

**ONLINE TOOLS FOR ANALYZING METAGENOMICS DATA**

BY

TAN KOK HOONG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF COMPUTER SCIENCE (HONS)**

Faculty of Information and Communication Technology

(Perak Campus)

JAN 2017

## REPORT STATUS DECLARATION FORM

**Title:** Online Tools For Analyzing Metagenomics Data

**Academic Session:** Jan 2017

I TAN KOK HOONG

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in

Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

\_\_\_\_\_  
(Author's signature)

\_\_\_\_\_  
(Supervisor's signature)

**Address:**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Supervisor's name

**Date:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**ONLINE TOOLS FOR ANALYZING METAGENOMICS DATA**

BY

TAN KOK HOONG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

JAN 2017

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**ONLINE TOOLS FOR ANALYZING METAGENOMICS DATA**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : \_\_\_\_\_

Name : \_\_\_\_\_

Date : \_\_\_\_\_

## **ACKNOWLEDGEMENTS**

First of all, I would like to express my sincere thanks to my supervisor, Dr. Ng Yen Kaow. He is such a good lecturer and always showing us his patient. It is my honour to engage in bioinformatics, which is quite unfamiliar to me. He always guides us, educates us and provide us the materials needed to make my life easier.

Besides, I would like to thank to Chen Jia Xing, a PhD student from City University of Hong Kong. She taught me from scratch and showed her patient and enthusiasm. She explained everything to me step by step to make sure I am understood.

Finally, I must say thanks to my parents and my family for their love, support and continuous encouragement throughout the course.

## ABSTRACT

This project developed four tools for studying metagenomics data. These four modules will become part of an online platform created by researchers from the City University of Hong Kong for metagenomics use.

The first tool constructed regulatory network module using the CDBN algorithm proposed by (Zhang, Ng & Li 2015). The algorithm constructed a directed biological network from gene expression data. The second tool implemented TIGRESS, another popular method for the inference of biological network. The third tool in this project was a tool for inferring biomarker that was based on the DNB method introduced by (Liu et al. 2012). The fourth and final tool was meant for imputation. The tool aimed to allow users to recover missing values in an abundance matrix.

As these four modules meant for the analysis of very massive data sets, their performance was important. In order to speed-up these tools, the Intel Threading Building Blocks (TBB) was used, a library that facilitated the distribution of processes to CPUs.

# TABLE OF CONTENTS

<b>TITLE</b>	<b>i</b>
<b>DECLARATION OF ORIGINALITY</b>	<b>ii</b>
<b>ACKNOWLEDGEMENT</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>iv</b>
<b>TABLE OF CONTENTS</b>	<b>v</b>
<b>LIST OF TABLES</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>x</b>
<b>CHAPTER 1: INTRODUCTION</b>	<b>1</b>
1.1 Motivation and Problem Statement	1
1.2 Project Scope	3
1.3 Objectives	4
1.4 Impact, significance and contribution	5
1.5 Background information	6
<b>CHAPTER 2: LITERATURE REVIEW</b>	<b>7</b>
2.1 Reviewed Papers	7
2.1.1 Reconstructing Directed Gene Regulatory Network Only Gene Expression Data	7
2.1.2 ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context	8
2.1.3 TIGRESS: Trustful Inference of Gene REgulation using Stability Selection	9
2.1.4 Identifying critical transitions and their leading biomolecular networks in complex diseases	10
2.2 Critical Remarks	11

<b>CHAPTER 3: PROPOSED METHOD / APPROACH</b>	<b>13</b>
3.1 Design Specification	13
3.1.1 Methodologies and General Work Procedures	13
3.1.2 Technologies Used	14
3.1.3 System Performance Definition	14
3.1.4 Verification Plan	14
3.2 System Design	15
3.2.1 Constructing Directed Network Module by Using CBDN	15
3.2.2 Biological Network Inference by Using TIGRESS	20
3.2.3 Inferring Biomarker Module	24
3.2.4 Imputation	27
3.3 Implementation Issues and Challenges	30
3.4 Timelines	31
<b>CHAPTER 4: EXPERIMENTAL RESULT</b>	<b>33</b>
4. 1 Constructing Directed Network Module by Using CBDN	33
4.2 Biological Network Inference by Using TIGRESS	35
4.3 Inferring Biomarker Module	37
4.4 Imputation	39
<b>CHAPTER 5: CONCLUSION</b>	<b>42</b>
<b>REFERENCE</b>	<b>43</b>
<b>APPENDIX A Online Platform Introduction</b>	<b>A-1</b>
<b>APPENDIX B Source Code</b>	<b>B-1</b>
B-1 Constructing Directed Network Module by Using CBDN (Source Code)	B-1
B-2 Constructing Directed Network Module by Using CBDN (Source Code)	B-4
B-3 Inferring Biomarker Module (Source Code)	B-15
B-4 Imputation (Source Code)	B-32



## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 2.2.1	Comparison of 3 algorithms	12
Table 4.1.1	Top 10 of important regulators	34
Table 4.3.1	Composite index of all potential dominant groups	38
Table 4.4.1	Imputation values before and after	41

## LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 2.1.1.1	Algorithm flow of CBDN	7
Figure 2.1.2.1	Algorithm flow of ARACNE	8
Figure 2..1.3.1	Algorithm flow of TRIGRESS	9
Figure 2.1.4.1	Disease Progression	10
Figure 2.1.4.2	Algorithm flow of DNB	10
Figure 3.1.1.1	Project Methodology	13
Figure 3.2.1.1	Directed Network	16
Figure 3.2.1.2	Influence values matrix	17
Figure 3.2.1.3	Two-gene cyclic interaction	17
Figure 3.2.1.4	Influence values matrix – after removing two-gene cyclic interactions	18
Figure 3.2.1.5	Influence values matrix – Final output	18
Figure 3.2.1.6	Regulatory Network	19
Figure 3.2.2.1	Gene expression data (left) and transcription factors (right)	21
Figure 3.2.2.2	Target gene expression values (left) and transcription factors expression values (right)	21
Figure 3.2.2.3	Separation of data	22
Figure 3.2.2.4	Frequency matrix for a gene	22
Figure 3.2.2.4	Frequency cube	23
Figure 3.2.3.1	Algorithm flow of DNB	
Figure 3.2.3.2	Correlation matrix before and after process	25
Figure 3.2.3.3	Composite index graph of a candidate group	16

Figure 3.2.4.1	Imputation process	28
Figure 3.2.4.2	Network constructed on control group	28
Figure 3.4.1	Project timeline (Text)	31
Figure 3.4.2	Project timeline (Diagram)	32
Figure 4.1.1	Human gut microbiome network	33
Figure 4.2.1	Biological network constructed by TIGRESS	35
Figure 4.2.2	<i>In silico</i> data network constructed by TIGRESS	36
Figure 4.3.1	Composite index of the first potential group	38
Figure 4.4.1	Biological network of the first 50 bacteria	39
Figure 4.4.2	First degree neighbours of gene 15	40
Figure 4.4.3	First degree neighbours of gene 33	40
Figure A.1	Main page of the online platform	A-1
Figure A.2	Visualizations that provided by the online platform	A-2
Figure A.3	Modules that provided by the online platform	A-3
Figure A.4	Gene DB of the online platform	A-4
Figure A.5	Delta team of the online platform	A-5

## LIST OF ABBREVIATIONS

<i>NGS</i>	Next-generation Sequencing
<i>ARACNE</i>	Algorithm for Reconstruction of Accurate Cellular Networks
<i>CBDN</i>	Context Based Dependency Network
<i>DPI</i>	Data Processing Inequality
<i>DDPI</i>	Directed Data Processing Inequality
<i>DNB</i>	Dynamic Network Biomarker
<i>GCC</i>	GNU Compiler Collection
<i>GRN</i>	Gene Regulatory Network
<i>LARS</i>	Least Angle Regression
<i>MI</i>	Mutual Information
<i>PCCs</i>	Pearson's Correlation Coefficients
<i>PPL</i>	Parallel Pattern Library
<i>TBB</i>	Thread Building Block
<i>TF</i>	Transcription Factor
<i>TG</i>	Target Gene
<i>TIGRESS</i>	Trustful Inference of Gene REgulation using Stability Selection
<i>PEP</i>	Proteomics Expansion Pipeline
<i>CPM</i>	Clique Percolation Method

## CHAPTER 1: INTRODUCTION

### 1.1 Motivation and Problem Statement

The general availability of Next-generation Sequencing (NGS) technology in the past decade has brought forth many advances in genomics. In particular, NGS techniques have been successfully applied to large-scale mixtures of unbiased genetic material, thus encouraging the analysis of whole-community of organisms in one single study, or *metagenomics*.

In this final year project, we were collaborating with City University of Hong Kong to come up a comprehensive platform for researchers to analyse metagenomics data. The platform would be made available to researchers as an online website. At present, this project had been involved with four modules of the website. Two of these were for constructing biological networks, one of these for inferring biomarkers, and the remaining one was for performing imputation on incomplete data.

A *biological network* was a network which models biological units such as genes as vertices, and the interactions or relationship of these units as edges. The properties of such networks, such as their topology, helped biologists to make important discoveries concerning the modelled biological units. Early studies of biological networks were largely concerned with the interactions of the proteins produced by the genes of a single species. However, scientists had realized the importance of the human microbiota in shaping our health, which had led to an urgency to study the metagenomics of microbial communities. One way to do so was to construct biological networks out of the collective genes of the microbial species within a community, with the hope of understanding how the community interacts.

## CHAPTER 1: INTRODUCTION

While software for the construction of such biological networks as well as for the biomarker discovery existed, they worked relatively slowly and inefficiently, due to the enormous volume of data involved in metagenomics. The current system also suffered from the inflexibility of being written in R and other language, and hence confined to the environment of that framework. My involvement in this project was to consider methods to speed-up the software, as well as to re-implement the software so that it was not constrained by the R framework and other platform.

Our target online platform was expected to be able to efficiently construct a network from metagenomics data automatically. The platform would incorporate recent advances which would also allow the automatic inference of the direction of the edges in the network. Users of the platform were expected to be researchers in fields such as ecology as well as the life sciences. The study of these biological networks was expected to assist the study of microbial community, including the human microbiota. In particular, we hoped that through a better understanding of the human microbiota, we could derive more hints on healthy living, or even bring forth better medicines. For more information about the online platform, please see Appendix A.

## CHAPTER 1: INTRODUCTION

### 1.2 Project Scope

At the end of the project, we delivered four software modules for the online platform, namely, two modules for network construction, one for biomarker discovery, and one for imputation. Together with the other modules, the platform provided researchers a complete software suite for analysing metagenomics data.

Our first and second module constructed biological networks from metagenomics data to model the interspecies interactions in its microbial community. This project rewrote the existing modules for the same purpose, which suffered from efficiency issues as well as being confined to the R framework and Matlab. We were expected to identify the bottleneck in its computation and rewrote the relevant codes to speed-up its process of network construction.

Besides, we delivered a module for inferring biomarkers. This project involved an enhanced algorithm for biomarker inference, as the current software was only applicable on gene expression data instead of our input of metagenomics data. This implied that a significant amount of rewrite was needed to develop the module.

The last module for this project was meant for the task of imputation that was the rescue or recovery of missing values. This module combined a few algorithms to recover the missing values in abundance matrix. The module made use of a case and control abundance matrix as input in order to evaluate the missing values.

## CHAPTER 1: INTRODUCTION

### 1.3 Project Objectives

Metagenomics study was currently difficult due to the lack of a good collection of software for its analysis. Many current softwares had different requirement for input, and typically lacked of documentation on how the software should work. This predicament was further exacerbated by the fact that many studies in metagenomics required a series of analyses which involved different tools from multiple parties.

The ultimate aim of this project was to provide a convenient online platform for the analysis of metagenomics data. The full platform was to facilitate an entire research workflow, from input processing to result interpretation. This final year project would be concerned with only four aspects of the platform. For more information about the platform, please refer to Appendix A.

An important task in metagenomics was the construction of biological regulatory network from metagenomics data (the first and second module in this project). Such networks allowed researchers in the field to infer the interaction and relationship of genes in microbial species.

The detection of biomarkers (the third module in this project) was another important task in metagenomics, since they would allow effective identification of the species within a sample.

The last module of this project was for recovering the missing values in an abundance matrix. This was achieved through imputation, the forth module of this project.

Codes for performing some of these modules may already exist in some form. However, these codes were unusable in the online platform due to programming language incompatibility, or they may suffer from performance issues. Their uses in the online platform may also be restricted due to the software licenses that they were released in. In order to make our modules available and open for all, this project re-developed them in an open source platform and on open source programming languages.



## CHAPTER 1: INTRODUCTION

### 1.4 Impact, significance and contribution

This project was a collaboration with the City University of Hong Kong, which would lead to the creation of a platform for researchers to carry out a series of analysis on metagenomics data. The platform strived to provide researchers to a convenient, all-in-one environment for metagenomics analyse. The platform provided researchers with the computing power required for their analyses, which would help them save cost.

First, constructing regulatory network on metagenomics data would help scientists interpret the interactions, or the relationships between the genes of the microbial species, in order to answer important scientific questions, such as: Does a specific gene influence another gene? Or, which genes have the highest influences on a specific gene? A typical research setup to answer such questions, would require the construction of a regulatory network for bacteria to study the interaction between bacteria. Through the network, scientists could figure out what was the bacteria that causes people to fall ill to certain diseases. The discoveries gleamed could then help medical practitioners devise medical procedures or cures for the disease.

If a specific interaction of microbial species is sufficiently significant, for example, if they precedes or describes a specific medical condition, researchers may want to find biomarkers. This would allow medical practitioners to identify early warning signs for the diseases and be able to treat the disease in its early stage.

Missing values recovery in the metagenomics data would help research to obtain more accurate and precise analysis outcome. Throughout the sampling data process, samples may be affected by various external factors and this may influence the data collected. Researchers were often at a lost in these situations since they may not have prior knowledge about what was going on. In the end, their analysis may ended up with inaccurate and imprecise output, causing them to miss important discoveries.

## CHAPTER 1: INTRODUCTION

### 1.5 Background information

Microbe (microscopic organism) is a very tiny organism that can only be seen under microscope. Microbe is a general term used to group several tiny organisms such as bacteria, viruses and so on (Genetic Science Learning Center 2014). When different types of microbes live and interact with each other in an environment, we call that microbial community.

Such communities are ubiquitous in nature, for example, in water, in soil, and most significantly, in the human body. The microbiota, a microbial community in the human body, has been found to contribute immensely to human health. This realization has spurred scientists to study the interaction and relationship among microbes in microbial communities. One way to study this is to construct a network structure based on metagenomics data collected from an environment sample.

Metagenomics is an emerging discipline which performs direct genetic analysis on the genomes in the microbial communities collected from the environment. Before emergence of metagenomics, microbes can only be studied through process of cultivation in laboratory (The Common Fund, n.d.). Due to presence of metagenomics, study of microbes can be easier than traditional approach. Metagenomics is a robust biological technique that does not need to go through the process of cultivation. By analysing metagenomics data, we may infer a biological network to examine the microbes' behaviour and interaction.

There has been many studies on various kind of networks for genes, diseases, symptoms, and so on. For instance, (Zyga 2014) constructed a diseases-symptoms network to study the relationship between symptoms and diseases. Based on the network, scientists were able to have a clearer picture of the interactions among them.

In order to retrieve more information from metagenomics data, biomarker can be inferred from the data. Biomarker is a biological sign or medical indicator for biological processes and state (Strimbu & Tavel 2010). For instance, biomarker can be used to predict risk of getting diseases, used to monitor and diagnose diseases and so on (Mayeux 2004).

## CHAPTER 2: LITERATURE REVIEWS

### 2.1 Reviewed Papers

#### 2.1.1 Reconstructing Directed Gene Regulatory Network by Only Gene Expression Data

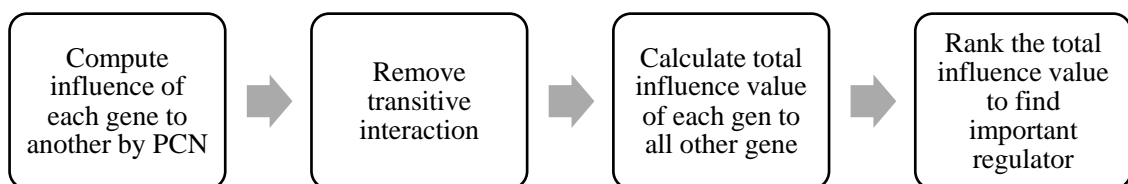
In this paper, the authors proposed a novel method, called as Context Based Dependency Network (CBDN) to infer the Gene Regulatory Network (GRN). This algorithm was proposed to construct the directed network with only gene expression data only, which mean it did not need any other additional information to infer GRN.

There are many methods were trying to compute the network by using correlation to infer the dependency of genes but those methods did not eliminate transitive interactions. However, there were few other methods were trying to fix the problem above but they did not consider of the edge direction.

In order to solve the problems, CBDN was introduced. Basically, this method could get rid of transitive interaction at the same time it could infer the edges' direction in the network. Besides, this method used only gene expression data, whereas other old methods were using gene expression data and other additional information. In this case, CBDN can be used to construct a directed network with minimal information. This approach also can be used to identify the important regulators.

In the other hand, this method may difficult to differentiate the interaction whether it is directed or transitive when the covariance is too large or too small, but it worked very well on medium covariance.

This methods had 3 steps to infer directed network. First of all, the influence value of each gene to another gene will be computed through Partial Correlation Network (PCN) then followed by removal of transitive interactions. Lastly, the total influence value of each gene will be calculated and ranked in order to get important regulators.



**Figure 2.1.1.1: Algorithm flow of CBDN**

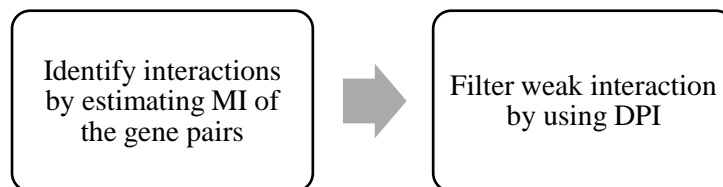
## CHAPTER 2: LITERATURE REVIEWS

### 2.1.2 ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context

In this paper, authors proposed a new algorithm that reconstruct Gene Regulatory Network (GRN) through Mutual Information (MI). The interaction between genes was identified by dependency value which could be calculated by using Mutual information approach. The method was known as Algorithm for the Reconstruction of Accurate Cellular Networks (ARACNE), it inferred transcriptional network by using information theoretic algorithm.

After that, transitive interactions would be included in the result and caused noises. In order to get rid of them, authors suggested to use Data Process Inequality (DPI) to filter the result. Basically those weak interaction gene pairs would be eliminated from the result.

The proposed method was estimating MI ranking by using Gaussian Kernel estimator instead of absolute MI because absolute MI was high sensitive to selection of Gaussian Kernel Width. DPI filtered the interactions by using appropriate MI threshold which calculated for a specific p-value in null hypothesis.



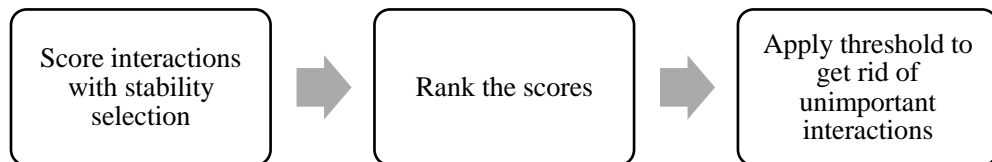
**Figure 2.1.2.1: Algorithm flow of ARACNE**

### 2.1.3 TIGRESS: Trustful Inference of Gene REGulation using Stability Selection

This paper, the proposed method was a feature selection method that using least angle regression (LARS) combined with stability selection to select target genes (TG) for transcription factor (TF). The method known as TIGRESS, treated gene regulatory network (GRN) inference as feature selection problems.

First, they considered GRN inference as a problem and divided the problem into sub-problems as many as TGs. Each sub-problem was taking a TG to find its TF. In order to find TF, each TG was calculated its score to all elements in the subset of TF with stability selection on top of LARS. Stability selection is a procedure that run a feature selection method multiple times on random data and evaluate the score of each feature. Using stability selection on top of LARS instead of using LARS directly is because using LARS is very sensitive and unstable when different variables had a very high correlations.

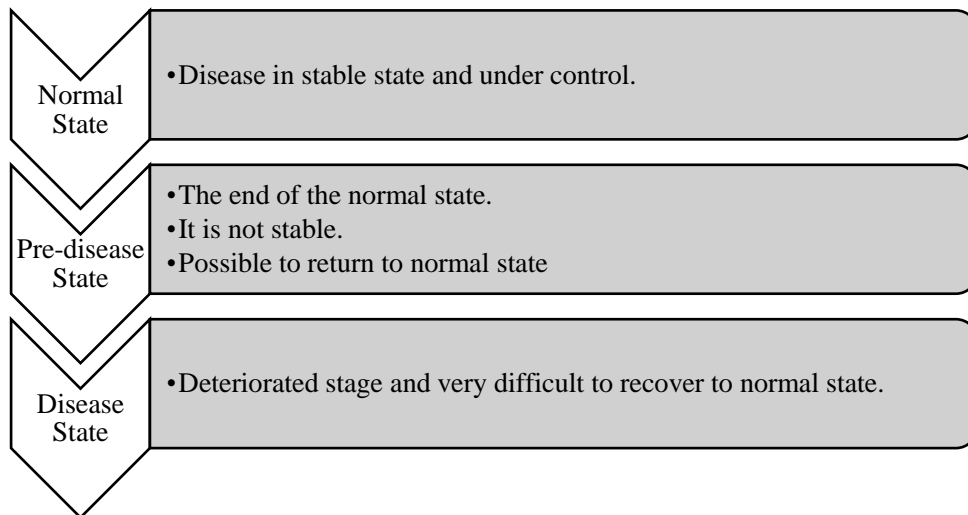
Once all the scores are computed, the scores would be ranked and applied threshold on them to get rid of unimportant interactions.



**Figure 2.1.3.1: Algorithm flow of TIGRESS**

### 2.1.4 Identifying critical transitions and their leading biomolecular networks in complex diseases

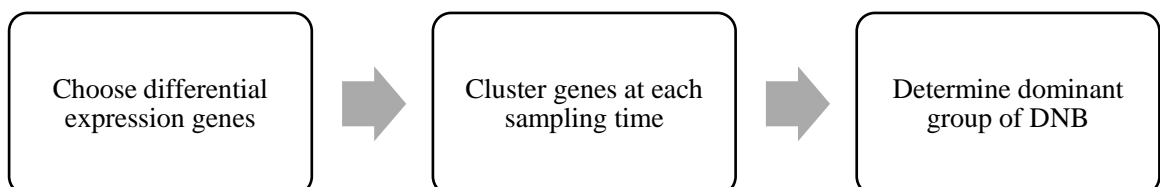
In disease progression, there are 3 states to categorize the disease state which are normal state, pre-disease state and disease state. **Figure 2.1.4.1** below explained each stage in disease progression.



**Figure 2.1.4.1: Disease progression**

This paper's aim was to detect pre-disease state to prevent deterioration of the disease. However, it was very difficult to detect the state because it had a very little change compare to normal state. Therefore, it was easy not to notice the changes of the disease.

This paper proposed a method to detect pre-disease state dynamically, called as Dynamical Network Biomarker (DNB). It acted like an early warning indicator to give out a sign when the disease is in pre-disease state. **Figure 2.1.4.2** is the algorithm flow of the paper.



**Figure 2.1.4.2: Algorithm flow of DNB**

## CHAPTER 2: LITERATURE REVIEWS

### 2.2 Critical Remarks

ARANCE is a method that compute dependency between transcription factor and target gene through Mutual Information (MI) from information theory. MI actually is a measurement of mutual dependency of triplet that use to determine how much the information held by one gene to another gene in the triplet. ARANCE uses this approach to evaluate the dependency of genes in gene expression data. Due to the used of MI, this method is suitable to use when the gene-gene relationships are nonlinear and irregular (Taylor, RC et al. 2008). MI does not make any assumption to the interaction between genes. In fact, MI can infer some interactions that may not find by Pearson's correlation. ARANCE method can get rid of transitive or indirect interactions by applying MI threshold. Although this method can get rid of unnecessary interactions but it cannot infer directionality of the directed interaction. In fact, it requires additional information which is not always available to us.

CBDN is a robust method that can infer a regulatory network from gene expression data only without any additional information. In order to infer a regulatory network, CBDN infer directionality of the edges by considering the influence value of the nodes and removing redundant interactions. CBDN uses Partial Correlation Network (PCN) to calculate dependency of each gene to another with the influence of third variables. However, the result will include transitive interactions. Therefore, a post-processing step is required to eliminate those interactions. Therefore, Directed Data Processing Inequality (DDPI) is introduced to remove transitive interactions. DDPI actually is modified from DPI from ARACNE method. With regulatory network, we can determine important regulators from the network without any additional information such as single nucleotide polymorphism data.

## CHAPTER 2: LITERATURE REVIEWS

A new scoring function is introduced by TIGRESS to evaluate the dependency between genes. It uses linear regression based approach which is Least Angle Regression (LARS) with Stability Selection. Because of using LARS, this method can apply on high dimensional data. This method will not include any redundant interactions. It is easier to separate indirect interactions from the network compared to CBDN and ARANCE. Due to sensitivity of LARS, Stability Selection is used on top of LARS to reduce its sensitivity toward the value of parameters. However, the value of the parameters will affect the performance of this method.

By comparing among this 3 algorithms, only CBDN can infer a causal and directed regulatory network with the minimal information needed. Normally these additional information is not that easy to collect and always not available to us. Inference of regulatory network could let researchers to retrieve more information from the network such as regulators compare to directed network only. CBDN and TIGRESS do assume the interactions between genes are linear and regular whereas ARANCE is not. ARANCE is a MI based approach, in fact any assumptions about the distribution of genes are not require (Taylor et al. 2008).

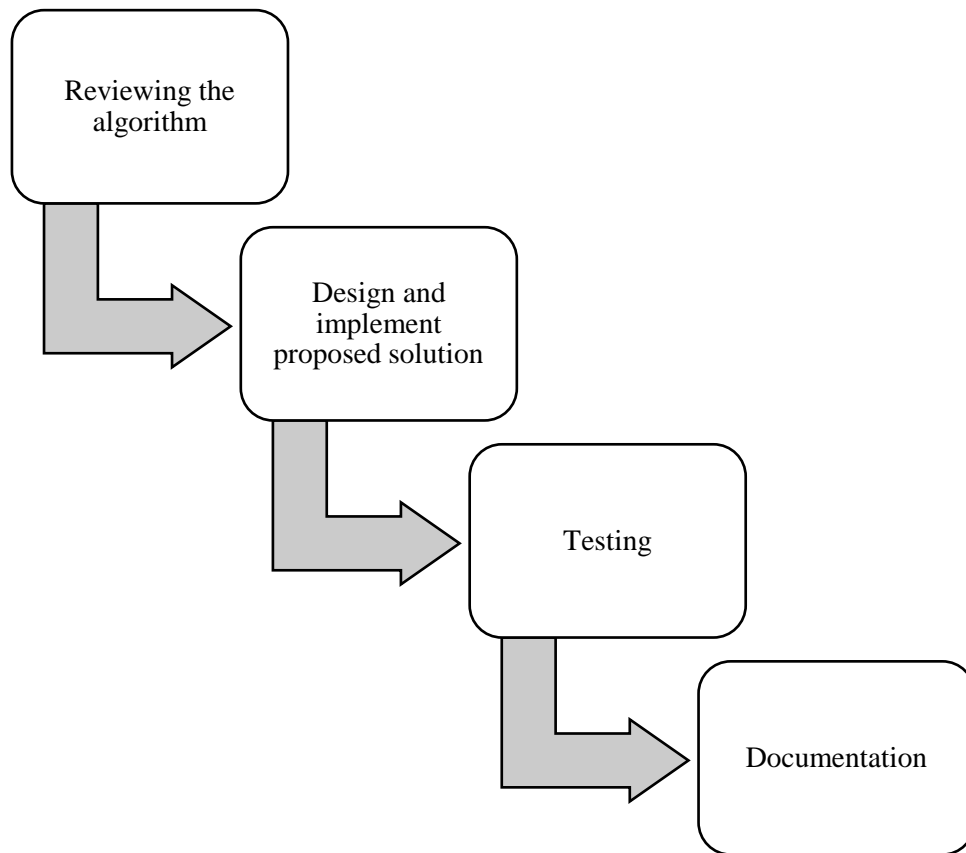
	<b>CBDN</b>	<b>ARANCE</b>	<b>TIGRESS</b>
Scoring Feature	Partial Correlation Network	Mutual Information	Least Angle Regression with Stability Selection
Transitive Inclusion	Yes	Yes	No
Removal Transitive Interaction	Yes	Yes	-
Edge Directionality	Yes	No	No
Required Additional Information	No	Yes	Yes

Table 2.2.1: Comparison of 3 algorithms



### 3.1 Design Specifications

#### 3.1.1 Methodologies and General Work Procedures



**Figure 3.1.1.1: Project methodology**

In this project, we used a simple waterfall approach to carry out the task. First, we would like to review the algorithm for constructing directed network and inferring biomarker. After reviewing the modules, we would like to implement the proposed solution on this four modules. After that, we would verify correctness of the outcome by testing on the dataset. However, our testing will be carried out on a server provided by City University of Hong Kong as their server has high processing power. As the data is huge and massive, basically it would take a few days or weeks to analyse the data. Once we successfully test out the correctness of the modules, we would proceed to documentation.

## CHAPTER 3: PROPOSED METHOD/ APPROACH

### 3.1.2 Technologies Used

1. Microsoft Visual Studio 2013 – Used to program modules.
2. C++ Language – Modules are written in C++ Language.
3. puTTY – Used to access the server provided by City University of Hong Kong.

### 3.1.3 System Performance Definition

1. Having a high processing speed.
2. Can produce accurate result.
3. Can be used in terminal, without any help of IDEs.

### 3.1.4 Verification Plan

All the metagenomics data would be retrieved from [EBI Metagenomics](#). The modules would use small data set as input to verify the correctness. So that time taken to complete the process would not be too lengthy. Once the output was accurate and correct, we would like to use a complete dataset to carry out modules verification.

### 3.2 System Design

#### 3.2.1 Constructing Directed Network Module by Using CBDN

In this module, Context Based Dependency Network (CBDN) which was introduced by (Zhang, Ng & Li 2015) was used to construct the metagenomics network. The proposed solution had 3 stages. The first stage determined edge direction by computing the influence value of each gene. When comparing influence value between two genes, the largest influence value would be selected as the parent of another gene. In order to calculate the influence value of a node  $X_j$  which lied between  $X_i$  to  $X_k$ , the differences between correlation and partial correlation was computed based on the equation below:

$$d(X_i, X_k | X_j) = Corr(X_i, X_k) - PC(X_i, X_k | X_j)$$

Before that, we should calculate the partial correlation by using the equation provided.

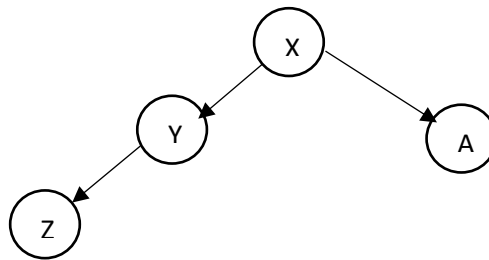
$$PC(X_i, X_k | X_j) = \frac{Corr(X_i, X_k) - Corr(X_i, X_j)Corr(X_k, X_j)}{\sqrt{[1 - Corr(X_i, X_j)^2][1 - Corr(X_k, X_j)^2]}}$$

Then the influence of  $X_j$  to  $X_i$  was the average of difference between correlations between partial correlations.

$$D(X_j \rightarrow X_i) = \frac{1}{n-1} \sum_{k \neq j}^{n-1} |d(X_i, X_k | X_j)|$$

## CHAPTER 3: PROPOSED METHOD/ APPROACH

After Step 1, we had successfully identified which nodes were the parents of another nodes, but there was no stated how many intermediaries were there, there may be one node in between the nodes. For example, suppose  $X$  is parent of  $Z$ , but there is an intermediary  $Y$  in between. In this case, we would like to know whether  $X$  directly influences  $Z$  or not. The figure below shown an example of this kind relationship. A properly constructed network should have this kind of transitive relationships removed. This was performed in the second step.



**Figure 3.2.1.1: Directed network**

In the second step, directed data processing inequality (DDPI) was used to remove the transitive relationships from the network. In order to remove transitive interaction, we found out the differences between directed interaction and transitive interaction, and determined whether the difference was larger than the threshold.

For the last step, each of the gene calculated their total influence value and ranked among themselves. The genes which had higher ranking were important regulators of the network.

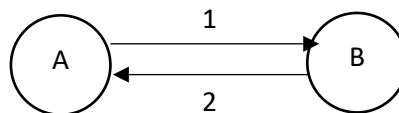
### CHAPTER 3: PROPOSED METHOD/ APPROACH

For a better understanding on how the module was developed, the figure below was an example of an output that had gone through step 1, that was, the calculation of influence values. Assuming that there were five genes, a 5 x 5 matrix was created to keep the influence values. Each row and column represented gene whereas the cell indicated the influence value. The influence value of diagonal cells all would be set to zero because CBDN assumed that a gene cannot influence itself.

	A	B	C	D	E
A	0	1	4	7	4
B	2	0	5	6	3
C	7	2	0	9	3
D	6	8	4	0	1
E	8	7	2	5	0

**Figure 3.2.1.2: Influence values matrix**

CBDN also assumed that there was no two-gene cyclic interaction. Therefore, we should remove the interaction that has the smallest value. For instance, in the case where  $A \rightarrow B$  and  $B \rightarrow A$ , with influence values 1 and 2 respectively, we would remove  $A \rightarrow B$  as it had the smallest value, 1.



**Figure 3.2.1.3: Two-gene cyclic interaction**

### CHAPTER 3: PROPOSED METHOD/ APPROACH

	A	B	C	D	E
A	0	0	0	7	0
B	2	0	5	0	0
C	7	0	0	9	3
D	0	8	0	0	0
E	8	7	0	5	0

**Figure 3.2.1.4: Influence values matrix – after removing two-gene cyclic interaction**

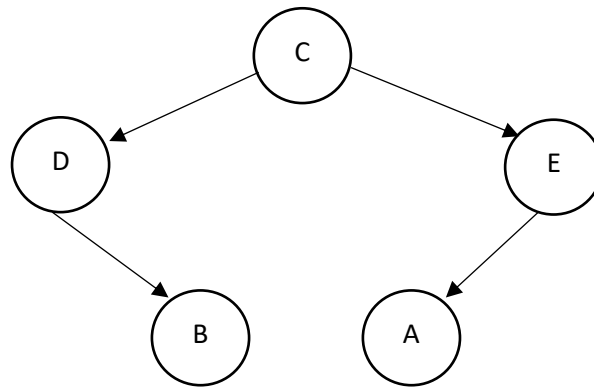
The matrix obtained provides information of parent-children relationship but it did not show any directed interaction, for example, gene B was the parent of gene A and gene C. In order to obtain directionality of genes, each gene could have only one incoming edge but multiple outgoing edges. Therefore, only the largest value in each column was retained whereas the rest were removed.

	A	B	C	D	E
A	0	0	0	0	0
B	0	0	5	0	0
C	0	0	0	9	3
D	0	8	0	0	0
E	8	0	0	0	0

**Figure 3.2.1.5: Influence values matrix – Final output**

### CHAPTER 3: PROPOSED METHOD/ APPROACH

The final matrix was then used as input to construct a network graph as shown in the Figure 3.2.1.6. Important regulators could be found by summing up the influence values of each gene and ranking them. Based on the final output, gene C was the most important regulator as it had the highest total influence value compared to other genes.



**Figure 3.2.1.6: Regulatory Network**

### 3.2.2 Biological Network Inference by Using TIGRESS

As an alternative to the CBDN, our online platform also provided the TIGRESS algorithm, proposed by Haury *et al.* (2012), to infer biological network. Instead of calculating mutual information or correlation between genes, TIGRESS used a feature selection method to score the interactions. The feature selection method used in this algorithm was the combination of least angle regression (LARS) and stability selection. The method ran the least angle regression for multiple times on random data and calculated the number of times each gene was selected.

In order to reduce the sensitivity of score when choosing number of times to run LARS, area under each curve was calculated instead of basic score. Below was the formula to calculate the area score:

$$s_{area}(t, g) = \frac{1}{L} \sum_{l=1}^L F(g, t, l)$$

The variable  $t$  was the element of the transcription factors (TFs) whereas  $g$  was the element of  $G$ , target genes. TIGRESS ran  $L$  LARS steps and  $F(g, t, l)$  was the frequency matrix from stability selection.

After scoring all the interactions, the interactions would be ranked in descending order (from highest score to lower score) and the interactions that were not significant (falling below a predefined cut-off value) would be removed.

Unlike the first module, this module required 2 input matrices which were gene expression data and transcription factors list, examples were shown in the figures below.



### CHAPTER 3: PROPOSED METHOD/ APPROACH

1	2	46	5	12
12	6	38	32	3
23	25	2	58	21
44	3	8	4	99
55	11	4	67	3
12	41	1	3	8

1	2	3
---	---	---

**Figure 3.2.2.1 Gene expression data (left) and transcription factors (right)**

Each row of the gene expression data represented an experiment or sample whereas each column represented a single gene. The transcription factors list kept the indices. There were five genes in the example above. The transcription factors were subset of this five genes.

In order to score each gene, we looped through every single gene (column). If the target gene (TG) was a transcription factor, the particular transcription factor (TF) would be removed from the transcription factor list. The target gene and transcription factors expression values would be abstracted from the expression data to perform stability selection.

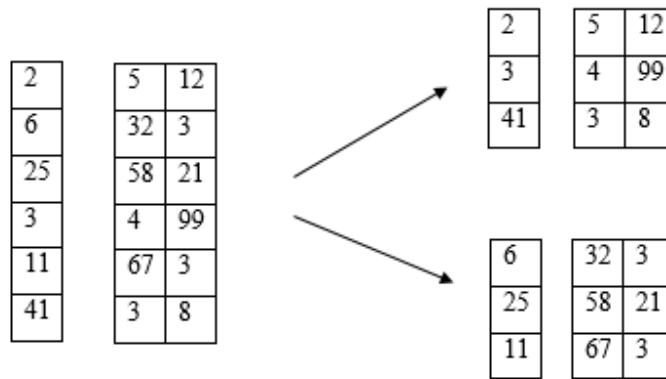
2
6
25
3
11
41

5	12
32	3
58	21
4	99
67	3
3	8

**Figure 3.2.2.2 Target gene expression values (left) and transcription factors expression values (right)**

### CHAPTER 3: PROPOSED METHOD/ APPROACH

The module would perform stability selection once the expression values of TG and TFs were ready. First, the module performed  $R$  times LARS on the data and each time the expression values would be randomly reweighted based on the alpha value. After that, the module would randomly separate the data into two equal or approximately equal size to run  $L$  LARS steps.



**Figure 3.2.2.3 Separation of data**

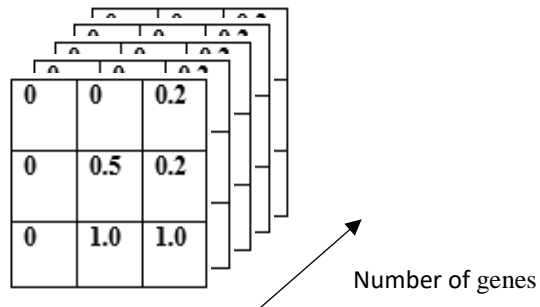
After running  $R$  times LARS, we would have a frequency matrix that kept the number of times each TF is selected for in each  $L$  steps of LARS. For the situation above, assuming that  $L=3$  and  $R=500$ , then we will have a 3 x 3 matrix. Each row represented  $L$  steps of LARS and each column represented the TFs. The column for the removed TF would be filled with zero to indicate that they were selected zero time in each  $L$  steps LARS.

0	0	100
0	250	100
0	500	500

**Figure 3.2.2.4 Frequency matrix for a gene**

### CHAPTER 3: PROPOSED METHOD/ APPROACH

The frequency matrix above was only for a gene. The same frequency matrix was repeated for every genes. There were five genes in the given initial example, which would give rise to a 3 x 3 x 5 cube. At the end of the computation, a cube was obtained which stored the score for each gene-gene interaction.



**Figure 3.2.2.4 Frequency cube**

### 3.2.3 Inferring Biomarker Module

In this module, we would like to use the algorithm proposed by (Liu *et al.* 2012) which was Dynamic Network Biomarker (DNB) to infer biomarker. In order to infer biomarker, our dataset must in the time series format which meant at each of the sampling point data would be collected.

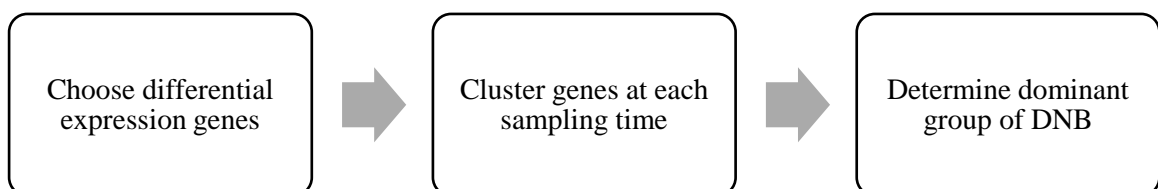
At each sampling point, we would analysed the data collected. At very first step, we would like to cluster the variables which is genes based on their correlation. Then, the following process was carried out on cluster-by-cluster bias. Each cluster we would like to compute their average Pearson's correlation coefficients (PCCs), average PCCs value between candidate group and others and average standard deviation value of the elements in the group. These values would be compared with previous sampling point. However, if they fulfilled all 3 criteria below then the particular cluster would be considered as biomarker.

Criteria for DNB:

1. PCCs value drastically increase in absolute value.
2. Average PCCs value of the cluster and any other cluster drastically decrease in absolute value.
3. Average standard deviation of the elements in the group drastically increase.

When all this criteria were put in together then the equation below was formed and we called it as composite index,  $I$ .

$$I =: \frac{SD_{d \times} \times PCC_d}{PCC_o}$$



**Figure 3.2.3.1: Algorithm flow of DNB**

### CHAPTER 3: PROPOSED METHOD/ APPROACH

In this module, two types of abundance matrices, namely, case abundance matrix and control abundance matrix, were needed. The case abundance matrix referred to the dataset that was collected from unhealthy patients whereas the control abundance matrix referred to the dataset collected from normal, healthy patients. Both matrices would be classified into time series format.

For the first step, at each sampling point we would choose the genes that show significant changes between case and control group by using student-t test with a critical value of 0.05. False Discovery Rate (FDR) and 2-fold change would be applied to increase the accuracy. For each sampling point, we would have a list of differential expression genes.

After that, the genes in each sampling point were clustered. In this module, a hierarchical clustering method with correlation as distance was used for the clustering. An  $N \times N$  correlation matrix was computed, where  $N$  was the number of differential expression genes in a sampling point. The matrix was pre-processed prior to the clustering. The upper triangle of the matrix and diagonal cells would be set to null as shown in the figure below. A user-controlled cut-off value was defined in order to stop the clustering when a certain number of clusters was obtained.

0	0.01	0.67	0.46	0.36
0.01	0	0.54	0.42	0.11
0.67	0.54	0	0.32	0.43
0.46	0.42	0.32	0	0.85
0.36	0.11	0.43	0.85	0

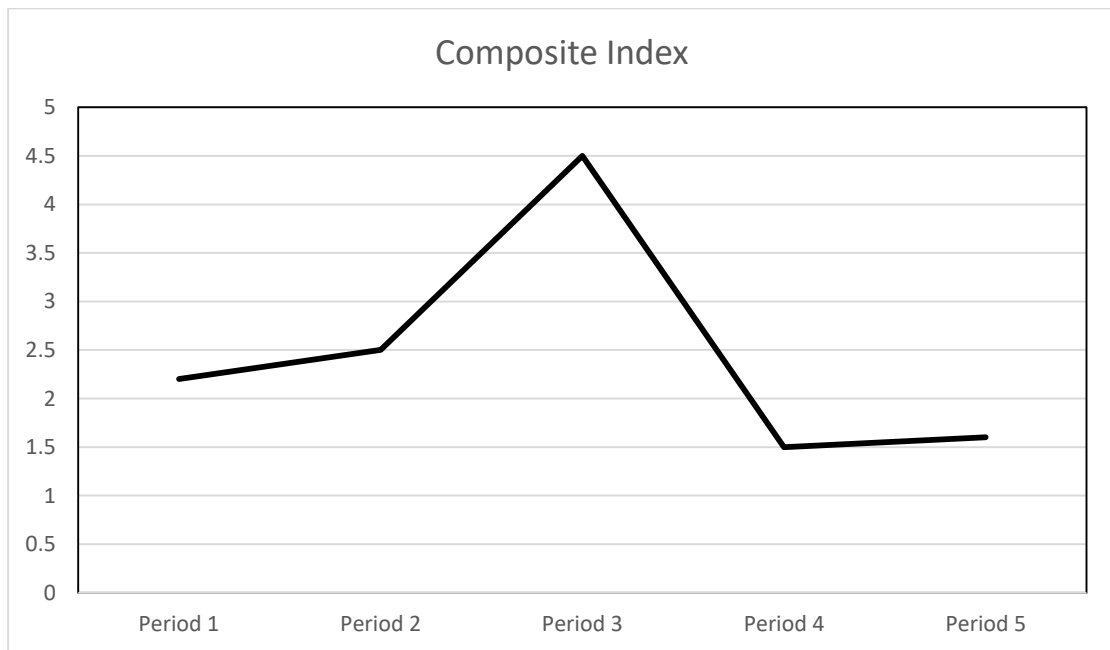
Null	Null	Null	Null	Null
0.01	Null	Null	Null	Null
0.67	0.54	Null	Null	Null
0.46	0.42	0.32	Null	Null
0.36	0.11	0.43	0.85	Null

**Figure 3.2.3.1: Correlation matrix before and after process**

### CHAPTER 3: PROPOSED METHOD/ APPROACH

All the clusters found throughout the entire timeline were used as candidates for the biomarkers. To find the biomarkers, an index was computed for each candidate group. If the candidate group fulfilled the criteria mentioned in previous part, the particular group would be the dominant group and the sampling point or time period was in pre-disease state.

For each candidate group, we would calculate its indices throughout the entire timeline and compare each of them. If the indices at particular period was higher than previous period and coming period, the particular period was what we are looking for and the candidate group would be our dominant group. The figure below showing the composite index found for a candidate group. The particular candidate group was potential dominant group because it had significant change at period 3 as the composite index of period 3 was higher than period 2 and period 4.



**Figure 3.2.3.2: Composite index graph of a candidate group**

### 3.2.4 Imputation

In this module, we integrated several algorithms to recover the missing values. Two abundance matrices were required as input for the module which were case abundance matrix and control abundance matrix, the input format which was exactly same as module 3 (Inferring Biomarker). We adopted the steps in PEP which was introduced by Goh *et al.* (2011). First, a network would be constructed on control abundance matrix. In order to construct the biological network, we integrated module 1 (CBDN) into this module to infer a network.

Next, differentially expressed bacteria would be detected and considered as seeds. Each of the seed indicated that there were missing values in the samples. In order to find differentially expressed bacteria, we implemented the first step in module 3 (DBN). Therefore, we would like to find the bacteria that had significant changes between case and control group by using student-t test with critical value of 0.05.

Once we found the seeds, each of the seed and their neighbour bacteria in the network would be clustered together. After expanding to their first degree neighbours, the clusters identified may overlapped with other clusters. Therefore, overlapping clusters were identified by using CPM which was introduced by Palla *et al.* (2005).

After obtaining overlapping clusters or communities, we calculated the seeds' missing value based on their community. Before calculating imputation value, we turned the network constructed into  $N \times N$  adjacency matrix,  $W$  where  $w_{ij} = 1$  if there was a link between bacterium  $i$  and bacterium  $j$ , else  $w_{ij} = 0$ .

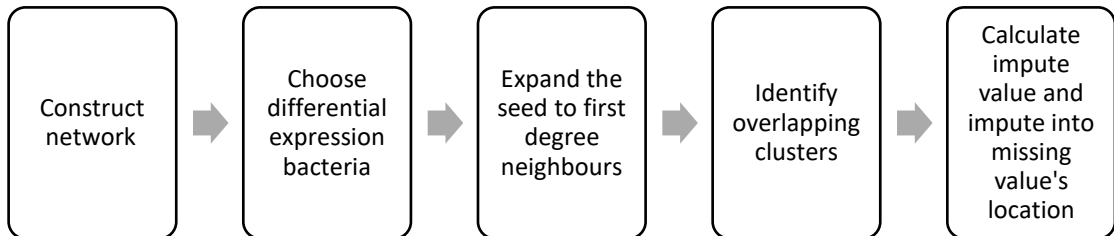
$$X_{ij} = \sum X_{kj}w_{kj} + c_{ij}$$

$X_{ij}$  was the expression value of seed  $i$  in sample  $j$  whereas  $w_{kj}$  was the interaction between bacterium  $k$  and bacterium  $j$ . In the equation above,  $c_{ij}$  was the constant value of bacterium  $i$  in the sample  $j$ . However, we had no prior information about this constant value. Therefore, we had to compute the constant value for each of the sample except the sample that had missing value, then average  $c$  would be calculated.

### CHAPTER 3: PROPOSED METHOD/ APPROACH

$$X_{ij} = \sum X_{kj}w_{kj} + \bar{c}$$

At the end, imputation value could be calculated for the samples which had missing value for bacterium *i*.



**Figure 3.2.4.1: Imputation process**

Now, we used the biological network matrix which I had illustrated in **3.2.1 Constructing Directed Network Module by Using CBDN** for explaining this module. The figure below was the network constructed on control abundance matrix.

	A	B	C	D	E
A	0	0	0	0	0
B	0	0	5	0	0
C	0	0	0	9	3
D	0	8	0	0	0
E	8	0	0	0	0

**Figure 3.2.4.1: Network constructed on control group**



### CHAPTER 3: PROPOSED METHOD/ APPROACH

Let say we found that bacterium  $C$  actually had missing values in its case abundance matrix. Then, we clustered it with its first degree neighbours which were bacterium  $B$ ,  $D$  and  $E$ . As there was only cluster we had, definitely there was no overlapping cluster.

In order to find average constant value for those samples which had missing value on bacterium  $C$ , we computed the average constant value by sum up all the constant values for the samples which are not missing value's sample and divide among themselves.

$$X_{Cj} = X_{Bj}w_{Bj} + X_{Dj}w_{Dj} + X_{Ej}w_{Ej} + \bar{c}$$

































By using the equation above, imputation value for bacterium  $C$  in sample  $j$  can be calculated.

### 3.3 Implementation Issues and Challenges

First of all, we would like to parallelize the module's analysis part. That was, our module should be able to distribute the tasks to the all processors. Our initial plan was to use Parallel Pattern Library (PPL), provided by Microsoft, to parallelize the modules. As our server was running in Linux OS, we had to use GNU Compiler Collection (GCC) to compile our module. Regrettably, we found that GCC does not provide this library yet. Therefore, we were forced to change to another library that provide similar feature which was Thread Building Block (TBB by Intel). In order to use TBB, our server must use Intel's processors. Luckily, this was indeed the case.

## CHAPTER 3: PROPOSED METHOD/ APPROACH

### 3.4 Timeline

		<b>Writing Constructing Network Module(CBDN)</b>	<b>41 days</b>	<b>Sat 10/1/16</b>	<b>Fri 11/25/16</b>
		Understand the algorithm for constructing network	7 days	Mon 10/3/16	Tue 10/11/16
		Select a programming language	2 days	Wed 10/5/16	Thu 10/6/16
		Write a simple workable program	7 days	Wed 10/12/16	Thu 10/20/16
		Test program	7 days	Fri 10/21/16	Mon 10/31/16
		Parallelise the program	5 days	Thu 11/10/16	Wed 11/16/16
		Test program	7 days	Thu 11/17/16	Fri 11/25/16
		Complete Constructing Network Module	0 days	Fri 11/25/16	Fri 11/25/16
		FYP 1 Documentation	14 days	Mon 10/24/16	Thu 11/10/16
		<b>Writing Inferring Biomarker Module</b>	<b>39 days</b>	<b>Mon 11/28/16</b>	<b>Thu 1/19/17</b>
		Understand the algorithm for inferring biomarker	7 days	Mon 11/28/16	Tue 12/6/16
		Modify the algorithm	7 days	Wed 12/7/16	Thu 12/15/16
		Write a simple workable program	7 days	Fri 12/16/16	Mon 12/26/16
		Test program	5 days	Tue 12/27/16	Mon 1/2/17
		Parallelise the program	5 days	Tue 1/3/17	Mon 1/9/17
		Test program	8 days	Tue 1/10/17	Thu 1/19/17
		Complete Inferring Biomarker Module	0 days	Thu 1/19/17	Thu 1/19/17
		<b>Writing Constructing Network Module(TIGRESS)</b>	<b>25 days</b>	<b>Fri 1/20/17</b>	<b>Thu 2/23/17</b>
		Understand the algorithm for constructing network	4 days	Fri 1/20/17	Wed 1/25/17
		Write a simple workable program	14 days	Thu 1/26/17	Tue 2/14/17
		Test program	7 days	Wed 2/15/17	Thu 2/23/17
		Complete Constructing Network Module	0 days	Thu 2/23/17	Thu 2/23/17
		<b>Writing Imputation Module</b>	<b>25 days</b>	<b>Fri 2/24/17</b>	<b>Thu 3/30/17</b>
		Understand the algorithm for constructing network	4 days	Fri 2/24/17	Wed 3/1/17
		Write a simple workable program	14 days	Thu 3/2/17	Tue 3/21/17
		Test program	7 days	Wed 3/22/17	Thu 3/30/17
		Complete Constructing Network Module	0 days	Thu 3/30/17	Thu 3/30/17
		FYP 2 Documentation	8 days	Thu 3/30/17	Mon 4/10/17

**Figure 3.4.1: Project timeline (Text)**

### CHAPTER 3: PROPOSED METHOD/ APPROACH

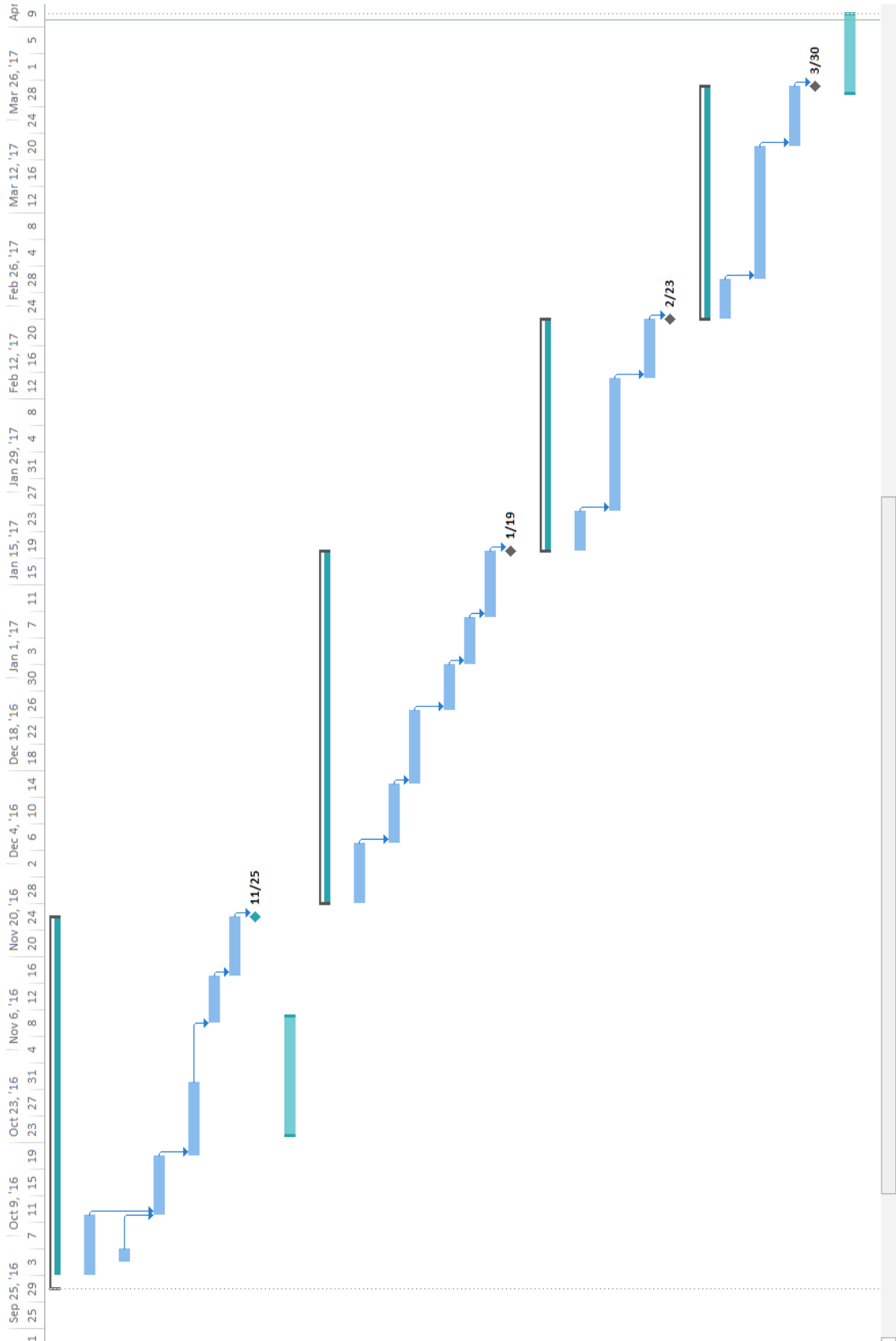
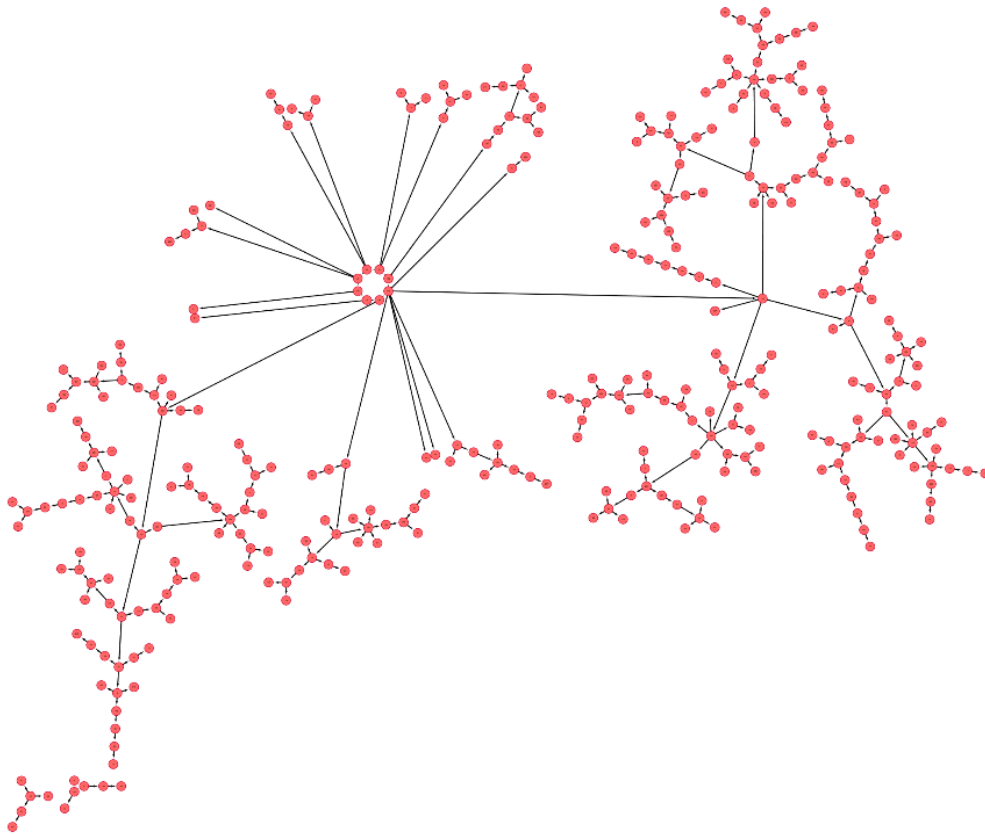


Figure 3.4.2: Project timeline (Diagram)

## CHAPTER 4: EXPERIMENTAL RESULT

### 4.1 Constructing Directed Network Module by Using CBDN

This module, we were using human gut microbiome which was from Li *et al.* (2014) to construct the network. This dataset was collected from different individuals which were Chinese, Danish, Spanish and American. The motive of the paper actually was to show the differences of gut microbial of different countries. However, we were using this dataset to construct the regulatory network of all microbes. This dataset contained 337 microbes and 1266 samples. The module would output 2 files which were interactions between microbes and ranking of microbes. In order to visualize the network, we turned our interactions result into Cytoscape and the figure below was the network diagram.



**Figure 4.1.1: Human Gut Microbiome Network**

## CHAPTER 4: EXPERIMENTAL RESULT

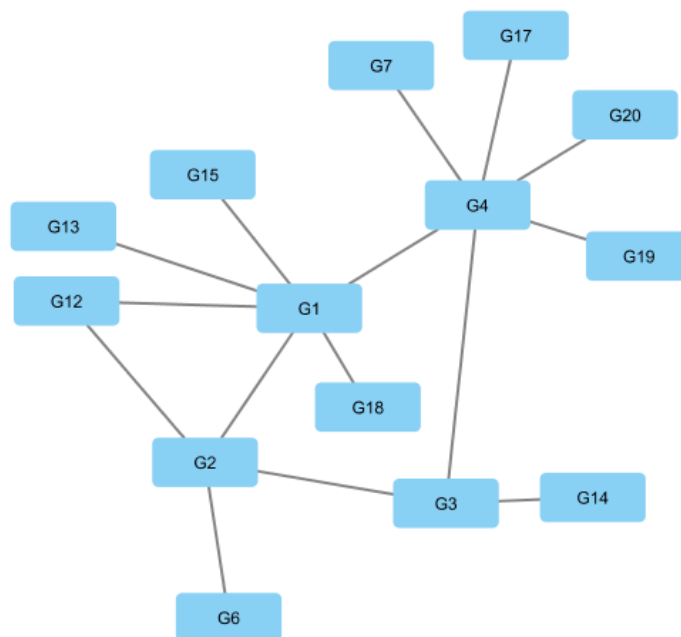
In this network, each of the node represented one kind of microbe and the edge represented the interaction between two microbes. The edges in the graph had their directionality which meant we could understand the microbes regulatory. From the Figure 4.1.1, three separated networks were found, and two of them actually were small networks with the size of 5 and 6, whereas the remaining microbes were in the large network. From the result, the microbe which had the highest total influence value was *Methylobacillus*. This bacterium actually regulated *Leptospira*, *Magnetococcus* and *Neisseria* in the community. The table below was the top 10 bacteria who had the highest total influence score among other bacteria.

<b>Bacteria Name</b>	<b>Total Influence Value</b>
<i>Methylobacillus</i>	0.160151
<i>Albidiferax</i>	0.148100
<i>Rubrivivax</i>	0.134855
<i>Nitrospira</i>	0.130282
<i>Listeria</i>	0.123234
<i>Desulfurivibrio</i>	0.115652
<i>Pediococcus</i>	0.111823
<i>Acetobacter</i>	0.109131
<i>Treponema</i>	0.104006
<i>Pelotomaculum</i>	0.099158

**Table 4.1.1: Top 10 of important regulators**

### 4.2 Biological Network Inference by Using TIGRESS

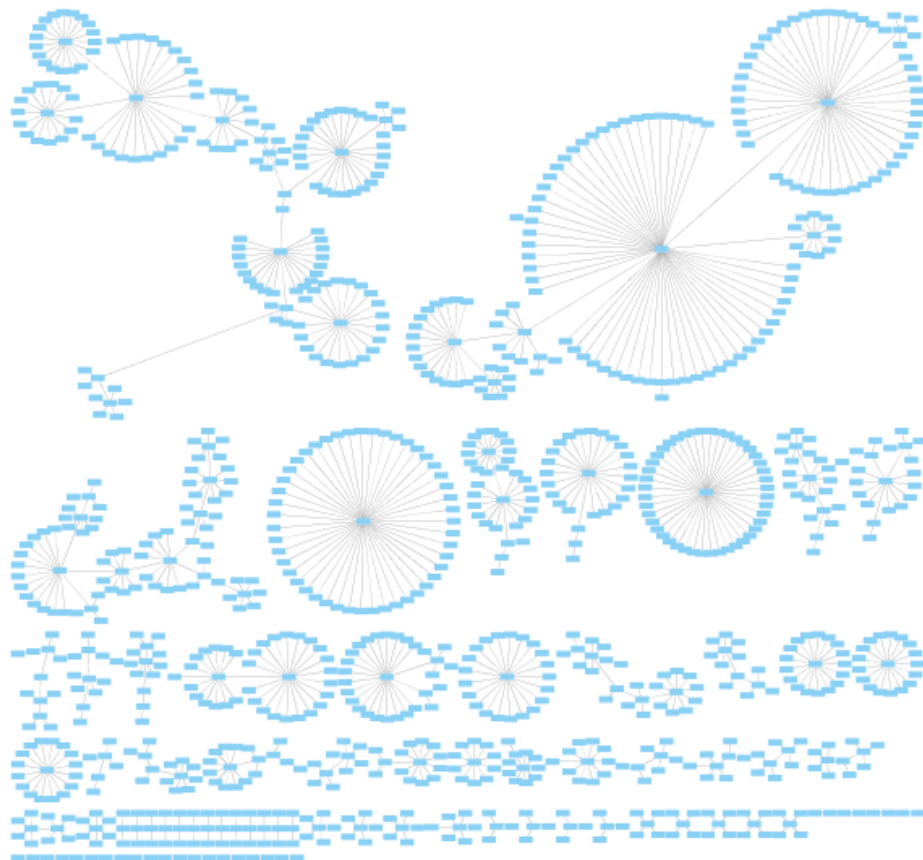
To test the module, we used the stimulation data from TIGRESS. The dataset obtained expression data of 20 genes and a list of transcriptional factors. The gene 1, 2, 3 and 4 were the transcriptional factors. In order not to output all the interactions, we were selected first 15 important interactions to construct the graph below. What make TIGRESS different from CBDN was the directionality of the edges. In TIGRESS, we did not know the directionality of the edges. In fact, the interaction between genes indicated the interaction only, but it did not tell us which gene was the parent node, and which gene was the child node.



**Figure 4.2.1 Biological network constructed by TIGRESS**

## CHAPTER 4: EXPERIMENTAL RESULT

We furthered our testing on *in silico* data from DREAM 5 challenge as TIGRESS took it as the benchmark dataset. In the dataset, there were 1643 genes and data were collected from 804 experiments. There were 195 transcriptional factors which were from gene 1 to gene 195. Among 320,190 interactions, we were selected top 1000 interactions and visualized the network below. The interesting fact was there were many sub-networks in the figure below and some of them even isolated from each other because we were selected only 0.3% of the edges. Besides, from the network below, we could determine which gene actually was important regulator.



**Figure 4.2.2 in silico data network constructed by TIGRESS**



## CHAPTER 4: EXPERIMENTAL RESULT

### 4.3 Inferring Biomarker Module

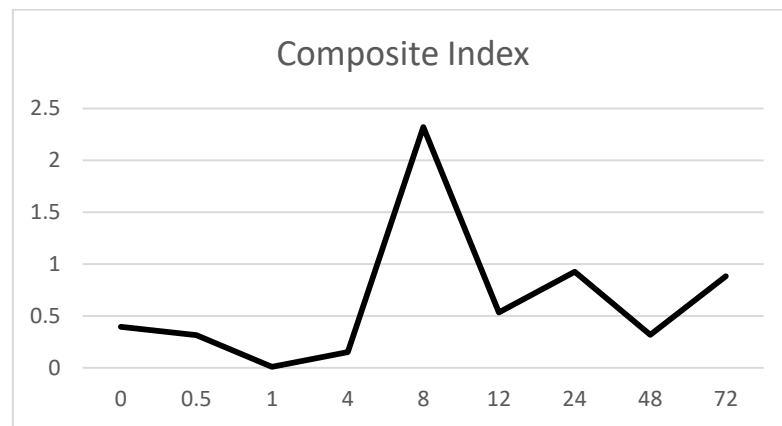
The dataset that we were using was genomics data about the lung injury with carbonyl chloride inhalation exposure which was being used by Liu *et al.* (2012). This dataset could be separated into two sets, one was case group and another one was control group. The dataset was collected by exposing mice to air (control) and phosgene (case). The dataset had 9 sampling points which were 0hr, 0.5hr, 1hr, 4hr, 8hr, 12hr, 24hr, 48hr and 72hr and each of them had 6 samples. In the dataset, there were 12871 genes. By using this module, we would like to detect early warning sign for acute lung injury.

After gone through FDR and 2-fold change, we obtained an array of number of differential expression genes throughout the entire timeline, [0, 29, 72, 195, 269, 173, 188, 176]. In order to find dominant group, we clustered the differential expression genes to maximum 40 clusters in each sampling points.

After clustering, each cluster in each sampling point would calculate their indices and checked with the criteria. In this case, we would have a list of number of potential dominant group throughout entire timeline, [0, 0, 0, 0, 1, 1, 1, 1, 0].

The first potential dominant group appeared in period 5 (8hr) and there were only 2 genes in the groups which were Ensmusg00000058905 and Tlr9. Figure below showing the composite index of the first potential dominant group that fulfilled all criteria. In the figure, we noticed the composite index increases sharply after 4hr and reached the peak on 8hr. In this case, the figure showed that pre-disease state was starting on 4hr.

## CHAPTER 4: EXPERIMENTAL RESULT



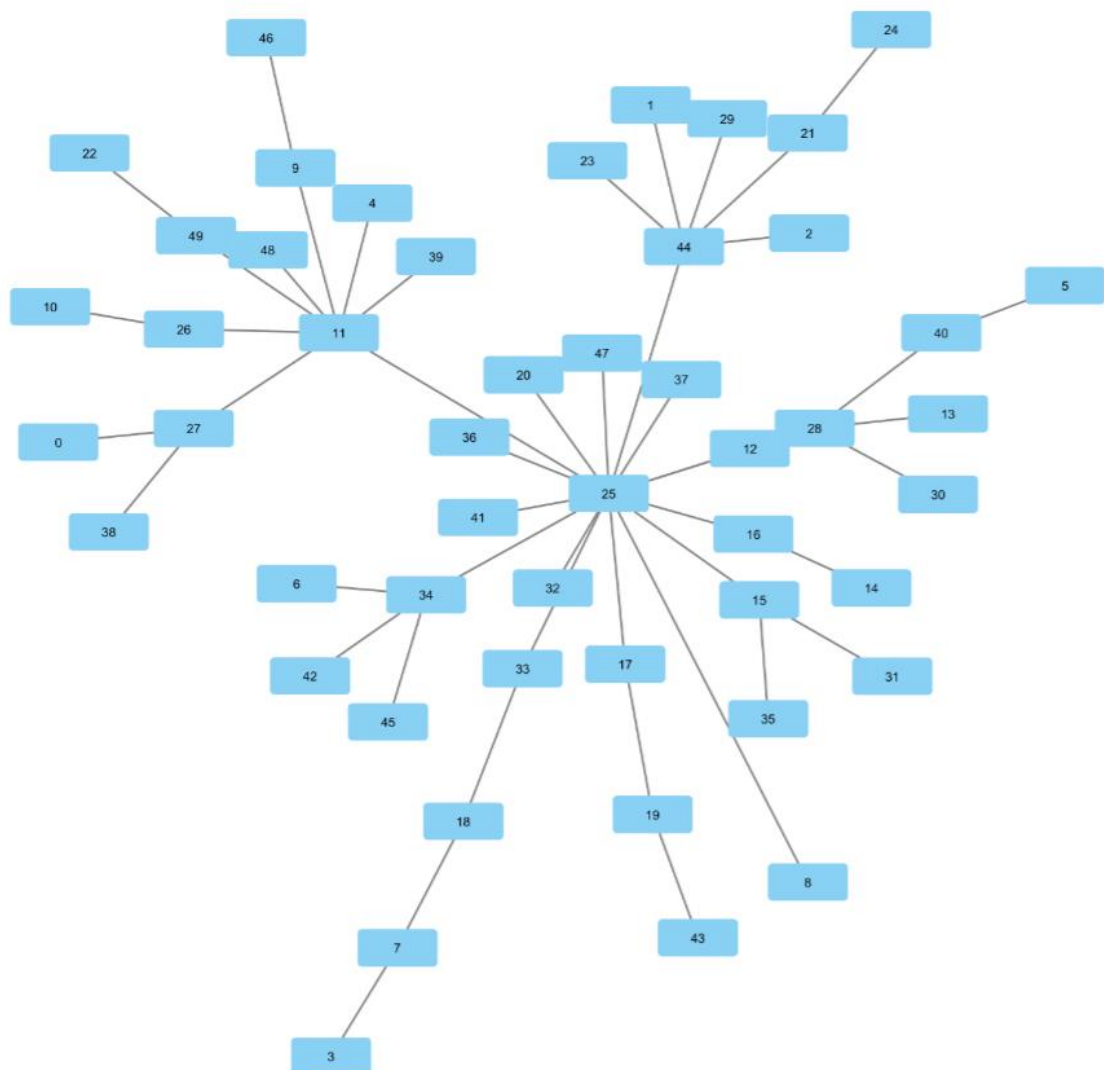
**Figure 4.3.1 Composite Index of the first potential dominant group**

	0	0.5	1	4	8	12	24	48	72
G 1	0.3959	0.3159	0.0105	0.1523	2.3230	0.5341	0.9283	0.3175	0.8850
G 2	0.1600	0.0735	0.0670	0.0600	0.1318	0.3386	0.1089	0.1122	0.1562
G 3	0.0659	0.1258	0.0707	0.0852	0.0672	0.1654	0.3889	0.0755	0.0946
G 4	6.0122 e-007	7.0229 e-007	9.5050 e-007	8.7669 e-007	1.3669 e-006	1.3700 e-006	1.1781 e-006	1.8187 e-006	1.3896 e-006

**Table 4.3.1 Composite Index of all potential dominant groups**

### 4.4 Imputation

To check whether the module worked fine or not, we used the dataset which was used in **4.3 Inferring Biomarker Module**. This was data about the lung injury with carbonyl chloride inhalation exposure. However, due to large volume of data, it was very lengthy to complete first step which was constructing biological network. Therefore, we selected first 50 genes to test the module and we made 6 expression values of gene 15 to be zero, which were considered as missing values.



**Figure 4.4.1 Biological network of the first 50 bacteria**

## CHAPTER 4: EXPERIMENTAL RESULT

First, a network had been constructed by using the control abundance data on first 50 genes (Figure 4.4.1). Among this 50 genes, we detected 2 seeds which were gene 15 and gene 33.

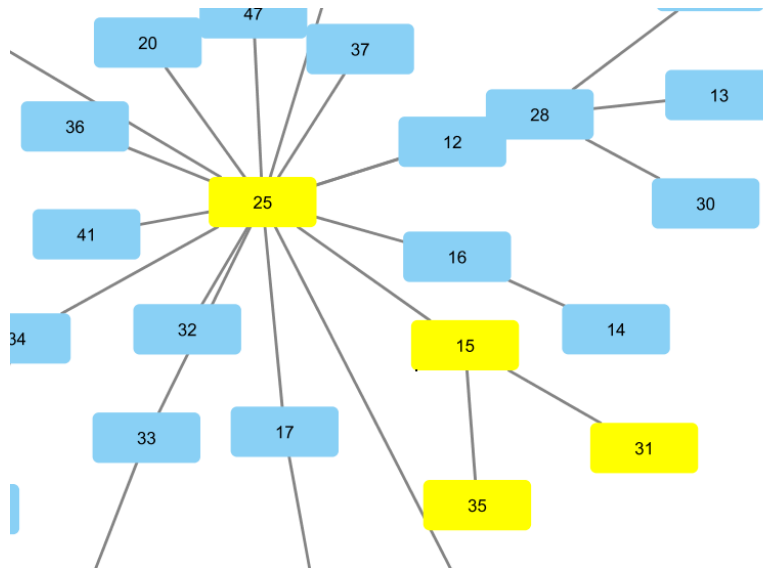


Figure 4.4.2 First degree neighbours of gene 15

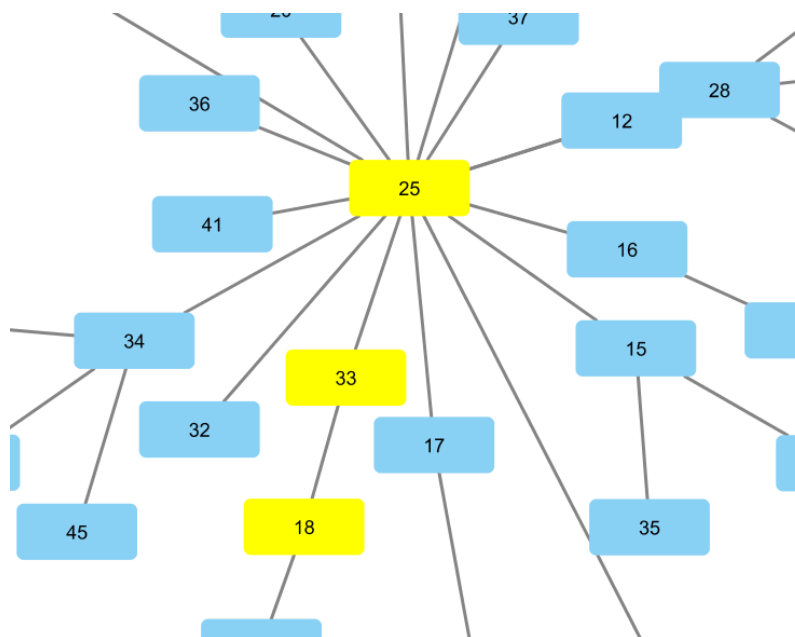


Figure 4.4.3 First degree neighbours of gene 33

## CHAPTER 4: EXPERIMENTAL RESULT

In fact, gene 33 had no missing values but it showed the significant changes between case and control group. Hence, this was not our imputation concern so far. After we found gene 15 as seed, we expanded the seed to its first degree neighbour genes to form a community which were shown in the Figure 4.4.2. The imputation values were shown in the table below. The initial average expression value for gene 15 was 1889.95 whereas the average expression value of the gene after imputation was 1889.81, different in 0.14.

	phosgene _0.5h_4A	phosgene _0.5h_5B	phosgene _0.5h_5A	phosgene _0.5h_6B	phosgene _0.5h_6A	phosgene _1h_10B
Case (Initial)	1845.85	1962.05	2164.8	1734.85	1858.45	1958.25
Case (Set as missing value)	0	0	0	0	0	0
Case (After)	1949.37	1970.41	1919.31	1888.44	1830.64	1958.41

**Table 4.4.1 Imputation values before and after**

## CHAPTER 5: CONCLUSION

Our ultimate aim was to develop a platform that provided a series of analysis services to researchers. The platform provided an all-in-one venue for such analyses. The aim of the platform was to allow researchers to perform all the standard analyses on the website, in an easy and convenient fashion.

This project involved the development of four modules of the platform, namely constructing directed network on metagenomics data by two different approaches, inferring its biomarkers and imputation. For all of the module, we were going to parallelize them so that they work effectively and efficiently. We used Thread Building Block (TBB) library to enable parallelism in the modules. The program would distribute the tasks to processors and fully utilize them.

For the constructing directed network module, we used CBDN method which was introduced by Zhang, Ng & Li (2015). First, influence value of each gene to others would be calculated then transitive relationship would be removed. After that, we would like to figure out the important regulator by ranking their total influence value. TIGRESS, another method that could be used to construct biological network, which was introduced by Haury *et al.* (2012). This method used feature selection approach to infer the relationships of each gene pair.

In the other hand, DNB was used to infer biomarker in this project. The method was proposed by Liu *et al.* (2015). First, differential expression genes would be selected and clustered at each sampling point then we would like to determine the criteria of each cluster at each sampling point. When the cluster fulfilled the criteria, then the particular cluster was considered as biomarker. The last module was imputation that was used to recover missing value in the expression data. Several algorithms were integrated to develop this module.

In order to speed up computational speed of this two modules, we implemented parallelism in our software. We used TBB library in the software as it provided sufficient functions to the software. The process would be distributed to all processors and fully utilized the resource available.

At the end of the project, we had achieved to complete 4 modules. However, the modules did not fully tested as there were limited datasets and processing power. Therefore, we forced to use genomics data for some test cases.

## REFERENCES

- Andrea Califano 2016, *Modeling Cell Regulatory Networks*. Available from:  
<<http://califano.c2b2.columbia.edu/modeling-cell-regulatory-networks>>. [19 November 2016].
- Efron, B, Hastie, T, Johnstone, I & Tibshirani, R 2004, 'Least Angle Regression', *The Annals of Statistics*, vol. 32, no. 2, pp. 407-499. [16 November 2016].
- EMBL-EBI, n.d., *Project: BGI Type 2 Diabetes study*. Available from:  
<<https://www.ebi.ac.uk/metagenomics/projects/SRP008047>> [31 March 2017].
- Genetic Science Learning Center 2014, *What are Microbes?*. Available from:  
<<http://learn.genetics.utah.edu/content/microbiome/intro/>>. [2 November 2016].
- Goh, W, Lee, Y, Zubaidah, R, Jin, J, Dong, D, Lin, Q, Chung, M & Wong, L 2011, 'Network-Based Pipeline for Analyzing MS Data: An Application toward Liver Cancer', *Journal of Proteome Research*, vol. 10, no. 5, pp. 2261-2272. [20 March 2017]
- Goh, W., Sergot, M., Sng, J. and Wong, L. 2013, 'Comparative Network-Based Recovery Analysis and Proteomic Profiling of Neurological Changes in Valproic Acid-Treated Mice', *Journal of Proteome Research*, vol. 12, no. 5, pp. 2116-2127. [1 March 2017].
- Haury, A, Mordelet, F, Vera-Licona, P & Vert, J 2012, 'TIGRESS: Trustful Inference of Gene REgulation using Stability Selection', *BMC Systems Biology*, vol. 6, no. 1. [20 October 2016].

## REFERENCES

- Li, J, Jia, H, Cai, X, Zhong, H, Feng, Q, Sunagawa, S, Arumugam, M, Kultima, J, Prifti, E, Nielsen, T, Juncker, A, Manichanh, C, Chen, B, Zhang, W, Levenez, F, Wang, J, Xu, X, Xiao, L, Liang, S, Zhang, D, Zhang, Z, Chen, W, Zhao, H, Al-Aama, J, Edris, S, Yang, H, Wang, J, Hansen, T, Nielsen, H, Brunak, S, Kristiansen, K, Guarner, F, Pedersen, O, Doré, J, Ehrlich, S, Pons, N, Le Chatelier, E, Batto, J, Kennedy, S, Haimet, F, Winogradski, Y, Pelletier, E, LePaslier, D, Artiguenave, F, Bruls, T, Weissenbach, J, Turner, K, Parkhill, J, Antolin, M, Casellas, F, Borrueal, N, Varela, E, Torrejon, A, Denariatz, G, Derrien, M, van Hylckama Vlieg, J, Viega, P, Oozeer, R, Knoll, J, Rescigno, M, Brechot, C, M'Rini, C, Mérieux, A, Yamada, T, Tims, S, Zoetendal, E, Kleerebezem, M, de Vos, W, Cultrone, A, Leclerc, M, Juste, C, Guedon, E, Delorme, C, Layec, S, Khaci, G, van de Guchte, M, Vandemeulebrouck, G, Jamet, A, Dervyn, R, Sanchez, N, Blottière, H, Maguin, E, Renault, P, Tap, J, Mende, D, Bork, P & Wang, J 2014, 'An integrated catalog of reference genes in the human gut microbiome', *Nature Biotechnology*, vol. 32, no. 8, pp. 834-841. [8 March 2017]
- Liu, R, Li, M, Liu, Z, Wu, J, Chen, L & Aihara, K 2012, 'Identifying critical transitions and their leading biomolecular network in complex diseases', *Scientific Report*, vol. 2.
- Margolin, A, Nemenman, I, Basso, K, Wiggins, C, Stolovitzky, G, Favera, R & Califano, A 2006, 'ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context', *BMC Bioinformatics*, vol. 7, no. 1, p. S7. [20 October 2016].
- Markowitz, F & Spang, R 2007, 'Inferring cellular networks – a review', Markowitz, F. and Spang, R. (2007). Inferring cellular networks – a review. *BMC Bioinformatics*, vol.8, no. 6, p. S5. [15 November 2016].



## REFERENCES

- Mayeux, R 2004, 'Biomarkers: Potential Uses and Limitations.', *NeuroRx*, vol. 1, no. 2, pp. 182-188. [3 November 2016].
- Members.cbio.mines-paristech.fr. n.d., *The TIGRESS page*. Available from: <<http://members.cbio.mines-paristech.fr/~ahaury/svn/dream5/html/index.html>>. [2 February 2017].
- Meta.genomics.cn. n.d., *Integrated reference catalog of the human gut microbiome*. Available from: <<http://meta.genomics.cn/meta/home>>. [1 April 2017].
- Morrison, JL, Breitling, R, Higham, DJ & Gilbert, DR 2005. 'GeneRank: Using search engine technology for the analysis of microarray experiments', *BMC Bioinformatics*, vol. 6, no. 1, p. 233. [25 February 2017]
- Palla, G, Derényi, I, Farkas, I & Vicsek, T 2005, 'Uncovering the overlapping community structure of complex networks in nature and society', *Nature*, vol. 435, no. 7043, pp. 814-818. [24 February 2017]
- Parallel Pattern Library*, n.d.. Available from: <<https://msdn.microsoft.com/en-us/library/dd492418.aspx>>. [23 October 2016].
- Taylor, RC, Acquah-Mensah, G, Singhal, M, Malhotra, D & Biswal, S 2008, 'Network Inference Algorithms Elucidate Nrf2 Regulation of Mouse Lung Oxidative Stress', *PLoS Comput Biol*, vol. 4, no. 8. [19 November 2016].
- The Common Fund n.d., *Human Microbiome Project*. Available from: <<https://commonfund.nih.gov/hmp/overview>>. [3 November 2016].

## REFERENCES

*Threading Building Blocks*, n.d.. Available from:

<<https://www.threadingbuildingblocks.org>>. [25 October. 2016].

Strimbu, K & Tavel, J 2010, 'What are biomarkers?', *Current Opinion in HIV and AIDS*, vol. 5, no. 6, pp. 463-466. [3 November 2016].

Wiki.c2b2.columbia.edu 2015, *ARACNe - Workbench*. Available from:

<<http://wiki.c2b2.columbia.edu/workbench/index.php/ARACNe>>. [19 November 2016].

Zhang, L, Ng, Y & Li, S 2015, 'Reconstructing directed gene regulatory network by only gene expression data', *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 163-170.

Zyga, L 2014, 'Diseases, symptoms, genes, and proteins linked together in giant network', *Medical Xpress* 18 July. Available from <<http://medicalxpress.com/news/2014-07-diseases-symptoms-genes-proteins-linked.html>>. [3 November 2016].

## APPENDICES

### Appendix A Online Platform Introduction

As our main objective was to collaborate with City University of Hong Kong to come out a comprehensive online platform. However, developing online platform was not the concern in this project, we were just in charge in four modules. <http://dl380a.cs.cityu.edu.hk/> is the link to the online platform.

RNA Data Analysis for Cancer Analysis Visualization Gene DB Publications Wiki About Us Login Register

### What We Do

An Integrated Cloud-Based System For Disease Gene Regulation Analysis

The genome of an organism consists of many genes which work in unison to orchestrate the organism's biological functions. Today, the entire genome of an organism can be deciphered with relative ease using next generation sequence (NGS) techniques, or single-nucleotide polymorphism (SNP) detection techniques. The information so obtained grants us great opportunities to glean into the workings of an organism's genes, allowing us to examine the developments, diseases, health, etc, of the organism. However, genes express differently with spatial and temporal changes, being regulated by a myriad of complicated processes. As a crucial step to their understanding, we must know how genes are regulated. An understanding of the genes' regulatory mechanisms will bridge the gap from genotype to phenotype, and enlighten us with insights on the synthesizing effects of different elements in cells

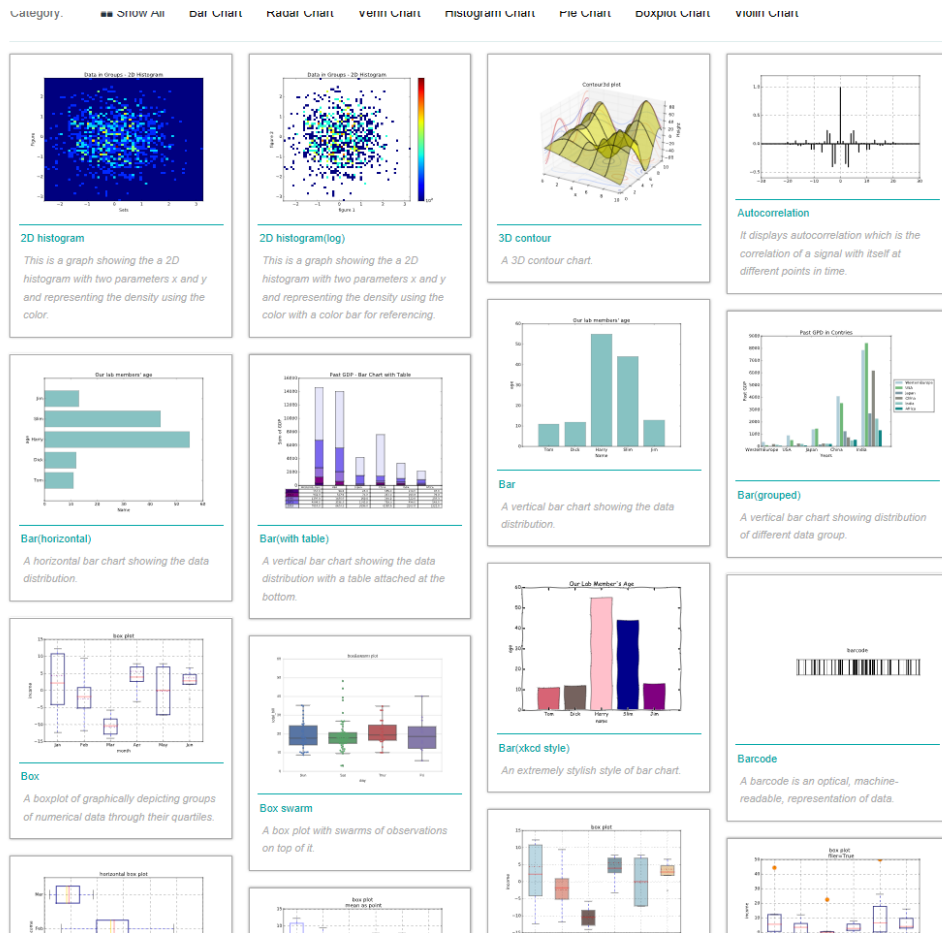
### What Do We Offer

- Data Collection**  
We collected raw data from existing databases and literature including the RNA-seq data and microarray data.
- Analysis Tools**  
We provided many commonly used analysis tools to reduce the work load for module development.
- Over 30 Modules**  
Over thirty modules have been developed for data analysis. Developer could develop and publish new modules in this platform.
- Omics Language**  
Faster data analysis and visualization. We implemented some of the popular visualizations, to enable users to visualize their data directly from the tool.

**Figure A.1 Main page of the online platform**

The online platform was a cloud-based system that provided free services to gene regulation analysis. Besides, they had their own gene database which had been processed from raw data and they offered over 30 modules to users to analyse the dataset. Other than that, users may visualize their result after analysed the data in over 70 ways such as scatter plots, histograms, pie charts, and so on.

# APPENDICES

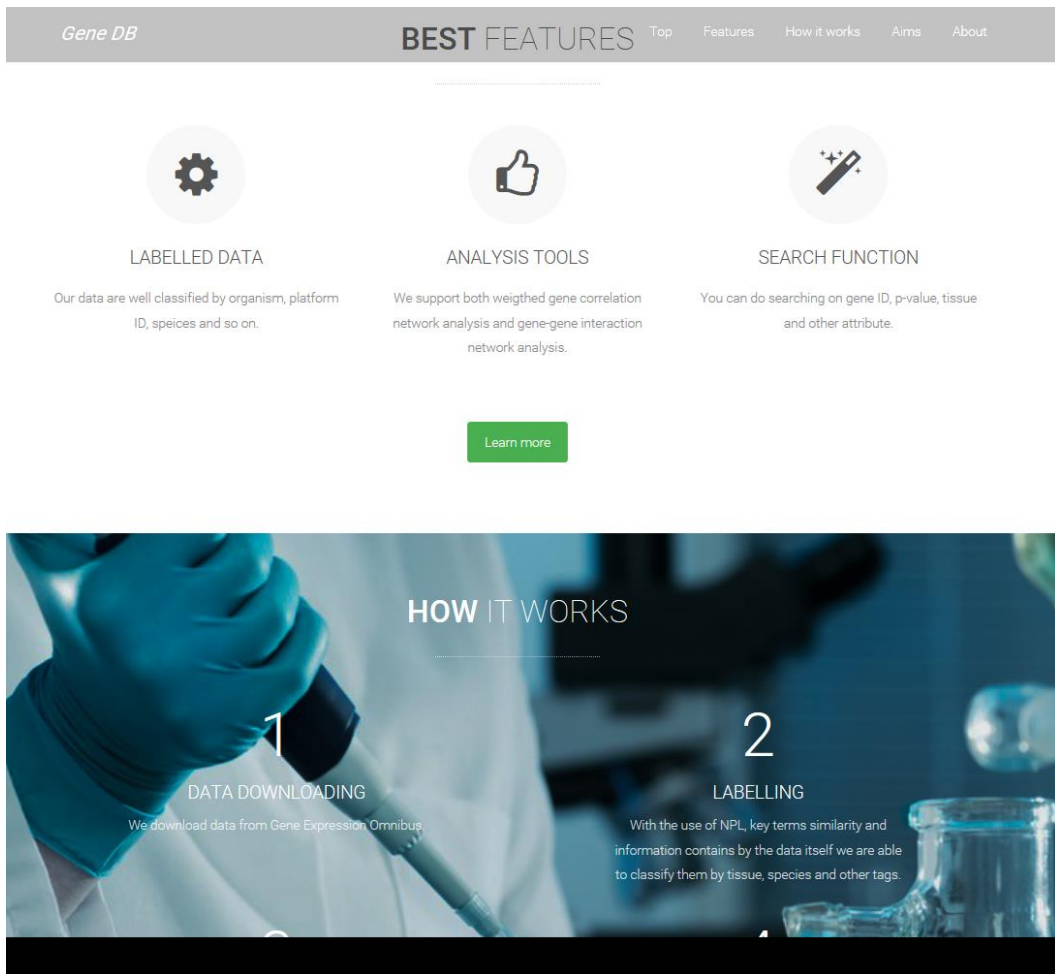


**Figure A.2 Visualizations that provided by the online platform**

# APPENDICES

The screenshot displays the 'Public Modules' section of a software interface. At the top, there is a dark blue header with 'Public Modules' on the left and 'Home / Modules' on the right. Below the header, there are two filter sections: 'Type:' with a 'Show All' button and 'Online Module' / 'Offline Module' options; and 'Category:' with a 'Show All' button and various category names including BWA, PersonalModule, Omics, mRNA, lncRNA, microRNA, circRNA, Meta Network Analysis, Meta Genomics Analysis, and Fraudulence Detection. The main content area is a grid of 12 module cards. Each card features a title, a 'MODULE' label with a yellow square icon, and a status indicator (radio button). The modules shown are: Deadapter (offline), BWA raw to bam (offline), BWA rm lib dup (offline), BWA merge bam (offline), BWA realign (offline), Personal Module 1 (offline), Personal Module 2 (offline), Personal Module 3 (offline), Prepare File (offline), Cut Adapter (offline), Cut Analyze (offline), and Alignment (offline).

**Figure A.3 Modules that provided by the online platform**




**Figure A.4 Gene DB of the online platform**

## APPENDICES

### Delta Team


Based in City University of Hong Kong, led by Dr. Shuaicheng Li, we are a multidisciplinary group of top-ranking and aspiring undergraduates, research assistants and Ph.D. students working tirelessly to bring you a brand-new experience in bioinformatics research that frees you from all the chores and empowers your journey of discovery.

- Algorithms
- Data Analysis
- Bioinformatics

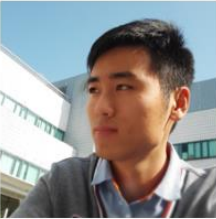


Dr. Shuaicheng Li  
Team Supervisor


### Platform Development Team




Mr. Jia Wenlong  
Ph.D. Student  
Works in cancer bioinformatics field for seven years, now focuses on oncovirus research.



Mr. Fang Zhou  
Research Assistant  
CityU 2015 graduate with three years of web development and testing experience.




Mr. Feng Xikang  
Ph.D. Student  
CityU Ph.D. student with three years' django experience.




Mr. Xu Chang  
Research Assistant  
I do a lot of stuff.

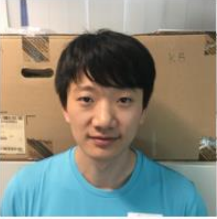
### Research Team




Mr. Ao Xiang  
Ph.D. Student  
Focus on alternative splicing



Miss. Chen Jiaxing  
Ph.D. Student  
Focus on gene network analysis.



Mr. Zhao Zicheng  
Ph.D. Student  
Ph.D. student



Mr. Tan Bowen  
Research Assistant  
CityU 2016 graduate. Bachelor in Mathematics, CityU.

**Figure A.4 Delta team of the online platform**

Development of this online platform was led by Dr. Shuaicheng Li with a group of enthusiastic students. In this project, Ms Chen JiaXing was the mentor who provided guidances and information about the modules to me. At the end of the project, we delivered 4 modules and all of the modules would be integrated with other modules to form a pipeline, then would be published under pipeline category in the online platform.

## APPENDICES

### B-1 Constructing Directed Network Module by Using CBDN (Source Code)

#### B-1.1 main.cpp

```
#include "tbb/blocked_range.h"
#include "tbb/parallel_for.h"
#include "tbb/task_scheduler_init.h"
#include "tbb/mutex.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "Dependency.h"
#include <armadillo>

using namespace std;
using namespace arma;
using namespace tbb;

int main(){
    string inputFile;

    //get input file from user
    cout << "Enter Abundance Matrix File Name: ";
    cin >> inputFile;

    //open input stream
    ifstream input(inputFile);

    //open output stream
    ofstream outputRankFull("Output-Rank(FullName).txt");
    ofstream outputRankIndex("Output-Rank(Index).txt");
    ofstream outputDependencyFull("Output-Dependency(FullName).txt");
    ofstream outputDependencyIndex("Output-Dependency(Index).txt");

    cout << "1. Retriving Data" << endl << endl;

    //This part of code is retrieve the gene ID and sample ID from the input file
    string line;
    getline(input, line);

    vector<string> sampleID;
    vector<string> geneID;

    line.erase(line.begin(), line.begin() + line.find(',') + 1);
    while (line.find(',') != -1){
        sampleID.push_back(line.substr(0, line.find(',')));
        line.erase(line.begin(), line.begin() + line.find(',') + 1);
    }
    sampleID.push_back(line);

    while (getline(input, line)){
        geneID.push_back(line.substr(0, line.find(',')));
    }

    //Load abundance matrix from input file
    mat exp;
```



## APPENDICES

```
exp.load(inputFile);

//remove first row and first column which are storing gene ID and sample ID
exp.shed_col(0);
exp.shed_row(0);

cout << "2. Constructing Network" << endl << endl;

//creating a matrix to store influence score of each gene pair
mat influenceScore(exp.n_rows, exp.n_rows, fill::zeros);

//calculate correlation of each gene to another gene
mat corrMat = cor(trans(exp));

int infrow = influenceScore.n_rows;

//parallelize the influence score calculation part to all processors
//parallelize row by row (each row represents a gene who influences other genes)
parallel_for(0, infrow, [&](int i){
    //loop through each column(each column represents a gene who is being influenced
    by another gene)
    for (int j = 0; j < influenceScore.n_cols; j++){
        //get rid of self regulatory interaction
        if (i == j){
            influenceScore(i, j) = 0;
            continue;
        }
        //influence score calculation part
        double totalInflu = 0.0;
        for (int k = 0; k < influenceScore.n_cols; k++){
            if (k != i && k != j){
                double pc = (corrMat(i, k) - (corrMat(i, j) * corrMat(k,
                j)))/ sqrt((1 - pow(corrMat(i, j), 2)) * (1 - pow(corrMat(k,
                j), 2)));
                totalInflu += abs(corrMat(i, k) - pc);
            }
        }
        influenceScore(j, i) = totalInflu / (influenceScore.n_rows - 1);
    }
});

//Remove cyclic interactions
for (int row = 1; row < influenceScore.n_rows; row++){
    for (int col = 0; col < row; col++){
        if (influenceScore(row, col) < influenceScore(col, row))
            influenceScore(row, col) = 0;
        else
            influenceScore(col, row) = 0;
    }
}

//Remove transitive interactions and final output will be stored in variable "graph"
urowvec maxIndex = index_max(influenceScore);
mat graph(influenceScore.n_rows, influenceScore.n_cols, fill::zeros);

for (int col = 0; col < maxIndex.n_cols; col++){
    graph(maxIndex(col), col) = influenceScore(maxIndex(col), col);
}
graph.save("graph.csv", csv_ascii);
```

## APPENDICES

```
cout << "4. Storing Result" << endl;
//calculate total influence score in row bias
vec totalInfluence = sum(graph, 1);

//sort the total influence value to find out important regulators
uvec indices = sort_index(totalInfluence, "descend");

//output ranked important regulators
cout << "Ranked regulators will be stored in Output-Rank(FullName).txt and Output-
Rank(Index).txt" << endl;
    for (int i = 0; i < indices.n_rows; i++){
        int index = indices.at(i);
        outputRankFull << geneID.at(index) << "\t" << totalInfluence(index) << endl;
        outputRankIndex << index << "\t" << totalInfluence(index) << endl;
    }

cout << "Dependency of gene pairs will be stored in Output-Dependency(Index).txt and
Output - Dependency(FullName).txt" << endl;
//output dependency of gene pairs
for (int i = 0; i < indices.n_rows; i++){
    int index = indices.at(i);
    if (totalInfluence(index) == 0)
        continue;
    for (int col = 0; col < indices.n_rows; col++){
        if (graph(index, col) == 0)
            continue;
        outputDependencyFull << geneID.at(index) << "\t" <<
        geneID.at(col) << "\t" << graph(index, col) << endl;
        outputDependencyIndex << index << "\t" << col << "\t" <<
        graph(index, col) << endl;
    }
}

cout << "Done." << endl;

input.close();
outputRankFull.close();
outputRankIndex.close();
outputDependencyFull.close();
outputDependencyIndex.close();
return 0;
}
```

## APPENDICES

### B-2 Biological Network Inference by Using TIGRESS (Source Code)

#### B-2.1 main.cpp

```
#include "tbb/blocked_range.h"
#include "tbb/parallel_for.h"
#include "tbb/task_scheduler_init.h"
#include <iostream>
#include <fstream>
#include <string>
#include <armadillo>
#include "tigress.h"

using namespace std;
using namespace arma;
using namespace tbb;

mat score_edges(cube, int);
mat rank_edges(mat, urowvec, int = 0);
urowvec indexing(vector<string>, vector<string>);

int main(){
    ofstream outputDependencyFull("Output-Dependency(Full).txt");
    ofstream outputDependencyIndex("Output-Dependency(Index).txt");

    cout << "----- TIGRESS Method -----" << endl << endl;
    cout << "----- 1. Retrieving Data -----" << endl;

    string dataset, tfset;

    //Request input file name from user and store it into variable inputTxt
    cout << "Input Dataset: ";
    cin >> dataset;

    //Request input file name from user and store it into variable inputTxt
    cout << "Input TF: ";
    cin >> tfset;

    //Declare an input stream for reading data from inputTxt
    ifstream input(dataset);

    //Declare an input stream for reading data from inputTxt
    ifstream input2(tfset);

    //This variable is used to keep all genes' name
    vector<string> geneName;

    vector<string> tfList;

    string tfName;
    while (getline(input2,tfName)){
        tfList.push_back(tfName);
    }
}
```

## APPENDICES

```
//This portion of code is used to move the pointer to next line in order to read a correct row of
data
string geneNameStr;
getline(input, geneNameStr);
geneNameStr.erase(geneNameStr.begin(), geneNameStr.begin() + geneNameStr.find(',') + 1);
//while (getline(input, geneNameStr)){
//    geneName.push_back(geneNameStr.substr(0, geneNameStr.find(',')));
//}
while (true){
    if (geneNameStr.find(',') == -1){
        geneName.push_back(geneNameStr);
        break;
    }

    geneName.push_back(geneNameStr.substr(0, geneNameStr.find(',')));
    geneNameStr.erase(geneNameStr.begin(), geneNameStr.begin() +
geneNameStr.find(',') + 1);

}

mat exp;
exp.load(dataset);
exp.shed_col(0);
exp.shed_row(0);

cout << "----- 2. Setting Parameters -----" << endl;

int R = 500;
int L = 3;
double alpha = 0.3;

cout << "R\t= " << R << endl;
cout << "L\t= " << L << endl;
cout << "Alpha\t= " << alpha << endl << endl << endl;

//Passing parameters to tigress function, refer to Tigress class
cout << "----- 3. Inferring Network -----" << endl;
//get list of tf index
urowvec tfIndex = indexing(geneName, tfList);

//cube freq = Tigress::tigress(R, L, alpha, trans(exp), tfIndex);
cube freq = Tigress::tigress(R, L, alpha, exp, tfIndex);

//scoring the interactions, refer to score_eges function in below
cout << "----- 4. Edges' Score -----" << endl;
mat scores = score_edges(freq, 2);

//Ranking the interactions, refer to rank_edges function in below
cout << "----- 5. Edges' Ranking -----" << endl;
mat sortedEdges = rank_edges(scores, tfIndex, 10000);
cout << "----- 6. Output Result -----" << endl;
//output ranked important regulators
for (int i = 0; i < sortedEdges.n_rows; i++){
    outputDependencyFull << geneName.at(sortedEdges(i, 0)) << "\t" <<
geneName.at(sortedEdges(i, 1)) << "\t" << sortedEdges(i, 2)<<endl;
    outputDependencyIndex << int(sortedEdges(i, 0)) << "\t" << int(sortedEdges(i, 1))
<< "\t" << sortedEdges(i, 2) << endl;
}
}
```

## APPENDICES

```
        outputDependencyFull.close();
        outputDependencyIndex.close();
        input.close();
        input2.close();
        return 0;
    }

//this function is used to score the interactions
mat score_edges(cube freq, int method){
    int ntf = freq.n_rows;
    int L = freq.n_cols;
    int ngene = freq.n_slices;
    mat scores;
    switch (method){
    case 1:
        scores = reshape(max(freq, 1), ntf, ngene, 1);
        break;

    case 2:
        scores = reshape((zeros(ntf, 1, ngene) + freq(span(), span(0,0), span())) / 2 +
            sum((freq(span(), span(1, L - 1), span()) + freq(span(),
            span(0, L - 2), span())) / 2, 1), ntf, ngene, 1);
        scores = 1 / (L - 0.5) * scores;
        break;
    }
    return scores;
}

//This method is used to rank the edges
mat rank_edges(mat scores, urowvec tfIndex, int cutoff){
    int ntf = tfIndex.n_cols;
    int ngene = scores.n_cols;
    int k = 0;
    mat edges((ngene * ntf) - ntf, 3);
    mat sortedEdges((ngene * ntf) - ntf, 3);

    for (int i = 0; i < ntf; i++){
        for (int j = 0; j < ngene; j++){
            if (tfIndex(i) != j){
                edges(k, 0) = tfIndex(i);
                edges(k, 1) = j;
                edges(k, 2) = scores(tfIndex(i), j);
                k++;
            }
        }
    }
    uvec indexList = sort_index(edges.col(2), "descend");
    for (int i = 0; i < indexList.n_rows; i++){
        sortedEdges.row(i) = edges.row(indexList(i));
    }

    sortedEdges = sortedEdges.rows(find(sortedEdges.col(2) > 0));
    int nedges = sortedEdges.n_rows;

    if (cutoff != 0)
        sortedEdges = sortedEdges.head_rows(cutoff);

    return sortedEdges;
}
```

## APPENDICES

```
//This function is used to get index number of tf gene
urowvec indexing(vector<string> geneName, vector<string> tfList){
    urowvec index(tfList.size());
    for (int i = 0; i < tfList.size(); i++){
        for (int j = 0; j < geneName.size(); j++){
            if (tfList.at(i) == geneName.at(j))
                index(i) = j;
        }
    }
    return index;
}
```

## APPENDICES

### B-2.2 Tigress.h

```
#ifndef TIGRESS_H
#define TIGRESS_H
#include <armadillo>

//This class is used to perform tigress
//removeTG: a function that remove TG from TF list
//stabilitySelection: a function that perform stability selection
//tigress: a function that perform tigress
//normalize_data: a function that used to normalize the expression data
class Tigress{
private:
    static arma::urowvec removeTG(int, arma::urowvec);

    static arma::mat stabilitySelection(arma::mat, arma::mat, int, int, double alpha,
arma::uvec, int);
public:
    static arma::cube tigress(int, int, double, arma::mat, arma::urowvec);
    static arma::mat normalize_data(arma::mat);

};
#endif
```

## APPENDICES

### B-2.3 Tigress.cpp

```
#include <string>
#include <vector>
#include <armadillo>
#include "tigress.h"
#include "lars.h"
#include "tbb/blocked_range.h"
#include "tbb/parallel_for.h"
#include "tbb/task_scheduler_init.h"

using namespace std;
using namespace arma;
using namespace tbb;

//The function that perform tigress
//stabilitySelection function can refer in below
cube Tigress::tigress(int R, int L, double alpha, mat expdata, urowvec tfIndex){
    expdata = normalize_data(expdata);
    int ntf = tfIndex.n_cols;
    cube freq(ntf, L, expdata.n_cols);

    //for (uword i = 0; i < expdata.n_cols;i++){
    parallel_for(uword(0), expdata.n_cols, [&](uword i){
        cout << i << endl;
        urowvec TFofTG = removeTG(i, tfIndex);

        mat x = expdata.cols(TFofTG);
        mat y = expdata.col(i);

        mat tmp = stabilitySelection(x, y, R, L, alpha, find(TFofTG >= 0), ntf);
        freq.slice(i) = trans(tmp);
    });
    return freq;
}

//Normilize expression data
mat Tigress::normalize_data(mat expdata){
    mat dataMean = mean(expdata);
    mat dataStd = stddev(expdata);

    return (expdata - (mat(expdata.n_rows, 1, fill::ones) * dataMean)) * diagmat(1 / dataStd);
}

//The function that used to perform remove TG from TF list
//before perform stability selection
urowvec Tigress::removeTG(int index, urowvec tfIndex){
    for(uword i = 0; i < tfIndex.size(); i++){
        if(tfIndex(i) != index) continue;
        tfIndex.shed_col(i);
    }
    return tfIndex;
}

//Feature selection function used to determine the interactions
mat Tigress::stabilitySelection(mat X, mat Y, int R, int L, double alpha, uvec predictorTF, int ntf){
```



## APPENDICES

```
mat freq(L, ntf, fill::zeros);
arma_rng::set_seed_random();
for(int i = 0; i < floor(R/2); i++){
    //reweight the expression data
    mat reweightedData = X % repmat(alpha + ((1 - alpha) * randu(1, X.n_cols)),
X.n_rows, 1);
    //create a list of number for randomly split the experiments into two sets
    uvec listExperiment(X.n_rows);
    for(int a = 0; a < X.n_rows; a++){
        listExperiment(a) = a;
    }
    //randomly shuffle the numbers
    listExperiment = shuffle(listExperiment);
    int half = floor(X.n_rows / 2);
    //variable Lars can refer to class Lars
    Lars l1(reweightedData.rows(listExperiment.subvec(0, half - 1)),
Y.rows(listExperiment.subvec(0, half - 1)), L);
    Lars l2(reweightedData.rows(listExperiment.subvec(half, listExperiment.n_rows -
1)), Y.rows(listExperiment.subvec(half, listExperiment.n_rows - 1)), L);
    //get result from lars
    mat result1 = l1.getBeta();
    mat result2 = l2.getBeta();
    //store the result
    freq.cols(predictorTF) = freq.cols(predictorTF) + abs(sign(result1(span(1, L),
span())));
    freq.cols(predictorTF) = freq.cols(predictorTF) + abs(sign(result2(span(1, L),
span())));
}

return freq / (R * 1.0);
}
```

## APPENDICES

### B-2.4 Lars.h

```
#ifndef LARS_H
#define LARS_H
#include <armadillo>

//The class that used to perform least angle regression
//Beta: variable to keep the result
//Lars: a constructor
//lars: a function that perform lars
//setDiff: a function that used to perform set different
//minplus: a function that used to find minimum positive feature
//normalize: a function that used to normalize expression data
//getBeta: a function that used to access private variable Beta
class Lars{
private:
    arma::mat Beta;
public:
    Lars(arma::mat, arma::mat, int);
    arma::mat lars(arma::mat, arma::mat, int, bool=false);
    arma::urowvec setDiff(arma::urowvec, arma::urowvec);
    double minplus(arma::mat, arma::uvec &, arma::urowvec &);
    arma::mat normalize(arma::mat);
    arma::mat getBeta();
};

#endif
```

## APPENDICES

### B-2.5 Lars.cpp

```
#include <string>
#include <vector>
#include <armadillo>
#include "lars.h"

using namespace std;
using namespace arma;

//A constructor
Lars::Lars(mat X, mat Y, int L){
    Beta = lars(X, Y, L);
}

//least angle regression function
mat Lars::lars(mat X, mat Y, int L, bool lasso){
    double eps = 1e-10;

    int n = X.n_rows;
    int p = X.n_cols;
    X = normalize(X);
    Y = Y - (mat(n, 1, fill::ones) * mean(Y));
    mat t;
    t << numeric_limits<double>::infinity();
    int T = t.size();

    mat beta(1, p, fill::zeros);
    mat mu(n, 1, fill::zeros);
    mat mu_old(n, 1, fill::zeros);

    mat gamma(L, 1, fill::zeros);

    urowvec A;
    urowvec Ac(p);
    for (int a = 0; a < p; a++){
        Ac(a) = a;
    }
    int nVars = 0;
    int signOk = 1;
    uword i = 0;
    int t_prev = 0;

    mat beta_t = zeros(T,p);

    int ii = 0;
    mat tt = t;

    uvec addVar;
    while (nVars < L){
        mat c = trans(X) * (Y - mu);
        double C = as_scalar(max(abs(c)));

        if (C < eps || t.is_empty())
            break;

        if (i == 0)
            addVar = find(C == abs(c));
```

## APPENDICES

```
    if (signOk){
        nVars++;
        A.insert_cols(A.size(), addVar);
    }
    mat s_A = sign(c(A));
    Ac = setDiff(Ac, A);
    int nZeros = Ac.n_cols;

    mat X_A = X.cols(A);
    mat G_A = trans(X_A) * X_A;
    mat invG_A = inv(G_A);
    double L_A = as_scalar (1 / sqrt(trans(s_A) * invG_A * s_A));
    mat w_A = L_A * invG_A * s_A;
    mat u_A = X_A * w_A;
    mat a = trans(X) * u_A;
    mat beta_tmp(p, 1, fill::zeros);
    mat gammaTest(nZeros, 2, fill::zeros);

    if (nVars == L){
        gamma(i) = C / L_A;
    }
    else {
        for (uword j = 0; j < nZeros; j++){
            double jj = Ac(j);
            gammaTest(j, 0) = (C - c(jj)) / (L_A - a(jj));
            gammaTest(j, 1) = (C + c(jj)) / (L_A + a(jj));
        }
        uvec min_i;
        urowvec min_j;

        gamma(i) = minplus(gammaTest, min_i, min_j);
        addVar = unique(trans(Ac.cols(min_i)));
    }

    for (uword o = 0; o < A.n_cols; o++){
        beta_tmp(A(o)) = beta(i, A(o)) + (gamma(i) * w_A(o));
    }

    mu = mu_old + gamma(i) * u_A;
    mu_old = mu;
    beta.insert_rows(beta.n_rows, trans(beta_tmp));

    i++;
}
return beta;
}

//set different function
urowvec Lars::setDiff(urowvec a, urowvec b){
    urowvec maxList;
    maxList << a.max() << b.max();
    urowvec sortMat(maxList.max() + 1, fill::zeros);
    urowvec result;

    for (int i = 0; i < a.n_cols; i++){
        if (sortMat(a(i)) != 0)
            continue;
        sortMat(a(i)) = 1;
    }
}
```

## APPENDICES

```
    for (int i = 0; i < b.n_cols; i++){
        if (sortMat(b(i)) == 0)
            continue;
        sortMat(b(i)) = 0;
    }

    result = trans(find(sortMat));
    return result;
}

//function used to get minimum positive feature
double Lars::minplus(mat X, uvec &i, urowvec &j){

    for (int col = 0; col < X.n_cols; col++){
        uvec a = find(imag(X.col(col)) != 0 );
        if (a.is_empty()) continue;
        for (int row = 0; row < a.n_rows; row++){
            X(a(row), col) = numeric_limits<double>::infinity();
        }
    }
    for (int col = 0; col < X.n_cols; col++){
        uvec a = find(X.col(col) <= 0);
        if (a.is_empty()) continue;
        for (int row = 0; row < a.n_rows; row++){
            X(a(row), col) = numeric_limits<double>::infinity();
        }
    }
    vector<int> tmpi, tmpj;
    double minValue = min(min(X));
    for (uword row = 0; row < X.n_rows; row++){
        for (uword col = 0; col < X.n_cols; col++){
            if (X(row, col) == minValue){
                tmpi.push_back(row);
                tmpj.push_back(col);
            }
        }
    }
    i = conv_to<uvec>::from(tmpi);
    j = conv_to<urowvec>::from(tmpj);

    return minValue;
}

//normalized expression data
mat Lars::normalize(mat X){
    int n = X.n_rows;
    X = (eye<mat>(n, n) - ((1 / (n * 1.0)) * ones<mat>(n, n))) * X;
    mat n22 = sqrt(diagvec(trans(X) * X));
    X = X / (ones<mat>(n, 1) * trans(n22));
    return X;
}

//getBeta function
mat Lars::getBeta(){
    return Beta;
}
```

## APPENDICES

### B-3 Inferring Biomarker Module (Source Code)

#### B-3.1 main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <armadillo>
#include <vector>
#include "Period.h"
#include "EntirePeriod.h"
#include "TTest.h"
#include "Cluster.h"
#include "Dominant.h"

using namespace std;
using namespace arma;

uvec clustering(mat,int);
uvec reset(uvec);

int main(){
    string fileNameCase;
    string fileNameControl;
    //Get Case file and control file from user
    cout << "Enter Case File Name and Control File Name." << endl;
    cout << "Case File: " << endl;
    cin >> fileNameCase;
    cout << "Control File: " << endl;
    cin >> fileNameControl;

    //open input and output stream
    ifstream caseExp(fileNameCase), controlExp(fileNameControl);
    ofstream out("Experiment Result.txt");

    //This variable is keep whole timeline for control and case
    //Refer to its class for functions
    EntirePeriod entirePeriodCase, entirePeriodControl;

    string line;
    vector<string> geneID1, geneID2;

    //Retrieve case samples data and store them into separate period
    //This part is reading case samples name and categorize them
    cout << "Retriving Case Samples Data..." << endl;
    getline(caseExp, line);
    line.erase(line.begin(), line.begin() + line.find(',') + 1);
    bool flag1 = true;
    int index1 = 0;
    while (flag1){
        string sample;
        if (line.find(',') == -1){
            sample = line;
            flag1 = false;
        }
        else{
            sample = line.substr(0, line.find(','));
            line.erase(line.begin(), line.begin() + line.find(',') + 1);
        }
    }
}
```

## APPENDICES

```
    }

    sample.erase(sample.begin(), sample.begin() + sample.find('_') + 1);

    string period = sample.substr(0, sample.find('_'));
    sample.erase(sample.begin(), sample.begin() + sample.find('_') + 1);

    entirePeriodCase.addPeriod(period, sample, index1);
    index1++;
}
//This part is reading expression value of each samples
mat expCase;
while (getline(caseExp, line)){
    vector<double> geneData;
    geneID1.push_back(line.substr(0, line.find(',')));
    line.erase(line.begin(), line.begin() + line.find(',') + 1);
    while (true){
        int curP = line.find(',');
        if (curP < 0){
            geneData.push_back(stod(line));
            break;
        }
        geneData.push_back(stod(line.substr(0, curP)));
        line.erase(line.begin(), line.begin() + curP + 1);
    }
    expCase.insert_rows(expCase.n_rows, conv_to<rowvec>::from(geneData));
}
entirePeriodCase.setExpData(expCase);
entirePeriodCase.setGeneName(geneID1);

//This part is similar to retriving case samples data
cout << "Retriving Control Samples Data..." << endl;
getline(controlExp, line);
line.erase(line.begin(), line.begin() + line.find(',') + 1);
bool flag2 = true;
int index2 = 0;
while (flag2){
    string sample;
    if (line.find(',') == -1){
        sample = line;
        flag2 = false;
    }
    else{
        sample = line.substr(0, line.find(','));
        line.erase(line.begin(), line.begin() + line.find(',') + 1);
    }

    sample.erase(sample.begin(), sample.begin() + sample.find('_') + 1);

    string period = sample.substr(0, sample.find('_'));
    sample.erase(sample.begin(), sample.begin() + sample.find('_') + 1);

    entirePeriodControl.addPeriod(period, sample, index2);
    index2++;
}
mat expControl;
while (getline(controlExp, line)){
    vector<double> geneData;
    geneID2.push_back(line.substr(0, line.find(',')));
    line.erase(line.begin(), line.begin() + line.find(',') + 1);
```

## APPENDICES

```
while (true){
    int curP = line.find(',');
    if (curP < 0){
        geneData.push_back(stod(line));
        break;
    }
    geneData.push_back(stod(line.substr(0, curP)));
    line.erase(line.begin(), line.begin() + curP + 1);
}
expControl.insert_rows(expControl.n_rows, conv_to<rowvec>::from(geneData));
}
entirePeriodControl.setExpData(expControl);
entirePeriodControl.setGeneName(geneID2);

//This part is choosing genes which have differential expression value
//between case and control for each period
cout << "\n-----1. Choosing Differential Expression Genes-----" << endl;

//Each Period
for (int i = 0; i < entirePeriodCase.size(); i++){
    cout << i;

    //Retrieve case and control Exp data for the period
    uvec caseIndex = conv_to<uvec>::from(entirePeriodCase.getPeriod(i).SampleIndex);
    uvec controlIndex =
        conv_to<uvec>::from(entirePeriodControl.getPeriod(i).SampleIndex);

    mat caseExp = entirePeriodCase.getExpData().cols(caseIndex);
    mat controlExp = entirePeriodControl.getExpData().cols(controlIndex);
    int controlsize = controlExp.n_rows;

    //Store the gene's P value
    vector<double> PValues;

    //Go Through Student T-test
    for (int j = 0; j < caseExp.n_rows; j++){
        //Retrieve case and control Exp data for each gene
        rowvec caseGene = caseExp.row(j);
        rowvec controlGene = controlExp.row(j);
        TTest tTest(caseGene, controlGene);

        PValues.push_back(tTest.pvalue);
    }

    vec PValuesList = conv_to<vec>::from(PValues);
    //Go through FDR and 2-Fold Change

    //Sort the Pvalues list and get the index of the gene
    uvec sortedPValuesList = sort_index(PValuesList, "ascending");
    vector<int> geneIndex;
    //For each index value in sorted index
    for (int j = 0; j < sortedPValuesList.n_rows; j++){
        //FDR
        if (PValuesList(sortedPValuesList.at(j)) < ((j + 1) / float(controlsize)) *
            0.05){
            //2-fold change
            if (stddev(caseExp.row(sortedPValuesList.at(j))) /
                stddev(controlExp.row(sortedPValuesList.at(j))) < 2)
                continue;
        }
    }
}
```



## APPENDICES

```
        //keep the index into a list
        geneIndex.push_back(sortedPValuesList.at(j));
    }
    else
        break;
}
//Keep differential expression genes and retrieve their expression value for case and
control
uvec selectedGene = conv_to<uvec>::from(geneIndex);
cout << selectedGene.n_rows << " ";
entirePeriodCase.addFilteredGene(selectedGene);
entirePeriodControl.addFilteredGene(selectedGene);
entirePeriodCase.addFilteredExp(caseExp.rows(selectedGene));
entirePeriodControl.addFilteredExp(controlExp.rows(selectedGene));
}

//Normalize the filtered expression data
vector<mat> normalizedExp;
for (int i = 0; i < entirePeriodCase.size(); i++){
    mat filteredGeneExpCase = entirePeriodCase.getFilteredExp(i);
    mat filteredGeneExpCon = entirePeriodControl.getFilteredExp(i);
    mat geneM = mean(filteredGeneExpCon, 1) * ones(1, filteredGeneExpCase.n_cols);
    mat geneStd = stddev(filteredGeneExpCon, 0, 1) * ones(1,
        filteredGeneExpCase.n_cols);

    mat normalized = (filteredGeneExpCase - geneM) / geneStd;

    normalizedExp.push_back(normalized);
}

//Output result to a file
out << "-----Choosing Differential Expression Genes-----" << endl;
for (int c = 0; c < entirePeriodCase.size(); c++){
    out << "\nPeriod " << entirePeriodCase.getPeriod(c).period << ":" << endl;
    if (entirePeriodCase.getFilteredGene(c).is_empty()){
        out << "None" << endl;
        continue;
    }
    for (int cc = 0; cc < entirePeriodCase.getFilteredGene(c).n_rows; cc++){
        out << entirePeriodCase.geneName
            .at(entirePeriodCase.getFilteredGene(c).at(cc)) << " ";
    }
    out << endl;
}

//Perform clustering for differential expression genes for each period
//We are using hierarchical clustering method to group the genes
//Clustering can refer to clustering function in below
//The result is storing in a Cluster variable, refer to Cluster class
cout << "\n\n-----2. Clustering The Genes-----" << endl;
//clusterPeriod is used to store clusters for all periods
vector<Cluster> clusterPeriod;
for (int i = 0; i < entirePeriodCase.size(); i++){
    cout << i << " ";
    uvec cluster = clustering(normalizedExp.at(i), 40);
    if (cluster.is_empty()){
        Cluster tmp;
        clusterPeriod.push_back(tmp);
        continue;
    }
}
```

## APPENDICES

```
        clusterPeriod.push_back(Cluster(cluster, entirePeriodCase.getFilteredGene(i),
        expCase, expControl));
    }

    //Calculating indices for all clusters found in whole timeline
    cout << "\n\n-----3. Calculating Indices-----" << endl;
    //Loop through every period to access every clusters
    vector<Dominant> candidateDNB;
    urowvec numDominatedGroup(clusterPeriod.size(), fill::zeros);
    for (int p = 0; p < entirePeriodCase.size(); p++){
        cout << p;
        int clustercount = clusterPeriod.at(p).clusterList.n_rows;

        //Go through each of the cluster and retrieve normalized exp based on the genes in
        each cluster
        for (int i = 0; i < clustercount; i++){
            uvec clustergene = find(clusterPeriod.at(p).clusterList == i);
            int numclustergene = clustergene.n_rows;
            mat nexpcase = clusterPeriod.at(p).sickExp.rows(clustergene);

            uvec othergene = find(clusterPeriod.at(p).clusterList != i);
            mat othergeneexp = clusterPeriod.at(p).sickExp.rows(othergene);
            int othergenenum = othergene.n_rows;

            vector<mat> pccallPeriod;
            vector<double> sumpccallPeriod;
            vector<double> sumstdPeriod;
            vector<double> sumMixpccPeriod;

            for (int period = 0; period < entirePeriodCase.size(); period++){
                uvec tmp =
                conv_to<uvec>::from(entirePeriodCase.getPeriod(period)
                .SampleIndex);

                //calculate pcc of the cluster in each period
                mat pccall = cor(trans(nexpcase.cols(tmp)));
                pccallPeriod.push_back(pccall);
                sumpccallPeriod.push_back((sum(abs(pccall)) -
                numclustergene) / 2);

                //calculate std of the cluster in each period
                vec stdEachGene = stddev(nexpcase.cols(tmp), 0, 1);
                double sumStd = sum(stdEachGene);
                sumstdPeriod.push_back(sumStd);

                //calculate sum of pcc of all other clusters
                mat sumMixpcc(1, 1, fill::zeros);
                mat tmpnexpcase = nexpcase.cols(tmp);
                mat tmpothergeneexp = othergeneexp.cols(tmp);
                for (int ii = 0; ii < numclustergene; ii++){
                    for (int jj = 0; jj < othergenenum; jj++){
                        sumMixpcc +=
                        abs(cor(trans(tmpnexpcase.row(ii)),
                        trans(tmpothergeneexp.row(jj))));
                    }
                }
                sumMixpccPeriod.push_back(sumMixpcc(0, 0));
            }
        }
    }
```

## APPENDICES

```
//calculate average of pcc of cluster, standard deviation of cluster, and sum
of pcc of others clusters
rowvec avgpccPeriod;
rowvec avgstdPeriod;
rowvec avgmixpccPeriod;

avgpccPeriod = conv_to<rowvec>::from(sumpccallPeriod) /
                (numclustergene * (numclustergene - 1)) * 2;
avgstdPeriod = conv_to<rowvec>::from(sumstdPeriod) / numclustergene;
avgmixpccPeriod = conv_to<rowvec>::from(sumMixpccPeriod) /
                numclustergene / othergenenum;

//Checking whether the cluster fulfil the criteria or not
for (int period = 1; period < entirePeriodCase.size() - 1; period++){
    if ((avgstdPeriod(period) > avgstdPeriod(period - 1)) &&
        (avgstdPeriod(period) > avgstdPeriod(period + 1)) &&
        (avgpccPeriod(period) > avgpccPeriod(period - 1)) &&
        (avgpccPeriod(period) > avgpccPeriod(period + 1)) &&
        (avgmixpccPeriod(period) < avgmixpccPeriod(period - 1))
        &&
        (avgmixpccPeriod(period) < avgmixpccPeriod(period +
        1)))){

        //keep the dominated group into a variable
        numDominatedGroup(p)++;
        rowvec compositeIndex = (avgstdPeriod / numclustergene)
        % (avgpccPeriod / (numclustergene * (numclustergene -
        1)) * 2) / avgmixpccPeriod;

        candidateDNB.push_back(Dominant(entirePeriodCase.getPeriod(p).period, p,
        trans(clustergene), compositeIndex));

        break;
    }
}

}

}

}

//output experimental result
out << "\n\n-----Experimental Result-----" << endl;
out << "Number of Candidate DNB In Each Sampling Point: " << numDominatedGroup
<< endl;
for (int a = 0; a < candidateDNB.size(); a++){
    Dominant tmpDNB = candidateDNB.at(a);
    out << "Period " << tmpDNB.period << ":" << endl;
    out << "Cluster : \n" << tmpDNB.cluster << endl;
    out << "Composite Index : \n" << tmpDNB.compositeIndex << endl << endl;
}
out.close();
caseExp.close();
controlExp.close();
return 0;
}

//Clustering function
uvec clustering(mat X, int cutoff){
    //if input X is empty
    if (X.n_rows == 0){
        uvec a;
```

## APPENDICES

```
        return a;
    }
    //declare a column vector to keep group number of each gene
    uvec cluster(X.n_rows, fill::zeros);
    //initialize the group number to each gene
    for (int i = 0; i < X.n_rows; i++){
        cluster(i) = i;
    }
    int clusterNo = max(cluster) + 1;
    //if group number is smaller than cutoff, return the result
    if (clusterNo <= cutoff){
        return cluster;
    }
    //calculating correlation of each gene
    mat corrMat = cor(trans(X), trans(X));
    corrMat = 1.000 - corrMat;

    mat ref = corrMat;
    //Set upper triangle area of correlation matrix to NaN
    for (int row = 0; row < corrMat.n_rows; row++){
        for (int col = row; col < corrMat.n_cols; col++){
            corrMat(row, col) = NAN;
        }
    }

    //Loop the clustering process until the total number of cluster is equal to cutoff
    while (clusterNo > cutoff){
        //find min value of each column and their index
        rowvec tmpMin = min(corrMat);
        urowvec tmpMinIndex = index_min(corrMat);

        //find the smallest value location
        uword minCol = index_min(tmpMin);
        uword minRow = tmpMinIndex.at(minCol);

        //perform cluster merging
        uvec clusterItems = find(cluster == minCol);

        for (int i = 0; i < clusterItems.n_rows; i++){
            cluster(clusterItems(i)) = minRow;
        }

        //reset cluster number, refer to reset function in below
        cluster = reset(cluster);
        clusterNo = max(cluster) + 1;

        //This part is reset the correlation matrix
        mat tempCorrMat;
        tempCorrMat = corrMat;

        //min col
        for (uword row = 0; row < tempCorrMat.n_rows; row++){
            if (std::isnan(tempCorrMat(row, minRow)) || (row == minCol)
                || (row == minRow))
                continue;

            tempCorrMat(row, minRow) = min(corrMat(max(row, minRow),
                min(row, minRow)), corrMat(max(row, minCol), min(row, minCol)));
        }
    }
}
```

## APPENDICES

```
//min row
for (uword col = 0; col < tempCorrMat.n_cols; col++){
    if (std::isnan(tempCorrMat(minRow, col)) || (col == minCol)
        || (col == minRow))
        continue;

    tempCorrMat(minRow, col) = min(corrMat(max(minRow, col),
        min(minRow, col)), corrMat(max(minCol, col), min(minCol, col)));

}

tempCorrMat.shed_col(minCol);
tempCorrMat.shed_row(minCol);

corrMat = tempCorrMat;

}
return cluster;

}

//This function is used to reset the cluster number
uvec reset(uvec X){
    int c = 0;
    for (int i = 0; i < X.n_rows; i++){
        uvec list = find(X == i);
        if (list.n_rows == 0)
            continue;
        for (int j = 0; j < list.n_rows; j++){
            X(list(j)) = c;
        }
        c++;
    }
    return X;
}
```

## APPENDICES

### B-3.2 Cluster.h

```
#ifndef CLUSTER_H
#define CLUSTER_H

#include <vector>
#include <armadillo>
#include <string>

using namespace std;
using namespace arma;

//This class is used to create an instance for each clustered genes
//clusterList = List of genes and their group
//geneList = List of genes' name
//clusterNo = List of clusters and number of their members
//sickExp = Case Expression Genes dataset
//healthyExp = Control Express Genes dataset
class Cluster{
private:

public:
    uvec clusterList;
    uvec geneList;
    urowvec clusterNo;
    mat sickExp;
    mat healthyExp;
    Cluster(uvec, uvec, mat, mat);
    Cluster();
};
#endif
```

## APPENDICES

### B-3.3 Cluster.cpp

```
#include "Cluster.h"
#include <vector>
#include <string>
#include <armadillo>

using namespace std;
using namespace arma;

Cluster::Cluster(){}

Cluster::Cluster(uvec list, uvec geneIndex, mat caseExp, mat controlExp){
    int NoCluster = max(list) + 1;
    clusterList = list;
    urowvec cNo(NoCluster, fill::zeros);
    for (int i = 0; i < list.n_rows; i++){
        cNo(list(i))++;
    }
    clusterNo = cNo;
    geneList = geneIndex;

    sickExp = caseExp.rows(geneIndex);
    healthyExp = controlExp.rows(geneIndex);
    mat sickM = mean(sickExp, 1) * ones(1, sickExp.n_cols);

    mat healthyM = mean(healthyExp, 1) * ones(1, healthyExp.n_cols);
    mat sickStd = stddev(sickExp, 0, 1) * ones(1, sickExp.n_cols);
    mat healthyStd = stddev(healthyExp, 0, 1) * ones(1, healthyExp.n_cols);
    sickExp = (sickExp - healthyM) / healthyStd;
    healthyExp = (healthyExp - healthyM) / healthyStd;
}
```

## APPENDICES

### B-3.4 Dominant.h

```
#ifndef DOMINANT_H
#define DOMINANT_H

#include <vector>
#include <armadillo>
#include <string>
#include "Cluster.h"

//This class is used to create an instance of dominant group
//period = period name of the dominant group
//periodIndex = index of the period
//cluster = list of genes
//compositeIndex = composite index of the dominant group throughout the timeline
class Dominant{
private:
public:
    string period;
    int periodIndex;
    urowvec cluster;
    Dominant(string, int, urowvec, rowvec);
    arma::rowvec compositeIndex;

};
#endif
```

### B-3.5 Dominant.cpp

```
#include "Dominant.h"
#include "Cluster.h"
#include <armadillo>
#include <string>

using namespace std;
using namespace arma;

Dominant::Dominant(string period, int periodIndex, urowvec cluster, rowvec compositeIndex){
    this->period = period;
    this->periodIndex = periodIndex;
    this->cluster = cluster;
    this->compositeIndex = compositeIndex;
}
```



## APPENDICES

### B-3.6 EntirePeriod.h

```
#ifndef ENTIREPERIOD_H
#define ENTIREPERIOD_H

#include "Period.h"
#include <vector>
#include <string>

using namespace std;

//This class is used to create a timeline that have several sampling point
//entirePeriod = list of periods
//expData = expression data
//filteredExp = List of filtered expression data (over periods)
//filteredGene = List of selected genes (over periods)
//geneName = List of genes' name

class EntirePeriod{
private:
    vector<Period> entirePeriod;
    arma::mat expData;
    vector<arma::mat> filteredExp;
    vector<arma::uvec> filteredGene;

public:
    vector<string> geneName;
    void setExpData(arma::mat);
    void setGeneName(vector<string>);
    arma::mat getExpData();
    void addPeriod(string, string, int);
    void addFilteredExp(arma::mat);
    void addFilteredGene(arma::uvec);
    arma::mat getFilteredExp(int);
    arma::uvec getFilteredGene(int);
    int size();
    bool isEmpty();
    Period getPeriod(int);
};

#endif
```

## APPENDICES

### B-3.7 EntirePeriod.cpp

```
#include "Period.h"
#include "EntirePeriod.h"
#include <vector>
#include <string>
using namespace std;

//Add period to the list and makesure there is no duplicated period
void EntirePeriod::addPeriod(string period, string sample, int index){
    //If list is empty, straight away add new period
    if (entirePeriod.size() == 0){
        entirePeriod.push_back(Period(period, sample, index));
        return;
    }

    //Check for existance of period. If yes then add sample to the period
    for (int i = 0; i < entirePeriod.size(); i++){
        if (entirePeriod.at(i).period == period){
            entirePeriod.at(i).SampleID.push_back(sample);
            entirePeriod.at(i).SampleIndex.push_back(index);
            return;
        }
    }
    //Else create a new period
    entirePeriod.push_back(Period(period, sample, index));
}

//Keep expression data
void EntirePeriod::setExpData(mat exp){
    this->expData = exp;
}

//Keep Genes Name
void EntirePeriod::setGeneName(vector<string> geneName){
    this->geneName = geneName;
}

//Retrieve expression data
mat EntirePeriod::getExpData(){
    return expData;
}

//Keep Filtered expression data
void EntirePeriod::addFilteredExp(mat exp){
    filteredExp.push_back(exp);
}

//Keep differential expression genes
void EntirePeriod::addFilteredGene(uvec geneList){
    filteredGene.push_back(geneList);
}

//determine number of list items
int EntirePeriod::size(){
    return entirePeriod.size();
}
```

## APPENDICES

```
//Get particular period
Period EntirePeriod::getPeriod(int index){
    return entirePeriod.at(index);
}

//Get filtered expression data
mat EntirePeriod::getFilteredExp(int index){
    return filteredExp.at(index);
}

//Get differential expression gene
uvec EntirePeriod::getFilteredGene(int index){
    return filteredGene.at(index);
}

//Check the list is empty or not
bool EntirePeriod::isEmpty(){
    if (entirePeriod.size() == 0)
        return true;
    return false;
}
```

## APPENDICES

### B-3.8 Period.h

```
#ifndef PERIOD_H
#define PERIOD_H

#include <vector>
#include <armadillo>
#include <string>

using namespace std;
using namespace arma;

//This class is used to create a period
//period = name of period
//SampleID = ID of the samples
//SampleIndex = Index of the samples
class Period{
    private:

    public:
        string period;
        vector<string> SampleID;
        vector<int> SampleIndex;
        Period(string, string, int);

};

#endif
```

### B-3.9 Period.cpp

```
#include "Period.h"
#include <armadillo>
#include <string>

using namespace std;
using namespace arma;

Period::Period(string period, string SampleID, int index){
    this->period = period;
    this->SampleID.push_back(SampleID);
    this->SampleIndex.push_back(index);
}
```

## APPENDICES

### B-3.10 TTest.h

```
#ifndef TTEST_H
#define TTEST_H

#include <boost/math/distributions/students_t.hpp>
#include <armadillo>

using namespace boost::math;

//This class is used to perform student-t test
//flag = return true if the p values is smaller than significant value
//pvalue = p value
class TTest{
private:

public:
    bool flag;
    double pvalue;
    TTest(arma::rowvec, arma::rowvec);
};

#endif
```

## APPENDICES

### B-3.11 TTest.cpp

```
#include <boost/math/distributions/students_t.hpp>
#include <armadillo>
#include "TTest.h"

using namespace std;
using namespace arma;

//T-test function
TTest::TTest(rowvec caseGene, rowvec controlGene){
    double caseMean = mean(caseGene);
    double controlMean = mean(controlGene);

    double caseVar = var(caseGene);
    double controlVar = var(controlGene);

    int caseSize = caseGene.n_cols;
    int controlSize = controlGene.n_cols;

    double alpha = 0.05;
    int df = caseSize + controlSize - 2;

    double diff = caseMean - controlMean;
    double pooledVar = ((caseSize - 1) * caseVar + (controlSize - 1) * controlVar) / float(df);

    double t = diff / sqrt(pooledVar * ((1 / float(caseSize)) + (1 / float(controlSize))));
    t = -abs(t);
    students_t dist(df);

    pvalue = 2 * cdf(complement(dist, fabs(t)));
    if (pvalue <= alpha)
        flag = true;
    else
        flag = false;
}
```

## APPENDICES

### B-4 Imputation (Source Code)

#### B-4.1 main.cpp

```
#include <iostream>
#include <armadillo>
#include <boost/math/distributions/students_t.hpp>
#include "tbb/blocked_range.h"
#include "tbb/parallel_for.h"
#include "tbb/task_scheduler_init.h"
#include "TTest.h"

using namespace std;
using namespace arma;
using namespace tbb;

int overlapping(urowvec, urowvec);

int main(){
    string casefileName;
    string controlfileName;

    //Get Case file and control file from user
    cout << "Enter Case File Name and Control File Name:" << endl;
    cout << "Case File: ";
    cin >> casefileName;
    cout << "Control File: ";
    cin >> controlfileName;
    ifstream inputCase(casefileName);
    ifstream inputCont(controlfileName);

    //This part of code is retrieving expression data from case and control file
    cout << "1. Retriving Data" << endl << endl;
    string line;
    getline(inputCont, line);

    vector<string> sampleID;
    vector<string> geneID;

    line.erase(line.begin(), line.begin() + line.find(',') + 1);
    while (line.find(',') != -1){
        sampleID.push_back(line.substr(0, line.find(',')));
        line.erase(line.begin(), line.begin() + line.find(',') + 1);
    }
    sampleID.push_back(line);

    while (getline(inputCont, line)){
        geneID.push_back(line.substr(0, line.find(',')));
    }

    mat expCase, expControl;
    expCase.load(casefileName);
    expCase.shed_col(0);
    expCase.shed_row(0);

    expControl.load(controlfileName);
    expControl.shed_col(0);
    expControl.shed_row(0);
```

## APPENDICES

```
//construct a biological network by using CBDN method
//refer to CBDN module
cout << "2. Constructing Network" << endl << endl;

mat influenceScore(expControl.n_rows, expControl.n_rows, fill::zeros);
mat corrMat = cor(trans(expControl));

int infrow = influenceScore.n_rows;
parallel_for(0, infrow, [&](int i){
    //for (int i = 0; i < influenceScore.n_rows; i++){
    for (int j = 0; j < influenceScore.n_cols; j++){
        if (i == j){
            influenceScore(i, j) = 0;
            continue;
        }
        double totalInflu = 0.0;
        for (int k = 0; k < influenceScore.n_cols; k++){
            if (k != i && k != j){
                double pc = (corrMat(i, k) - (corrMat(i, j) * corrMat(k, j))
                    )/ sqrt((1 - pow(corrMat(i, j), 2)) * (1 - pow(corrMat(k, j),
                    2)));
                totalInflu += abs(corrMat(i, k) - pc);
            }
        }
        influenceScore(j, i) = totalInflu / (influenceScore.n_rows - 1);
    }
});
for (int row = 1; row < influenceScore.n_rows; row++){
    for (int col = 0; col < row; col++){
        if (influenceScore(row, col) < influenceScore(col, row))
            influenceScore(row, col) = 0;
        else
            influenceScore(col, row) = 0;
    }
}

urowvec maxIndex = index_max(influenceScore);
mat graph(influenceScore.n_rows, influenceScore.n_cols, fill::zeros);

for (int col = 0; col < maxIndex.n_cols; col++){
    graph(maxIndex(col), col) = influenceScore(maxIndex(col), col);
}

//choosing differential expression gene like DNB module
cout << "3. Detect Differentially Expressed Genes" << endl << endl;

//Store the gene's P value
vector<double> PValues;

//Go Through Student T-test
for (int j = 0; j < expCase.n_rows; j++){
    //Retrieve case and control Exp data for each gene
    rowvec caseGene = expCase.row(j);
    rowvec controlGene = expControl.row(j);
    TTest tTest(caseGene, controlGene);

    PValues.push_back(tTest.pvalue);
}
}
```



## APPENDICES

```
vec PValuesList = conv_to<vec>::from(PValues);
//Go through FDR and 2-Fold Change

//Sort the Pvalues list and get the index of the gene
uvec sortedPValuesList = sort_index(PValuesList, "ascending");
vector<int>geneIndex;
//For each index value in sorted index
for (int j = 0; j < sortedPValuesList.n_rows; j++){
    //FDR
    if (PValuesList(sortedPValuesList.at(j)) < ((j + 1) / float(expControl.n_rows)) * 0.5){
        //2-fold change
        if (stddev(expCase.row(sortedPValuesList.at(j))) /
            stddev(expControl.row(sortedPValuesList.at(j))) < 2)
            continue;

        //keep the index into a list
        geneIndex.push_back(sortedPValuesList.at(j));
    }
    else
        break;
}

//Expanding the seeds to include their neighbor genes
cout << "4. Expanding to Neighbor genes" << endl << endl;
//a list used to keep all clusters
vector<vector<int>> clusters;
//get list of different expression genes
urowvec diffExpGene = conv_to<urowvec>::from(geneIndex);
//loop through each of them, each of them will include their neighbor genes and consider as a
cluster
//neighbor genes are determined by using the network constructed in step 1
for (int i = 0; i < diffExpGene.n_cols; i++){
    vector<int> clustergene;
    clustergene.push_back(diffExpGene(i));
    for (int row = 0; row < graph.n_rows; row++){

        if (graph(row, diffExpGene(i)) == 0)
            continue;
        else
            clustergene.push_back(row);
    }

    for (int col = 0; col < graph.n_cols; col++){
        if (graph(diffExpGene(i), col) == 0)
            continue;
        else
            clustergene.push_back(col);
    }
    clusters.push_back(clustergene);
}

//Identify which clusters have overlapping genes
//merge them as are considered as one community
//overlapping function can refer below
cout << "5. Identifying overlapping clusters" << endl << endl;

mat overlappingMat(clusters.size(), clusters.size(), fill::zeros);
int kc = 3;
for (int row = 0; row < overlappingMat.n_rows; row++){
```

## APPENDICES

```
        for (int col = 0; col < overlappingMat.n_cols; col++){
            overlappingMat(row, col) =
                overlapping(conv_to<urowvec>::from(clusters.at(row)),
                    conv_to<urowvec>::from(clusters.at(col)));
        }
    }
    //set diagonal elements to 1 if they are greater than k value
    for (int i = 0; i < overlappingMat.n_rows; i++){
        if (overlappingMat(i, i) < kc)
            overlappingMat(i, i) = 0;
        else
            overlappingMat(i, i) = 1;
    }

    //set each cells to 0 or value greater than 1 for number of similir genes
    for (int row = 0; row < overlappingMat.n_rows; row++){
        for (int col = 0; col < overlappingMat.n_cols; col++){
            if (row == col)
                continue;
            if (overlappingMat(row, col) < kc - 1)
                overlappingMat(row, col) = 0;
            else
                overlappingMat(row, col) = 1;
        }
    }
    //sum columnny by column
    rowvec sumMat = sum(overlappingMat, 0);
    //find which cluster is overlapping clusters
    uvec clique = find(sumMat != 0);
    uvec cliqueList(clique.n_rows);
    //create a list of number for each overlapping clusters
    for (int i = 0; i < cliqueList.n_rows; i++){
        cliqueList(i) = i + 1;
    }

    //group clusters which are same community
    for (int i = 0; i < clique.n_rows; i++){
        for (int j = 0; j < clique.n_rows; j++){
            if (clique(i) == clique(j))
                continue;

            if (overlappingMat(clique(i), clique(j)) == 1)
                if (cliqueList(j) != cliqueList(i))
                    cliqueList(j) = cliqueList(i);
        }
    }

    //reset the communities number
    int nc = 1;
    for (int i = 1; i <= cliqueList.n_rows; i++){
        uvec list = find(cliqueList == i);
        if (list.n_rows == 0)
            continue;
        for (int j = 0; j < list.n_rows; j++){
            cliqueList(list(j)) = nc;
        }
        nc++;
    }

    //merge the clusters from same communities to form a larger cluster
```

## APPENDICES

```
int size = max(cliqueList);
vector<vector<int>>cliques;
for (int i = 0; i < size; i++){
    uvec group = find(cliqueList == i + 1);
    vector<int>tmp;
    for (int j = 0; j < group.n_rows; j++){
        for (int k = 0; k < clusters.at(clique(group(j))).size(); k++){
            tmp.push_back(clusters.at(clique(group(j))).at(k));
        }
    }
    cliques.push_back(tmp);
}

//checking for duplicated genes
vector<vector<int>> tmpList;
for (int i = 0; i < cliques.size(); i++){
    urowvec tmp = conv_to<urowvec>::from(cliques.at(i));
    tmp = unique(tmp);
    tmpList.push_back(conv_to<vector<int>>::from(tmp));
}
cliques = tmpList;

//score the communities/clusters
cout << "6. Calculating Imputation Value" << endl << endl;
rowvec score(cliques.size(), fill::zeros);
int n = expCase.n_cols;
for (int i = 0; i < cliques.size(); i++){
    mat genesMat = expCase.rows(conv_to<urowvec>::from(cliques.at(i)));
    rowvec E = sum(genesMat, 0) / (cliques.at(i).size() * 1.0);
    score(i) = accu(E) / (n * 1.0);

    uvec missingValueIndex = find(genesMat.row(0) == 0);
    if (missingValueIndex.n_rows == 0)
        continue;
    rowvec cValues = genesMat.row(0) - sum(genesMat.rows(1, genesMat.n_rows - 1),
    0);
    double sumMissC = sum(cValues(missingValueIndex));
    double c = (sum(cValues) - sumMissC) / (cValues.n_cols -
    missingValueIndex.n_rows);

    for (int k = 0; k < missingValueIndex.n_rows; k++){
        expCase.at(cliques.at(i).at(0), missingValueIndex(k)) = c +
        abs(cValues(missingValueIndex(k)));
        cout << c + abs(cValues(missingValueIndex(k))) << " ";
    }
}
cout << mean(expCase.row(15))<<endl;
expCase.save("newCase.csv", csv_ascii);
return 0;
}

//overlapping function, used to determine how many genes exist in both cluster
int overlapping(urowvec a, urowvec b){
    urowvec maxVal;
    maxVal << a.max() << b.max() << endl;
    urowvec list(maxVal.max() + 1, fill::zeros);
    for (int i = 0; i < a.n_cols; i++){
        int x = a(i);
        list(a(i))++;
    }
}
```

## APPENDICES

```
    for (int i = 0; i < b.n_cols; i++){  
        list(b(i))++;  
    }  
    uvec result = find(list == 2);  
    return result.n_rows;  
}
```

## APPENDICES

### B-4.2 TTest.h

```
#ifndef TTEST_H
#define TTEST_H

#include <boost/math/distributions/students_t.hpp>
#include <armadillo>

using namespace boost::math;

//This class is used to perform student-t test
//flag = return true if the p values is smaller than significant value
//pvalue = p value
class TTest{
private:

public:
    bool flag;
    double pvalue;
    TTest(arma::rowvec, arma::rowvec);
};

#endif
```

## APPENDICES

### B-4.3 TTest.cpp

```
#include <boost/math/distributions/students_t.hpp>
#include <armadillo>
#include "TTest.h"

using namespace std;
using namespace arma;

//T-test function
TTest::TTest(rowvec caseGene, rowvec controlGene){
    double caseMean = mean(caseGene);
    double controlMean = mean(controlGene);

    double caseVar = var(caseGene);
    double controlVar = var(controlGene);


    int caseSize = caseGene.n_cols;
    int controlSize = controlGene.n_cols;

    double alpha = 0.05;
    int df = caseSize + controlSize - 2;

    double diff = caseMean - controlMean;
    double pooledVar = ((caseSize - 1) * caseVar + (controlSize - 1) * controlVar) / float(df);

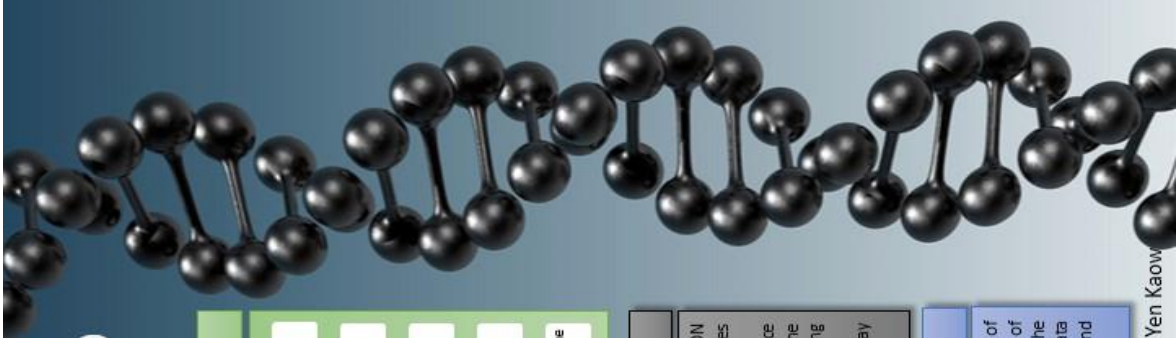
    double t = diff / sqrt(pooledVar * ((1 / float(caseSize)) + (1 / float(controlSize))));
    t = -abs(t);
    students_t dist(df);

    pvalue = 2 * cdf(complement(dist, fabs(t)));
    if (pvalue <= alpha)
        flag = true;
    else
        flag = false;
}
```



**UTAR**  
UNIVERSITI TUNKU ABDUL RAHMAN

**Faculty of Information and Communication Technology (FICT)**  
**Online Tools For Analyzing Metagenomics Data**



**Introduction**

In recent years, scientists have realized the importance of human microbiota in shaping our health. This has led to an urgency to study the human microbiota. In order to study the microbial community or microbiota, we would like to develop modules to analyze metagenomics data

**Project Scope**

In this project, we would like to collaborate with City University of Hong Kong to come up a comprehensive platform for researchers to analyse metagenomics data. This project, we are involved in four modules of the platform.

**Constructing Biological Network by TIGRESS**

Score interactions with stability selection

Rank the interactions

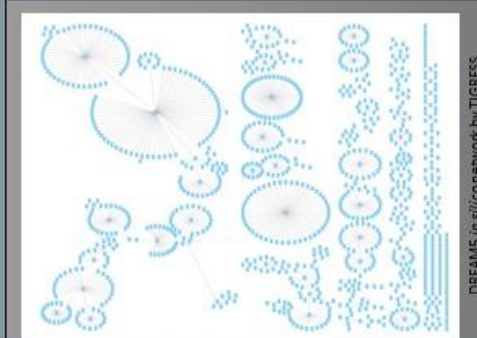
Apply threshold to remove unimportant interactions

**Inferring Biomarker by DNB**

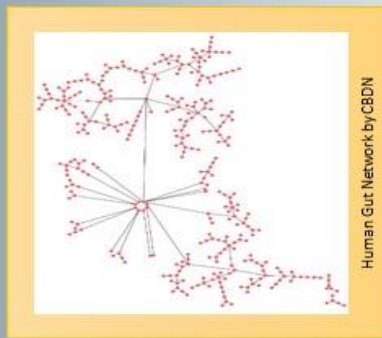
Choose differential expression genes

Cluster genes at each sampling time

Determine dominant group of DNB



DREAMS in silico network by TIGRESS



Human Gut Network by CBDN

**Constructing Regulatory Network by CBDN**

Compute influence of each gene to another by PCN

Remove transitive interaction

Calculate total influence value of each gene to all other gene

Rank the total influence value to find important regulator

**Imputation**

Construct network

Choose differential expression bacteria

Expand the seed to first degree neighbours

Identify overlapping clusters

Calculate imputation value and impute into the missing value

**Discussion**

CBDN and TIGRESS are the methods that can infer a biological network. However, CBDN requires minimal information to construct a regulatory network whereas TIGRESS requires additional information in order to infer directionality of edges. Biomarker is a biological sign or medical indicator for biological processes and state. Inference of biomarker on metagenomics data can let researchers to retrieve more information from the data. For instance, biomarker can act as an early warning sign to predict risk of getting diseases. Imputation is used to recover missing value in abundance matrix so that researchers may obtain accurate and precise output from the analysis.

**Conclusion**

In a nutshell, we would like to come up a comprehensive platform that provide a series of analysis functions to researchers. We do hope that our contribution can facilitate the study of microbiology as there are many microbes out there still remain unexplored. At the end of the project, we delivered 4 modules. However, some of them were tested by genomics data instead of metagenomics data because they do not have proper and complete dataset and some of them require very long processing time to complete the test.

By: Tan Kok Hoong

Supervised By: Dr. Ng Yen Kaow

# PLAGIARISM CHECK SUMMARY

FYP2 FYP2 report - DUE 31-Jul-2017

Originality GradeMark PeerMark ONLINE TOOLS BY KOK HOONG TAN turnitin 4% SIMILAR -- OUT OF 0

This project developed four tools for studying metagenomics data. These four modules will become part of an online platform created by researchers from the City University of Hong Kong for metagenomics use.

The first tool constructed regulatory network module using the CDBN algorithm proposed by (Zhang, Ng & Li 2015). The algorithm constructed a directed biological network from gene expression data. The second tool implemented TIGRESS, another popular method for the inference of biological network. The third tool in this project was a tool for inferring biomarker that was based on the DNB method introduced by (Liu et al. 2012). The fourth and final tool was meant for imputation. The tool aimed to allow users to recover missing values in an abundance matrix.

As these four modules meant for the analysis of very massive data sets, their performance was important. In order to speed-up these tools, the Intel Threading Building Blocks (TBB) was used, a library that facilitated the distribution of processes to CPUs.

### Match Overview

3 matches

1	tolstoy.newcastle.edu.au Internet source	1%
2	doc.aporc.org Internet source	1%
3	cbio.ensmp.fr Internet source	1%
4	Haury, Anne-Claire, Fa... Publication	<1%
5	bmcgenomics.biomedc... Internet source	<1%
6	Li, Guangzhong, Jiaqin... Publication	<1%
7	amdec-bioinfo.cu-geno... Internet source	<1%
8	saicdods.saicocean.com Internet source	<1%
9	"Security in Computing... Publication	<1%
10	Kiani, Narsis A., Hector... Publication	<1%
11	"Research Results fro... Publication	<1%

PAGE: 1 OF 42 Text-Only Report



# PLAGIARISM CHECK SUMMARY

**turnitin** Originality Report

Processed on: 10-Apr-2017 00:23 MYT  
ID: 796668706  
Word Count: 7435  
Submitted: 1

ONLINE  
AN  
MET

Similarity Index  
**4%**

Similarity by Source	
Internet Sources:	3%
Publications:	3%
Student Papers:	N/A

Document Viewer

DATA  
By Kok Hoong Tan

include quoted include bibliography excluding matches < 8 words mode: show highest matches together

This project developed four tools for studying metagenomics data. These four modules will become part of an online platform created by researchers from the City University of Hong Kong for metagenomics use. The first tool constructed regulatory network module using the CDBN algorithm proposed by (Zhang, Ng & Li 2015). The algorithm constructed a directed biological network from gene expression data. The second tool implemented TIGRESS, another popular method for the inference of biological network. The third tool in this project was a tool for inferring biomarker that was based on the DNB method introduced by (Liu et al. 2012). The fourth and final tool was meant for imputation. The tool aimed to allow users to recover missing values in an abundance matrix. As these four modules meant for the analysis of very massive data sets, their performance was important. In order to speed-up these tools, the Intel Threading Building Blocks (TBB) was used, a library that facilitated the distribution of processes to CPUs. 1.1 Motivation and Problem Statement The general availability of Next-generation Sequencing (NGS) technology in the past decade has brought forth many advances in genomics. In particular, NGS techniques have been successfully applied to large-scale mixtures of unbiased genetic material, thus encouraging the analysis of whole-community of organisms in one single study, or metagenomics. In this final year project, we were collaborating with City University of Hong Kong to come up a comprehensive platform for researchers to

- 1% match (Internet from 11-Mar-2017)  
<http://tolstoy.newcastle.edu.au>
- 1% match (Internet from 15-Mar-2016)  
<http://cbio.ensmp.fr>
- 1% match (Internet from 22-May-2016)  
<http://doc.aporc.org>
- < 1% match (publications)  
[Haury, Anne-Claire, Fantine Mordelet, Paola Vera-Licona, and Jean-Philippe Vert. "TIGRESS: Trustful Inference of Gene REgulation using Stability Selection", BMC Systems Biology, 2012.](#)
- < 1% match (Internet from 23-Sep-2016)  
<https://bmcgenomics.biomedcentral.com/article/10.1186/s12859-016-2791-2>
- < 1% match (publications)  
[Li, Guangzhong, Jiaqing Zhu, and Jie Li. "Understanding bilateral exchange rate risks", Journal of International Money and Finance, 2016.](#)
- < 1% match ()  
<http://amdec-bioinfo.cu-genome.org>
- < 1% match (Internet from 13-May-2015)  
<http://saicdods.saicocean.com>
- < 1% match (publications)  
[Kiani, Narsis A., Hector Zenil, Jakub Olczak, and Jesper Tegner. "Evaluating network](#)

**PLAGIARISM CHECK SUMMARY**

<b>Universiti Tunku Abdul Rahman</b>			
<b>Form Title : Supervisor’s Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1



**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	Tan Kok Hoong
<b>ID Number(s)</b>	1302418
<b>Programme / Course</b>	Computer Science
<b>Title of Final Year Project</b>	Online Tools For Analyzing Metagenomics Data

<b>Similarity</b>	<b>Supervisor’s Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)</b>
<b>Overall similarity index: _____ %</b>  <b>Similarity by source</b> Internet Sources: _____ % Publications: _____ % Student Papers : _____ %	
<b>Number of individual sources listed of more than 3% similarity: _____</b>	
<b>Parameters of originality required and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

***Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.***

\_\_\_\_\_  
Signature of Supervisor  
  
Name: \_\_\_\_\_  
  
Date: \_\_\_\_\_

\_\_\_\_\_  
Signature of Co-Supervisor  
  
Name: \_\_\_\_\_  
  
Date: \_\_\_\_\_

