negation of any $x \in \mathcal{B}$ by $\bar{x}$.

The dual of a Boolean function f is the function f^d defined by:

$$f^d(x) = \overline{f(\overline{x})} \tag{2.1}$$

for all $x = (x_1, x_2, \ldots, x_n) \in \mathcal{B}^n$, where $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$.

A monotone Boolean function f is said to be in Disjunctive Normal Form (DNF) if it is represented as an OR of ANDs, i.e. as

$$f = \bigvee_{j=1}^{m} M_j, \text{ where } M_j = \bigwedge_{i \in T_j} x_i \tag{2.2}$$

for a collection of m sets $T_1, T_2, \ldots T_m$.

Here, the monomials M_j are called *implicants* of f. If the underlying sets T_j satisfy the *Sperner property*, i.e. $T_j \not\subset T_k$ whenever $j \neq k$, then each M_j is a *prime implicant* of f and m is called the *size* of f. In this case, the point $x \in \mathcal{B}^n$ defined by

$$x_i = \begin{cases} 1 & \text{if} \quad i \in T_j \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

is a *minimal true point* of f; indeed, for this x we have $f(x) = 1$ and $f(y) = 0$ for any $y < x$, where $y < x$ means that $y \leq x$ and $y \neq x$.

Similarly, a monotone Boolean function f is said to be in Conjunctive Normal Form (CNF) if it is represented as an AND of ORs, i.e. as

$$f = \bigwedge_{j=1}^{m} C_j, \text{ where } C_j = \bigvee_{i \in S_j} x_i \tag{2.4}$$

for a collection of m sets $S_1, S_2, \dots S_m$.

Here, the clauses C_j are called *implicates* of f. Once again, if the underlying sets S_j satisfy the Sperner property, then each C_j is also called a *prime implicate* of f and m is called the *size* of f. In this case, the point x defined by

$$x_i = \begin{cases} 0 & \text{if} \quad i \in S_j \\ 1 & \text{otherwise} \end{cases} \tag{2.5}$$

is a *maximal false point* of f; indeed, for this x we have $f(x) = 0$ and $f(y) = 1$ for any $y > x$.

Lastly, we define the *support* of x, denoted *supp*(x), as the set $\{i \in \{1, 2, \dots, n\} \mid x_i = 1\}$.

We focus on two related problems for monotone Boolean functions, duality and dualization:

1. **Duality**: are two monotone Boolean functions defined by a DNF and a CNF equivalent?

2. **Dualization**: compute the CNF equivalent to a given monotone DNF.

These two problems can be easily transformed into one another, as we explain below. The dualization problem is equivalent to *Transversal Hypergraph Generation*, also called the *Minimal Hitting Set Enumeration* problem. For background on these problems and applications, we refer the reader to [1, 7, 10, 11] and references therein.

The algorithm with the best known worst-case performance guarantee for the dualization of a monotone Boolean function f, called the FK-B algorithm, has incremental quasi-polynomial running time [4]. More precisely, starting from a description of f in DNF, each iteration obtains an additional clause or verifies duality of the equivalent CNF, in $N^{o(logN)}$ time, where N is the total size of the DNF and the current, possibly incomplete, CNF.

2.1 The FK-Dualization Algorithm

Algorithms 1 and 2 show the FK dualization and FK-B duality checking procedure respectively, following the presentations of [11] and [12].

Algorithm 1 starts with an empty CNF, called C, and a complete DNF, called D. In a loop, it checks the equivalence of C and D using the FK-B algorithm. This algorithm either certifies duality, in which case the dualization is complete and the CNF is returned, or it returns a *conflicting assignment (CA)*, that is, an assignment X^* on which the CNF and the

Algorithm 1 Fredman-Khachiyan Dualization

Input: A monotone Boolean function f on $\mathcal{B}^n$ expressed by its complete DNF D.
Output: The complete CNF of $C = D^d$.

```
 1: function $FK$-Dualization($C$)
 2:     $C = \emptyset$;
 3:     Call $FK$-$B$ on the pair $(C, D)$;
 4:     if the returned value is $\emptyset$ then
 5:         return C
 6:     else
 7:         let $X^* \in \mathcal{B}^n$ be the point returned by $FK$-B;
 8:         compute a maximal false point of $C$, say $Y^*$, such that $X^* \leq Y^*$;
 9:         $C = C \wedge \bigvee_{j \in supp(\overline{Y^*})} x_j$ ;
10:         Go to Step 3.
```

DNF take different values, i.e. either $CNF(X^*) = 1$ and $DNF(X^*) = 0$, or $CNF(X^*) = 0$ and $DNF(X^*) = 1$. It then identifies a maximal false point Y^* greater than X^*, and adds its complement to the current CNF as a new clause.

Algorithm 2, FK-B, takes two Boolean functions in the form of one CNF and one DNF, and checks if the inputs are equivalent for all possible Boolean assignments via a recursive approach. If they are not equivalent, it returns a conflicting assignment. The first step in this algorithm is to remove redundant clauses from both the CNF and the DNF, which is accomplished by comparing every pair of clauses c, c' separately in the CNF and the DNF, and eliminating c' whenever $c \leq c'$.

Line 3 checks three necessary conditions for a CNF and DNF to be equivalent. These are:

1. Existence of a non-empty intersection between every clause in CNF and every monomial in DNF; if this condition is violated, a conflicting assignment is a monomial m from the DNF that does not intersect some clause of the CNF.

2. The presence of exactly the same variables in the CNF and the DNF; if this condition is violated, with x being a variable in the DNF that does not appear in the CNF, a conflicting assignment is obtained by $m - \{x\}$, where m is a monomial of the DNF that includes x. Alternatively, if x is a variable in the CNF that does not appear in the DNF and c is a clause of the CNF that includes x, a possible conflicting assignment is obtained by $\{V - c\} \cup x$, where V is the set of all variables.

3. The maximum length of a monomial in the DNF is at most the number of clauses in the CNF, and the maximum length of a clause in the CNF is at most the number of monomials in the DNF. In a case that there is a monomial $m \in DNF$ that contains more variables than there are clauses in CNF, if $m' \subset m$ is a proper subset of m

satisfying $m' \cap c \neq \emptyset$ for every clause c of CNF, a conflicting assignment is m'. In the other case that there is a clause $c \in CNF$ that contains more variables than number of monomials in DNF, if $c' \subset c$ is a proper subset of c satisfying $c' \cap m \neq \emptyset$ for every monomial m of DNF, the conflicting assignment would be $V - c'$, where V is the set of all variables.

We note that in each case, the conflicting assignment can be found in polynomial time in the length of the CNF and the DNF.

Line 4 addresses the case in which either the CNF or the DNF is very small, and the equivalence can be checked directly via exhaustive search through the tree of assignments in the CNF or the DNF, whichever is smaller. If they are inequivalent the procedure returns a conflicting assignment; otherwise, it returns $\emptyset$.

The recursive part of the algorithm starts from line 6 where a splitting variable is selected; based on its frequency in the CNF and the DNF, the splitting variable is set to either $False$ or $True$, and after that the current CNF and DNF are simplified by fixing this variable assignment and generating a recursive call to the FK-B algorithm on the new, smaller problem. Note that in line 7, a variable x is called at most μ-frequent in D if its frequency in D is at most $1/\mu(|D| \cdot |C|)$, i.e. $|\{m \in D : x \in m\}|/|D| \leq 1/\mu(|D| \cdot |C|)$, where $\mu(n) \sim \log n / \log \log n$ is the largest integer k such that $k^k \leq n$. A similar definition applies to C.

Given that the FK-B algorithm, Algorithm 2, returns the first conflicting assignment (CA) that it finds between the given CNF and DNF, computing the dual of a given DNF using Algorithm 1 requires $N_{CNF} + 1$ iterations, where N_{CNF} is the size of the CNF that is dual to the given DNF.

2.2 Mapping the Dualization Problem to Metabolic Network Structures

The dualization problem can be mapped to a variety of problems. The problem of interest in our application is that of analyzing the structure of a metabolic network model by finding its Elementary Flux Modes (EFMs) and Minimal Cut Sets (MCSs). The problem of finding the smallest size EFM or MCS in a metabolic network is NP-hard [13], while the problem of finding all the EFMs or all the MCSs has an unknown complexity status, and can only be solved in reasonable time for small or medium-size metabolic networks [1, 3, 14–17].

In the context of a metabolic network model M, the monotone Boolean function f is defined on the characteristic vectors of subsets of reactions via $f(x) = 1$ if and only if the support of x enables biomass production. In this setting the minimal true points of f are

Algorithm 2 The Fredman-Khachiyan Algorithm B (*FK*-B)

Input: Monotone DNF D and CNF C.

Output: $\emptyset$ in case of equivalence; otherwise, an assignment $\mathcal{A}$ with $D(\mathcal{A}) \neq C(\mathcal{A})$.

```
function $FK$-B$(C, D)$
    make $D$ and $C$ irredundant;
    if a necessary condition is violated then return conflicting assignment;
    if $\min\{|D|, |C|\} \leq 2$ then return $\emptyset$ or the conflicting assignment found by a direct check;
    else
        choose a splitting variable $x$
        if $x$ is at most $\mu$-frequent in $D$ then
            $\mathcal{A} \leftarrow FK$-B$(D_1^x, C_0^x \wedge C_1^x)$ // recursive call for $x$ set to $False$
            if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A}$
            for all clauses $c \in C_0^x$ do
                $\mathcal{A} \leftarrow FK$-B$(D_0^{c,x}, C_1^{c,x})$ // see ⟨1⟩ below
                if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{x\}$
        else if $x$ is at most $\mu$-frequent in $C$ then
            $\mathcal{A} \leftarrow$ FK-B$(D_0^x \vee D_1^x, C_1^x)$ // recursive call for $x$ set to $True$
            if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{x\}$
            for all monomials $m \in D_0^x$ do
                $\mathcal{A} \leftarrow FK$-B$(D_1^{m,x}, C_0^{m,x})$ // see ⟨2⟩ below
                if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{m\}$
        else
            $\mathcal{A} \leftarrow FK$-B$(D_1^x, C_0^x \wedge C_1^x)$ // recursive call for $x$ set to $False$
            if $\mathcal{A} = \emptyset$ then
                $\mathcal{A} \leftarrow FK$-B$(D_0^x \vee D_1^x, C_1^x)$ // recursive call for $x$ set to $True$
                if $\mathcal{A} \neq \emptyset$ then return $\mathcal{A} \cup \{x\}$
    return $\mathcal{A}$
```

⟨1⟩: $D_1^x \equiv C_0^x \wedge C_1^x$: recursive call for all maximal non-satisfying assignments of C_0^x for x set to $True$. $D_0^{c,x}$ and $C_1^{c,x}$ denote the formulae we obtain by setting all the variables in c to $False$.

⟨2⟩: $D_0^x \vee D_1^x \equiv C_1^x$: recursive call for all minimal satisfying assignments of D_0^x for x set to $False$. $D_1^{m,x}$ and $C_0^{m,x}$ denote the formula we obtain by setting all the variables in m to $True$.

called elementary flux modes (EFMs) and the maximal false points of f are called minimal cut sets (MCSs), see for example [18, 19]. In the experimental results section we apply the *FK*-B algorithm with the proposed improvements to metabolic network models and study the impact these improvements have on its performance in generating the MCSs based on the pre-computed set of EFMs.

When analyzing metabolic networks to obtain the MCSs from the EFMs, it is beneficial to pre-process the given EFMs before starting the dualization procedure. The preprocessing involves three steps:

1. removing any reactions that are not part of any EFMs (also known as blocked reactions [20, 21]), which correspond to unused variables;
2. removing any reactions involved in all the EFMs (also referred to as essential reactions [20, 21]), adding them as singleton MCSs during post-processing;
3. collapsing any group of k reactions whose presence/absence patterns in the EFMs are identical (a special case of this is referred to as enzyme subsets [20, 21]) into a single reaction, expanding each of the final MCSs involving this reaction into k copies during post-processing.

The pre-processing and post-processing steps are not necessary and can be omitted. However, they reduce the original problem and make the dualization procedure faster, so we routinely perform them. Please note that, although they are motivated by our specific application, they may be applied to any dualization problem without modification.

Chapter 3

Methods

3.1 Reducing the Number of Redundancy Tests in the FK-B Algorithm

In Algorithm 2, the first step (line 2) removes redundancy in both the CNF and the DNF. Redundancy removal involves an all-pairs comparison of the clauses (the monomials) in the CNF (the DNF) and removes any supersets found. In logic, removing redundancy is equivalent to applying the absorption rule to simplify the Boolean function. While there are algorithms that slightly improve the asymptotic running time, say by a log factor, not practical improvement in the quadratic running time is known, and it may be that none exists [22].

This procedure is a bottleneck due to the large number of pairwise comparisons that must be performed in each recursive call, and this is compounded by the fact that when we perform dualization, the FK-B algorithm is called many times to find the clauses of the CNF.

We reduce the number of redundancy tests performed in FK-B, and consequently in FK-dualization, by noting that when we set a variable to $True$ ($False$) in the CNF (DNF), there is no need to check the redundancy of the CNF (DNF) in the next recursive call because such a setting results in clauses (monomials) in the CNF (DNF) being removed, which cannot generate any additional redundancy.

This is implemented using two binary flags, which are set if the redundancy in the CNF (the DNF) needs to be checked, and cleared otherwise. FKR, the algorithm with reduced redundancy checks, differs from the baseline, algorithm 2, in the following ways. First, the redundancy of the CNF (the DNF) is only checked if the corresponding flag is set. Second, in lines 8 and 20, where a variable x is set to $False$, the flag for the CNF is set and the flag for the DNF is cleared, since we only need to check the redundancy in the CNF, not the DNF. Conversely, in lines 14 and 22, the variable x is set to $True$, so the flag for the DNF is set and the flag for the CNF is cleared. Note that in lines 11 and 17, it is assumed that variable x is respectively set to $True$ and $False$, and then the variables in c and m are respectively set to $False$ and $True$. For this reason, redundancy can be produced in those lines, so the next call to FKR needs to check the redundancy in both the CNF and the DNF.

3.2 Finding Multiple Conflicting Assignments

Given that we can use any conflicting assignment between the current CNF and DNF to compute a new clause in the CNF, we can find Multiple Conflicting Assignments (MCAs) at the same time to generate more than one clause per iteration of the dualization procedure, and reduce the running time of the algorithm by reducing the total number of required iterations.

To this end, MCAs can be computed in the three situations below without a significant increase of computational effort. The first two situations arise during the assessment of the first two conditions necessary for equivalence in FK-B.

The first condition that we assess in the FK-B algorithm is the existence of a non-empty intersection between every clause in CNF and every monomial in DNF. If there is no intersection between monomial m in the DNF and clause c in the CNF, then m makes the DNF *True* and the CNF *False*, so it is a CA. During the dualization procedure, especially early on, many of the monomials and clauses have no intersection. We thus consider intersections between every clause in the CNF and every monomial in the DNF at once and can return more than one CA.

The second condition that we assess in the FK-B algorithm is the presence of exactly the same variables in the CNF and the DNF. If this condition is not met, a CA is determined from the extra variable(s) in the CNF or the DNF. If multiple variables are present in exactly one of the CNF and the DNF, we consider all possible conflicting assignments instead of returning only one.

The third situation in which we compute MCAs is in the case where $\min(|C|, |D|) \leq 2$. In such cases, the conflicting assignments are directly derived from Boolean algebra. We only consider the case $|C| \leq 2$ here; the case $|D| \leq 2$ is symmetric and is processed analogously.

The following cases may happen during this step:

- $|C| = 1$

  Here, we look for any variable x in the unique CNF clause, denoted $C[1]$, such that D does not contain the singleton monomial x, in which case $\{x\}$ is a conflicting assignment.

- $|C| = 2$

  In this case, we denote the two clauses by $C[1]$ and $C[2]$. There are three sub-cases:

  - Let $A_0 := C[1] \cap C[2]$. If $x \in A_0$ is a variable such that D does not contain the singleton monomial x, then $\{x\}$ is a conflicting assignment.

- Let $A_1 := C[1] - C[2]$ and $A_2 := C[2] - C[1]$. Note that $A_1, A_2 \neq \emptyset$ because the CNF is non-redundant. If some monomial m in D is a subset of one of the A_i's, then $\{x | x \in m\}$ is a conflicting assignment.
- Let $(x, y) \in A_1 \times A_2$, with A_1 and A_2 defined above. If no monomial in D is a subset of $\{x, y\}$ then $\{x, y\}$ is a conflicting assignment.

In all the aforementioned cases, whenever more than one conflicting assignment is found, we return all of them. An issue regarding the MCAs is that sometimes more than one CA can be mapped to a single clause in the CNF. In this case, we use the unique clauses resulting from the MCAs.

3.3 Dealing with Repeated Subproblems

Given that FK-B is a recursive algorithm which is called many times during dualization, certain subproblems are solved very frequently. In this case, memoizing (storing for future retrieval) these subproblems and their solutions via certificate CA's (with dual pairs being characterized by an empty set of CAs) is beneficial, as it can reduce the running time of both the FK-B algorithm as well as FK-dualization as a whole.

To this end, we use a hash table whose keys are combination of the CNF and the DNF and whose values are the CAs. To implement this idea, we compute the key for a given CNF and DNF prior to calling the FK-B algorithm. If it is already in the hash table, we retrieve the value, i.e. the corresponding CAs, bypassing a recursive call to FK. Otherwise, we call FK-B and store any computed CAs as a new record in the hash table.

As we experimented with different settings in the implementation of this memoization technique, we realized that solving small subproblems, with $|C| \leq 2$ or $|D| \leq 2$, from scratch was faster than storing them in the hash table and retrieving the CAs because the special case in line 4 of Algorithm 2 applies to them. Thus, we do not use hashing on these subproblems in our implementation.

On the other hand, it is challenging to store large subproblems because there are so many of them. This may be reduced to a degree by storing functions only up to symmetry. An MBF $f(x_1, x_2, \ldots, x_n)$ is *equivalent* to another MBF $g(x_1, x_2, \ldots, x_n)$ if there is a permutation $\sigma \in S_n$ such that $f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}) = g(x_1, x_2, \ldots, x_n)$. For example, the function $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$ is equivalent to $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$ via the permutation $\sigma = (12)(3)$. We can then try to put the MBF corresponding to the DNF into a *canonical form* for hashing. In general this is difficult, but we can do it for functions on $n \leq 6$ variables by simply generating all $n!$ permutations, multiplying the function's representation by a fixed matrix, and scanning through it once, choosing the lexicographically smallest function, as described in [23]. The same permutation is applied to the CNF, leading to the overall representation used for hashing.

3.4 Choosing the Splitting Variable by a Weighting Approach

Selecting the splitting variable in the FK-B algorithm plays a key role in the speed of the algorithm. The usual method of choosing the splitting variable is to select the common most frequent variable in both the CNF and the DNF. Here, we present a weighting approach to choosing the splitting variable which may reduce the number of FK calls required.

Algorithm 3 Updating variable weights. This algorithm is called between dualization iterations.

Input: A table W with rows indexed by the variables, and two columns: 1) Weight and 2) Depth.

Output: The updated weight table W.

```
function UpdatingWeights($W$)
    $W[Depth] = \boldsymbol{Normalize}(W[Depth])$;
    $W[Weight] = W[Weight] \odot W[Depth]$; // element-wise multiplication
    $W[Weight] = \boldsymbol{Normalize}(W[Weight])$;
    return $W$;
```

In this approach, each variable x_i is associated with a weight w_i and a depth d_i. The weight w_i is an estimate of the suitability of variable x_i to be chosen as the splitting variable, and the depth d_i is the smallest recursion depth at which variable x_i has been selected as the splitting variable during the FK-dualization procedure so far . The initial values for w_i and d_i are 100 and ∞ for each i, respectively.

The depth value d_i is updated during a dualization iteration, when the variable x_i is selected as the splitting variable. When this happens, if the current depth is d^*, d_i is updated to $\min(d_i, d^*)$. On the other hand, the weight w_i is updated between dualization iterations according to Algorithm 3. Briefly, the depths are normalized, the current weights are multiplied by the normalized depths, and then they are themselves normalized to form a discrete probability distribution.

In this approach, the splitting variable in each iteration of FK-dualization is chosen at random from the distribution defined by the weights; a higher weight corresponds to a higher probability of being selected as the splitting variable.

3.5 Choosing the Order of Settings for a Non-μ-frequent Splitting Variable

In Algorithm 2, if the chosen splitting variable is not μ-frequent in either the CNF or the DNF, the splitting variable is first set to *False*, and if a conflicting assignment is not found, it is set to *True*. There is no theoretical reason behind this order for the settings, and we now present two heuristic approaches for deciding which of the two possible orders can lead to a conflicting assignment faster.

Both approaches make decisions based on the history of setting a variable to *True* or to *False* first in the previous iterations of FK-dualization. The first approach considers the variable's entire history, i.e. the values assigned to it in all the conflicting assignments found so far, and decides to first set the splitting variable to the value found in the *majority* of previously identified conflicting assignments. The second approach makes the decision in the exact same way, but only considers the conflicting assignments found in the $K = 5$ most recent iterations of FK-dualization.

Note that in the both approaches, if there is no history of assignments to a variable, the default order, first setting *False*, then setting *True*, is chosen.

3.6 Shrinking the CNF and the DNF in the Intermediate Steps of FK-B

In the intermediate steps of running the FK-B algorithm on a CNF and a DNF, it frequently happens that a variable appears as a singleton in the CNF, and also appears in all the monomials in the DNF. Alternatively, it can happen that a variable appears as a singleton in the DNF, and also appears in all the clauses in the CNF. In such a situation, the variable in question can be removed from both the CNF and the DNF without affecting equivalence (i.e. the original DNF and CNF are equal if and only if the reduced ones are). This operation reduces the problem size and lowers the number of recursive calls to FK-B.

This shrinkage step can be performed as a pre-processing step in the FK-B algorithm, before checking the three necessary conditions for equivalence.

Chapter 4

Experimental Results

As mentioned in Section 2.2, the dualization problem can be used to solve the problem of finding the MCSs given the EFMs in a metabolic network model. Here, we use the metabolic models available in the BioModels database[1] to assess the proposed algorithms. To this end, we selected 19 small and medium-size models to be able run several FK variants on them in a reasonable time. Table 4.1 shows the characteristics of the models.

To prepare these models for the application of our dualization algorithm, we performed the following steps:

1. Parsing the biological models using the SBML parser in *MATLAB* to obtain a stoichiometric matrix containing the metabolites as rows and the reactions as columns;

2. Applying EFMTool [14] or FluxModeCalculator [24] to extract the EFMs into a matrix;

3. Converting the matrix into a binary one by setting all non-zero values to one, and preprocessing it using the steps outlined in Section 2.2; the resulting matrix is used as the input DNF for the dualization problem in order to find the MCSs, corresponding to the CNF.

Different experiments have been designed to elucidate the efficiency of the proposed modifications to the original FK algorithm, individually and in combination. For this purpose, different metrics have been used to compare them to the original FK algorithm.

In the following, we indicate the modification(s) made to the algorithm 2 using the letters *F*, *W*, *S*, *H*, *C*, *O* and *R*, defined as follows:

F : The splitting variable is the most **f**requent variable in the CNF and the DNF;

W : The splitting variable is chosen using the **w**eighting approach;

[1] `http://www.ebi.ac.uk/biomodels-main/publmodels`

Table 4.1: Characteristics of models. Metabolites: number of metabolites; EFMs: number of elementary flux modes (monomials in the DNF); $n_r^{<pre}$: Number of reactions (variables) before the preprocessing steps; $n_r^{>pre}$: Number of reactions (variables) after the preprocessing steps; $n_{MCS}^{<post}$: Number of minimal cut sets (clauses in the CNF) before the postprocessing steps; $n_{MCS}^{>post}$: Number of minimal cut sets (clauses in the CNF) after the postprocessing steps.

Model	Metabolites	EFMs	$n_r^{<pre}$	$n_r^{>pre}$	$n_{MCS}^{<post}$	$n_{MCS}^{>post}$
BIOMD0000000034	9	13	22	22	56	56
BIOMD0000000042	15	35	25	20	56	188
BIOMD0000000048	23	63	25	14	320	12960
BIOMD0000000089	16	20	36	28	192	15552
BIOMD0000000093	34	24	46	24	293	2001
BIOMD0000000094	34	23	45	23	293	667
BIOMD0000000106	25	12	32	17	14	512
BIOMD0000000107	14	11	23	13	14	448
BIOMD0000000108	9	50	17	17	60	60
BIOMD0000000110	15	12	22	15	48	864
BIOMD0000000162	32	60	45	20	675	1928934
BIOMD0000000163	16	12	26	21	156	1296
BIOMD0000000165	37	20	30	9	16	576
BIOMD0000000166	3	18	9	9	27	27
BIOMD0000000169	11	17	27	23	128	1536
BIOMD0000000170	7	10	17	15	32	128
BIOMD0000000171	12	16	26	23	65	264
BIOMD0000000173	26	14	26	10	17	4617
BIOMD0000000228	9	13	22	20	128	512

S : The CNF and the DNF are **s**hrunk during the intermediate steps of running FK;

H : A **h**ash table is used for small subproblems;

C : The **c**anonical form of the CNF and the DNF is stored in the hash table;

O : A heuristic is used to find the optimal **o**rder of settings for non-μ-frequent splitting variables;

R : The $K = 5$ most **r**ecent splits, not the full history, are used in the order heuristic (O) above.

We quantify the effectiveness of the proposed improvements using the following metrics:

1. Backtrack count: Quantifies how many recursive calls to the FK-B algorithm return no conflicting assignment. This count decreases with better splitting variable selection strategies.

2. Backtrack length: Quantifies the total depth of the recursive calls that return no conflicting assignment. This length decreases with better splitting variable selection strategies as well.

3. Iteration count: Quantifies the number of recursive calls to the FK-B algorithm. This count decreases with improved search strategies, and many of our heuristics may contribute to this.

4. Hash table hits: Quantifies the number of successful hash queries to bypass additional calls to the FK-B algorithm (used when the CNF and the DNF have size at least 3). This count increases when the same (or, with option C, equivalent) subproblem is solved multiple times.

5. Node count: Quantifies the total number of variable settings required to find all the conflicting assignments. This count decreases with a faster identification of the conflicting assignments.

In the following, we present the experiments we carried out and discuss their results. We note that the FK-dualization procedure is inherently stochastic, and we used a fixed *seed* to make the results reproducible. Changing this seed may alter the results, but our sensitivity analysis suggests that there is no significant impact on the relative contributions of each modification (data not shown).

4.1 Reducing the Number of Redundancy Tests

The redundancy test is a key bottleneck of FK-dualization, being called on each input to the FK-B algorithm. In this experiment we measure how much the flags proposed in Section 3.1 reduce the number of redundancy tests.

Figure 4.1 shows the total number of pairwise comparisons in the redundancy tests performed during FK-dualization. It suggests that the proposed modification decreases the number of comparisons required by 20% to 70%.

Given that this modification significantly reduces the running time of FK-dualization, we only discuss the FK-B algorithm with the modification reducing the number of redundancy tests from now on.

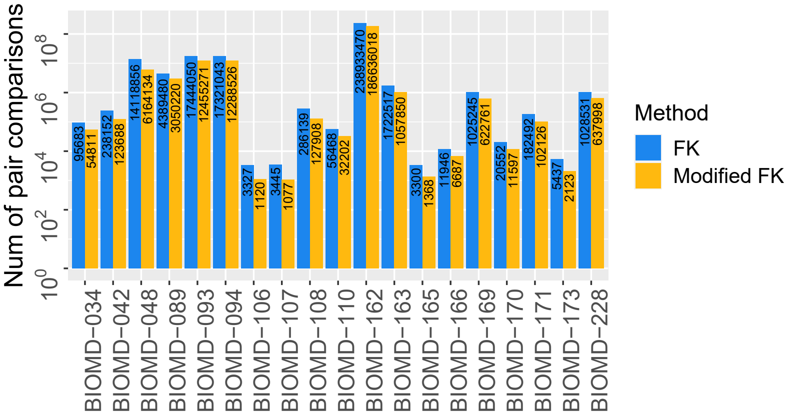


Figure 4.1: Total number of pairwise comparisons performed in the removing redundancy during FK-dulaization using the original FK algorithm and the modified version of FK which the number of redundancy tests are reduced. The numbers on each bar show the exact number of performed comparisons. Note that in the name of models on the horizontal axis, we have replaced seven '0' by '-' for the sake of readability.

4.2 Finding Multiple Conflicting Assignments

As discussed in Section 3.2, finding multiple CAs at the same time can reduce the iteration count for FK-dualization, each of which may require multiple FK iterations. Figure 4.2 shows the iteration counts for the original FK algorithm and the variant of FK that finds multiple CAs at the same time. It suggests that this modification can reduce the number of required iterations by 10% to 65%. Figure A.2 additionally shows the size of the CNF at each iteration.

Similar to the previous section, because this modification is clearly beneficial and independent of other changes, it will become part of the baseline for subsequent experiments.

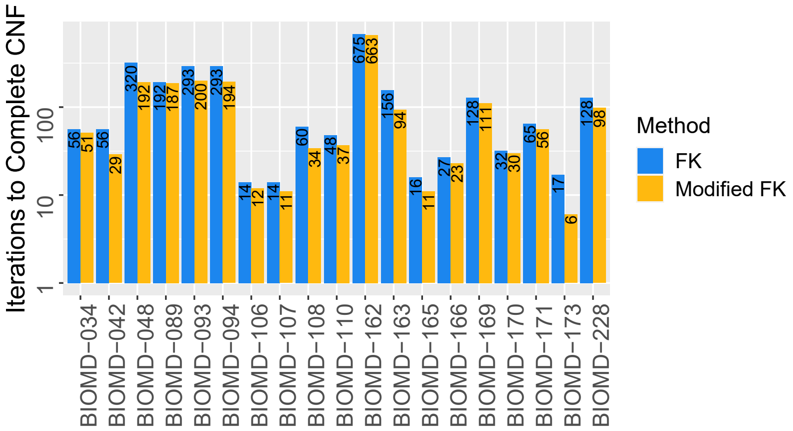


Figure 4.2: Number of FK-dulaization iterations to complete the CNF given the DNF using the original FK algorithm and the modified version of FK which multiple CAs are found at once. The numbers on each bar show the exact number of iterations. Note that in the name of models on the horizontal axis, we have replaced seven '0' by '-' for the sake of readability.

4.3 Comparing FK Variants Based on Splitting Variable Decisions

In this experiment, we compare seven variants of the FK-B algorithm (note that the results of the other possible variants are available as Supplementary File 1).

F : A baseline FK-B algorithm, following Algorithm 2 with reduced redundancy tests, multiple conflicting assignments and using the most frequent variable as the splitting variable;

FH : Same as the F variant, but also uses a hash table;

FHC : Same as the FH variant, but uses the subproblems' canonical form as the hash key;

FO : Same as the F variant, but uses the whole history to optimize the assignment order.

FOR : Same as the F variant, but uses the recent splits to optimize the assignment order.

FS : Same as the F variant, but also uses the shrinkage process.

W : Like the F variant, but uses the weighting approach to select the splitting variable instead.

Figure 4.3 displays the results of this analysis for each of the models. The left panels show the size of the CNF at each iteration of the FK-dualization algorithm. It suggests that for most of the models, W is the fastest method while F is the slowest one. The other modifications have similar effects to one another, but are superior to the baseline F method.

The right panels compare the methods based on the other metrics that we introduced. In these figures, the measurements whose values are zero are shown as bars under the baseline. Since hash tables are only used in the FH and FHC variants, only two bars appear above the baseline for the hash table hits. These variants also perform better on the backtrack count and backtrack length for 15/19 of the models. The iteration count is the lowest for the W variant for 12/19 of the models, and the node counts are strongly correlated with the iteration counts. Interestingly, the FH and FHC variants have the exact same number of hash table hits, meaning that storing the subproblems in canonical form to take equivalence into account provides no additional benefit.

In conclusion, considering both the left and the right panels suggests that W is generally the best at reducing the number of FK calls as well as the number of iterations required to compute the CNF.

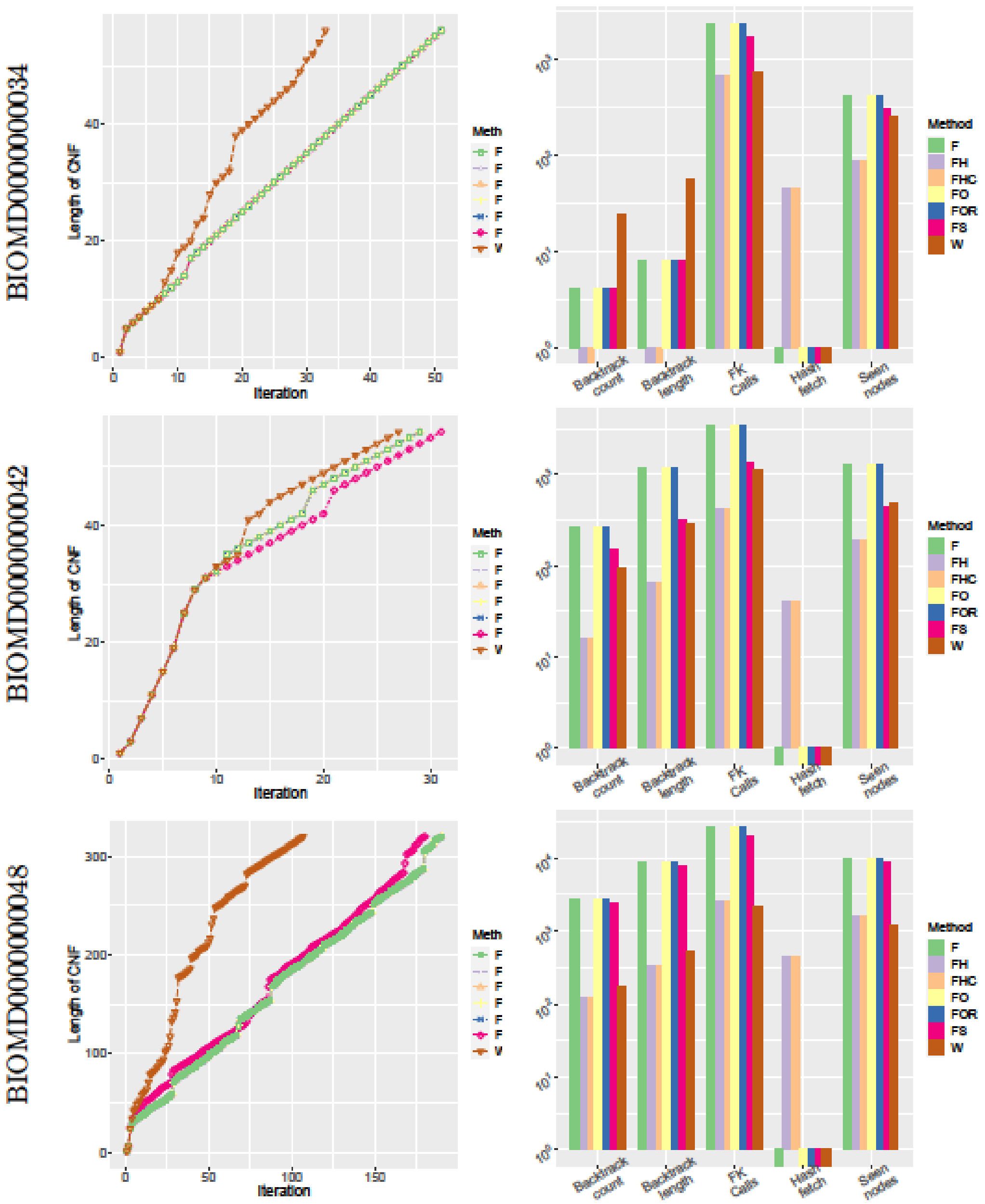


Figure 4.3: The figures in each row belongs to a model. The figures in the first column show progression of constructing CNF versus iterations in FK-dualization algorithm. The figures in the second column illustrate how beneficial each improvement is in FK-dualization based on five measures: 'Backtrack count' shows how many times wrong branches of tree of assignments have been chosen to go through that finally it had to return to the higher levels; 'Backtrack length' shows how deep it has gone through the wrong branches; 'Seen nodes' shows the number of variables that have been set to either *true* or *false* or both to reach to the conflicting assignment(s); 'FK Calls' indicates to the number of recursive calls to FK algorithm; and 'Hash fetch' shows the number of successful fetches to the hash table in case that keys are not stored in the canonical form and if $|C| < \tau$ and $|D| < \tau$ where $\tau = 3$, the hash table is not used.

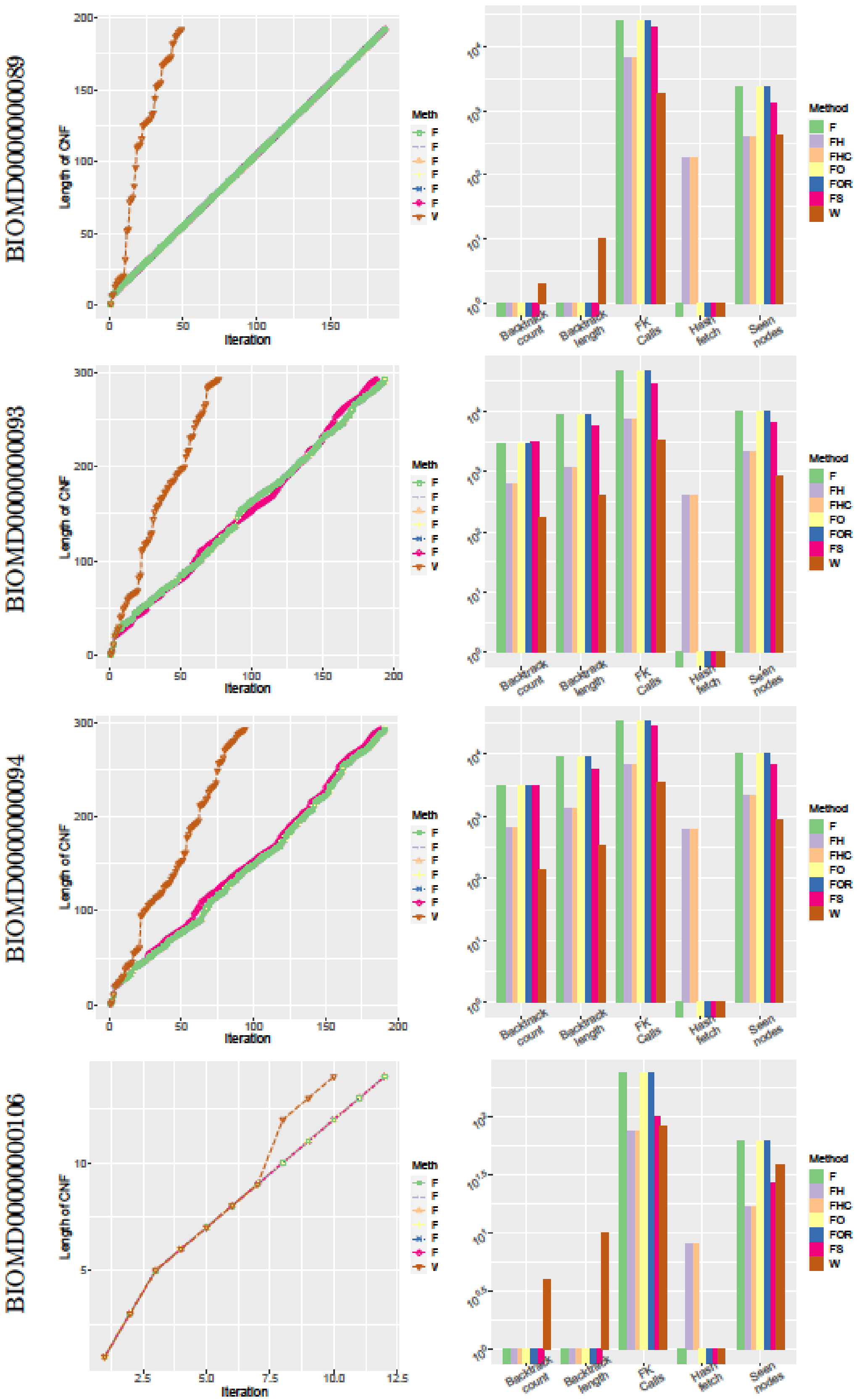


Figure 4.3: Continued.

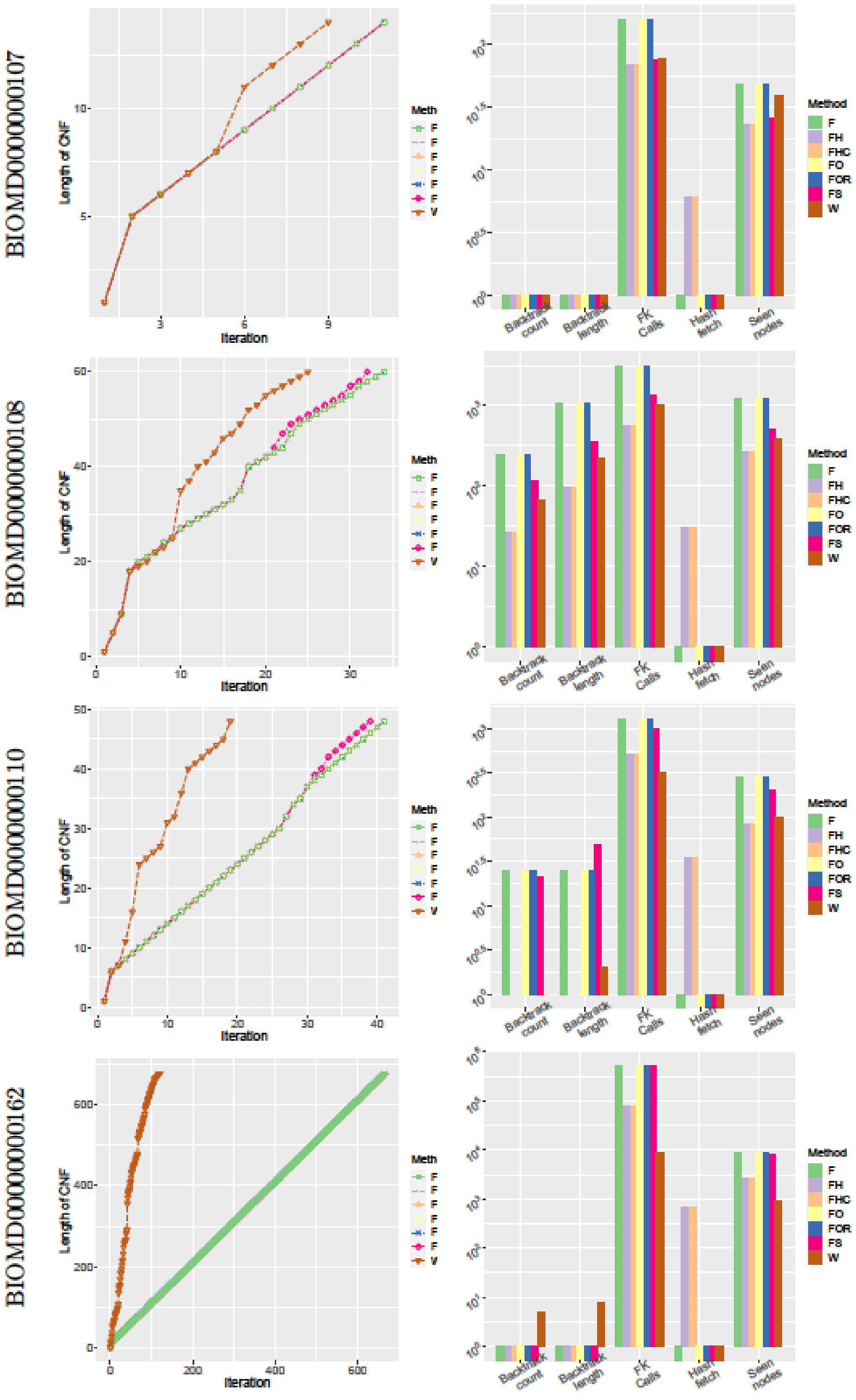


Figure 4.3: Continued.

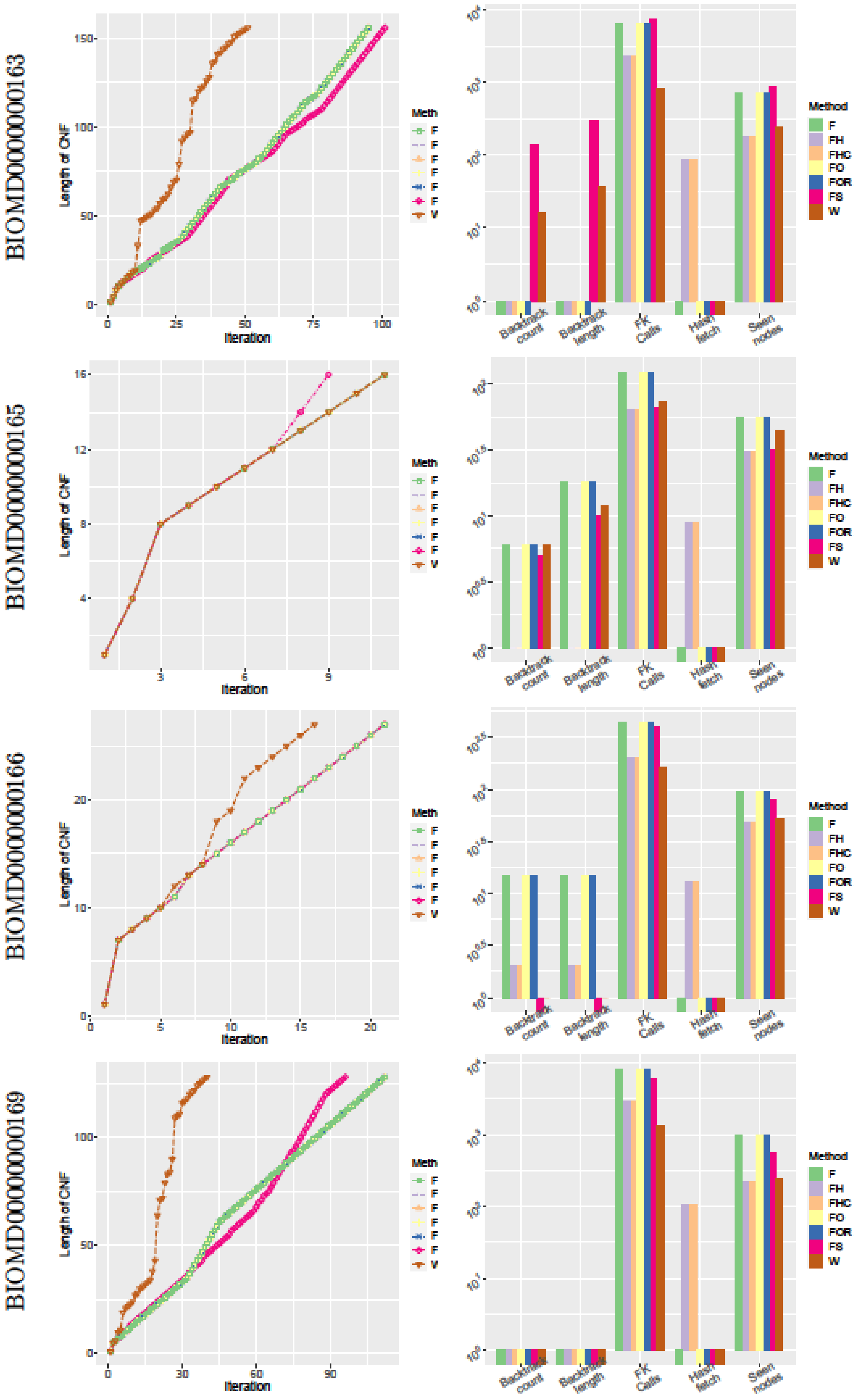


Figure 4.3: Continued.

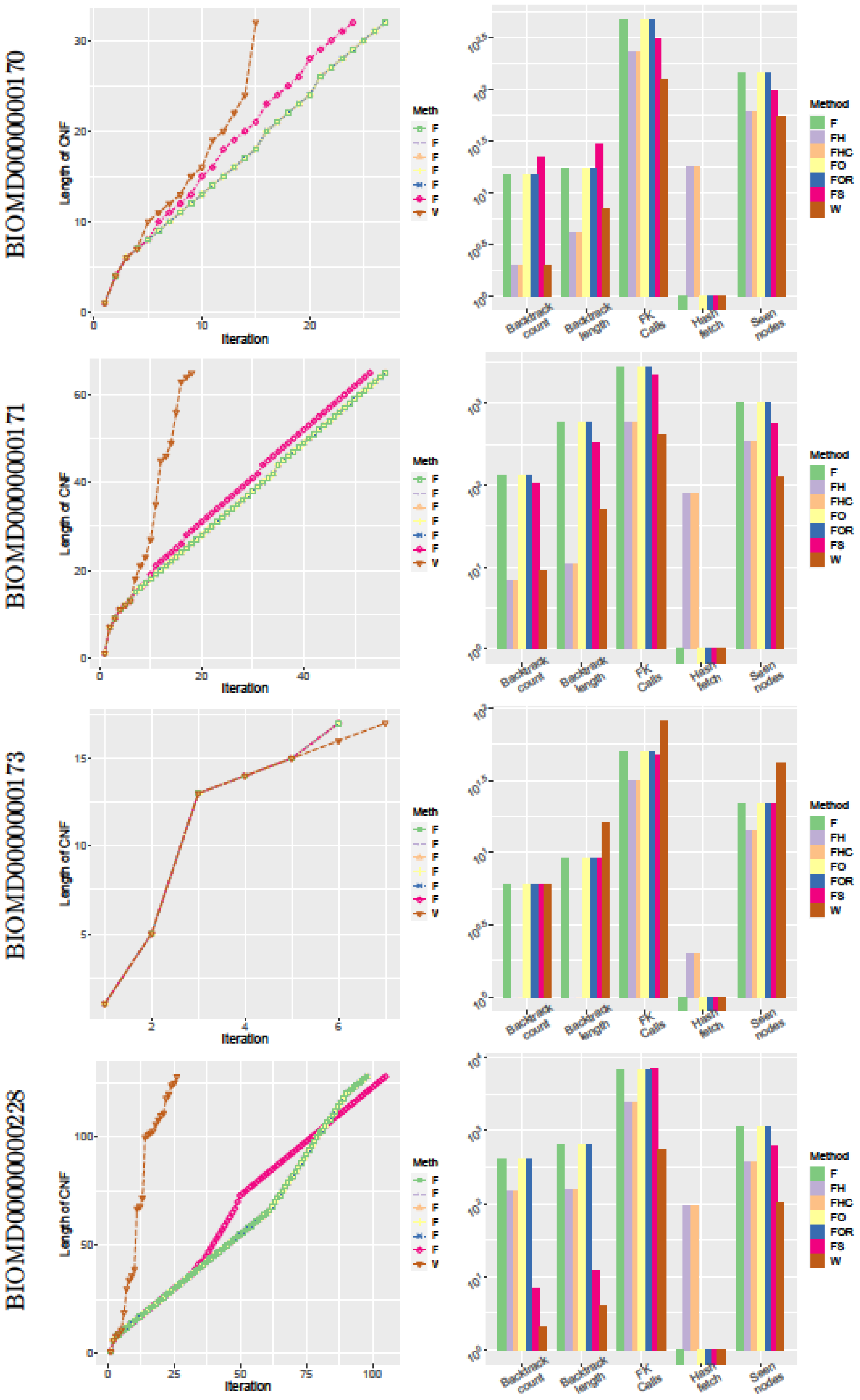


Figure 4.3: Continued.

4.4 Analysis of the Pareto Frontier

In this experiment, we try to find the *minimal* subsets of modifications to the FK algorithm required to achieve the minimum value of three key metrics - iteration count, backtrack length, and node count - for all of the models (all possible combinations of the modifications that minimize each of these metrics are shown in Supplementary File 2). Tables 4.2-4.4 show the results. As it can be seen some modifications, e.g. W (weighting) or S (shrinkage), are seen in several rows in all the tables. Such pattern can give us an idea about the benefit of each modification.

We also created Table 4.5 to summarize the optimal modifications across all three tables for each model. The last column in this table shows the common modifications, i.e. those needed to achieve an optimal value of each of the metrics. For 14/19 of the models, there is at least one common modification. The two most frequently seen common modifications are S (shrinkage) and H (hash table). However, W (weighting) is also a frequently occurring modification in the rest of the table.

Table 4.2: Achieving the minimum calls to FK using minimal FK improvements.

Model	FK calls	Successful HashFetch	Backtrack counts	Backtrack length	Seen nodes	Method
BIOMD0000000034	666	1	23	54	249	WH
BIOMD0000000042	253	46	8	12	126	FHS
BIOMD0000000048	2176	0	175	543	1207	W
BIOMD0000000089	1700	0	2	8	269	WS
BIOMD0000000093	2748	0	96	226	695	WS
BIOMD0000000094	2748	0	96	226	695	WS
BIOMD0000000106	63	6	0	0	15	FHS
BIOMD0000000107	37	0	0	0	16	WS
BIOMD0000000108	292	28	8	24	169	FHS
BIOMD0000000110	319	0	1	2	100	W
BIOMD0000000162	8244	0	4	12	853	WS
BIOMD0000000163	790	1	4	11	220	WH
BIOMD0000000165	62	0	3	6	32	WS
BIOMD0000000166	141	0	0	0	48	WS
BIOMD0000000169	868	1	0	0	225	WH
BIOMD0000000170	122	0	2	7	55	W
BIOMD0000000171	229	0	7	18	84	WS
BIOMD0000000173	29	2	1	1	14	FHS
BIOMD0000000228	331	0	4	8	107	WS

Table 4.3: Achieving the minimum backtracking length using minimal FK improvements.

Model	FK calls	Successful HashFetch	Backtrack counts	Backtrack length	Seen nodes	Method
BIOMD0000000034	678	45	0	0	86	FHS
BIOMD0000000042	253	46	8	12	126	FHS
BIOMD0000000048	2288	322	109	292	1436	FHS
BIOMD0000000089	6769	181	0	0	347	FHS
BIOMD0000000093	2748	0	96	226	695	WS
BIOMD0000000094	2748	0	96	226	695	WS
BIOMD0000000106	63	6	0	0	15	FHS
BIOMD0000000107	37	0	0	0	16	WS
BIOMD0000000108	292	28	8	24	169	FHS
BIOMD0000000110	505	35	1	1	82	FH
BIOMD0000000162	79013	650	0	0	2584	FHS
BIOMD0000000163	2337	86	0	0	176	FH
BIOMD0000000165	64	9	1	1	31	FH
BIOMD0000000166	141	0	0	0	48	WS
BIOMD0000000169	868	1	0	0	225	WH
BIOMD0000000170	232	18	2	4	60	FH
BIOMD0000000171	449	68	6	8	223	FHS
BIOMD0000000173	29	2	1	1	14	FHS
BIOMD0000000228	2530	89	2	2	216	FHS

Table 4.4: Achieving the minimum number of seen nodes using minimal FK improvements.

Model	FK calls	Successful HashFetch	Backtrack counts	Backtrack length	Seen nodes	Method
BIOMD0000000034	678	45	0	0	86	FHS
BIOMD0000000042	253	46	8	12	126	FHS
BIOMD0000000048	2220	4	187	563	1169	WHS
BIOMD0000000089	1700	0	2	8	269	WS
BIOMD0000000093	2748	0	96	226	695	WS
BIOMD0000000094	2748	0	96	226	695	WS
BIOMD0000000106	63	6	0	0	15	FHS
BIOMD0000000107	37	0	0	0	16	WS
BIOMD0000000108	292	28	8	24	169	FHS
BIOMD0000000110	453	33	3	7	79	FHS
BIOMD0000000162	8244	0	4	12	853	WS
BIOMD0000000163	2337	86	0	0	176	FH
BIOMD0000000165	64	9	1	1	31	FH
BIOMD0000000166	196	13	0	0	46	FHS
BIOMD0000000169	2256	87	0	0	205	FHS
BIOMD0000000170	122	0	2	7	55	W
BIOMD0000000171	229	0	7	18	84	WS
BIOMD0000000173	29	2	1	1	14	FHS
BIOMD0000000228	563	0	2	4	103	W

Table 4.5: Modifications helped to achieve minimum value for each measurement.

Model	Minimizing FK calls	Minimizing backtracking length	Minimizing seen nodes	Common modification
BIOMD0000000034	WH	FHS	FHS	H
BIOMD0000000042	FHS	FHS	FHS	FHS
BIOMD0000000048	W	FHS	WHS	
BIOMD0000000089	WS	FHS	WS	S
BIOMD0000000093	WS	WS	WS	WS
BIOMD0000000094	WS	WS	WS	WS
BIOMD0000000106	FHS	FHS	FHS	FHS
BIOMD0000000107	WS	WS	WS	WS
BIOMD0000000108	FHS	FHS	FHS	FHS
BIOMD0000000110	W	FH	FHS	
BIOMD0000000162	WS	FHS	WS	S
BIOMD0000000163	WH	FH	FH	H
BIOMD0000000165	WS	FH	FH	
BIOMD0000000166	WS	WS	FHS	S
BIOMD0000000169	WH	WH	FHS	H
BIOMD0000000170	W	FH	W	
BIOMD0000000171	WS	FHS	WS	S
BIOMD0000000173	FHS	FHS	FHS	FHS
BIOMD0000000228	WS	FHS	W	

4.5 Comparison of Processing Times

Table 4.6 presents the processing time for each metabolic model using different FK variants that gives a better understanding on how each improvement can affect processing time. As shown, W variant is the fastest one in 17 models out of 19 models.

Additionally, we tested the F and W variants on four Ecoli metabolic networks including acetate, succinate, glycerol and glucose [3, 18]. Table 4.7 presents their characteristics and the required time to find the MCSs. As shown, W variant works well in the two biggest networks and significantly reduces the processing times in comparison with F variant. It is worthfully to mention that the processing times we obtained here are much higher than the reported processing times in [3, 18]. The reason stems from the difference in the essence of the algorithms and in our project we aimed to speed up the FK-B algorithm because of its application in joint-generation problem and we did not aim to compete with the present algorithms that generate MCSs.

Table 4.6: Comparing processing times. The numbers show the time in seconds and in each row the lowest number is bold.

Model	**F**	**FH**	**FHC**	**FO**	**FOR**	**FS**	**W**
BIOMD-034	0.81	1.69	1.7	0.75	0.78	0.63	**0.51**
BIOMD-042	2.04	1.85	1.94	2	1.96	**1.22**	1.25
BIOMD-048	25.43	12.02	12.15	25.58	25.76	22.18	**7.39**
BIOMD-089	14.66	7.82	7.73	14.64	14.71	10.81	**4.71**
BIOMD-093	46.87	20.46	19.79	47.46	47.01	38.1	**11.28**
BIOMD-094	40.68	20.03	20.16	40.44	40.64	36.57	**10.8**
BIOMD-106	0.08	1.45	1.44	0.08	0.08	**0.05**	**0.05**
BIOMD-107	0.07	1.45	1.45	0.07	0.07	0.05	**0.04**
BIOMD-108	1.71	1.97	2	5.34	2.34	1.08	**0.62**
BIOMD-110	0.47	1.64	1.64	0.48	0.48	0.41	**0.19**
BIOMD-162	649.62	310.64	310.99	650.85	656.5	683.58	**64.21**
BIOMD-163	4.28	3.97	3.86	4.3	4.29	4.49	**1.65**
BIOMD-165	0.06	1.48	1.47	0.06	0.07	0.04	**0.04**
BIOMD-166	0.14	1.53	1.53	0.14	0.14	0.15	**0.06**
BIOMD-169	4.15	3.38	3.38	4.16	4.16	3.04	**1.41**
BIOMD-170	0.2	1.51	1.51	0.2	0.2	0.15	**0.06**
BIOMD-171	1.26	2	1.99	1.18	1.2	0.9	**0.28**
BIOMD-173	**0.03**	1.49	1.49	**0.03**	0.04	0.04	0.05

Table 4.7: Comparing processing times for Ecoli metabolic networks. In each row the lowest processing time is bold.

Model	**# of reactions**	**# of EFMs**	**# of MCSs**	**Processing Time (seconds)**	
				F	W
Acetate	21	363	54	2.47	**1.94**
Glucose	34	21592	857	317905.62 (88.3 hrs)	**225271.3** **(62.58 hrs)**
Glycerol	28	9479	376	19131.28 (5.31 hrs)	**9005.03** **(2.5 hrs)**
Succinate	26	3421	159	**179.43**	220.95

Chapter 5

Discussion

In 1996, Fredman and Khachiyan [4] proposed two novel algorithms, so-called FK-A and FK-B, to identify duality between two monotone Boolean functions whose time complexities are quasi-polynomial time of the input size. Considering the duality test algorithm, if one of the monotone Boolean function is known, its dual function can be produced using the conflicting assignment returning from FK-A or FK-B algorithms in an incremental manner. The computational complexity of such dualization algorithm would still be quasi-polynomial. However, the dualization algorithm is not fast enough in practice when the given monotone Boolean function is of medium or large size.

In this project, we proposed several improvements/techniques to reduce the FK-B running time in practice. These improvements do not affect the theoretical time complexity of the FK-B algorithm, however, they make it possible to use the FK-B to solve medium-to-large-scale dualization problem in practice.

The first improvement is using flags to reduce the number of redundancy tests, pair comparisons, in each recursive call of the FK-B algorithm. The second improvement is about returning multiple conflicting assignment instead of only one. In dualization, this helps to produce more than one monomial in the DNF in each iteration. The third one is about using a hash table and instead of solving repeated subproblems several times, fetch the conflicting assignments. In the fourth improvement, instead of choosing the most frequent variable in both the CNF and the DNF as splitting variable, we choose splitting variable randomly while the chance of a variable being selected is based on its history of acting as splitting variable in previous iterations. In this way, if a variable has been acted as splitting variable in previous iterations and led to a conflicting assignment fast it gets more chance to be selected as a splitting variable in the current point. In the fifth improvement, we focused on a situation where the splitting variable is not μ-frequent in either of CNF or DNF. In this case, instead of first setting the variable to *False* and search for a conflicting assignment(s) and in case of not finding any conflicting assignment(s) setting the variable to *True* and repeat the search again, we consider history of the variable when it has appeared in previous conflicting assignments. It is firstly set to *False* if in majority of previous identified conflicting assignments it is *False*, otherwise it is firstly set to *True* then if any conflicting assignments is found, the other option, i.e. *True* or *False*, is tested. The last improvement is about shrinking the CNF and the DNF in the middle of FK-B recursive calls where one variable appears as a singleton in the CNF/DNF and appears in all monomials/clauses in DNF/CNF. In this case, this variable can safely be removed from both CNF and DNF which results in having a smaller problem to solve.

The proposed modifications have been applied on FK-B algorithm and FK dualization and as an application we have used the modified algorithm in finding minimal cut sets based on given elementary flux modes in metabolic networks. The results show that the proposed improvements can reduce the running time by an order of magnitude on most of the examples.

It is noteworthy to highlight that the proposed modifications/techniques are general and applicable to any problems, e.g. transversal hypergraph generation problem [10], that can be mapped to Monotone boolean function dualization problem.

Part II

Prediction of Drug Resistance in Tuberculosis Using Modular Neural Network

Chapter 1

Introduction

1.1 Mycobacterium Tuberculosis

Tuberculosis (TB) is an airborne infectious disease caused by the *Mycobacterium tuberculosis* (MTB) bacterium. The TB death rate puts TB in the list of top 10 leading causes of death in the world. According to the World Health Organization (WHO) report published in October 2020[1] the number of people who get infected with TB is gradually decreasing every year and about 10 million people became infected with TB in 2019. It is known that people with weak immune systems like those undergoing chemotherapy and those with HIV/AIDS are at a high risk of developing TB disease when they are exposed to MTB. The WHO reports that in 2019, about 1.4 million people died because of TB, of whom about $208,000$ were HIV-positive people.

TB generally attacks the lungs, but can also attack other parts of the body. There are two types of TB infection: active TB and latent TB Infection (LTBI). When someone is sick with active TB he/she feels sick[2] and can spread the disease to other people. Such a person needs to get treated as soon as possible. On the other side, in case of LTBI, although a person is infected by TB and has the TB bacteria in their body, the body's immune system works properly and the person does not feel sick. Importantly, the people with LTBI do not spread the disease. The diagnosis of TB is performed using chest X-rays and the culture of bodily fluids (primarily sputum), as well as the tuberculin skin test.

According to the WHO, about 10% of latent infections develop into active TB which, if it does not get treated, has the potential of killing 1 out of 2 among those affected.

[1] https://apps.who.int/iris/bitstream/handle/10665/336069/9789240013131-eng.pdf

[2] The typical symptoms of active TB are a chronic cough with blood-containing mucus, fever, night sweats, and weight loss.

To treat TB, either active TB or LTBI, it is needed to take (multiple) antibiotics over a long period of time, i.e. six to nine months. The drugs used for treating TB are grouped in two categories: first-line drugs and second-line drugs. The most frequent and effective antibiotics used for TB treatment include Isoniazid (INH), Rifampin (RIF), Ethambutol (EMB), Pyrazinamide (PZA) and Streptomycin (SM), so-called first-line drugs. It is common that treating TB using the first-line drugs fails due to the emergence of drug resistant bacteria. In this case, the first-line drugs should be replaced with the second-line drugs that are expensive and much more toxic than the first-line drugs. The second-line drugs are Fluoroquinolones group including Ofloxacin (OFX), Levofloxacin (LEV), Moxifloxacin (MOX) and Ciprofloxacin (CIP), injectable drugs including Kanamycin (KAN), Amikacin (AMK) and Capreomycin (CAP), and less effective drugs including Ethionamide (ETH)/prothionamide (PTH), Cycloserine (CS)/terizidone and P-aminosalicylic acid (PAS).

Drug resistance in MTB is primarily caused by spontaneous mutations in genes that are drug targets or drug-activating enzymes. These mutations occur in the form of Single Nucleotide Polymorphisms (SNPs), insertions or deletions and rarely large deletions. It is estimated that the rate of spontaneous mutations in prokaryotes is 0.0033 per replication, or about 10^{-9} mutations per base pair (bp) per generation, meaning that roughly 2 mutations a year occur in MTB given its genome's length of 4.5 Mbp and replication time of 20 hours.

However, when the mutation numbers in MTB increase in the population this can result in the emergence of drug resistance. It is known that the rate of mutation depends on the drugs which are taken and typically the patients are prescribed combination of more than one drug to decrease the risk of MTB strains being able to survive, as the risk of a strain containing multiple resistance mutations is a lot lower. The mechanisms of resistance to some drugs like INH and RIF have now been discovered by identifying the genomic regions where the relevant mutations occur [25–28]. Table 1.1 shows this information for several drugs and one can see that for some drugs, mutations in more than one gene are responsible for MTB developing resistance. The information provided in the table is based on the current knowledge of drug resistance mechanism and the table is still being completed by many researchers across the world who study resistance mechanism.

According to WHO, the TB drug resistance types are as follows:

- Mono-resistance that is resistance to only one first line drug,
- Rifampicin resistance (RR) that is resistance to Rifampicin with or without resistance to other drugs.
- Poly-resistance that is resistance to more than one first line drug, other than both Rifampicin and Isoniazid,

Drug	Gene	Role of gene product
Isoniazid	katG	catalase/peroxidase
	inhA	enoyl reductase
	ahpC	alkyl hydroperoxide reductase
Rifampicin	rpoB	b-subunit of RNA polymerase
Pyrazinimide	pncA	PZase
Streptomycin	rpsL	S12 ribosomal protein
	rrs	16S rRNA
	gidB	7-methylguanosine methyltransferase
Ethambutol	embB	arabinosyl transferase
Fluoroquinolones	gyrA/gyrB	DNA gyrase
Amikacin	rrs	16S rRNA
Kanamycin	rrs	16S rRNA
	eis	aminoglycoside acetyltransferase
Capreomycin	tlyA	rRNA methyltransferase
	rrs	16S rRNA
Ethionamide	inhA	enoyl reductase
	ethA	enoyl-APC reductase
	katG	catalase/peroxidase
P-aminosalicylic acid	thyA	thymidylate synthase A

Table 1.1: First- and second-line drugs and mechanisms of drug resistance.

- Multidrug resistance (MDR) that is resistance to at least both Isoniazid and Rifampicin,
- Extensive drug resistance (XDR) that is MDR plus resistance to any fluoroquinolone, and at least one of three second line injectable drugs (Capreomycin, Kanamycin and Amikacin),

As mentioned above, the emergence of drug resistance in TB patients makes the treatment process harder and it would be beneficial to identify drug resistance of a patient to a specific drug in the beginning of the treatment. According to WHO, 3.3% of new TB cases were MDR-TB or RR-TB in 2019.

Generally, drug resistance can be detected using special laboratory tests which test the bacteria for sensitivity to the drugs or detect resistance patterns. These tests can be culture-based which consider the growth of bacteria in presence of drugs, and molecular-based which pay attention to genetic mutations. The main problem with culture-based tests is that they require professional laboratory and trained staff. Additionally, these tests are too lengthy due to the slow growth rate of MTB bacteria that is doubling roughly once per day [29].

Given that many resistance-causing mutations are known, a molecular-based test can be done fast but its sensitivity is low for some drugs and it can be an expensive test. Another problem with this type of tests is the need for specialists to perform the test. Recently, some assays like GeneXpert have been commercialized that are easy to use for DST; however, they have some limitations in detecting resistance to some drugs, and they are also expensive. A new test called SPIT SEQ[3] was introduced in August 2019 in India that tests sputum and detects all the resistant mutations within 10 days. This test is not yet commercially available but a potential problem with this test is its cost that is estimated at $100 for each sample.

Considering the difficulty of these tests due to length of process, accessibility and cost, some efforts have been made to get help from the computer science field to create tools that are faster and cheaper to predict resistance to the drugs. In the following section we introduce some tools and approaches proposed to predict drug resistance.

1.2 Drug Resistance Prediction Using Machine Learning Techniques

TBProfiler [30,31], MyKrobe [32], KvarQ [33], PhyResSE [34] use presence of the resistance-causing mutations in the isolates to predict resistance to the drugs. According to [35] these tools have good performance for INH and RIF, the two most effective first-line drugs, but poor performance for fluoroquinolones and injectable second-line drugs. Similar to these tools, in [36], the authors find that detecting premature stop codons in resistance-associated genes (katG, ethA, pncA, and gidB) can improve performance of resistance prediction based on Whole Genome Sequencing (WGS) data.

In [37] the authors have applied several ML algorithms including Support Vector Machine (SVM), regularized Logistic regression (LR) and product-of-marginals (PM) as well as three ensemble learning methods including Random Forest (RF), Adaboost, and Gradient Boosting Trees (GBT) to predict resistance to 11 drugs based on WGS data. To this end, they have used three different sets of features: all variants found within the 23 candidate genes, predetermined resistance-associated variants as listed in [38], and a subset of the first set including only resistance-associated genes for the particular drug. They have also used linear dimension reduction techniques like SPCS/SNMF to handle sparsity in the data and enhance the performance. Yang *et al.* [39] have also done such research. They have applied several ML algorithms like LR and SVM with two different kernels, RF, PM and a class-conditional Bernoulli mixture model (CBMM) on SNP data from selected 23 candidate genesIn another research paper [40], the authors have created and trained a GBT model on SNP data to predict drug resistance.

[3] `https://www.biospectrumindia.com/news/78/14407/medgenome-unveils-novel-dna-test-for-all-dr-mutations-in-tb.html`

Chen *et al.* [41] have used wide&deep multi-task neural network (WDNN) to predict resistance of isolates to 10 drugs using WGS data. They have compared the obtained performance with a single-drug WDNN, a single-drug WDNN trained on known resistant mutations for each drug, RF and regularized LR models.

Yang *et al.* [42] proposed two deep denoising auto-encoders augmented by multi-task classifier and a clustering layer. The former is called DeepAMR and the latter is called DeepAMR-cluster. DeepAMR is used to predict drug resistance while DeepAMR-cluster is used to cluster the lineages. These models have been applied on SNP data to predict resistance to 10 drugs. Hyperparameter tuning in DeepAMR has been done by minimizing the Hamming loss that is fraction of wrong labels to the total number of labels. In addition to the deep models, the authors have used Multi-label K-nearest Neighbor (MLKNN) and Ensemble Classifier Chain (ECC) with LR as base classifiers on the data. The authors compare the results of DeepAMR with MLKNN, ECC, RF, SVM and Direct Association (DA) methods. In DA method, an isolate is labeled as resistant to a drug if any of known resistance mutations from [38] is present in the isolate. Additionally, the authors rank the SNPs by permutation feature importance model. In this approach the order of a feature column is permuted and sensitivity decrease is computed.

DeepARG [43] is another tool using deep learning to predict drug resistance. To this end, the authors first align short read sequences with known Antibiotic Resistant Genes (ARGs) and compute a bit score that measures similarity between two aligned sequences. Bit scores are fed into a fully connected NN to predict resistance of the input sequence to the drugs.

Karmakar *et al.* [44] have pursued a different research direction from the previously mentioned ones, and have proposed to use the structural information of pncA gene to predict drug resistance to Pyrazinamide. In this regard, they consider the changes to the 3D structure of pncA when a variant occurs in this gene. The features they have used are related to stability, dynamics, and evolutionary conservation of the gene. In this study, RF has been used as prediction model.

Zhang *et al.* [45] use a Deep Convolutional Neural Network (DCNN) with four two-dimesional convolutional layers to predict Pyrazinamide resistance in TB strains. To this end, they reshape one-dimensional SNP data to two-dimensional shape to be able to apply two-dimensional convolution operator. They also use SVM model along with a recursive feature selection procedure to find resistance genes and mutations.

Chapter 2

Data Processing and Feature Engineering

2.1 SNP Detection in Raw Data

We have used the European Nucleotide Archive (ENA) to download the DNA short reads. The raw sequence data are mapped to the H37Rv reference genome[1] using `bwa-mem` software, and then two SNP callers GATK [46] and SAMTools [47] are run to detect the SNPs, insertions and deletions (indels)[2]. At the end, the intersection of these two SNP callers is extracted that results in $742,620$ SNPs. Using the SNPs we create a binary matrix whose rows and columns correspond to isolates and SNPs, respectively. Entry $[i, j]$ in this matrix is '1' when SNP j is present in isolate i, otherwise it is '0'. Figure 2.1 shows histogram of SNP occurrence across isolates. As shown there are $\sim 100,000$ SNPs that have detected in only one isolate. As we get away from one, i.e. the number of SNP occurrence increases, the number of such SNPs decreases. This histogram show that we have a very sparse matrix.

2.2 Gene/Protein Sequences

According to the TB database there are $\sim$ 4000 known genes in TB bacteria that can be downloaded from EPFL database[3] or NCBI. For each gene we have its start and end locations corresponding the M. tuberculosis H37Rv reference genome as well as its orientation that is either of 'minus' or 'plus'. To reconstruct the sequence of each gene in each isolate we follow the below steps:

1. Finding the start and end locations of the gene and its orientation,

[1] `https://www.ncbi.nlm.nih.gov/assembly/GCF_000195955.2/`

[2] Note that from here for the sake of simplicity we refer to all of them as SNPs.

[3] `https://mycobrowser.epfl.ch/`

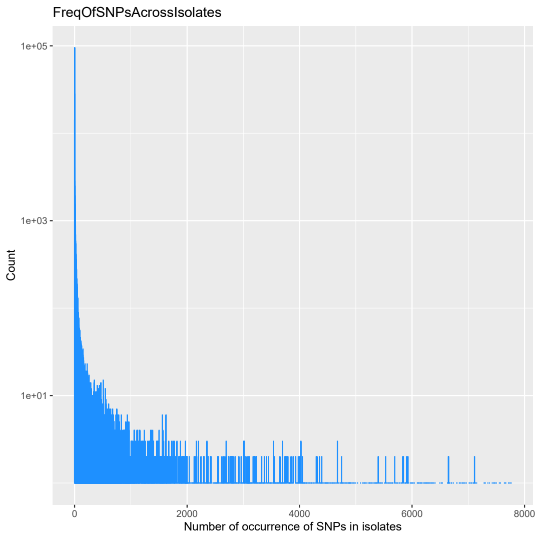


Figure 2.1: Histogram of SNP presence across isolates.

2. Mapping the locations on reference genome and extracting the sequence between start and end locations,

3. If the orientation is 'plus' there is no need to change the sequence, if the orientation is 'minus', the sequence gets reversed, then complemented by replacing 'A' with 'T' and replacing 'C' with 'G' and vice versa.

4. Finding the SNPs occurred in the gene and the corresponding isolate,

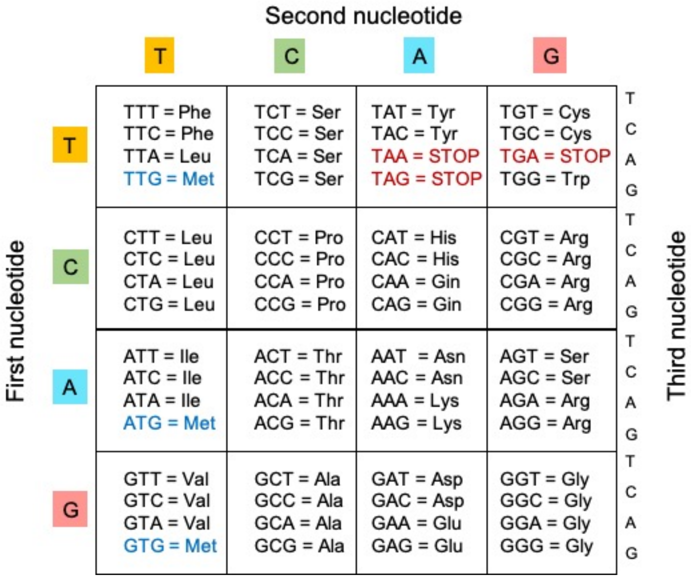
5. Sorting the SNPs based on their position in order of end to beginning of the sequence,

6. Modifying the sequence based on each SNP.

Note that when two indels are overlapping and conflicting, we choose the one that is shorter and affects a smaller part of the sequence. At the end, we have 4024 gene sequences per sample with an average sequence length across all isolates of 995.29 bases.

Reconstructing gene sequences is followed by reconstructing protein sequences. According to [48] the start codons in TB are 'GTG', 'ATG' and 'TTG', and the stop codons are 'TAG', 'TAA' and 'TGA'. To translate a gene sequence to its protein sequence we start from the beginning of the sequence looking for a start codon and then we use the table shown in Figure 2.2 to translate the codons to corresponding amino acids. Translation ends when a stop codon is met. We use this procedure to translate the gene sequences we already have and end up having 3990 proteins whose average length across isolates is 201.6[4].

[4] Note that in some genes, there is a stop codon in the middle of the reference gene sequence that is confusing as stop codons are expected to be at the end of gene sequences. We have skipped those genes and that is why the number of proteins is less than the number of genes.



Second nucleotide
First nucleotide
Third nucleotide

First nucleotide	T	C	A	G	Third nucleotide
T	TTT = Phe	TCT = Ser	TAT = Tyr	TGT = Cys	T
	TTC = Phe	TCC = Ser	TAC = Tyr	TGC = Cys	C
	TTA = Leu	TCA = Ser	TAA = STOP	TGA = STOP	A
	TTG = Met	TCG = Ser	TAG = STOP	TGG = Trp	G
C	CTT = Leu	CCT = Pro	CAT = His	CGT = Arg	T
	CTC = Leu	CCC = Pro	CAC = His	CGC = Arg	C
	CTA = Leu	CCA = Pro	CAA = Gin	CGA = Arg	A
	CTG = Leu	CCG = Pro	CAG = Gin	CGG = Arg	G
A	ATT = Ile	ACT = Thr	AAT = Asn	AGT = Ser	T
	ATC = Ile	ACC = Thr	AAC = Asn	AGC = Ser	C
	ATA = Ile	ACA = Thr	AAA = Lys	AGA = Arg	A
	ATG = Met	ACG = Thr	AAG = Lys	AGG = Arg	G
G	GTT = Val	GCT = Ala	GAT = Asp	GGT = Gly	T
	GTC = Val	GCC = Ala	GAC = Asp	GGC = Gly	C
	GTA = Val	GCA = Ala	GAA = Glu	GGA = Gly	A
	GTG = Met	GCG = Ala	GAG = Glu	GGG = Gly	G

Figure 2.2: Genetic code chart: It shows translating each nucleotide triplet in DNA into an amino acid or a termination signal in a protein.

Now, by having the gene/protein sequences the following preprocessing steps are needed to prepare the sequence data to be fed into the network:

- To enable tracking each gene/protein whose lengths are variable, the sequences are chunked into subsequences with length 200 for genes and 50 for proteins. Then, the last chunk in each gene and protein is filled with a dummy symbol to make the length of the last chunk equal to the other chunks.

- All obtained chunks are concatenated in order and a very long sequence for whole genes/proteins is created.

After the preprocessing steps, the final length of gene and protein sequences are $4,402,400$ and $822,100$, respectively. Given that gene sequences are too lengthy and needs huge amount of computational resources, we only use protein sequences in our study.

2.3 Premature Stop Codon

It happens that a SNP changes a codon from a regular one to a *stop* codon, e.g. changing 'TAC' to 'TAA'. When it occurs before reaching the end of gene sequence it is called *premature* stop codon. In [36], the authors show that considering the premature stop codons and ignoring the occurring SNPs after these stop codons can improve prediction of drug resistance in TB. Thus, we follow their suggestion and identify premature stop codons and also ignore the SNPs that occur after the stop codon.

2.4 Mutation Type

Mutation type is another source of information that might be helpful in improving performance of prediction models. In this regard, we consider four types of mutations: missense, nonsense, silent and frameshift [49].

Missense mutation is a mutation whose effect in changing a nucleotide in a codon results in changing the protein, e.g. changing 'ACT' to 'ATT' which results in changing 'Threonine' protein to 'Isoleucine' protein. Nonsense mutation changes a codon to a stop codon which can cause a premature stop codon, e.g. changing 'TAT' to 'TAG'. Silent mutation does not affect the translated protein, e.g. changing 'CTT' to 'CTC' which both are translated to 'Leucine' protein. Finally, frameshift corresponds to indels which can shift the following codons. If the length of an indel is not divisible by three, it affects the reading frame of the current and following codons. Thus, frameshifts can shorten or enlarge a protein sequence and convert the protein sequence into an abnormal protein product.

Note that if we have two indels whose sum of shift lengths is divisible by three, all mutations between these two indels are considered a frameshift, and after the second indel the following codons are intact and their types are determined based on their effect on the translated proteins.

2.5 TB Core Genome

The number of genes in TB is 4024; many of them are unrelated to the drug resistance. Thus, we only consider TB core genes that are 2891 genes to reduce size of the data. The list of TB core genes can be downloaded from `https://www.cgmlst.org/ncs/schema/741110/locus/` [50].

2.6 Pathways in Tuberculosis

A pathway consists of genes and their interactions which perform a specific task in a cell. According to the Kyoto Encyclopedia of Genes and Genomes (KEGG) database[5] there are 114 pathways in MTB whose genes and other related information can be downloaded using the *R* package `KEGGREST` [51]. Such information is used to create a binary matrix whose rows and columns correspond to genes and pathways, respectively, and entry $[i, j]$ is 1 if gene i belongs to pathway j. Such matrix/mask is used when there is protein sequence input data to the neural network.

[5] `https://www.genome.jp/kegg/`

2.7 Feature Selection Using Chi-squared Test

It is common to do feature selection in case of having a large number of features when the number of samples is limited. One of the simplest and most useful methods to perform feature selection is the Chi-squared test that tests the independence of two variables. In this regard, the inputs of the test are a feature/variable and a label. This test is applied on every feature and returns a score. When a variable and the corresponding label are independent the score is low. Thus, to do feature selection, variables whose Chi-squared scores are higher than others are the ones that are informative. Note that to prevent data leakage from the training part to the testing part of the data, feature selection is only applied on the training data, not the validation and testing data.

Chapter 3

Deep Neural Network Designed to Predict Drug Resistance

3.1 Introduction to Deep Neural Networks

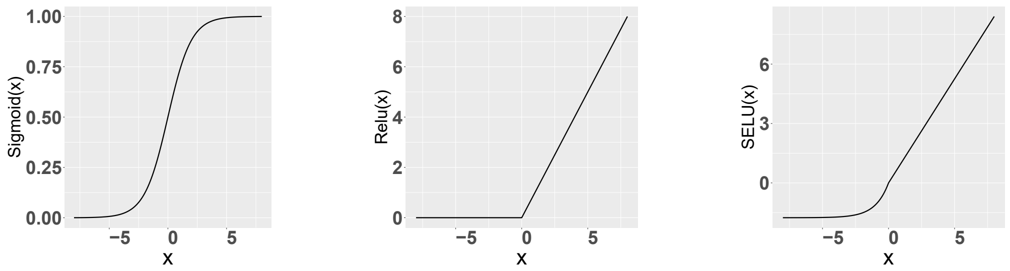
A neural network is composed of neurons, layers and connections between neurons which take input variable(s) and produce output(s) based on a chain of equations. The layers in a basic neural network are an input layer, hidden layer(s) and an output layer. In each layer there is one or more neurons which are connected to the neurons in previous and next layer and each connection has a specific weight. Each neuron has two functions including a linear function that computes the weighted summation of inputs to the neuron, $z = X_0W_0 + X_1W_1 + \ldots$, and an activation function that is a linear or non-linear function that is applied to z and produces an output value, $f(z)$.

Training a neural network is an iterative procedure including forward and backward passes. In forward pass, the input data feeds into the network and goes through the network layer by layer, then an output is generated by the output layer. Afterwards, the error, so-called loss, is computed based on the difference between the network output(s) and the true output(s). At this point, if a predetermined stop condition is met the training procedure stops, otherwise the forward pass is followed by a backward pass. In the backward pass, so-called back propagation, gradient of loss with respect to the weights of connections is computed and the weights are updated accordingly and then the forward pass starts again.

In the following sections, we briefly talk about different parts of the proposed neural network model to predict drug resistance in TB and details of the training procedure.

3.1.1 Activation Functions

In a neural network, for each neuron an activation function is defined that maps the input of the node to the output. Activation functions can be linear or non-linear. Linear activation



(a) Sigmoid.

(b) ReLU.

(c) SELU.

Figure 3.1: Activation functions.

functions are not popular since they cannot capture the complexity of the input data to the NN. Popular non-linear activation functions includes Sigmoid (Logistic) and ReLU (Rectified Linear Unit) activation functions which are defined as:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

$$ReLU(x) = \max(0, x). \tag{3.2}$$

Figure 3.1(a) shows the Sigmoid activation function. As shown, Sigmoid activation function maps the input values to a value between 0 and 1. In case of having binary classification problem, Sigmoid is the activation function that is used for the neuron(s) in the last layer and its output values are considered as probability of belonging to the positive class.

ReLU activation function is mostly used for the neuron(s) in the hidden layer(s). According to its formula if the input value is greater than 0 its output is equal to the input value, otherwise it deactivates the neuron(s) by mapping the input value to 0. In this case, during the backpropagation pass, the weights and biases of such neuron(s) are not updated which may result in making neuron(s) dead, i.e. they never become activated (See Figure 3.1(b)). In this regard, some variants of ReLU like SELU (Scaled Exponential Linear Unit) [52] has been proposed that do not deactivate the neuron(s). The SELU activation function is defined as:

$$SELU(x) = \begin{cases} \alpha x & x > 0 \\ \alpha\lambda(e^x - 1) & x \leq 0 \end{cases} \tag{3.3}$$

where $\alpha = 1.05070098$ and $\lambda = 1.67326324$ are constants. Noteworthy to mention that these values have been computed by solving some equations to preserve the mean and variance of the input values between consecutive layers. Figure 3.1(c) shows the SELU activation function.

3.1.2 Weight Initialization: He Method

The initial values of parameters of a layer including weights and biases can affect performance of a neural network since outputs of one layer are the inputs to the activation functions of the layer, and the outputs of the activation functions go to the subsequent layer and pass forward through the whole network. Considering this, weights and biases can potentially cause or prevent exploding gradients, i.e. too large loss gradients during backward pass, or vanishing gradients, i.e. too small loss gradients during backward pass. In either case, the network might not be able to converge or it can take a long time to converge. Thus, selecting proper initial values for the wights and biases is an important task. One of the popular initialization methods is random initialization where random numbers based on a predetermined distribution, say Gaussian, are assigned to the weights and biases.

In 2015, He *et al.* [53] proposed an activation aware initialization approach which assumes *ReLU* or *Leaky ReLU* are used as activation functions in the hidden layers. In this case, the weights of layer l are:

$$W^{[l]} = W_r^{[l]}\sqrt{\frac{2}{N_{l-1}}} \tag{3.4}$$

where $W^{[l]}$ is weight matrix of layer l, $W_r^{[l]}$ is randomly initialized weight matrix of layer l, and N_{l-1} is the number of neurons in layer $l-1$. In the He approach, the initial value of biases is set to zero. In their paper they have shown that their initialization approach can help convergence even in very deep networks.

3.1.3 Optimization

Optimization plays a key role in deep learning when we try to minimize the loss function, i.e. empirical risk, with respect to some parameters on a training set. Gradient descent (GD) is an iterative algorithm that is used for neural network optimization. The main idea behind GD approach for a minimization problem is taking steps toward steepest descent direction that is in the opposite direction of the gradient of the loss function.

The basic formulation of GD is as follows:

$$w_{t+1} = w_t - \eta_t \nabla J(w_t), \tag{3.5}$$

where η is the step-size, so-called learning rate, w is the parameter and $\nabla J(w_t)$ is gradient of the loss function for the t^{th} iteration. $\nabla J(w_t)$ can be written as:

$$\nabla J(w_t) = \frac{1}{n} \sum_{i=1}^{n} \nabla J_i(w_t), \tag{3.6}$$

where $J_i(w_t)$ is the loss for sample i in iteration t and n is the number of training samples.

Stochastic Gradient Decent (SGD) is one of the variants of GD where the weights (parameters) are updated after each training sample. Thus, when one iteration, a pass on all training samples, is done, the weights have been updated n times.

AdaGrad [54] is another variants of GD where instead of using a fixed learning rate to update the parameters w, it uses a different learning rate for every parameter at each iteration based on past gradients of the parameter:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}} + \epsilon} \cdot \nabla J(w_{t,i}), \tag{3.7}$$

where ϵ is smoothing constant, say $1e - 8$, to prevent division by zero and G is gradient matrix:

$$G = \sum_{\tau=1}^{t} g_\tau g_\tau^T, \tag{3.8}$$

where $g_\tau = \nabla J_i(w)$ that is the gradient vector at iteration τ and the diagonal element of matrix G is computed as:

$$G_{ii} = \sum_{\tau=1}^{t} g_{\tau,i}^2. \tag{3.9}$$

$\sqrt{G_{t,ii}}$ can be considered as ℓ_2-norm of previous gradients such that parameters with small updates get larger scaling factor of the learning rate and parameters with large updates get smaller scaling factor of the learning rate.

The weakness of AdaGrad is diminishing the learning rate as accumulation of gradients over time gets very large. In this regard, Hinton [55] has proposed Root Mean Square Propagation (RMSprop) method to resolve the issue. In RMSprop, the weights are updates as:

$$v(w,t) = \gamma v(w,t-1) + (1-\gamma)(\nabla J(w))^2, \tag{3.10}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v(w,t)}} \cdot \nabla J(w) \tag{3.11}$$

where γ is called forgetting factor, suggested to be set to 0.9 and $v(w,t)$ computes the moving average. According to the formula, the learning rate is divided by a factor that normalizes the gradient for weight w considering magnitude of recent gradients for that weight. In this manner, the learning rate is decreased when the gradient is large and increased when the gradient is small; this behaviour avoids exploding and vanishing gradients in the network, respectively.

It has been observed that AdaGrad works well on problems with sparse gradients and RMSprop works well on non-stationary problems which might be noisy. Adaptive Moment Estimation (Adam) [56] method combines benefits of both AdaGrad and RMSprop methods and similar to them adapts the learning rate for each parameter. Adam considers averages of gradients as well as the second moments of gradients when it updates the weights:

$$m_{t+1,w} = \beta_1 m_{t,w} + (1-\beta_1)\nabla J(w_t) \tag{3.12}$$

$$v_{t+1,w} = \beta_2 v_{t,w} + (1-\beta_2)(\nabla J(w_t))^2 \tag{3.13}$$

$$\hat{m}_w = \frac{m_{t+1,w}}{1-\beta_{t+1,1}} \tag{3.14}$$

$$\hat{v}_w = \frac{v_{t+1,w}}{1-\beta_{t+1,2}} \tag{3.15}$$

$$w_{t+1} = w_t - \eta\frac{\hat{m}_w}{\sqrt{\hat{v}_w}+\epsilon} \tag{3.16}$$

where ϵ is smoothing constant, say $1e-8$, to prevent division by zero, and and β_1 and β_2 are forgetting factors for gradients and second moments of gradients, suggested to be set at $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

3.1.4 Preventing Overfitting

Overfitting is one of the main challenges in the field of deep neural networks and it occurs when the training data is small while the network, the number of trainable parameters, is big. If overfitting occurs, the network works well on training data, however its performance on test (unseen) data is poor and it is said that the network cannot generalize. Different techniques have been proposed to resolve the issue including regularization, dropout and early stopping techniques.

Regularization

In regularization technique, a term is added to the loss function aiming to make the weights smaller with expecting to obtain a simpler network. In this regard, the total loss of the network for ℓ_2-regularization is defined as:

$$\text{Total loss} = \text{Loss} + \frac{\lambda}{2n}\sum ||w||_2 \tag{3.17}$$

where λ is the regularization parameter, i.e. penalty, and n is the number of samples. The second term computes the ℓ_2 norm of all weights exist in the neural network that helps to have smaller weights. For ℓ_1-regularization the second term in the total loss is ℓ_1 norm that is written as:

$$\text{Total loss} = \text{Loss} + \frac{\lambda}{2n} \sum ||w||_1. \tag{3.18}$$

Dropout

In dropout technique, which is considered a regularization method, some neurons are randomly ignored at each epoch during the training process, both forward and backward passes. In the other words, at each epoch each neuron and its connections are ignored with a given probability that makes the training process noisy and prevent the network from overfitting and learning noise in the training data [57].

Early stopping

One of the challenges in training a neural network is when to stop training process. If training stops early, the model has high bias (underfits), meaning it has not yet learned the training data. On the other hand, if training continues for a long time, the model has a high variance (overfits). In the early stopping approach, the validation loss is monitored [1]. If the validation loss does not improve for a predetermined epoch number the training process is stopped and the network weights from the epoch with minimum value of validation loss is restored [58].

3.1.5 Batch Normalization

In real data, sometimes the range of features is different from each other, which may cause biases in the learning process. To prevent this, normalization technique is applied on features to map them to a common scale while the distribution in the original data is maintained.

As mentioned above, in a deep neural network with several hidden layers the weights are updated in a backward pass, layer-by-layer, from the last layer to the input layer. Updating weights of a layer is performed with the assumption that outputs of the prior layers follow a specific distribution. However, the distribution of outputs changes after each weight update, which may make the network unstable and requiring a long training process. In this regard, Ioffe and Szegedy in [59] proposed batch normalization technique that standardizes outputs of layers that come from activation functions. Using batch normalization in a layer spreads through the subsequent layers and it is expected that the distribution of inputs

[1] Note that the validation data is never used to train the model and we just measure network performance on the validation data after each epoch to decide about stopping or continuing the training process.

does not change dramatically during backward updates, resulting in faster training process and network stability.

3.1.6 Learning Rate Scheduler

Learning rate is a key hyperparameter, usually in the range between 0 and 1, which scales the magnitude of the neural network weight updates aiming to minimize the loss function, and thus can affect the speed of training process. Setting the learning rate to a very small value causes slowing down of training progress or even getting stuck in a suboptimal solution. Setting the learning rate to a very large value can cause quick convergence to a suboptimal solution or seeing drastic variation in the loss function such that it may jump over the optimal point.

Considering these points, changing the learning rate over time is the simplest method to prevent oscillation and getting stuck in a local minimum. In this approach, the learning rate is decreased by multiplying with a momentum, say 0.99, after each epoch. This method causes large weight updates early and small updates later during the training process [60].

3.1.7 Loss Function

One of the important components of neural networks is the loss function that measures the performance, the prediction error, of the network by comparing the network output with the true output. Considering that the loss value is used for computing gradients and subsequently updating weights in the network, choosing the appropriate loss function can impact the performance of the final model. Different loss functions have been introduced, of which the appropriate ones for classification problems are Binary Cross-Entropy (BCE) and Categorical Cross-Entropy. The first one is applicable for binary classification problems while the latter is appropriate for multi-class classification problems. Given that drug resistance prediction is a binary classification problem we focus only on BCE loss function.

Binary Cross-Entropy (BCE), also called Sigmoid cross-entropy, loss is used when we have a binary classification problem and the activation function of the last layer is Sigmoid. BCE loss is written as:

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)), \tag{3.19}$$

where n is the number of samples, y_i is the label of the i-th sample and $p(y_i)$ is the predicted probability of sample i belonging to the positive class.

In case of an imbalanced data, i.e. the number of positive samples is significantly smaller than negative samples, some researchers suggest to use a weighted BCE (WBCE) as follows:

$$WBCE = -\frac{1}{n}\sum_{i=1}^{n} Wy_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)) \tag{3.20}$$

where W is the weight assigned to the positive class. In WBCE, if $W > 0$ it penalizes the wrong answers for the positive class which is considered the minority in an imbalanced data set. If $W = 1$ then we have $WBCE = BCE$.

Generalizing BCE/WBCE for multi-label classification problem is performed via computing summation of losses over all labels. Note that in case of multi-label classification the parameter W in the WBCE is computed for each label separately. Additionally, considering that in our problem, in which there are some unknown resistance labels for some drugs for some isolates, during the computation of loss values for each drug we just consider the samples with known labels. Considering these points the multi-label WBCE (ML-WBCE) is as follows:

$$ML\text{-}WBCE = -\frac{1}{n}\sum_{d=1}^{D} \sum_{\substack{i=1 \\ y_i^d \text{ is known}}}^{n} W^d y_i^d \log(p(y_i^d)) + (1 - y_i^d)\log(1 - p(y_i^d)) \tag{3.21}$$

where n is the number of samples, D is the number of drugs, y_i^d is label of sample i for drug d and $p(y_i^d)$ is predicted probability of belonging sample i to the positive class for drug d, W^d is the weight assigned to the positive class for drug d.

3.2 Choosing Metric(s) for Performance Comparison

In recent years some papers [61–63] have discussed importance of performance metric selection in applied papers. According to these papers, in case of having imbalanced data, using accuracy and Receiver Operating Characteristic (ROC) curve measurements can be misleading such that with a low *true positive* rate we get nice results for ROC and accuracy. To this end, using Precision-Recall (PR) curve measurements is recommended as in computing precision ($\frac{TP}{TP+FP}$) and recall ($\frac{TP}{TP+FN}$) values the true negative value, that is from major class, is never used and they just measure correct predictions for the positive/minor class.

3.3 Multi-headed Multi-task Neural Network to Predict Drug Resistance

To improve the performance of prediction model we propose to integrate multiple data types, i.e. gene/protein sequence and SNP absence/presence data, extracted from TB isolates. Each data has its own shape and type thus we need to choose appropriate architecture (head)

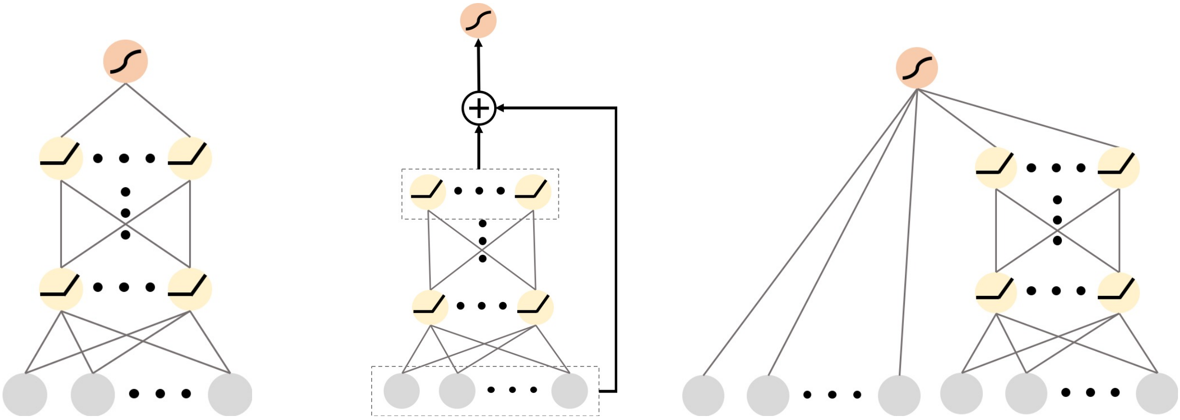

(a) Fully connected block.
(b) Residual block.
(c) Wide-n-deep block.

Figure 3.2: Three general heads.

to feed the data into the network. For example, for SNP absence/presence data we can use a simple fully-connected network, however for gene/protein sequence data that are too lengthy we need to consider using convolution layers in the proposed head.

In this thesis, we use three general heads including residual block, wide-n-deep (WDNN) block and fully-connected (FC) block applicable for numerical data. Figure 3.2 shows these three blocks. In each block the number of layers and nodes are adjusted during hyperparameter optimization. Note that for categorical data, e.g. SNP type data, we add an embedding layer as the first layer of the block.

For sequence data, Long-Short term Memory (LSTM) or Recursive Neural Network (RNN) are the usual architectures in the field of deep neural network. However in case of our problem, having about 4000 genes/proteins with average length of $\sim$ 900 for genes and $\sim$ 200 for proteins, using these elements in the architecture is not practical. In this case, we propose the architecture shown in Figure 3.3 that uses Convolutional Neural Network (CNN) layers to get a separate feature map for each gene/protein. This sequence processing block can be connected to a fully connected block whose connections can be masked using background knowledge like gene-pathway data.

3.4 Fine Tuning of Hyperparameters

In machine learning algorithms changing some specific parameters can usually positively/negatively affect the results and thus, they should be predetermined before the algorithms start. In the field of deep neural networks such parameters are number of layers, number of nodes in a layer, learning rate, etc. Considering the number of parameters and also their types, i.e. whether they are discrete or continuous, we face a search space that must be explored to find the best setting for the ML algorithm.

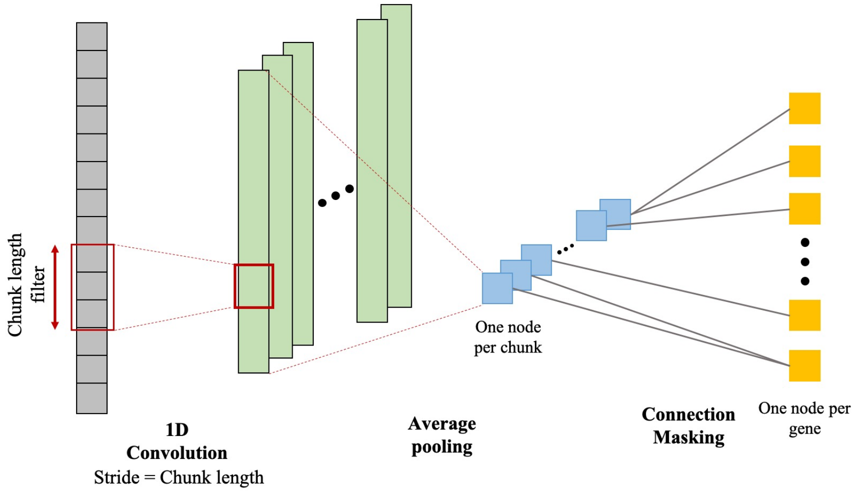


Figure 3.3: Sequence processing block.

The naive and basic approach of finding the best setting is doing grid search and giving every combination of all the hyperparameter values a try. However, grid search is not a good solution when it comes to deep neural networks as there are several continuous and discrete hyperparameters, e.g. learning rate and number of layers. Additionally, giving every possible setting a try makes grid search a very time-consuming approach; of course, the procedure can be run in parallel, but this increases the requirement of computational resources.

The second approach of exploring the search space is performing random search. In this approach, a sampling distribution is provided for each hyperparameter and values are randomly sampled. The philosophy behind random search is that for most of the data sets, not all hyperparameters are equally important, and random search does not spend much time on an unimportant parameter [64]. This approach can also be run in parallel similar to the grid search approach and is faster than grid search approach; however, there is no guarantee that it finds the best setting.

The third approach for fine tuning hyperparameters is Bayesian optimization approach, which is a sequential model-based optimization. In this approach, unlike grid search or random search where each run is independent from the other runs for selecting the next candidate setting to test, the information obtained from previous runs is utilized to improve the sampling method of the next experiment. In brief, a model is initialized with hyperparameter vector λ which, after training, is scored S according to a predetermined metric. Then, the previously evaluated hyperparameter values are used to compute a posterior expectation of the hyperparameter space. Afterwards, the optimal hyperparameter values are chosen according to the posterior expectation as the next set of hyperparameters. This procedure is repeated iteratively until converging to an optimum or reaching the predetermined maximum number of iterations. Considering the advantage of Bayesian optimization over grid and random search approaches we use Bayesian optimization approach to adjust hyperparameters in our project.

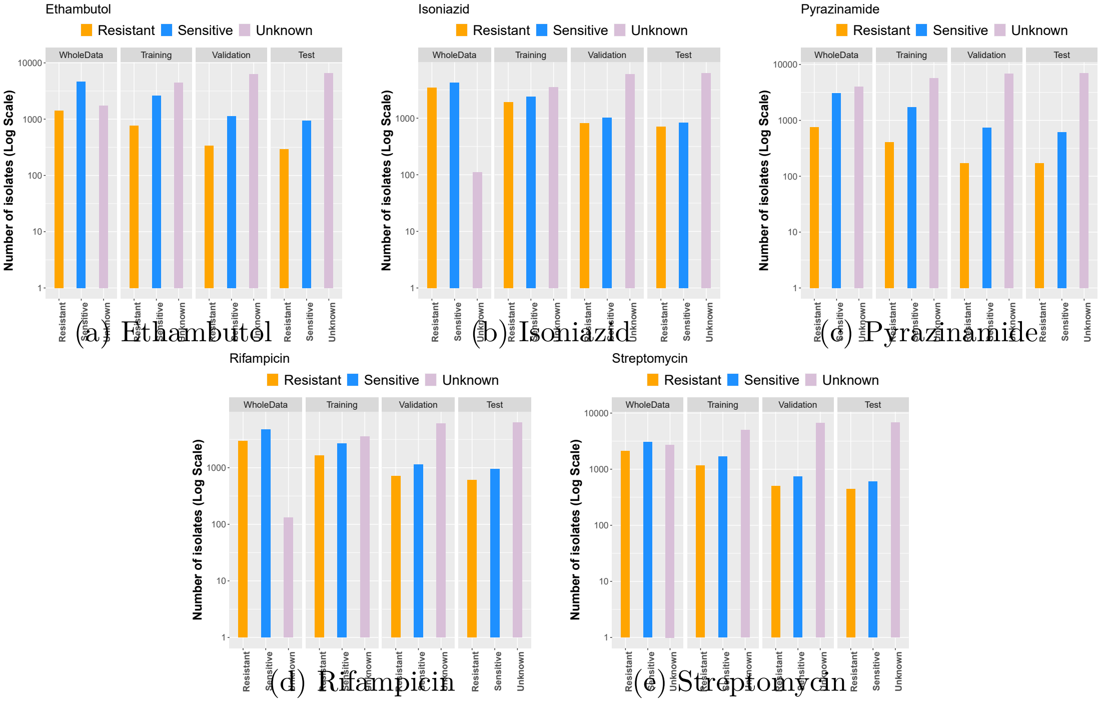
Chapter 4

Experimental Results

4.1 Settings of Experiments

There are 7845 isolates whose labels are partially known for twelve drugs including 'Ethambutol', 'Isoniazid', 'Pyrazinamide', 'Rifampicin' and 'Streptomycin' as first-line drugs and 'Amikacin', 'Capreomycin', 'Ciprofloxacin', 'Ethionamide', 'Kanamycin', 'Moxifloxacin' and 'Ofloxacin' as second-line drugs. In this project, we mainly focus on first-line drugs, however, the results on all drugs are available in the supplementary section.

To use the data for creating prediction models we divide the data to three parts: 50% as training, 30% as validation and 20% as test data. To have similar distribution of the labels in these three parts we use the stratified approach in *scikit-learn* package [65] considering the first-line drugs.



(a) Ethambutol (b) Isoniazid (c) Pyrazinamide

(d) Rifampicin (e) Streptomycin

Figure 4.1: Distribution of labels for each drug in training, validation and test splits.

Figure 4.1 shows the number of isolates from each label in each drug for whole data, training, validation and test data. As shown in the part of *WholeData* the number of sensitive isolates are more than the resistant ones and we have many isolates with partial labels, i.e. for a specific isolate the labels for all 12 drugs are not known. Also, as seen the stratified approach have split the data into training, validation and test data with similar label distributions which is helping to conduct a fair performance evaluation. In this regard, note that we use the same data division in all experiments.

In this study to reduce the size of data, we focus on only TB core genome, thus for SNP related data we only keep SNPs occurred on this set of genes and also, for sequence data we only keep sequences of TB core genes. In addition, in SNP related data features whose number of non-zero values is less than 6 are removed from the data as they are considered as rare SNPs.

In the next step, to reduce the number of input features, in all data types except for sequence data, we use Chi-square feature selection test on training data and chosen features are fed into the NNs. In this test, the features whose scores are higher than the average of scores across all features and have a p-value less than 0.05 are selected as final features. Note that Chi-square test is applied on each feature and each drug, that means we get a separate set of selected features for each drug. At this point, the final set of selected features is produced considering union of the selected features for each drug. Table 4.1 shows the number of selected variables in each drug and the size of final set of selected variables.

	Ethambutol	Isoniazid	Pyrazinamide	Rifampicin	Streptomycin	Total number of selected features
SNP Absence/Presence Data	2164	2725	1928	2618	2499	3758
SNP Type Data	2063	2613	1802	2484	2388	3590
Premature Flag Data	226	314	193	295	251	465
Translation Fraction Data	103	109	81	132	99	143

Table 4.1: Number of selected variables for each drug using Chi-square method. The last column shows the final number of selected variables that is obtained by computing the union of selected variables for all drugs.

Data types	Variable Name	Variable type, Range, Distribution
SNP Absence/Presence Data, SNP Type Data, Translation Fraction Data, Premature Stop Codon Flag Data	Number of nodes	Integer, (5, 50), uniform
	Number of layers	Integer, (2, 20), uniform
	Dropout rate	Real, (1e-1, 5e-1), log-uniform
	$\ell 2$ Regularization penalty	Real, (1e-2, 5e-1), log-uniform
	Type of NN head	'WDNN', 'ResidualNN', 'FC'
Gene Sequence Data, Protein Sequence Data	Number of nodes	Integer, (5, 50), uniform
	Number of filters	Integer, (5, 100), uniform
	Dropout rate	Real, (1e-1, 5e-1), log-uniform
	$\ell 2$ Regularization penalty	Real, (1e-2, 5e-1), log-uniform
Shared Layers	Number of nodes	Integer, (5, 100), uniform
	Number of layers	Integer, (1, 20), uniform
	Dropout rate	Real, (1e-1, 5e-1), log-uniform
	$\ell 2$ Regularization penalty	Real, (1e-2, 5e-1), log-uniform
Global Parameters	Learning rate	Real, (1e-6, 1e-1), log-uniform
	Activation function of hidden layers	'ReLU', 'SELU'

Table 4.2: Hyperparameter list. This table shows the name, type and range of each hyperparameter defined for each input data type as well as shared part of the NN and global hyperparameters. Note that in case of using two input data types, given that one head is used for each input data, two sets of the hyperparameters are considered for optimization.

To build the NN, we use He method to initialize the weights and Adam as optimizer in the model. ML-BCE is used for measuring loss. A learning rate scheduler is used such that it reduces the learning rate by factor 0.98 in each epoch after epoch 10.

Bayesian optimization approach is used to fine-tune the network based on several hyperparameters listed in the Table 4.2. This table shows the name, type and initial range of each hyperparameter that is used for each data type. Noteworthy to mention that the range of each hyperparameter has been chosen empirically.

Fine-tuning of the network is performed on training data and validation data. To this end, we test 50 different sets of hyperparameters to find the setting whose obtained AUC-PR on validation data is maximized. Network training stops when it reaches to maximum number of epochs, 2000, or early stopping condition is met (patience is set to 50). After finding the best setting, we start to train the network with the chosen hyperparameters while the maximum number of training epochs is set to 4000. In case of meeting early stopping condition, model weights from the epoch with the minimum validation loss is restored.

In the following, we present results obtained from NNs with different settings and architectures.

4.2 Baselines to Compare Performance of the Proposed Model with

Logistic regression (LR) is a statistical method that is applicable for binary classification problems. LR method works well in the field of drug resistance prediction in TB on SNP absence/presence data, hence it is used as the main benchmark here. Note that LR is applicable for single label classification problems not multi-label ones. In this case, we run LR for each drug separately and report its performance. Additionally, Chen *et al.* [66] propose to use a multi-task wide&deep NN (WDNN) to predict drug resistance in TB. As we performed some experiments we noticed that residual NN or even fully connected NN sometimes works better than the WDNN for our data. Thus, we set the NN architecture as a hyperparameter to find the best architecture. Figure 4.2 shows the obtained results on first-line drugs from LR, multi-task NN (NN5) as well as single-label NN (SNN) using just SNP abs

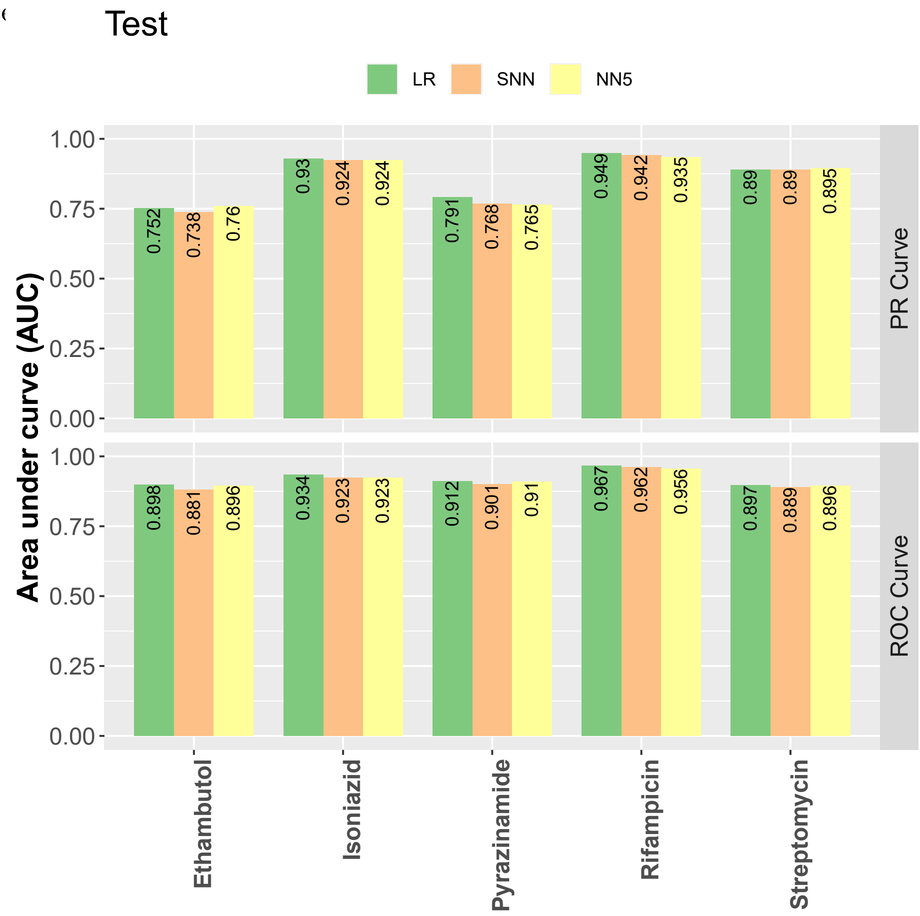


Figure 4.2: Obtained AUC values from LR, SNN and NN5 methods using SNP absence/presence data on the first-line TB drugs. Values on each bar shows the exact AUC value.

As seen in terms of AUC-PR, that is the main performance metric in our study, LR method works better for Isoniazid, Pyrazinamide and Rifampicin, however, the difference between LR's and NN5's values is not considerable. Noteworthy to mention that according to the Bayesian hyperparameter tuning, in NN5 WDNN outperforms the other ones, i.e. Residual or FC blocks, and is selected as the main model and in SNN, WDNN has been used for Isoniazid, Pyrazinamide and Streptomycin, residualNN has been used for Ethambutol and FC has been used for Rifampicin.

4.3 Using Multiple Input Data Types

In this experiment, we have applied NN5 on different input data types. Figure 4.3 shows the results. According to the results, 'TransFrac' and 'PrematFlag' data are not good sources of data and their AUC-PR values are the lowest for each drug while using three sources of data, 'SNPAbs-ProtSeq-TransFrac' produces the highest AUC-PR for 4 drugs out of 5 drugs and also, according to Figure 4.4, the average AUC of using 'SNPAbs-ProtSeq-TransFrac' as well as 'SNPType-ProtSeq-PrematFlag' data types outperform the other data sources.

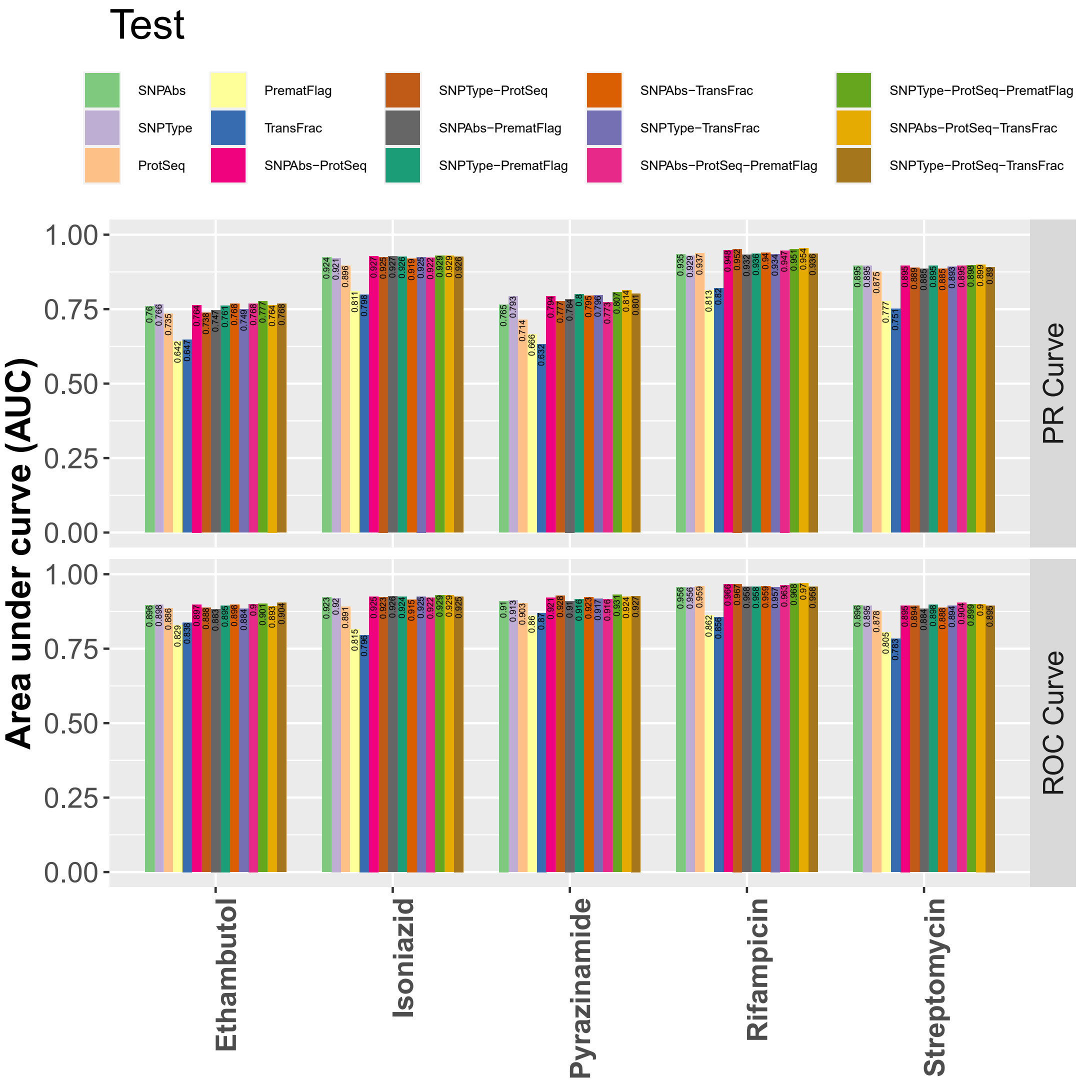


Figure 4.3: Obtained AUC from NN5 on first-line drugs.

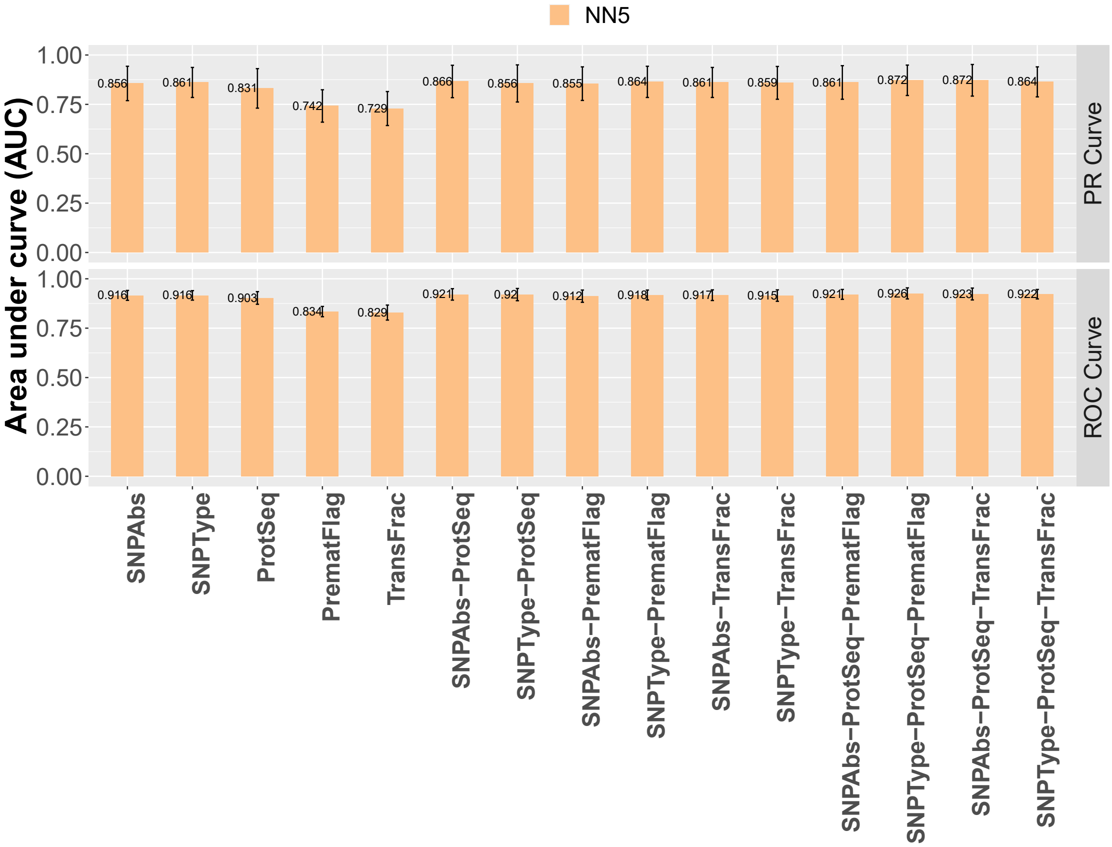
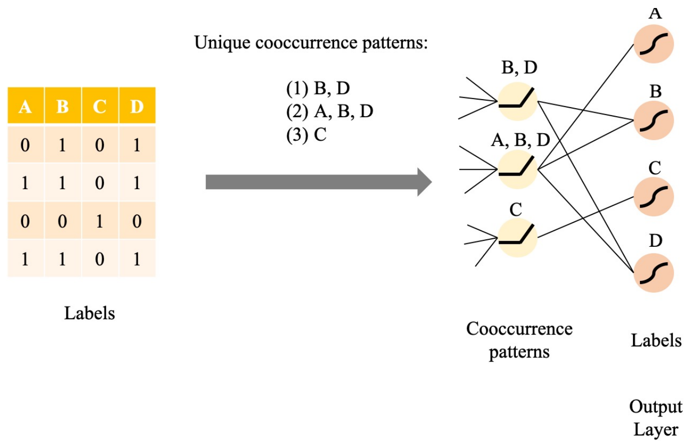


Figure 4.4: Average AUC from NN5 across first-line drugs. Each bar corresponds to the average AUC values across the set of drugs. Error bars show the standard deviation across drugs.

4.4 Considering Co-occurrence of Resistance Labels in NN5

According to [67], when we face with a multi-label classification problem considering the co-occurrence of labels and adding a layer before the output layer can be helpful. To this end, unique patterns of resistance co-occurrence in drugs are extracted and a dense layer, whose number of neurons/nodes is set to the number of unique co-occurrence patterns, is added to the network right before the output layer. We refer to this model as 'NN5_COOCC'. Regarding the connections, there is a connection between a node, say A, in the co-occurrence layer and a node, say B, in the output layer if $B \in A$ (See Figure 4.5).



A	B	C	D
0	1	0	1
1	1	0	1
0	0	1	0
1	1	0	1

Figure 4.5: Adding a label co-occurrence layer right before the output layer. As seen these two layers are not fully connected. Neuron $\{A, B, D\}$ is only connected to neurons A, B and D and not C in the output layer.

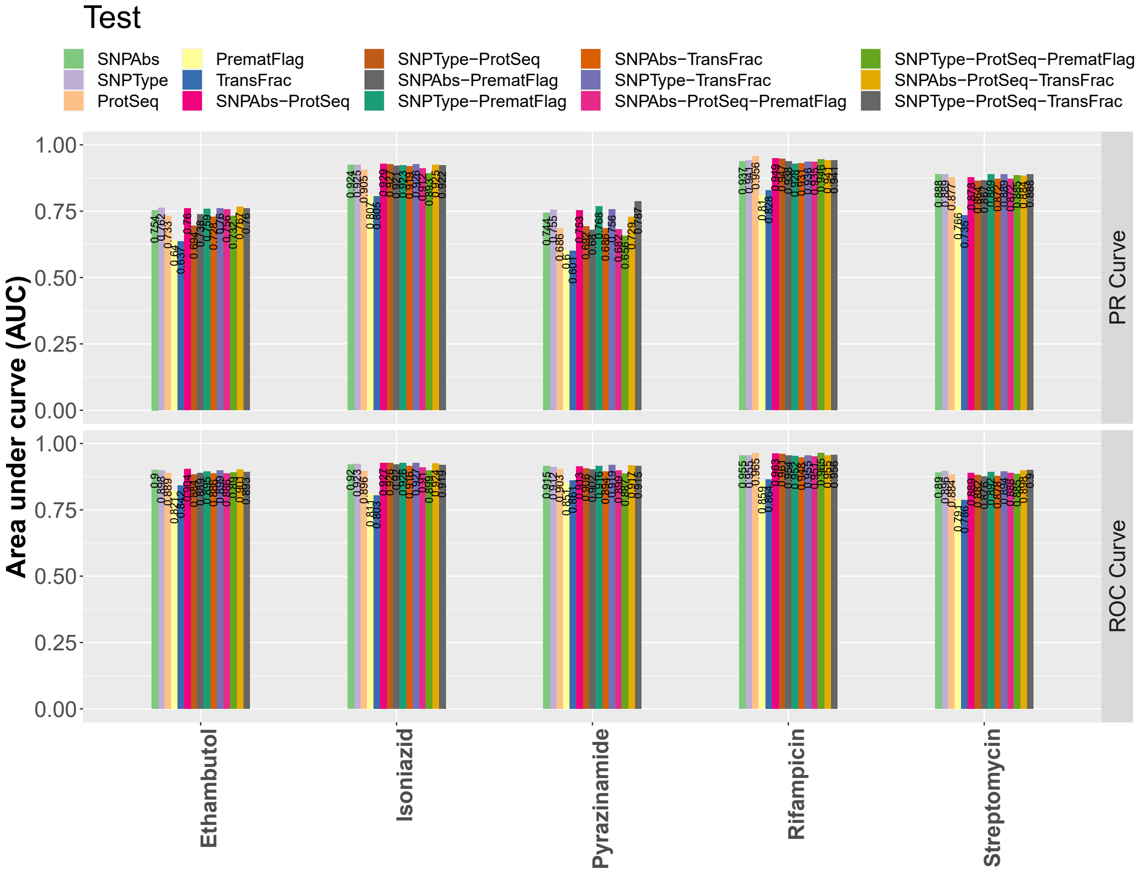


Figure 4.6: Obtained AUC from NN5_COOCC.

Figure 4.6 shows the results of this experiment using different input data types. The AUC-PR of ‘TransFrac’ and ‘PrematFlag’ data are the lowest while performance of the other data types are too close to each other such that we cannot say which data source(s) is better than the other ones. Figure 4.7 compares the average AUC values of 'NN5_COOCC' and 'NN5' across first-line drugs using different input data sources. According to average AUC-PR values, data 'SNPType-ProtSeq-TransFrac' produces the best result for NN5_COOCC model, however, in comparison with NN5 its AUC-PR is a bit, 0.4% lower.

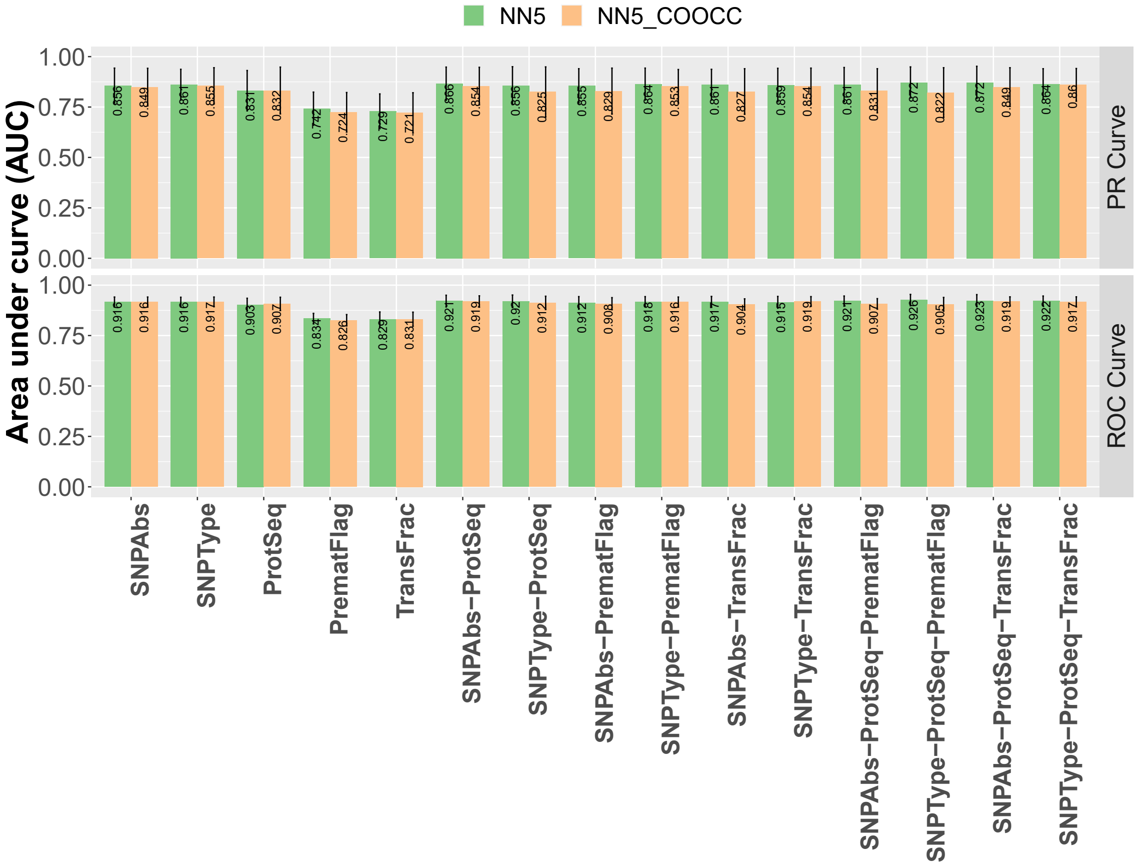
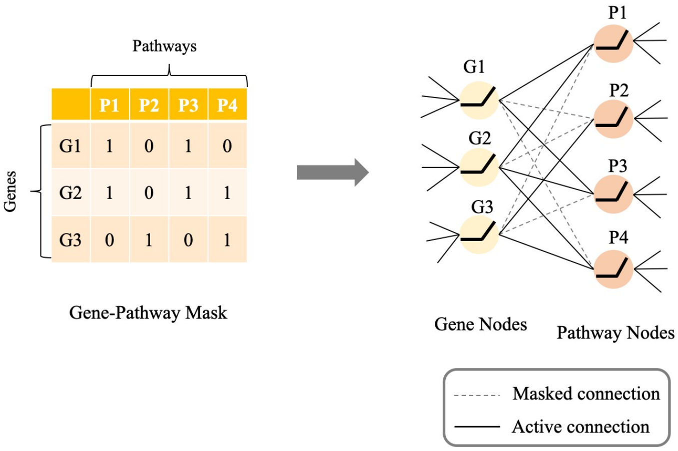


Figure 4.7: Average AUC from NN5 and NN5_COOCC across first-line drugs. Each bar corresponds to the average AUC values across the set of drugs. Error bars show the standard deviation across drugs.

4.5 Masking Connections Using Gene-Pathway Data in Sequence Processing Block

In this experiment, we try to add domain knowledge on TB pathways to the neural network when one of the input data is protein sequence data. To this end, with help of KEGG database we create a gene-pathway mask, and also a layer is added right after the last layer of sequence processing block. We call this model 'NN_GPM'. If the value in the entry $[gene_i, pathway_j]$ in the mask is 0 we set the connection weight between gene i and pathway j in the network to 0, otherwise we do not change the connection weight (See Figure 4.8).



	P1	P2	P3	P4
G1	1	0	1	0
G2	1	0	1	1
G3	0	1	0	1

Figure 4.8: Connecting the last layer of sequence processing block to a layer whose neurons correspond to the pathways. If entry $[G_i, P_j]$ is one the connection between node G_i and P_j is active, otherwise it is masked, i.e. its weight set to 0.

Figure 4.9 illustrates the obtained results from NN5_GPM for different combination of input data. Note that protein sequence data is always used since gene-pathway mask is only applied on sequence processing block. As shown, 'SNPAbs-ProtSeq' data obtains higher AUC-PR values in comparison with the other data sources.

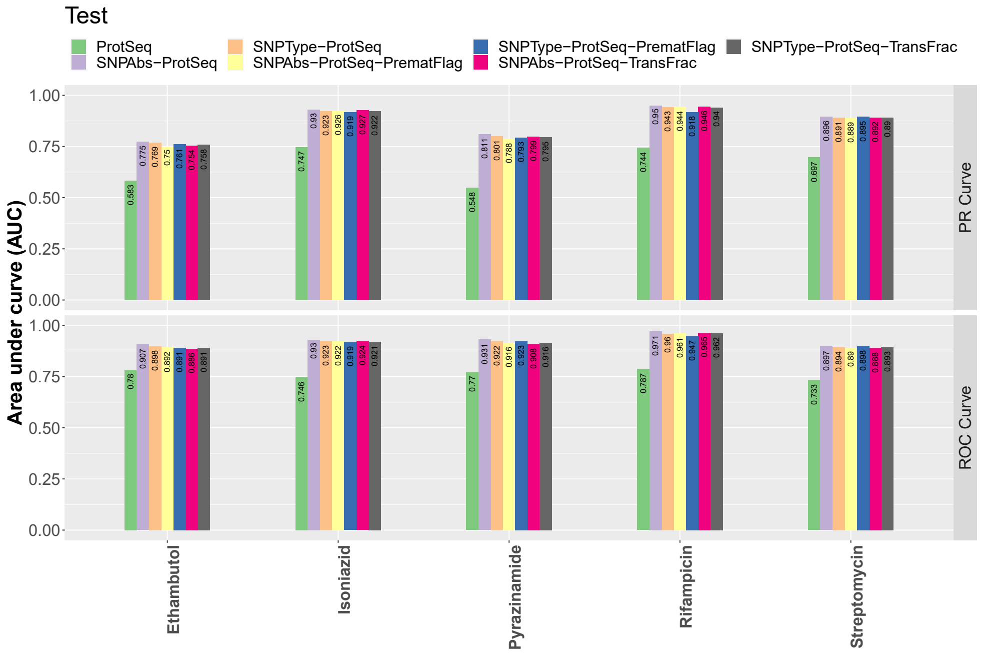


Figure 4.9: Obtained AUC from NN5_GPM.

Figure 4.10 shows average AUC-PR across first-line drugs for NN5 and NN5_GPM. As shown, using 'SNPAbs-ProtSeq' and 'SNPType-ProtSeq' data for NN5_GPM produce a slightly better AUC-PR in comparison with NN5, however, considering the other data sources NN5 is slightly better than NN5_GPM.

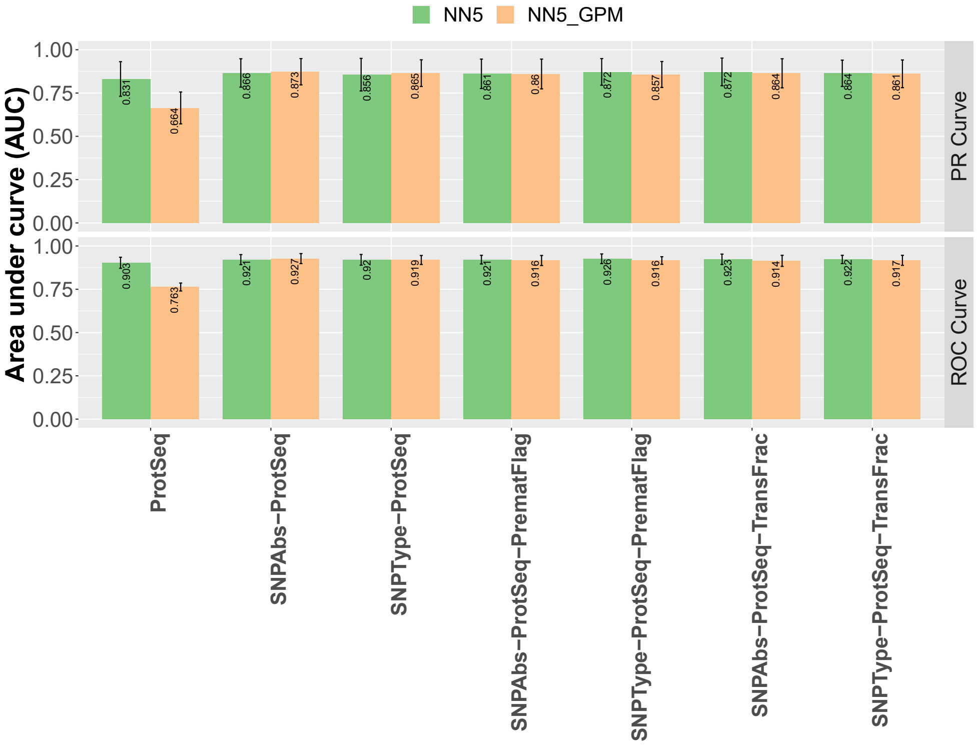


Figure 4.10: Average AUC from NN5 and NN5_GPM across first-line drugs. Each bar corresponds to the average AUC values across the set of drugs. Error bars show the standard deviation across drugs.

4.6 Weighted Loss to Handle Imbalance Data

As shown in Figure 4.1, the number of resistant isolates is less than the number of susceptible ones for each drug that makes the data imbalanced. To handle the imbalance data one of the popular method in the field of deep neural network is using weights for each class for each label. Here, we have used WBCE formula as introduced in Section 3.1.7 to see if it can improve the results. We refer to this model as ‘NN5_WBCE’. Figure 4.11 shows the results for different types of input data on first-line drugs. Other than ‘PrematFlag’ and ‘TransFrac’ data the AUC-PR for different drugs and input data are very close.

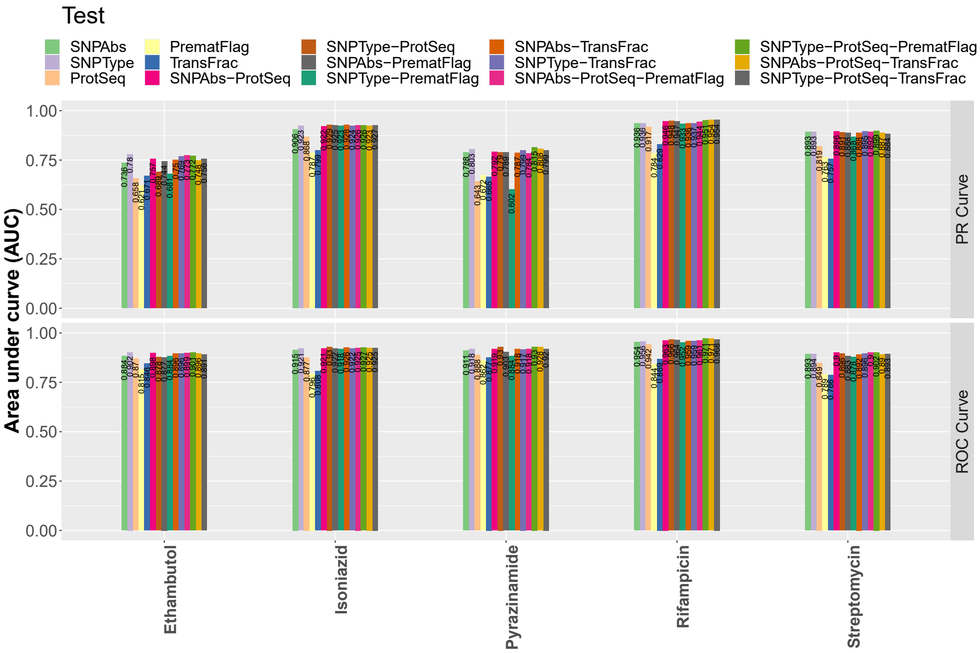


Figure 4.11: Obtained AUC from NN5_WBCE.

To find if WBCE can help to improve the results we have compared the average AUC values across first-line drugs obtained by NN5 and NN5_WBCE in Figure 4.12. As seen, the results are very close and there is no significant difference between them.

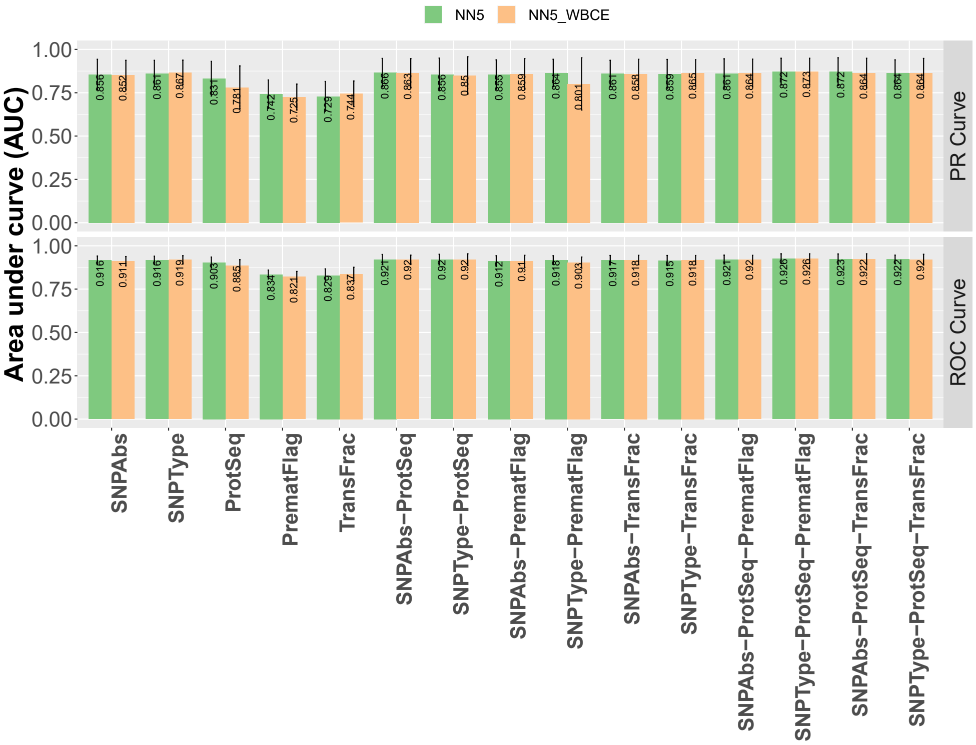


Figure 4.12: Average AUC from NN5 and NN5_WBCE across first-line drugs. Each bar corresponds to the average AUC values across the set of drugs. Error bars show the standard deviation across drugs.

4.7 Summarizing the Results

Figure 4.13 helps to get a complete insight on the results reported in previous sections. It plots top three methods in terms of AUC-PR and AUC-ROC. As shown, 'ProtSeq', 'SNPAbs' and 'SNPType' data are seen frequently in this figure that indicates these three data sources are the most informative ones. As shown, LR is included in the top three methods only for Isoniazid while its AUC-PR is still equal to NN5_GPM (SNPAbs-ProtSeq). Among different NN5 variants and different data inputs NN5 (SNPAbs-ProtSeq-TransFrac) and NN5_GPM (SNPAbs-ProtSeq) show a good performance which appear in the top three list for four and three drugs, respectively.

Figure 4.14 shows the best average AUC across first-line drugs for each method. Note that the data input for both LR and SNN is only SNPAbs while for the other methods different sources of data have been used. As seen NN5_GPM (SNPAbs-ProtSeq), and NN5_WBCE (SNPType-ProtSeq-PrematFlag) have obtained higher average AUC-PR, however, they are very close to the other combination of methods and input data.

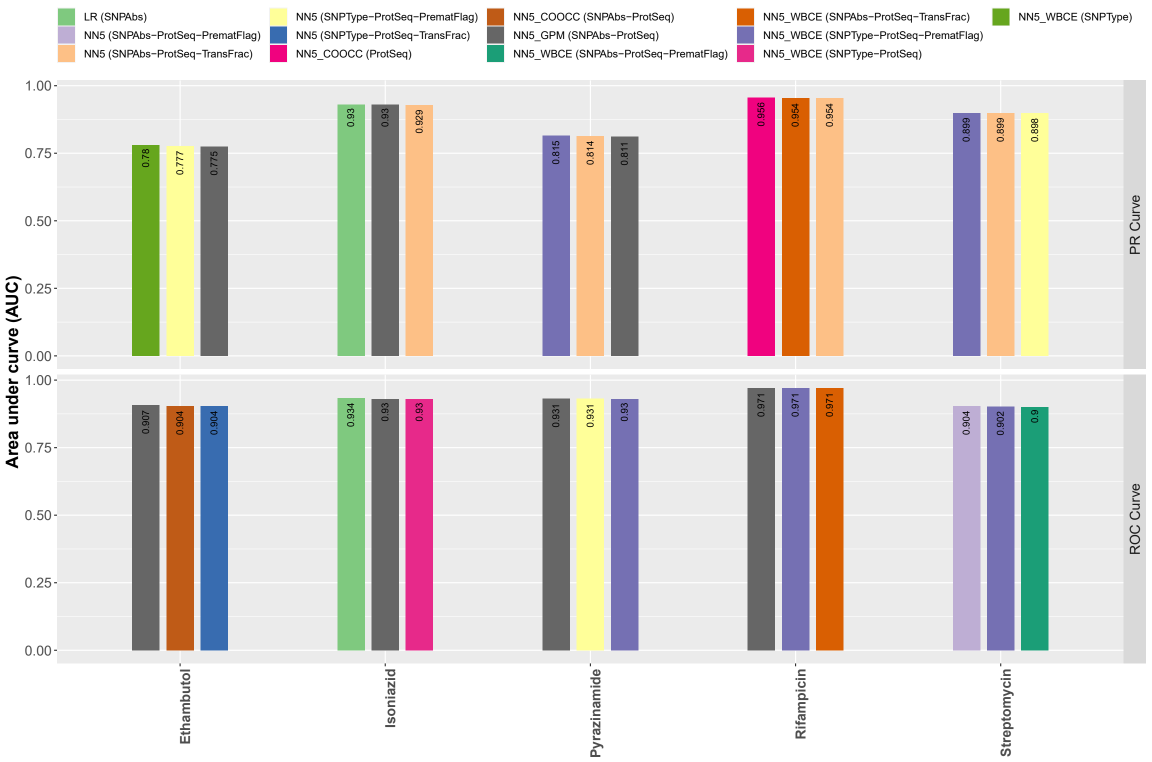


Figure 4.13: Top three AUC results for each drug. The legend shows the name of method and data input(s) for each bar color.

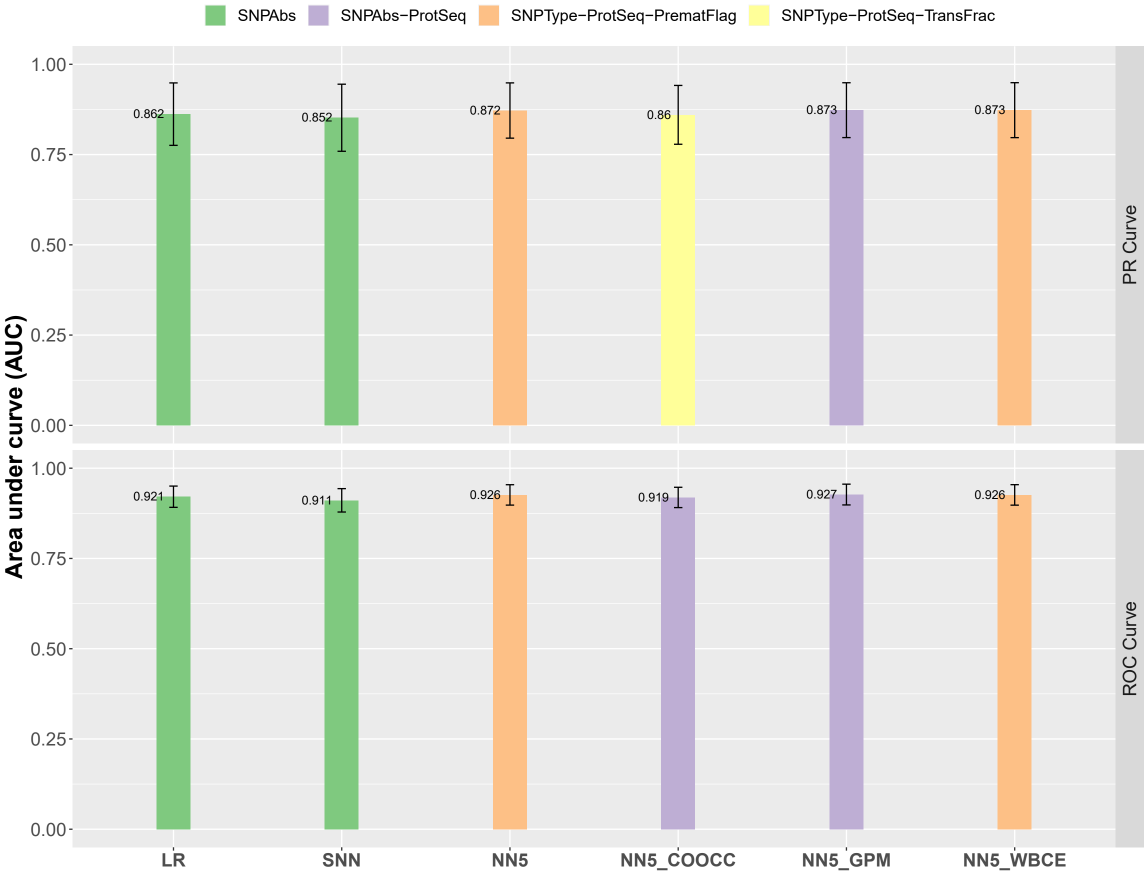


Figure 4.14: Average AUC results for first-line drugs. Each bar corresponds to a method and the color of each bar corresponds to the used data input(s) that are shown in the legend. The error bars indicates to the standard deviation across AUC values in first-line drugs.

Table 4.3: The statistics of AUC-PR values obtained from 10 runs and corresponding %95 confidence intervals (CI). The last column shows if the obtained results are significant when we compare with LR AUC-PR.

	Min	Max	Mean	Std	95% CI	Significant
LR	-	-	0.862	-	-	
SNN	0.849	0.858	0.854	0.002	[0.852, 0.856]	
NN5	0.924	0.927	0.926	0.001	[0.925, 0.926]	*
NN5_COOCC	0.923	0.926	0.925	0.001	[0.924, 0.926]	
NN5_GPM	0.914	0.922	0.919	0.003	[0.917, 0.921]	*
NN5_WBCE	0.915	0.919	0.917	0.001	[0.916, 0.918]	

To make sure that the results shown in Figure 4.14 are robust, we ran the NNs, while the weights are randomly initialized, 10 times using the best parameter settings found by the Bayesian optimization. Table 4.3 shows the statistics computed based on the obtained AUC-PR as well as the corresponding %95 confidence intervals (CIs) and the significance. To compute the CI, given that we have limited number of runs ($n = 10 < 30$), we use the t-distribution. As seen, the CIs are very small indicating to robustness of the NNs. Additionally, performing significance test with considering that critical value of t at 0.05 and $dof = 9$ is 1.833, it turns out that the $NN5$ and $NN5\_GPM$ are significant when we compare their mean with LR.

Considering these figures that summarize results of more than 70 experiments we can say:

- Multi-task NNs outperform SNN and LR as in Figure 4.13 all the three top AUC results except for Isoniazid come from NN5. Figure 4.14 also confirms this as the average AUC values for NN5 methods are higher than SNN and LR. These results can be justified as multi-task NNs are able to capture the relationship between drugs.
- The prediction performance of models using 'TransFrac' and 'PrematFlag' as a single input data is not good (see Figures 4.3, 4.6, 4.11) but when they are combined with the other data types they can slightly improve the performance as it is seen in Figure 4.13 where these two data sources appear 7 times in the legend, and 9 bars out of 15 bars in the PR curve part.
- As seen in the Figure 4.13, 'ProtSeq' data exist in all the bar colors except than the LR and NN5_WBCE bars and also in Figure 4.14 all the best results from NN5 variants use 'ProtSeq' data. Thus, we can say that 'ProtSeq' data is a good source of information to predict drug resistance. Additionally as seen in these two figures, in almost all cases when 'ProtSeq' data is used, one of the SNP data sources, i.e. 'SNPAbs' or 'SNPType', is also used. This indicates that using these two data sources,

'ProtSeq' data and SNP data, improves the final results. The only problem regarding using protein sequence data is its high demand for computational resources.

- As seen the best results based on both AUC-PR and AUC-ROC in the Figure 4.14 come from NN5_GPM using two sources of data and the second best results come from NN5_WBCE using three sources of data. Considering the data sources in the Figure 4.13 we also see that about half of the bars come from NNs with three data sources. Based on this we can say that using three sources of the data in NN5 models can produce good results but seems that such results or even slightly better results can be produced using the other variants of NN5. i.e. NN5_COOCC, NN5_GPM or NN5_WBCE, with less input data sources. Thus, changing the NN architecture in a good manner can positively affect the model performance.

- Although the obtained results confirm that the SNP data is informative for predicting drug resistance as several research use it, we believe that 'ProtSeq' data has also a good amount of information regarding drug resistance mechanism. There are a few research papers discussing the roles of TB proteins in drug resistance. For example Sharma *et al.*in [68] talk about isolates that show drug resistance to some first-line and second-line drugs while no mutations in genes, who are responsible for drug resistance, exist. Such thing can only be explained by studying proteomics in TB as proteins can control biological processes.

- Last but not least, computational cost of using neural networks is much higher than methods like LR. While training process in LR takes less than half an hour and it does not need powerful processors, training process in a neural network takes from few hours in a small neural network to 8 hours in a big neural network on graphics processing units (GPUs). Therefore, considering that the performance of LR and NNs are very close, in case of having limited time and computational resources LR seems a better choice than neural networks. However, if the goal is not only getting a good performance but studying the important factors of drug resistance specifically based on different data types, NNs are better as different types of data can be fed into the networks.

Chapter 5

Discussion

Tuberculosis (TB), caused by Mycobacterium tuberculosis (MTB) bacteria, is a serious airborne disease that can be spread via coughing or sneezing from a sick person to another person. TB primarily occurs in the lungs; however, it can also affect other parts of the body like the spine, the brain and the kidneys. According to WHO, TB is one of the top 10 leading causes of death in the world. Two types of TB infection are active TB and latent TB infection (LTBI). Although both of these can be completely cured using antibiotics, not every drug can cure the disease due to the emergence of drug resistant TB strains in recent years. Treating drug resistant TB is harder than normal TB and it is better to find if the TB is drug resistant prior to prescribing medicine. The common method of determining if the TB is drug resistant is via laboratory experiments, but this method is expensive, time consuming and not always accessible. Considering the time and cost of identifying drug resistant TB in the lab, researchers proposed machine learning techniques to predict drug resistance based on Whole Genome Sequencing (WGS) data. WGS data is used as a main source of data to predict drug resistance since it has been discovered that some mutations in specific genes can make a TB isolate resistant to a specific drug. In order to use WGS data to predict drug resistance in TB, SNP calling software is used to detect the SNPs in the sequence of TB isolates and then the absence/presence of SNPs is used to develop and train machine learning models.

In this project, we first detect the SNPs using GATK and SAMTools SNP callers and identify the common SNPs so as to reduce false positive SNPs. Then, using the detected SNPs, we generate new types of data to see if the data representation can affect model performance. The data produced based on SNP absence/presence data include SNP type, protein sequence, translation fraction and premature stop codon flag data. We use the data to train several multi-input multi-task NNs to predict drug resistance labels for TB isolates. The obtained results show that integrating more than one source of data can positively affect the model performance, and protein sequence data and SNP data, i.e. SNP absence/presence data or SNP type data, are more informative than the other types. On the other side, manipulating the NN architecture using external knowledge can boost the results.

As we worked on this project, we became aware of some possibilities that can be considered in the future in order to improve the model performance. The first one is about producing SNP absence/presence data that is the main source of data in the field of drug resistance prediction. As mentioned, we have detected more than $700,000$ SNPs, which we know to be primarily false positives (not relevant to drug resistance), and tried to refine them using Chi-squared tests. However, recently some advanced approaches like DeepVariant [69] have been proposed to produce more reliable SNPs. We think that using the advanced SNP callers can increase the quality of SNP absence/presence data which directly affect the final results.

Another aspect of the project that can be investigated is the significant number of partially-labeled TB isolates. In this project, we took it in consideration when we define the loss function; however, this is not the only way to handle the missing labels. Durand *et al.* [70] propose the use of Graph Neural Networks (GNNs) to model the correlation between labels. We think GNNs can be helpful in this matter since TB drug resistance phenotypes are correlated; for example, TB strains that are resistant to Rifampicin are more likely to be resistant to Isoniazid, as well [71].

Furthermore, as seen in our results, protein sequence data is an informative source of data. In this regard, Sharma *et al.*in [68] talk about isolates that show drug resistance to some first-line and second-line drugs while no mutations exists in the genes that are responsible for drug resistance. Such a phenomenon can only be explained by studying proteomics in TB and considering gene/protein expressions. However, processing sequence data is challenging due to its large size and high amount of computational resources required. One way to handle this amount of data is limiting the study to a subset of genes, as we did by focusing on the core TB genome. In spite of reducing the amount of sequence data, feeding the protein sequence data into NNs is tricky and to this end, we chunked them into equal parts, merged the parts and defined specific filters in the convolution layers. Such a method can be used in any other applications where the input data is sequences and is desired that the whole sequence be fed into the network. Given that processing sequence data is not always possible due to lack of computational resources, another way of using sequence data is extracting new features from sequences that capture overall sequence patterns, as was done in [72]. Therefore, instead of feeding the large sequence data to the NN, an extracted tabular data with reasonable size can be used as input.

Lastly, another direction that can be followed is the idea of ensemble learning where instead of a single model, several sub-models are created and trained and then using ensemble approaches like stacking one meta-learner is created. This is also promising since the sub-models can cover weak points of each other and make the final model stronger.

Bibliography

[1] Andrew Gainer-Dewar and Paola Vera-Licona. The minimal hitting set generation problem: algorithms and computation. *SIAM J. Discrete Math.*, 31(1):63–100, 2017.

[2] J.C. Bioch and T. Ibaraki. Complexity of Identification and Dualization of Positive Boolean Functions. *Information and Computation*, 123(1):50 – 63, 1995.

[3] Utz-Uwe Haus, Steffen Klamt, and Tamon Stephen. Computing knock-out strategies in metabolic networks. *J. Comput. Biol.*, 15(3):259–268, 2008.

[4] Michael L Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.

[5] V. Gurvich and L. Khachiyan. On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Discrete Appl. Math.*, 96/97:363–373, 1999.

[6] Utz-Uwe Haus. `cl-jointgen`, a common lisp implementation of the joint-generation method. Available from `https://sourceforge.net/projects/cl-jointgen/` and in C at `https://sourceforge.net/projects/jointgen-c.cl-jointgen.p/`.

[7] Matthias Hagen, Peter Horatschek, and Martin Mundhenk. Experimental comparison of the two Fredman-Khachiyan-algorithms. In *Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 154–161. SIAM, 2009.

[8] Khaled Elbassioni. C implementation of Fredman and Khachiyan's algorithm A. Available from `https://github.com/VeraLiconaResearchGroup/MHSGenerationAlgorithms/blob/master/containers/fka-begk`.

[9] Nafiseh Sedaghat, Tamon Stephen, and Leonid Chindelevitch. Speeding up dualization in the Fredman-Khachiyan algorithm B. In *17th International Symposium on Experimental Algorithms (SEA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[10] Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: a brief survey. *Discrete Appl. Math.*, 156(11):2035–2049, 2008.

[11] Yves Crama and Peter L Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

[12] Martin Mundhenk and Robert Zeranski. How to apply SAT-solving for the equivalence test of monotone normal forms. In Karem A. Sakallah and Laurent Simon, editors,

Theory and Applications of Satisfiability Testing - SAT 2011, pages 105–119, Berlin, 2011. Springer.

[13] V Acuña, F Chierichetti, V Lacroix, A Marchetti-Spaccamela, M-F Sagot, and L Stougie. Modes and cuts in metabolic networks: complexity and algorithms. *BioSystems*, 95:51–60, 2009.

[14] Marco Terzer and Jörg Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24(19):2229–2235, 2008.

[15] Radhakrishnan Mahadevan, Axel von Kamp, and Steffen Klamt. Genome-scale strain designs based on regulatory minimal cut sets. *Bioinformatics*, 31(17):2844–2851, 2015.

[16] Hyun-Seob Song, Noam Goldberg, Ashutosh Mahajan, and Doraiswami Ramkrishna. Sequential computation of elementary modes and minimal cut sets in genome-scale metabolic networks using alternate integer linear programming. *Bioinformatics*, 33(15):2345–2353, 2017.

[17] Steffen Klamt, Radhakrishnan Mahadevan, and Axel von Kamp. Speeding up the core algorithm for the dual calculation of minimal cut sets in large metabolic networks. *BMC Bioinformatics*, 21(1):510, 2020.

[18] Steffen Klamt and Ernst Dieter Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–234, 2004.

[19] Jürgen Zanghellini, David E. Ruckerbauer, Michael Hanscho, and Christian Jungreuthmayer. Elementary flux modes in a nutshell: Properties, calculation and applications. *Biotechnology Journal*, 8(9):1009–1016, 2013.

[20] Leonid Chindelevitch, Jason Trigg, Aviv Regev, and Bonnie Berger. An exact arithmetic toolbox for a consistent and reproducible structural analysis of metabolic network models. *Nature Communications*, 5(4893):165–182, 2014.

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

[22] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes in Theoretical Computer Science*, 322:51–67, 2016.

[23] Tamon Stephen and Timothy Yusun. Counting inequivalent monotone Boolean functions. *Discrete Applied Mathematics*, 167:15–24, 2014.

[24] Jan Bert Van Klinken and Ko Willems van Dijk. FluxModeCalculator: an efficient tool for large-scale flux mode computation. *Bioinformatics*, 32(8):1265–1266, 2015.

[25] Pedro Eduardo Almeida Da Silva and Juan Carlos Palomino. Molecular basis and mechanisms of drug resistance in Mycobacterium tuberculosis: classical and new drugs. *Journal of antimicrobial chemotherapy*, 66(7):1417–1430, 2011.

[26] Quang Huy Nguyen, Lucie Contamin, Thi Van Anh Nguyen, and Anne-Laure Bañuls. Insights into the processes that drive the evolution of drug resistance in Mycobacterium tuberculosis. *Evolutionary applications*, 11(9):1498–1511, 2018.

[27] SH Gillespie. Tuberculosis: evolution in millennia and minutes, 2007.

[28] M McGrath, NC Gey van Pittius, PD Van Helden, RM Warren, and DF Warner. Mutation rate and the emergence of drug resistance in mycobacterium tuberculosis. *Journal of Antimicrobial Chemotherapy*, 69(2):292–302, 2014.

[29] Dany JV Beste, Mateus Espasa, Bhushan Bonde, Andrzej M Kierzek, Graham R Stewart, and Johnjoe McFadden. The genetic requirements for fast and slow growth in mycobacteria. *PLoS One*, 4(4):e5349, 2009.

[30] Francesc Coll, Ruth McNerney, Mark D Preston, José Afonso Guerra-Assunção, Andrew Warry, Grant Hill-Cawthorne, Kim Mallard, Mridul Nair, Anabela Miranda, Adriana Alves, et al. Rapid determination of anti-tuberculosis drug resistance from whole-genome sequences. *Genome medicine*, 7(1):51, 2015.

[31] Jody E Phelan, Denise M O'Sullivan, Diana Machado, Jorge Ramos, Yaa EA Oppong, Susana Campino, Justin O'Grady, Ruth McNerney, Martin L Hibberd, Miguel Viveiros, et al. Integrating informatics tools and portable sequencing technology for rapid detection of resistance to anti-tuberculous drugs. *Genome medicine*, 11(1):41, 2019.

[32] Phelim Bradley, N Claire Gordon, Timothy M Walker, Laura Dunn, Simon Heys, Bill Huang, Sarah Earle, Louise J Pankhurst, Luke Anson, Mariateresa De Cesare, et al. Rapid antibiotic-resistance predictions from genome sequence data for Staphylococcus aureus and Mycobacterium tuberculosis. *Nature communications*, 6(1):1–15, 2015.

[33] Andreas Steiner, David Stucki, Mireia Coscolla, Sonia Borrell, and Sebastien Gagneux. KvarQ: targeted and direct variant calling from fastq reads of bacterial genomes. *BMC genomics*, 15(1):881, 2014.

[34] Silke Feuerriegel, Viola Schleusener, Patrick Beckert, Thomas A Kohl, Paolo Miotto, Daniela M Cirillo, Andrea M Cabibbe, Stefan Niemann, and Kurt Fellenberg. PhyResSE: a web tool delineating Mycobacterium tuberculosis antibiotic resistance and lineage from whole-genome sequencing data. *Journal of clinical microbiology*, 53(6):1908–1914, 2015.

[35] Tra-My Ngo and Yik-Ying Teo. Genomic prediction of tuberculosis drug-resistance: benchmarking existing databases and prediction algorithms. *BMC bioinformatics*, 20(1):68, 2019.

[36] Camilla Hundahl Johnsen, Ole Lund, Philip TLC Clausen, and Frank M Aarestrup. Improved resistance prediction in Mycobacterium tuberculosis by better handling of insertions and deletions, premature stop codons, and filtering of non-informative sites. *Frontiers in microbiology*, 10:2464, 2019.

[37] Samaneh Kouchaki, Yang Yang, Timothy M Walker, A Sarah Walker, Daniel J Wilson, Timothy EA Peto, Derrick W Crook, CRyPTIC Consortium, and David A Clifton. Application of machine learning techniques to tuberculosis drug resistance analysis. *Bioinformatics*, 35(13):2276–2282, 2019.

[38] Timothy M Walker, Thomas A Kohl, Shaheed V Omar, Jessica Hedge, Carlos Del Ojo Elias, Phelim Bradley, Zamin Iqbal, Silke Feuerriegel, Katherine E Niehaus, Daniel J Wilson, et al. Whole-genome sequencing for prediction of Mycobacterium tuberculosis drug susceptibility and resistance: a retrospective cohort study. *The Lancet infectious diseases*, 15(10):1193–1202, 2015.

[39] Yang Yang, Katherine E Niehaus, Timothy M Walker, Zamin Iqbal, A Sarah Walker, Daniel J Wilson, Tim EA Peto, Derrick W Crook, E Grace Smith, Tingting Zhu, et al. Machine learning for classifying tuberculosis drug-resistance from DNA sequencing data. *Bioinformatics*, 34(10):1666–1671, 2018.

[40] Wouter Deelder, Sofia Christakoudi, Jody Phelan, Ernest Diez Benavente, Susana Campino, Ruth McNerney, Luigi Palla, and Taane Gregory Clark. Machine learning predicts accurately mycobacterium tuberculosis drug resistance from whole genome sequencing data. *Frontiers in Genetics*, 10:922, 2019.

[41] Michael L Chen, Akshith Doddi, Jimmy Royer, Luca Freschi, Marco Schito, Matthew Ezewudo, Isaac S Kohane, Andrew Beam, and Maha Farhat. Deep learning predicts tuberculosis drug resistance status from whole-genome sequencing data. *BioRxiv*, page 275628, 2018.

[42] Yang Yang, Timothy M Walker, A Sarah Walker, Daniel J Wilson, Timothy EA Peto, Derrick W Crook, Farah Shamout, Tingting Zhu, and David A Clifton. DeepAMR for predicting co-occurrent resistance of mycobacterium tuberculosis. *Bioinformatics*, 35(18):3240–3249, 2019.

[43] Gustavo Arango-Argoty, Emily Garner, Amy Pruden, Lenwood S Heath, Peter Vikesland, and Liqing Zhang. DeepARG: a deep learning approach for predicting antibiotic resistance genes from metagenomic data. *Microbiome*, 6(1):1–15, 2018.

[44] Malancha Karmakar, Carlos HM Rodrigues, Kristy Horan, Justin T Denholm, and David B Ascher. Structure guided prediction of pyrazinamide resistance mutations in pnca. *Scientific Reports*, 10(1):1–10, 2020.

[45] Andrew Zhang, Ling Teng, and Gil Alterovitz. An explainable machine learning platform for pyrazinamide resistance prediction and genetic feature identification of mycobacterium tuberculosis. *Journal of the American Medical Informatics Association*, 28(3):533–540, 2021.

[46] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, Jared R Maguire, Christopher Hartl, Anthony A Philippakis, Guillermo Del Angel, Manuel A Rivas, Matt Hanna, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491, 2011.

[47] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[48] Michael A DeJesus, James C Sacchettini, and Thomas R Ioerger. Reannotation of translational start sites in the genome of Mycobacterium tuberculosis. *Tuberculosis*, 93(1):18–25, 2013.

[49] Daniel L Hartl. *Essential genetics: A genomics perspective.* Jones & Bartlett Publishers, 2014.

[50] Thomas A Kohl, Roland Diel, Dag Harmsen, Jörg Rothgänger, Karen Meywald Walter, Matthias Merker, Thomas Weniger, and Stefan Niemann. Whole-genome-based Mycobacterium tuberculosis surveillance: a standardized, portable, and expandable approach. *Journal of clinical microbiology*, 52(7):2479–2486, 2014.

[51] Dan Tenenbaum and Bioconductor Package Maintainer. *KEGGREST: Client-side REST access to the Kyoto Encyclopedia of Genes and Genomes (KEGG)*, 2020. R package version 1.30.1.

[52] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[54] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[55] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a: overview of mini–batch gradient descent. *Coursera Lecture slides, https://class. coursera. org/neuralnets-2012-001/lecture*, 2012.

[56] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[57] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[58] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, pages 402–408, 2001.

[59] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[60] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach.* " O'Reilly Media, Inc.", 2017.

[61] Jan Brabec and Lukas Machlica. Bad practices in evaluation methodology relevant to class-imbalanced problems. *arXiv preprint arXiv:1812.01388*, 2018.

[62] Fayyaz Minhas, Amina Asif, and Asa Ben-Hur. Ten ways to fool the masses with machine learning. *arXiv preprint arXiv:1901.01686*, 2019.

[63] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[64] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[66] Michael L Chen, Akshith Doddi, Jimmy Royer, Luca Freschi, Marco Schito, Matthew Ezewudo, Isaac S Kohane, Andrew Beam, and Maha Farhat. Beyond multidrug resistance: Leveraging rare variants with machine and statistical learning models in Mycobacterium tuberculosis resistance prediction. *EBioMedicine*, 43:356–369, 2019.

[67] Gakuto Kurata, Bing Xiang, and Bowen Zhou. Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 521–526, 2016.

[68] Divakar Sharma, Deepa Bisht, and Asad U Khan. Potential alternative strategy against drug resistant tuberculosis: a proteomics prospect. *Proteomes*, 6(2):26, 2018.

[69] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T Afshar, et al. A universal SNP and small-indel variant caller using deep neural networks. *Nature biotechnology*, 36(10):983–987, 2018.

[70] Thibaut Durand, Nazanin Mehrasa, and Greg Mori. Learning a deep convnet for multi-label classification with partial labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 647–657, 2019.

[71] Akos Somoskovi, Linda M Parsons, and Max Salfinger. The molecular basis of resistance to isoniazid, rifampin, and pyrazinamide in mycobacterium tuberculosis. *Respiratory research*, 2(3):1–5, 2001.

[72] Zhen-Ling Peng, Jian-Yi Yang, and Xin Chen. An improved classification of G-protein-coupled receptors using sequence-derived features. *BMC bioinformatics*, 11(1):1–13, 2010.

Appendix A

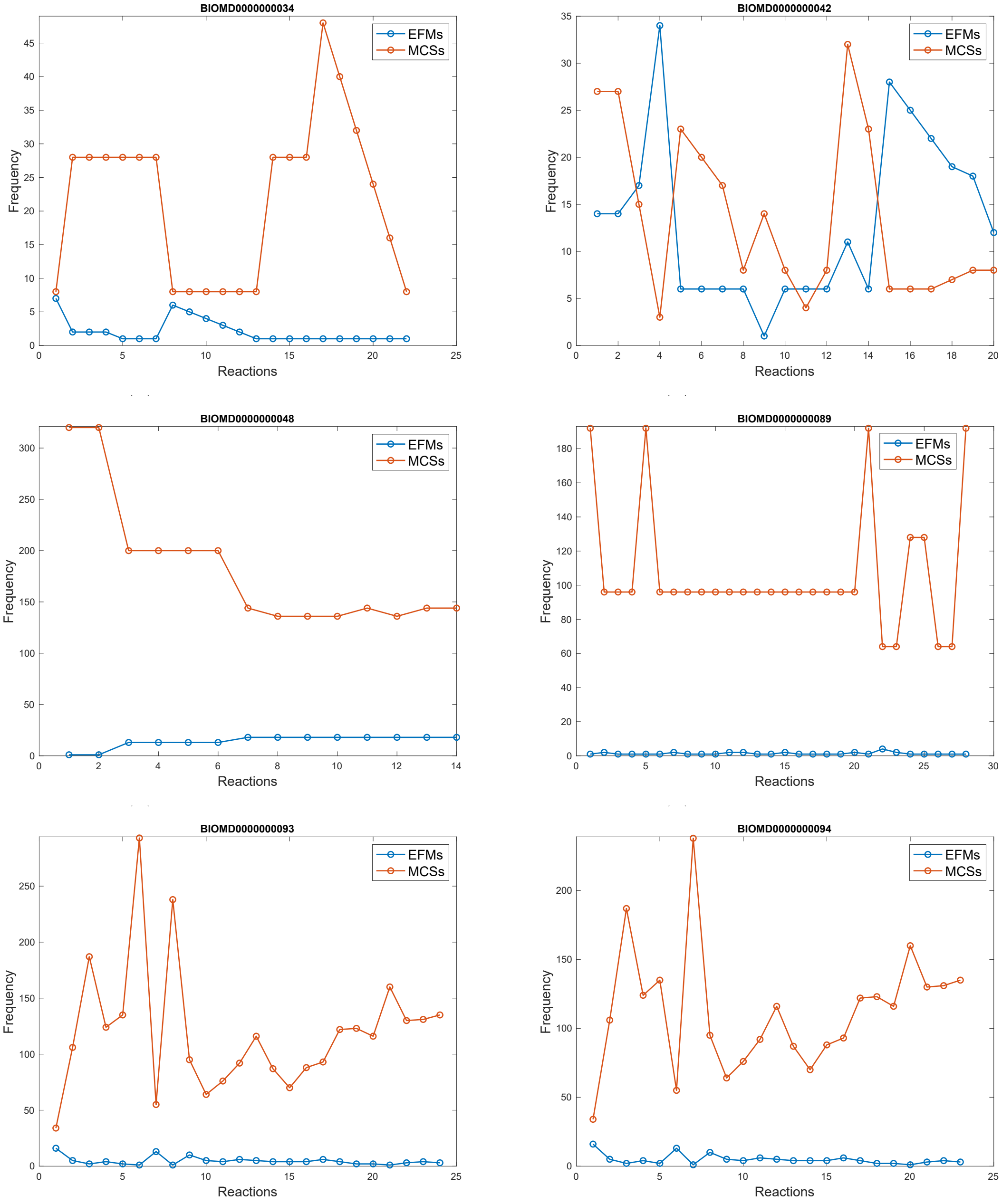
Speeding Up the Structural Analysis of Metabolic Network Models Using the Fredman-Khachiyan Algorithm B

A.1 Code and Data Avaiability

The data and code are available on GitHub Repository `https://github.com/NaSed/Modified_FK_B_Algorithm` .

A.2 Analyzing Reactions in the Biological Models

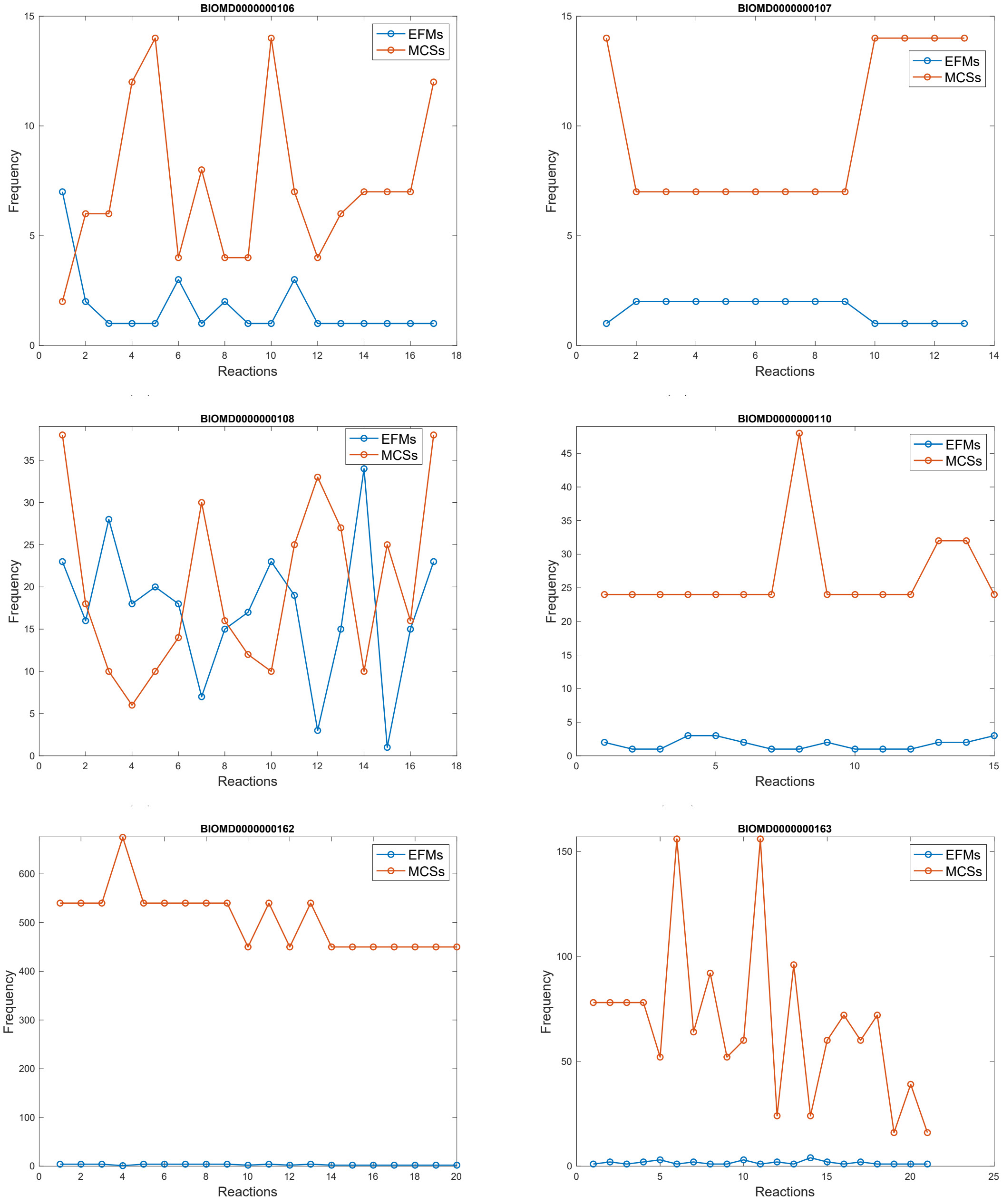
Figure A.1 shows the occurrence frequency of reactions in EFMs and MCSs for each model. As shown, in most of the models, occurrence frequency of reactions in the EFMs are less than the MCSs, e.g. *BIOMD0000000034* and *BIOMD0000000048*.



(5) BIOMD0000000093

(6) BIOMD0000000094

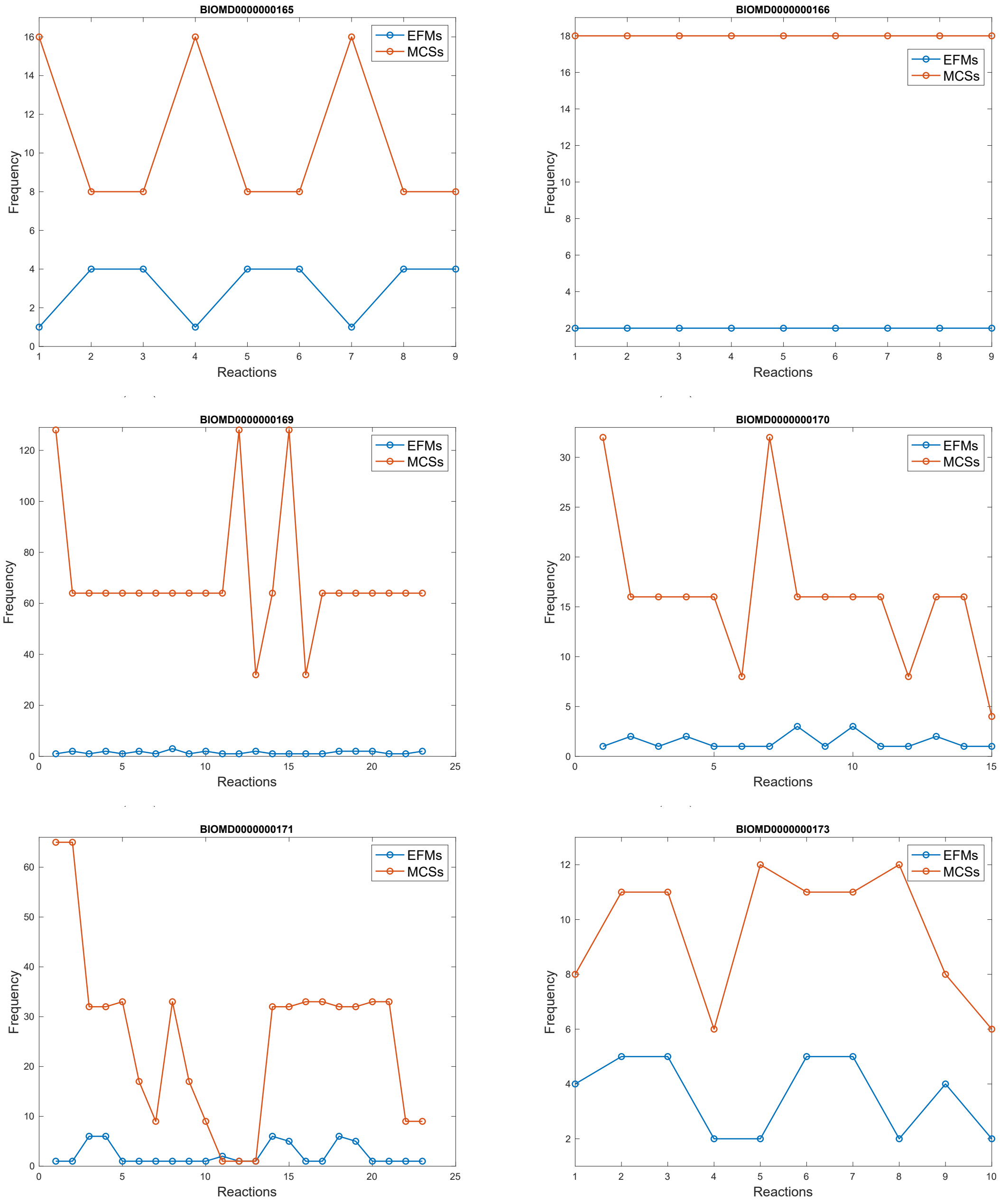
Figure A.1: Frequency of occurrence of reactions in EFMs and MCSs.



(11) BIOMD0000000162

(12) BIOMD0000000163

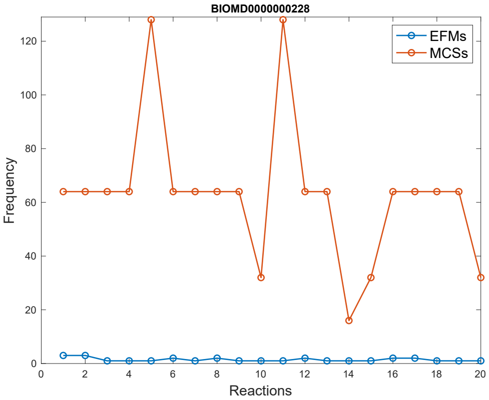
Figure A.1: Continued.



(17) BIOMD0000000171

(18) BIOMD0000000173

Figure A.1: Continued.

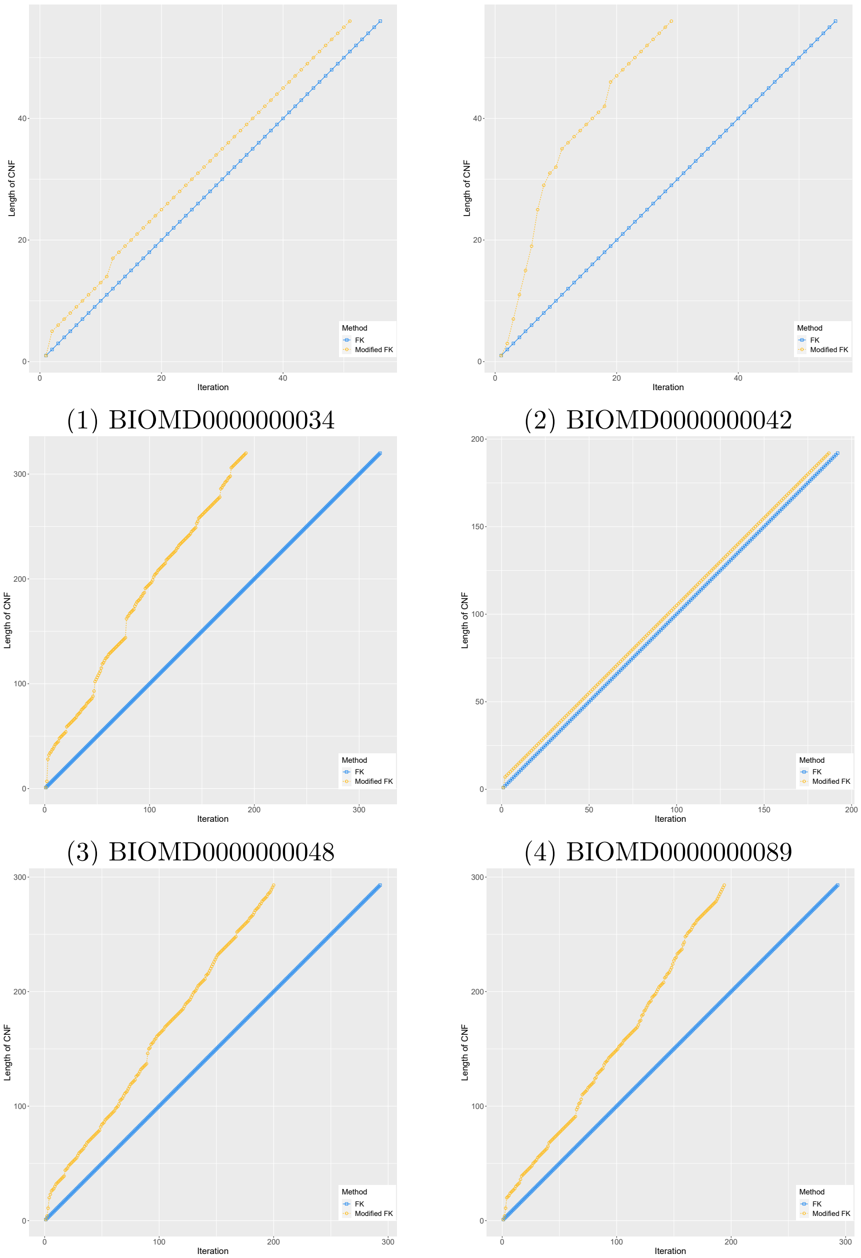


(19) BIOMD0000000228

Figure A.1: Continued.

A.3 CNF Completion Progress in Case of Using FK and modified FK considering finding multiple CAs in the FK-dualization Algorithm

Figure A.2 demonstrates the progression of constructing CNF when FK and modified FK in the dualization procedure.



(1) BIOMD0000000034

(2) BIOMD0000000042

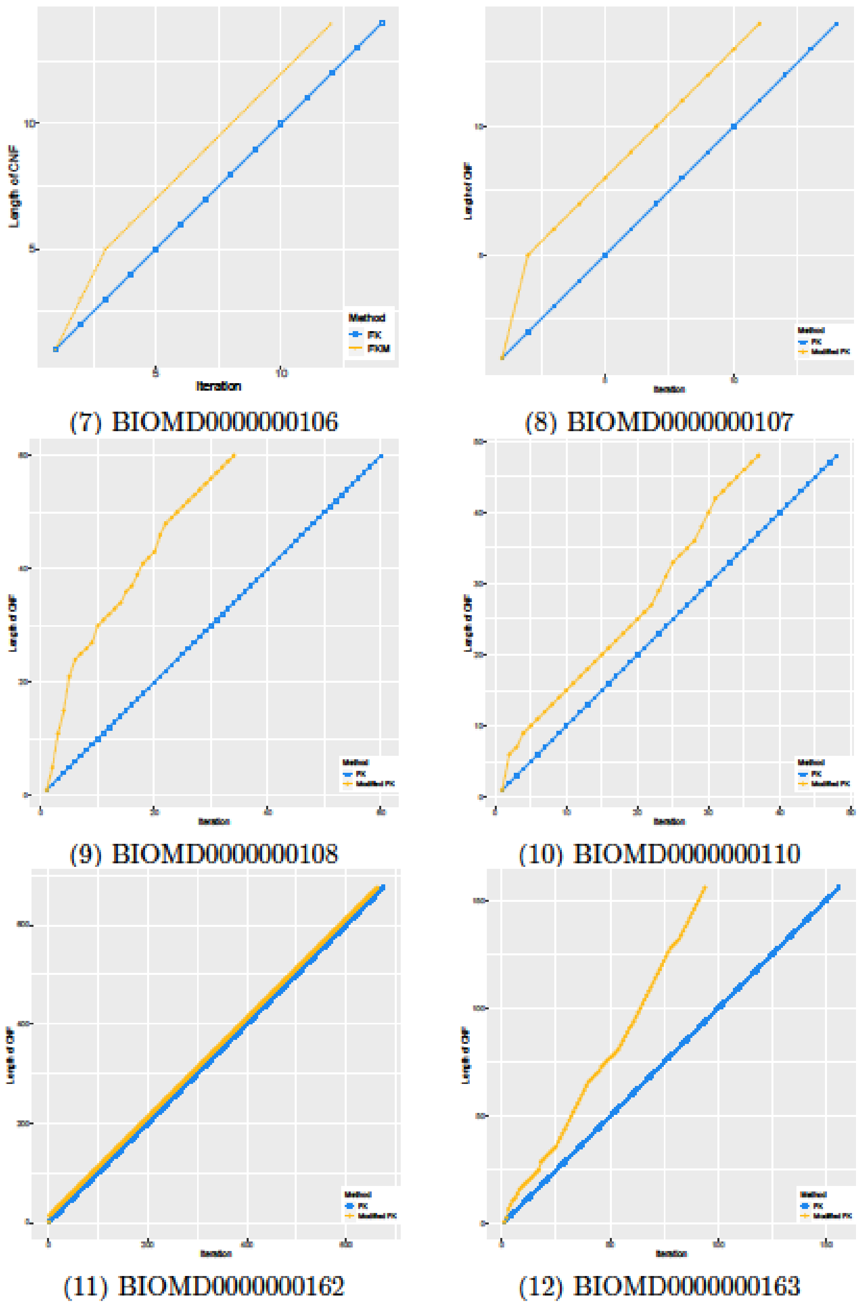
(3) BIOMD0000000048

(4) BIOMD0000000089

(5) BIOMD0000000093

(6) BIOMD0000000094

Figure A.2: Progression of constructing CNF across FK-dualization iterations when FK and FKM have been used for equivalency check between the CNF and the DNF.



(7) BIOMD0000000106

(8) BIOMD0000000107

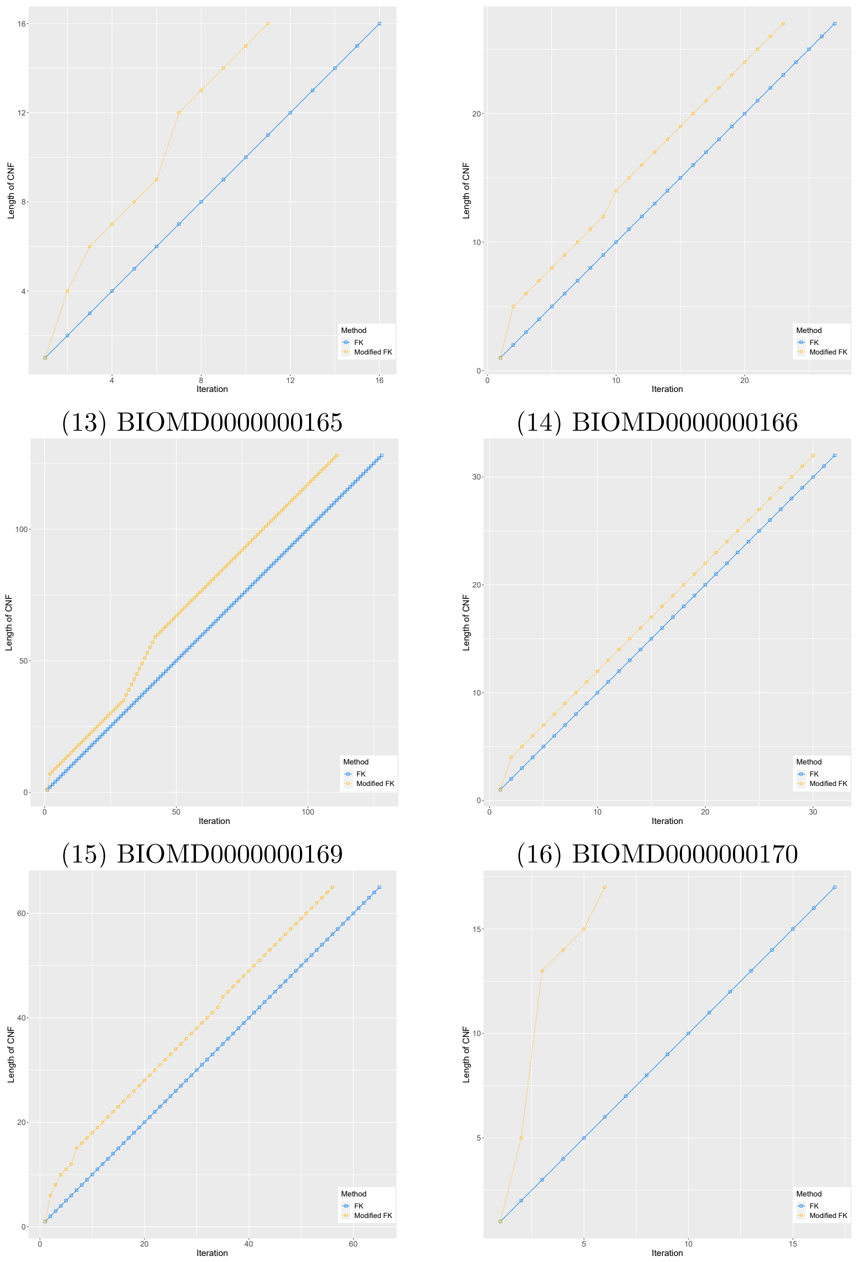
(9) BIOMD0000000108

(10) BIOMD0000000110

(11) BIOMD0000000162

(12) BIOMD0000000163

Figure A.2: Continued.



(13) BIOMD0000000165

(14) BIOMD0000000166

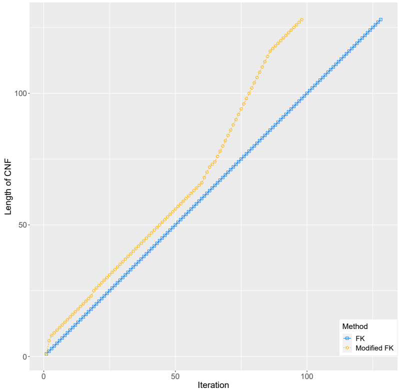
(15) BIOMD0000000169

(16) BIOMD0000000170

(17) BIOMD0000000171

(18) BIOMD0000000173

Figure A.2: Continued.



(19) BIOMD0000000228

Figure A.2: Continued.

Appendix B

Prediction of Drug Resistance in Tuberculosis Using Modular Neural Network

B.1 Code and Data Availability

The original SNP data, and code to produce the other data types, running the model and visualizing the results, are available on GitHub Repository `https://github.com/NaSed/TBDrugResistancePrediction`.

B.2 Considering Co-occurrence of Resistance Labels and Weighted Loss in NN5

In this experiment, we tried adding co-occurrence layer to the network while we use the weighted loss function. We call this 'NN5_COOCC_WBCE'. Figure B.1 represents the AUC values for different sources of data and Figure B.2 compares the results of 'NN5_COOCC_WBCE' with the three variants of 'NN5', 'NN5_COOCC' and 'NN5_WBCE'.

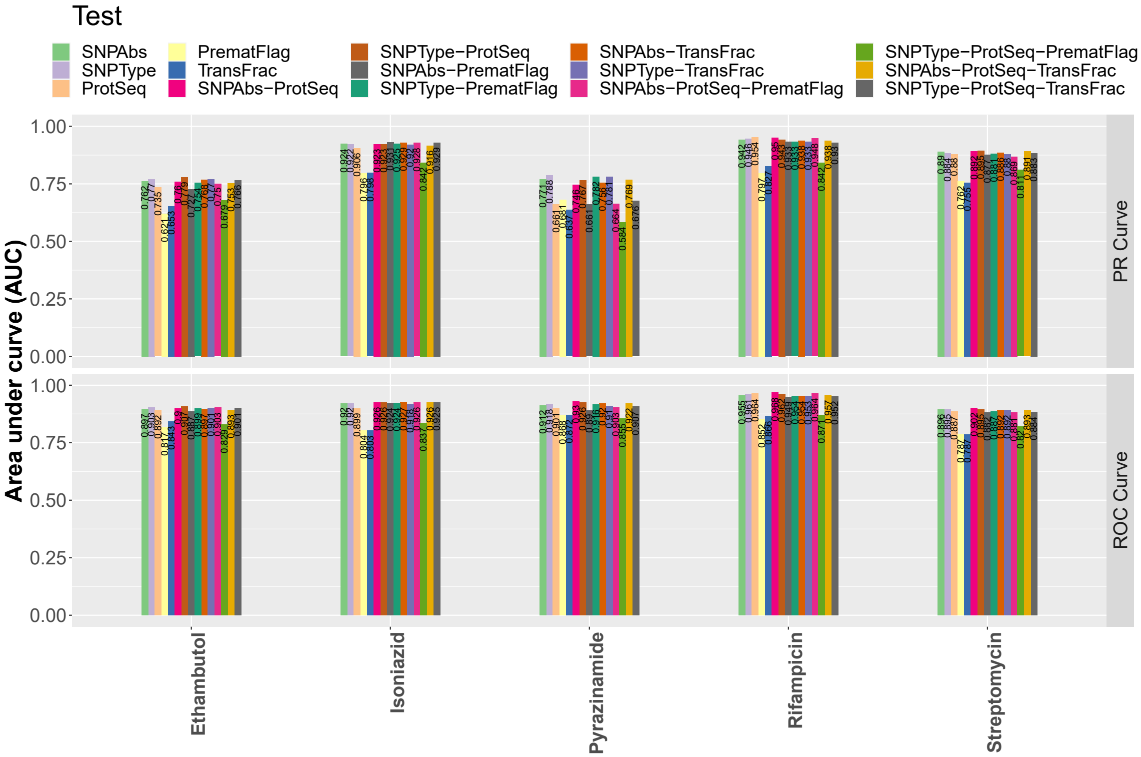


Figure B.1: Obtained AUC from NN5_COOCC_WBCE.

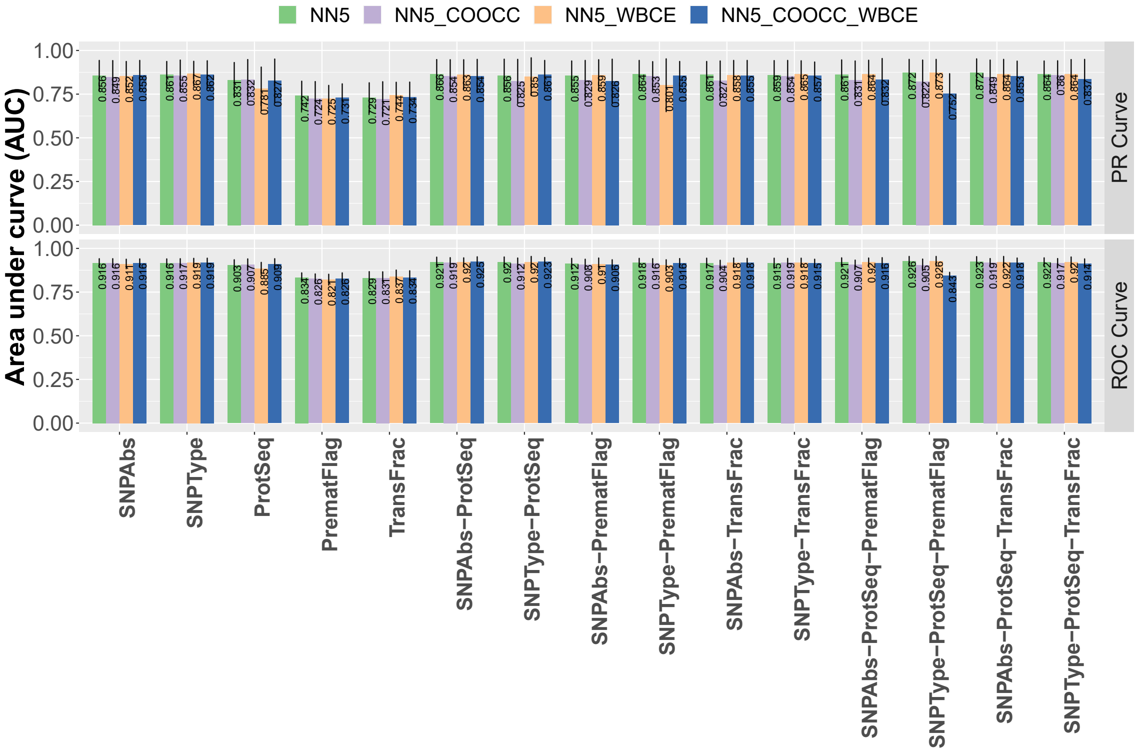


Figure B.2: Average AUC from NN5, NN5_COOCC, NN5_WBCE and NN5_COOCC_WBCE across first-line drugs. Each bar corresponds to the average AUC values across the set of drugs. Error bars show the standard deviation across drugs.

B.3 Considering Masking Connections Using Gene-Pathway Data and Weighted Loss in NN5

In this experiment, we use gene-pathway connection masking as well as weighted loss function in the network. We call this 'NN5_GPM_WBCE'. Figure B.3 represents the AUC values for different sources of data and Figure B.4 compares the results of 'NN5_GPM_WBCE' with the three variants of 'NN5', 'NN5_GPM' and 'NN5_WBCE'.

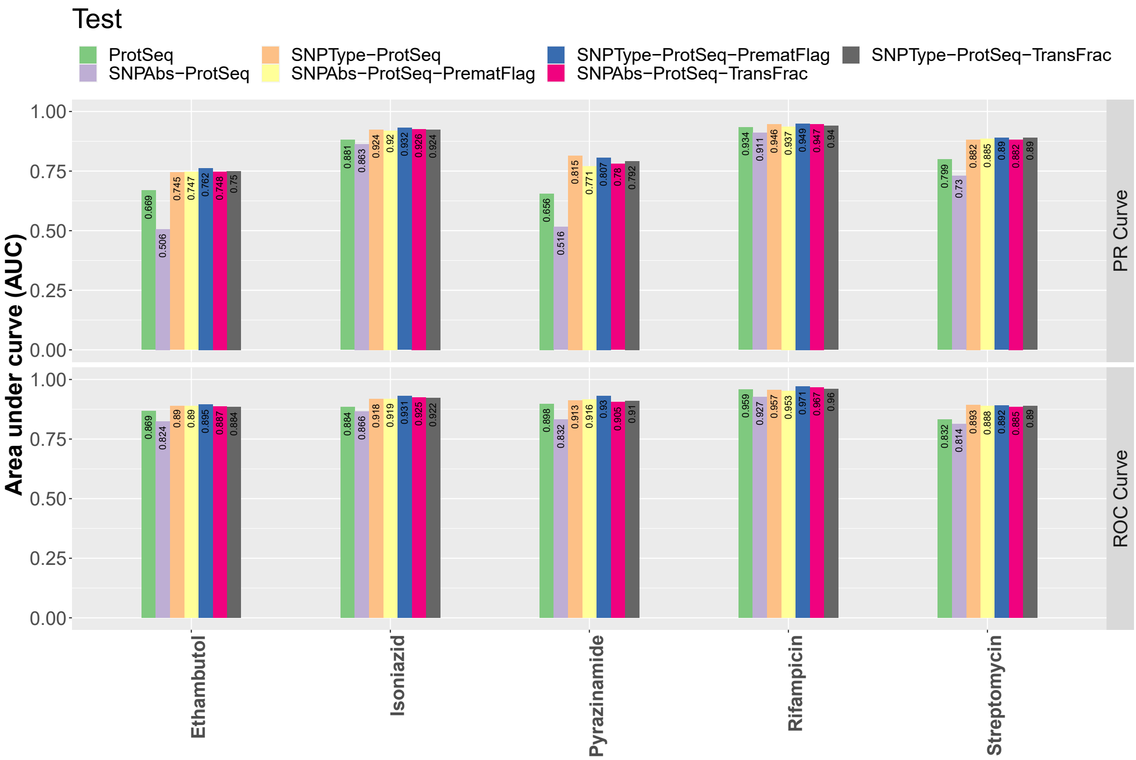


Figure B.3: Obtained AUC from NN5_GPM_WBCE.

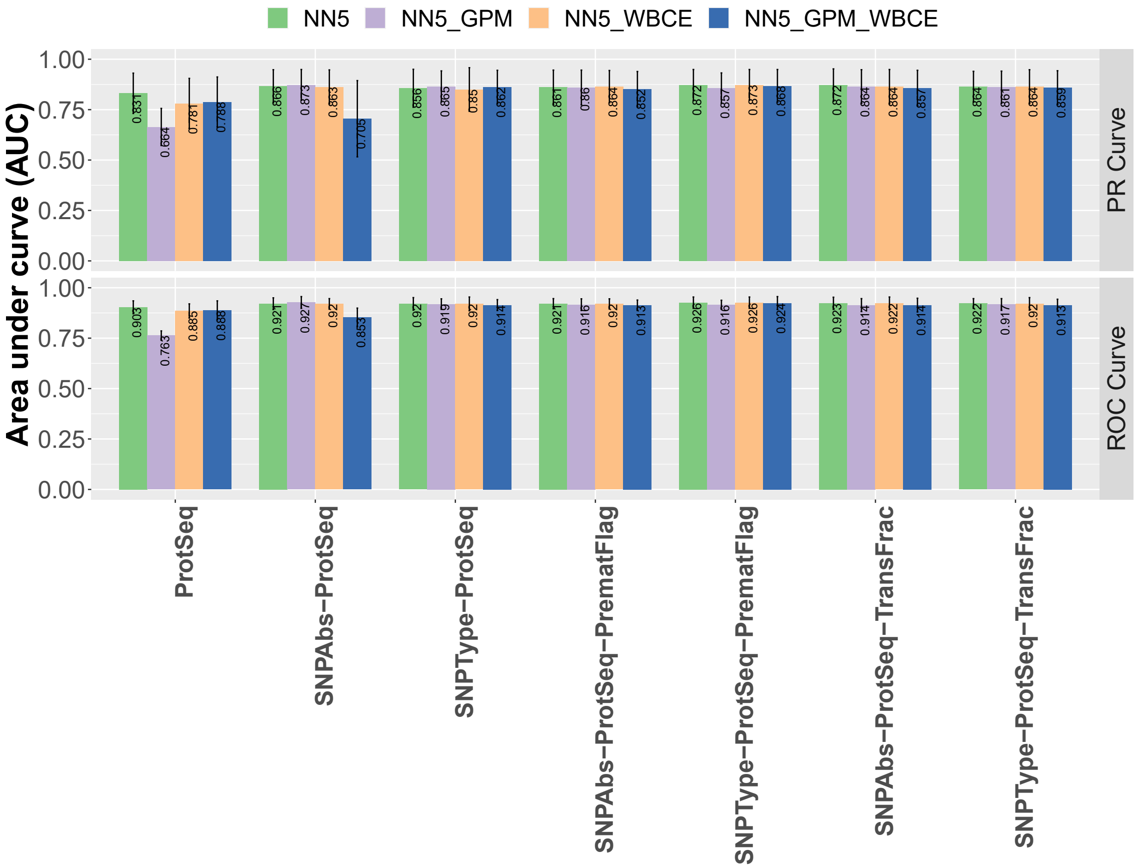


Figure B.4: Average AUC from NN5, NN5_GPM, NN5_WBCE and NN5_GPM_WBCE across first-line drugs. Each bar corresponds to the average AUC values across the set of drugs. Error bars show the standard deviation across drugs.