2014

Inferring Gene Regulatory Networks from Expression Data using Ensemble Methods

Janusz Slawek
Virginia Commonwealth University

Inferring Gene Regulatory Networks from Expression Data using Ensemble methods

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

Janusz Sławek

Master of Science from AGH University of Science and Technology in Cracow, 2010

Bachelor of Science from AGH University of Science and Technology in Cracow, 2010

Director: Dr. Tomasz Arodź

Department of Computer Science, School of Engineering

Virginia Commonwealth University

Richmond, Virginia

April, 2014

Contents

List of Tables

List of Figures

List of Algorithms

Abstract

Inferring Gene Regulatory Networks from
Expression Data using Ensemble methods

By Janusz Sławek

A thesis submitted in partial fulfillment of the requirements for the degree
of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2014.

Director: Dr. Tomasz Arodź
Department of Computer Science, School of Engineering

High-throughput technologies for measuring gene expression made inferring of the genome-wide Gene Regulatory Networks an active field of research. Reverse-engineering of systems of transcriptional regulations became an important challenge in molecular and computational biology. Because such systems model dependencies between genes, they are important in understanding of cell behavior, and can potentially turn observed expression data into the new biological knowledge and practical applications. In this dissertation we introduce a set of algorithms, which infer networks of transcriptional regulations from variety of expression profiles with superior accuracy compared to the state-of-the-art techniques. The proposed methods make use of ensembles of trees, which became popular in many scientific fields, including genetics and bioinformatics. However, originally they were motivated from the perspective of classification, regression, and feature selection theory. In this study we exploit their relative variable importance measure as an indication of the presence or absence of a regulatory interaction between genes. We further analyze their predictions on a set of the universally recognized benchmark expression data sets, and achieve favorable results in compare with the state-of-the-art algorithms.

Chapter 1

Introduction

Cellular phenotypes are determined by large networks of regulated genes, which are also called Gene Regulatory Networks (GRNs) or Transcriptional Regulatory Networks (TRNs) [87]. A problem of reverse-engineering GRNs from expression data is a long-standing challenge in molecular and computational biology. High-throughput technologies for measuring gene expression, such as microarrays and RNA-Seq, made it possible to collect genome-scale snapshots of gene expression across different experiments, such as diverse treatments to cells, which in result made inferring large networks of transcriptional interactions an active field of research [27]. Challenges principally arise from the nature of the data: they are typically noisy, high dimensional, and sparse [41]. Therefore, some of the regulatory interactions might be missed by the inference methods, causing false negative predictions. Moreover, discovering causal relationships between products of genes is not a trivial task without dedicated experiments. Especially, frequent recurrent patterns observed in transcriptional networks, also called network motifs, often confuse the inference algorithms [74]. An example of such a misclassification is predicting the presence of a regulatory interaction between two co-regulated genes, i.e. genes controlled by the same regulator. Even though their expression might be relatively strongly correlated, they are not in a regulatory relation with each other, and cause false positive predictions. Therefore by far the biggest challenge

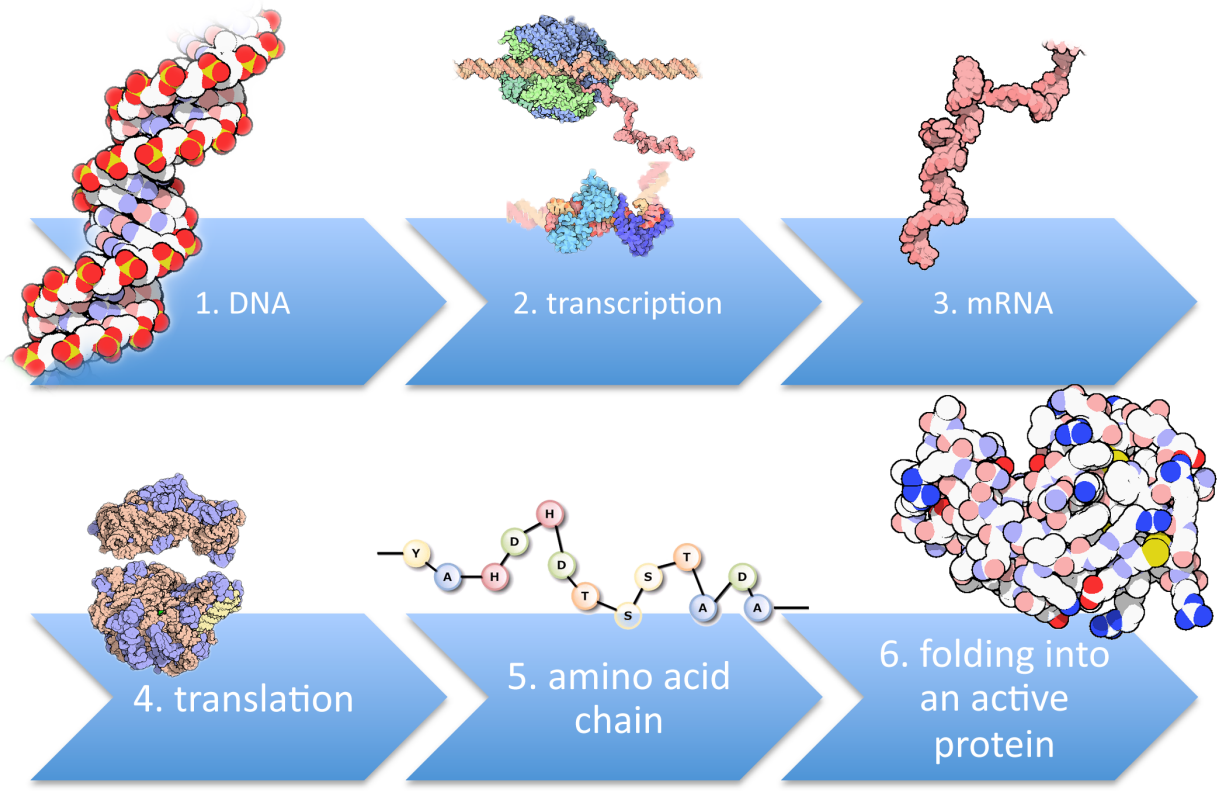

Figure 1.1: Gene expression in essence is a three-step process. In a process of transcription (2) a region of DNA (1) is transcribed into RNA (3). In a process of translation (4) a RNA transcript is translated into a chain of amino acids (5). A ribosome (4) serves as a "workbench", and a RNA transcript serves as a "recipe" for creating such amino acid chains. In a process of folding an amino acid chain is folded into an active protein (6). Transcriptional regulation occurs only in a process of transcription (2). Protein structures taken from PDB [8] .

in reverse-engineering GRNs is to design an universal off-the-shelf algorithm which would process non-specific expression data and calculate highly accurate predictions for a wide range of organisms.

The scientific community demands turning genome-scale data, such as genome-wide expression profiles, into fundamentally new biological knowledge and practical applications. However, raw expression data is of limited use for furthering biological understanding. Inference of transcriptional regulatory networks is an example of a computational analysis

of expression data, which can possibly support new biological discoveries, help cure diseases, or engage cells in biomaterial production [94]. For example, for a newly sequenced organism, interactions inferred from gene expression data can serve as the first draft of a genome-wide transcriptional regulatory network. Based on that raw approximation, network regions utilized in a condition-specific manner might identify the list of active subnetworks. Those identified as responsive to drug treatment can be studied in detail to combat drug resistance. Apart from identifying functional modules of a regulatory network, inference algorithms might be helpful in predicting behavior of a system following perturbations, a situation often encountered in the drug discovery process [56]. However, to identify real physical interactions between gene products, expression profiles must be reinforced with additional sequence data. In other words, expression data alone can only indicate if a particular regulatory interaction occurs, but do not explain how it happens.

Gene expression is a process in which proteins are synthesized based on instructions encoded into DNA [60], as shown in Figure 1.1. RNA molecules play an important role in this process. Francis Crick, a Nobel Prize laureate, who co-discovered the structure of the DNA molecule in 1953 together with James Watson, summarized the relationships among DNA, RNA, and protein by what he called the central dogma of molecular biology: *"DNA directs its own replication and its transcription to RNA which, in turn, directs its translation to proteins"* [22]. In essence, gene expression is considered a three-step process:

1. RNA polymerase transcribe a region of DNA, i.e. a gene, into a messenger RNA (mRNA) molecule, also called a transcript. Many eukaryotic genes contain introns and exons fragments. Introns do not correspond to amino acid sequences in the expressed protein. The splicing mechanism at the mRNA level permits alternative combinations of exons, which are merged into one piece of message and ultimately translated into a protein. This process greatly increases variability of synthesized proteins.

2. Ribosome process the mRNA transcript and translate it into a polypeptide chain.

3. The polypeptide chain undergoes many post-translational modifications and folds into a biochemically active protein.

The above process allows for a variety of different forms of expressed proteins, e.g., alternatively spliced forms. However, in the process of reverse-engineering transcriptional regulatory networks we only consider those that are measured in expression profiles. If different isoforms of genes are available in expression data we process them as if they were different genes.

Regulatory molecules, also called transcription factors (TF), control the formation of transcripts in the transcription phase. These regulators are usually fully-formed proteins: repressors or inducers. Even though in eukaryotes smaller molecules other than proteins, such as siRNA and miRNA, also play an important role in that process, we only consider regulations between transcription factors and regulated genes. A major turning point in our understanding of the regulation mechanisms of gene expression was the discovery of operons. In essence an operon is a functioning unit of DNA containing a cluster of genes under the control of a single regulatory signal or a promoter, i.e. a region of DNA that initiates transcription of a particular gene. Moreover, a transcription factor can only bind to DNA at a specific position. Such a position, or a binding site, is determined by the sequence of nucleotides in DNA. As a consequence, an operon constitutes a well-defined system of genes and regulatory interactions. In fact, for the discovery of the *lac* operon in *E. coli*, which is responsible for lactose uptake and utilization, François Jacob, André Michel Lwoff, and Jacques Monod were awarded a Nobel Prize, since it was indeed a true breakthrough in genetics and molecular biology.

Regulation of an operon occurs through repressors or inducers. They are DNA-binding proteins that prevent or initiate the transcription of genes, respectively. Because both

types of transcriptional regulations might be positive or negative, there are four possible operons:

1. **Positively inducible operons** are controlled by activator proteins, which under normal circumstances do not bind to the regulatory DNA region. Transcription is switched on only if a specific inducer binds to the activator.

2. **Positively repressible operons** are also controlled by activator proteins, but these are normally bound to the regulatory DNA region. When a repressor protein binds to the activator, it dissociates from the DNA and transcription is switched off.

3. **Negatively inducible operons** are controlled by repressor proteins, which are normally bound to the operator of the operon and in consequence prevent transcription. Specific inducers can bind to repressors, so that binding to DNA is no longer effective, and thus initiate transcription.

4. **Negatively repressible operons** are normally transcribed. However, they have binding sites for repressor proteins, which prevent transcription only if a specific co-repressors are present. Binding of the repressor and co-repressor prevents transcription.

A repressor mode is advantageous if a regulated gene determines a function that is in low demand within the organism's natural environment. By contrast, if the function is in high demand, it is the activator that controls the corresponding gene. Operons occur primarily in prokaryotes but also in some eukaryotes, including nematodes such as *C. elegans* and the fly, *Drosophila melanogaster*. The control of gene expression is far more complex in eukaryotes than in bacteria, however the same basic principles apply. As in prokaryotes, transcription is controlled by proteins that bind to specific motifs and modulate the activity of RNA polymerase.

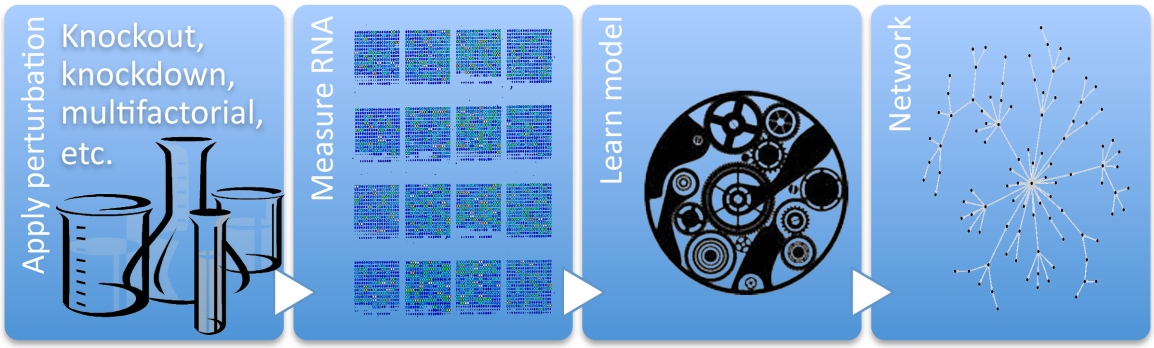

Figure 1.2: The general strategy for reverse-engineering transcription control systems. (1) Cells are initially perturbed with various treatments to elicit distinct responses. (2) After each perturbation, the expression (concentration) of RNA transcripts in the cells is measured. (3) A learning algorithm calculates the parameters of a model that describes the transcription control system underlying the observed responses. (4) The resulting model may then be used in the analysis and prediction of the control system function.

Some transcription factors might be responsible for several operons, and a single operon may be regulated by several transcription factors as well. A convenient representation of such a many-to-many relationship between transcription factors and regulated genes is a network of regulatory interactions: a gene regulatory network. It hides biological complexity of a transcriptional regulation of mRNA expression, and model complex molecular interactions as a single edge in a graph [42, 41, 44]. In other words, physical relationships between regulators and transcripts are reduced to a simple concept: is a particular transcription factor involved in regulatory interaction with a particular transcript or not. In this way, gene regulatory networks emphasize functional dependencies between genes, and therefore are important in understanding of cell behavior. As shown in Figure 1.2, using expression data as input, an inference method predicts GRN defined as a set of nodes representing genes, and directed edges representing regulatory interactions between them. GRNs are usually conveniently represented as directed graphs, as shown in Figure 2.1. In such a graph nodes correspond to genes, and edges correspond to the regulatory interactions between transcription factors and genes. Transcription is often hierarchically controlled, which

means that transcription factors not only regulate expression of the end product proteins, but also formation of the other transcription factors. Therefore, exploring a complex system of regulatory interactions is a challenge. Rigorous computational methods, including the ones described later in this study, are of great importance for biological discovery.

1.1 Contributions of the Thesis

In this study we predict gene regulatory networks from variety of expression data types of diverse origins. Even though it is a long-standing challenge in molecular and computational biology, the individual state-of-the-art methodologies do not show robust performance across different networks. We integrate variety of expression profile types and design two algorithms ADANET and ENNET, which perform robustly across diverse data sets. Moreover, the algorithms do not assume a specific mathematical model of a transcriptional regulation, e.g., a linear model, and do not require fine-tuning of their parameters, which promises accurate predictions for the future networks. We construct genome-wide networks of regulatory interactions, which include up to few hundreds transcription factors, and up to few thousands regulated genes. The proposed algorithms compare favorably to the state-of-the-art methods on the universally recognized benchmark networks. The major contributions of this dissertation are:

1. Integrating different kinds of expression data under the single algorithm of inferring gene regulatory networks.

2. Introduction of gene regulatory algorithms:

   (a) ADANET, which uses boosted ensemble of decision stumps.

   (b) ENNET, which uses boosted ensemble of regression stumps and perform robustly across diverse data sets.

3. Introduction of two algorithms of refining the prediction:

   (a) Analyzing variance of the edge predictions.

   (b) Analyzing the effects of a possible regulation.

4. Constructing regulatory networks on a genome scale, which consist of up to few hundreds transcription factors, and up to few thousands regulated genes.

5. Quantitative analysis of the results achieved by the proposed methods and the comparison with the state-of-the-art algorithms.

The rest of this dissertation is organized as follows:

In Chapter 2 we describe the background of our research. In Section 2.1 we characterize the existing inference methods, their working principles, advantages, and disadvantages. Later, in Section 2.2, we introduce benchmark regulatory networks, which we use as a gold standard to evaluate our proposed algorithms, together with the state-of-the-art methods. In addition to this, in Section 2.3, we present methods of evaluating the inference algorithms, including the analysis of Precision-Recall and ROC curves. Also, in Section 2.4, we discuss different types of expression profiles, which we use as an input for our predictors.

In Chapter 3 we give theoretical foundations of building an ensemble of weak predictors, and using their weighted outcomes as a final prediction. We distinguish between parallel ensembles, which we describe in Section 3.1, and serial ensembles, which are characterized in Section 3.2. On top of that, we describe how their relative importance measure of variables can indicate the presence or absence of a regulatory interaction between genes.

In Chapter 4 we describe our proposed approaches to the problem of Gene Regulatory Network inference. In Section 4.1 we give an overview of the main steps of our proposed procedure, leading to the calculation of a network of regulatory interactions. In Section 4.2

we explain in details the process of creating independent gene selection problems, whereas in Section 4.3 we describe how we solve them using two new algorithms, ADANET and ENNET. Moreover, in Section 4.4 we introduce two methods of re-ranking our predictions to further improve the final results.

In Chapter 5 we present the quantitative analysis of our proposed algorithms, and compare them with the state-of-the-art methods. We report our predictions in several sections, each of them corresponding to one popular benchmark regulatory network. For large networks of regulatory interactions we also present Precision-Recall dependency of the predictions in more details.

In Chapter 6 we discuss the results we achieved on benchmark data sets, as well as the general properties of our proposed methods. In Section 6.1 we present the default set of parameters of ADANET, one of the proposed algorithms. In Section 6.2 we calculate its computational complexity. In Section 6.3 we discuss in details the choice of parameters of ENNET algorithm. In Section 6.4 we show computational complexity of ENNET. Finally, in Section 6.5 we show that our proposed method calculates stable predictions.

Finally, in Chapter 7, we conclude this dissertation and present future directions of our research.

Chapter 2

Background

High throughput techniques allow for collecting genome-wide snapshots of gene expression across different experiments, such as diverse treatments and other perturbations to cells [27]. Analyzing these data to infer the regulatory network is one of the key challenges in the computational systems biology. The difficulty of this task arises from the nature of the data: they are typically noisy, high dimensional, and sparse [41]. Moreover, discovering direct causal relationships between genes in the presence of multiple indirect ones is not a trivial task given the limited number of knockouts and other controlled experiments. Attempts to solve this problem are motivated from a variety of different perspectives. Most existing computational methods are examples of the influence modeling, where expression of a target transcript is modeled as a function of expression levels of transcription factors. Such a model does not aim to describe physical interactions between molecules, but instead uses an inductive reasoning to find a network of dependencies that could explain the regularities observed among the expression data. In other words, it does not explain mechanistically how transcription factors interact with regulated genes, but indicate candidate interactions with strong a evidence in expression data. This knowledge is crucial to prioritize a detailed study of the mechanics of the transcriptional regulation.

2.1 Existing inference methods

Researchers devoted considerable attention to the problem of reverse-engineering gene regulatory networks from expression data. Solutions to this problem are motivated from variety of different perspectives, which are grouped into the following categories in this study:

1. Boolean network models.
2. Probabilistic graphical models: Bayesian Networks (BN) and Markov Networks (MN).
3. Ordinary differential equations (ODE) models.
4. Correlation and information theoretic models.
5. Machine learning models.
6. Meta-predictors.

All of the above methods, except from meta-predictors, might be considered as different examples of the *influence* models. In such an approach, an expression of a target transcript is a response determined by the expression of its transcription factors. Such a model does not describe physical interactions between molecules, but rather finds a network of regulatory interactions, which could explain the regularities observed among the expression data. In that sense, the above methods are *inductive* learning techniques for inferring new information from data. The two major limitations of inductive techniques with respect to gene regulatory networks are the following:

1. The first limitation is that the model can be difficult to interpret in terms of the physical structure, and therefore difficult to integrate or extend with further research. In other words, a model inducted from expression data does not explain how a certain transcription factor regulates a certain target gene.

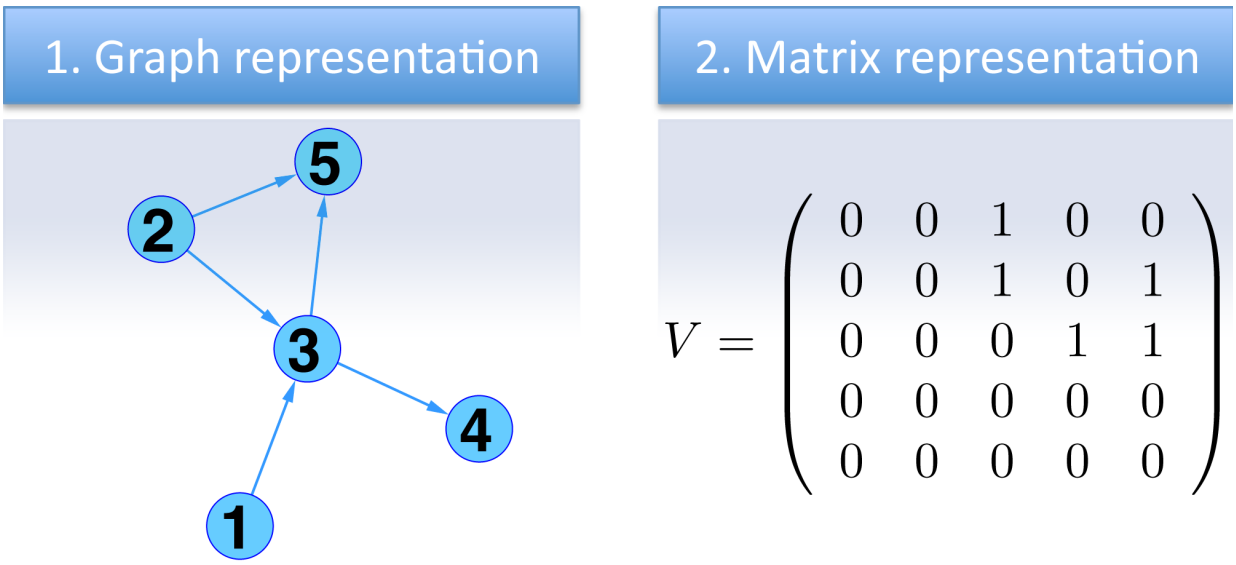

Figure 2.1: An example transcript control network with five transcripts and five connections (1). Different models may be used to represent the relationships between products of genes, but all of them are designed to derive an adjacency matrix V (2). Rows of V correspond to outgoing edges from a single node, columns correspond to incoming edges to a single node. For example, an edge from node 3 to node 5 is depicted in V as binary "1" in in 3-rd row and 5-th column.

2. The second limitation is that inductive models are correct for the input data but only plausible outside of them [21]. In other words, a network inferred from one set of expression data can not be assumed correct for the other one.

In spite of these evident limitations, a vast of gene regulatory network inference algorithms use inductive reasoning, as shall be described below at this section.

2.1.1 Boolean network models

The first group of methods describe GRN as a system of binary variables, as shown in Formula 2.1, where a gene takes one of the two states: expressed, or not-expressed:

$$\psi_{t+1}(X_i) = f_i^b(\psi_t(X_1), ..., \psi_t(X_n)), \tag{2.1}$$

where ψ is an expression pattern representing the states of genes, X_i is expression of the target transcript across different experiments, and $\{X_1, ..., X_n\}$ is a set of expressions of its potential transcription factors across different experiments. A binary state of a transcript is determined by a Boolean function f_i^b of the binary states of the input transcripts. Here the output expression pattern ψ_{t+1} at time t+1 is determined by input expression pattern ψ_t at time t. A discretization scheme ψ must be first introduced to convert concentration levels of transcripts X into binary values $\psi(X)$. This task might be a challenge because measurement error on expression data is often large [1]. After discretization phase, there are two main strategies to learn the topology of regulatory interactions:

1. The first one finds the minimal set of input TFs whose expression provides complete information on the output transcript, with the use of Mutual Information (MI) between them [58].

2. The second approach uses Occam's Razor paradigm of finding the most parsimonious set of input TFs whose expression is coordinated or consistent with the output transcript [2, 50, 55].

Both approaches may apply some additional post-processing to improve the resulting inferred network by removing redundant connections. Nevertheless, Boolean logic requires considerably more data samples than the other approaches [25]. Among other issues, this limitation impedes their practical use in a process of reverse-engineering GRNs [97].

2.1.2 Probabilistic graphical models

Probabilistic graphical models exploit causal relationships between RNA transcripts [41]. They analyze multivariate joint probability distributions over the observations, usually with the use of Bayesian Networks (BN) [39,72,96] and Markov Networks (MN) [83]. A Bayesian Network represents the state of a transcript as a random variable, which is specified by the

joint probability distribution function, as shown in Formula 2.2:

$$P(X_1, ..., X_n) = \prod_i P(X_i, U_i), \tag{2.2}$$

where $U_i \subseteq X$ is a set of variables that directly influence value of X_i, also called parents of X_i. The graphical representation is given by a directed graph, see Figure 2.1, where directed edges are originated from parents U_i to X_i. The product decomposition in Equation 2.2 is guaranteed to be a coherent probability distribution if a directed graph is acyclic. In Markov Networks the multivariate joint probability P is represented as a product of potentials, each of which captures the interaction among a set of variables and specifies the "desirability" of joint value assignments to these variables [38], as shown in Formula 2.3:

$$P(X_1, ..., X_n) = \frac{1}{Z} \prod_j \pi_j[Cj]. \tag{2.3}$$

Here $\pi_j[Cj]$ is the j-th potential over the variables $C_j \subseteq X$, and Z is a normalizing constant which ensures that the total probability mass is 1.

An objective of graphical modeling is to learn a network of regulatory interactions which fits to the underlying distribution from which the observations were made. It is done in two steps:

1. **Model selection.** In this step a general structure of a model is selected, which best reflects the dependencies between gene products. For example, one can choose if the multivariate joint probability P should be represented as in Equation 2.2 or as in Equation 2.3. Usually, there is a trade-off between the complexity of a model and the number of parameters to learn, and consequently the amount of the expression data required for a statistically significant estimate. Computational overhead is also a limiting factor.

2. **Parameter estimation**. Once the general structure of a graphical model is selected, the parameters of the conditional probabilities are estimated. This task is often addressed as an optimization problem, of which the exact form depends on the selection made in the first step. The final adjacency matrix V, see Figure 2.1, is predicted from the parameters learned in this step.

The form of the conditional probability function is usually implied a priori, including Boolean and linear functions, which helps to minimize the number of parameters of the model to learn. An advantage of Bayesian Network models over Boolean models is the fact that a set of partially complete distribution functions are sufficient to determine the topology of a network, which helps overcoming a problem of the training data being incomplete. However, since learning optimal Bayesian networks from expression data is NP-hard, i.e. too computationally intensive for any existing computer, most networks of a practical interest are far too large for the exact methods to be feasible. Even finding approximations to the optimal network is computationally infeasible for problems involving several thousands of genes. Various heuristic search schemes must be applied in order to find parameters of a model, such as greedy-hill climbing or Markov Chain Monte Carlo approach [71], and consequently infer the network. Nevertheless, ease of incorporating of prior knowledge to the structure of a regulatory network and the ability to deal with undersampled data made Bayesian networks popular in reverse-engineering of small GRNs [39, 70, 81, 82].

2.1.3 Ordinary differential equations models

Another group of methods describe GRN as a system of differential equations, as shown in Formula 2.4:

$$\frac{dX_i}{dt} = f_i(X_1, ..., X_n). \tag{2.4}$$

The rate of change in expression of transcript X_i is given by a function f_i of the concentration levels of its transcription factors $\{X_1, ..., X_n\}$. Similar to probabilistic graphical

models, network inference is obtained in two steps: model selection and parameter estimation. Popular models imply linear functions f_i a priori [20, 26, 42, 95] because they reduce the number of parameters to learn, which is especially important when predicting a network from sparse data sets. Bayesian Best Subset Regression (BBSR) [43] has been proposed as a novel model selection approach, which uses Bayesian Information Criterion (BIC) to select an optimal model for each target gene. The amount of data required to solve a linear model is much less than what more complex nonlinear models require. However, choosing such a simple relation comes at the cost of placing strong constraints on the mathematical model of a regulatory interaction.

Both time series expression profiles, and steady state data, as described later in Section 2.4, might be used as an input for ODE models. If an algorithm uses time series data, it estimates the rates of change of the target transcript, as given in Formula 2.4, from the series of experiments. Alternatively, if the algorithm uses steady-state data, than it does not need to calculate derivatives: $\frac{dX_i}{dt} = 0$. However, a measure of the external perturbation to the level of a target transcript is needed to keep it in balance with the steady-state rate of its synthesis. In either way, the number of experimental data points is typically less than the number of parameters to learn. Multiple solutions have been proposed to overcome that problem, usually implying additional constraints on the model and employing various search schemes [7, 93]. An example of such a constraint is a limiting threshold on the number of transcription factors that each transcript can have. Usually that number is much lower than the total number of possible transcription factors in a network. It does not seem to restrict the nature of a regulatory interaction, since only a small subset of TFs is observed to regulate a particular transcript.

2.1.4 Correlation and information theoretic models

The following approach is motivated from statistics and information theory. TwixTwir [75] uses double two-way t-test to score transcriptional regulations. A null-mutant z-score algorithm [74] scores interactions based on a z-score transformed knockout expression matrix. Various information theoretic algorithms are estimating and analyzing cross-correlation and mutual information matrix MI of gene expression [6, 17, 57, 66], including ANOVA η^2 method [54]. Of these two measures, mutual information estimates are by far more popular, as they capture non-linear relations in expression data. Formally, mutual information of the expression of a target transcript X_i and its transcription factor X_j, perceived as two discrete random variables, is defined in Formula 2.5:

$$\mathrm{MI}_{ij} = \sum_{x_j \in X_j} \sum_{x_i \in X_i} p(x_i, x_j) \log\left(\frac{p(x_i, x_j)}{p(x_i) \cdot p(x_j)}\right), \tag{2.5}$$

where $p(x_i, x_j)$ is the joint probability distribution function of X_i and X_j, and $p(x_i)$ and $p(x_j)$ are the marginal probability distribution function of X_i and X_j respectively. The simplest information theoretic model, known as a relevance network [17], ranks all the possible edges between X_i and X_j based on their score MI_{ij}, and predicts the presence of a regulatory interaction if the MI_{ij} coefficient is higher than a given threshold. Note that such a prediction is undirected, since $\mathrm{MI}_{ij} = \mathrm{MI}_{ji}$ for any given i and j. Mutual information coefficient does not determine the causal relationship between products of genes. In other words, even if a pair (X_i, X_j) is found to have a relatively high MI_{ij} coefficient, it is not clear which element of this pair is a transcription factor, and which is a regulated gene.

ARACNE [64, 65] method was proposed as an improvement to relevance network. It removes possibly indirect edges from triples of genes with the use of Data Processing Inequality (DPI). The DPI states that if the two genes X_a and X_c interact through a third one

X_b, i.e. in Figure 2.1 gene 1 interacts with gene 4 only through gene 3, then:

$$\mathrm{MI}_{ac} \leq \min\left(\mathrm{MI}_{ab}; \mathrm{MI}_{bc}\right). \tag{2.6}$$

Therefore the least of the three: MI_{ac}, MI_{ab}, MI_{bc} may indicate an indirect edge in the network. Another improvement to the relevance network was introduced in CLR [29] algorithm, which applies an adaptive background correction step to eliminate false correlations and indirect influences from MI matrix. Here, the final confidence measure for an edge between genes X_i, and X_j is given in Formula 2.7:

$$f(Z_i, Z_j) = \sqrt{Z_i^2 + Z_j^2}, \tag{2.7}$$

where Z_i and Z_j are the z-scores of MI_{ij} from the marginal distributions: $Z_i = |\frac{\mathrm{MI}_{ij} - \mu_i}{\sigma_i}|$, where μ_i and σ_i are the mean and the standard deviation of the empirical distribution of MI values, respectively. Finally, C3NET [3, 4] method further filters out irrelevant edges. It only keeps an edge between the a transcription factor and a transcript if their shared MI value is at least for one of them maximal with respect to all the other transcripts. Another method, NARROMI [98], eliminates redundant interactions from MI matrix by applying the ODE-based recursive optimization, which involves solving a standard linear programming model.

2.1.5 Machine learning models

Recently, machine learning theory, especially classification, regression, and feature selection theory, successfully applied in various other domains, brought promising new possibilities to the field of inferring biological networks. One example of such an algorithm is MRNET [68], which applies maximum relevance/minimum redundancy (MRMR) [24] feature selection method. It formulates the network inference problem as a series of supervised gene selection procedures, where each gene in turn is designated as the target output. The method ranks

the set of transcription factors according to the difference between the mutual information with the target transcript (maximum relevance) and the average mutual information with the previously ranked transcription factors (minimum redundancy). In other words, it is a greedy search approach, which starts by selecting the transcription factor X_1 having the highest coefficient MI_{k1} to the target transcript X_k. The second selected transcription factor X_2 is the one with a high MI_{k2} coefficient to the target transcript and at the same time a low MI_{12} coefficient to the previously selected transcription factor. In the following steps, given a set S of already selected transcription factors, the algorithm updates S by choosing the transcription factor X_j as shown in Formula 2.8:

$$S = S \cup \{\arg \max_{X_j \in S'} (u_j - r_j)\}, \tag{2.8}$$

where S' is a complement of S with respect to the set of all the transcription factors, u_j is a relevance term: $u_j = \mathrm{MI}_{kj}$, and r_j is a redundancy term: $r_j = \frac{1}{|S|} \sum_{X_i \in S} \mathrm{MI}_{ki}$. At each step of the algorithm the selected transcription factor is expected to balance a trade-off between relevance and redundancy.

Another example of a machine learning inference procedure is recently published GENIE3 [51] algorithm, which makes use of Random Forest algorithm to score important transcription factors. Random Forest algorithm is described in details in Chapter 3. In essence, it exploits the embedded relative importance measure of input variables as a feature ranking criterion in a series of gene selection procedures, similar to MRNET algorithm. However, unlike MRNET algorithm, it combines a set of regression trees, and use their averaged outcome as a final prediction of a network of regulatory interactions. TIGRESS [46] follows a similar approach but is based on the least angle regression (LARS).

A possible improvement to the accuracy of any learning algorithm might be done with

the use of boosting [30, 79]. It is a technique for empirical risk minimization of exponential loss function [13, 35, 80], first applied in AdaBoost [32, 33] algorithm. This approach has been used in OKVAR-Boost [59] method. Our ADANET and ENNET algorithms for reverse-engineering transcriptional networks use respectively boosted ensemble of decision stumps and boosted ensemble of regression stumps to score important transcription factors. Serial ensembles, like Random Forest algorithm, and parallel ensembles, like Adaboost algorithm, will be later described in details in Chapter 3.

2.1.6 Meta-predictors

It is known, and will be shown empirically later in Chapter 5, that not a single inference method performs optimally across all the popular expression data sets. Therefore, the last group of inference methods, also called meta-predictors, use not only one approach to reverse-engineer a GRN, but rather combine multiple different strategies. For example a meta-predictor could combine predictions made by all the inference methods described above in this chapter. In this way a consensus network of regulatory interactions is created by re-ranking interactions according to their average score across all the participating methods. It has been empirically shown [61] that the community networks are consistently as good or better than the top individual methods. It is especially important that such a meta-predictor performs robustly across diverse data sets, since this is by far the most important quality of an inference method. Only then such an algorithm promises accurate predictions for unknown regulatory networks, given the biological variation among organisms and the experimental variation among gene-expression data sets.

A possible explanation of the superior performance of the community method is the fact that they exploit the diversity of the participating inference methods. In other words, individual inference methods might be biased towards discovering certain types of regulatory interactions. Due to the stabilizing effect of averaging, a community score turns out to be

more reliable. Moreover, it has also been empirically shown [61] that consensus predictions of methods which use similar methodologies, for example methods which belong to the same category of models, were outperformed by consensus predictions from diverse methodologies of the same size. Another interesting fact found in this study was that combining accurate individual inference models results in high-quality consensus prediction, whereas poor predictors essentially contribute noise. This highlights an importance of developing high-quality individual predictors.

Another explanation of the superior performance of the community method is the fact that they combine methods, which operate on diverse data. This second point is especially important in this study, because it gave foundations for an algorithm, introduced later in Chapter 4, which integrates different types of expression profiles. In other words, this study shows if processing diverse types of expression data by a single inference algorithm could be an alternative to integrating diverse types of individual predictors, which is done by a meta-predictor.

2.2 Benchmark gene regulatory networks

Researchers devoted considerable attention in recent years to the problem of evaluating performance of the inference methods on adequate benchmarks [61, 78]. Three main strategies have been proposed to generate benchmark GRNs:

1. Well studied *in vivo* networks from model organisms, such as those of *E. coli* [40], and *S. cerevisiae* [52].

2. Genetically engineered synthetic *in vivo* networks [18, 19].

3. Artificially simulated *in silico* networks [23, 53, 67, 78, 92].

Table 2.1: Summary of the data sets from the popular benchmarks for evaluating inference algorithms of gene regulatory networks.
#TFs denotes the number of known transcription factors, #RGs- the number of regulated transcripts, #TPs- the number of true regulatory interactions, #S- number of training samples including wildtype, knockout, knockdown, multifactorial, and time series expression data, as described in Section 2.4, D3- DREAM3, D4- DREAM4, D5- DREAM5, MF- multifactorial.

Network	model organism	expression data	#TFs	#RGs	#TPs	#S
D3 size 10 net. 1	*E. coli*	*In silico*	10	10	11	106
D3 size 10 net. 2	*E. coli*	*In silico*	10	10	15	106
D3 size 10 net. 3	*S. cerevisiae*	*In silico*	10	10	10	106
D3 size 10 net. 4	*S. cerevisiae*	*In silico*	10	10	25	106
D3 size 10 net. 5	*S. cerevisiae*	*In silico*	10	10	22	106
D3 size 50 net. 1	*E. coli*	*In silico*	50	50	62	585
D3 size 50 net. 2	*E. coli*	*In silico*	50	50	82	585
D3 size 50 net. 3	*S. cerevisiae*	*In silico*	50	50	77	585
D3 size 50 net. 4	*S. cerevisiae*	*In silico*	50	50	160	585
D3 size 50 net. 5	*S. cerevisiae*	*In silico*	50	50	173	585
D3 size 100 net. 1	*E. coli*	*In silico*	100	100	125	1168
D3 size 100 net. 2	*E. coli*	*In silico*	100	100	119	1168
D3 size 100 net. 3	*S. cerevisiae*	*In silico*	100	100	166	1168
D3 size 100 net. 4	*S. cerevisiae*	*In silico*	100	100	389	1168
D3 size 100 net. 5	*S. cerevisiae*	*In silico*	100	100	551	1168
D4 size 10 net. 1	N/A	*In silico*	10	10	15	136
D4 size 10 net. 2	N/A	*In silico*	10	10	16	136
D4 size 10 net. 3	N/A	*In silico*	10	10	15	136
D4 size 10 net. 4	N/A	*In silico*	10	10	13	136
D4 size 10 net. 5	N/A	*In silico*	10	10	12	136
D4 size 100 net. 1	N/A	*In silico*	100	100	176	411
D4 size 100 net. 2	N/A	*In silico*	100	100	249	411
D4 size 100 net. 3	N/A	*In silico*	100	100	195	411
D4 size 100 net. 4	N/A	*In silico*	100	100	211	411
D4 size 100 net. 5	N/A	*In silico*	100	100	193	411
D4 size 100 net. 1 MF	N/A	*In silico*	100	100	176	100
D4 size 100 net. 2 MF	N/A	*In silico*	100	100	249	100
D4 size 100 net. 3 MF	N/A	*In silico*	100	100	195	100
D4 size 100 net. 4 MF	N/A	*In silico*	100	100	211	100
D4 size 100 net. 5 MF	N/A	*In silico*	100	100	193	100
D5 net. 1	N/A	*In silico*	195	1643	4012	805
D5 net. 3	*E. coli*	*In vivo*	334	4511	2066	805
D5 net. 4	*S. cerevisiae*	*In vivo*	333	5950	3940	536
M^{3D}	*E. coli*	*In vivo*	160	1443	2873	466

The main disadvantage of the first group of well studied *in vivo* benchmark networks is the fact that experimentally confirmed pathways can not be assumed complete, regardless of how well is a model organism known. Such networks are assembled from known transcriptional interactions with a strong experimental support. The gold standard networks based on them are expected to have few false positives (FP). However, they contain only a subset of the true interactions, i.e. they are likely to contain many false negatives (FN). This implies additional error when evaluating network predictions because the false positive edges found by predictor can either by truly not regulatory interactions or novel interactions.

Because the second group of genetically engineered synthetic *in vivo* networks is only very small, it does not provide enough data to sufficiently evaluate inference methods on a genome scale. An example of such a network is a synthetic gene network of *S. cerevisiae* [18]. However, it consists of only five genes, which is not enough for a comprehensive assessment of the inference algorithms. Ideally, a library of diverse synthetic *in vivo* gold standards should be composed of variety of networks of different sizes and topologies. Unfortunately, such a library does not exist.

Therefore the last group of artificially simulated *in silico* networks is commonly used to produce artificial gene expression data. *In silico* benchmark data sets are easier, faster, and cheaper to reproduce than the real biological experiments. Also the amount of expression data is only limited by the user. The latest simulators [78] mimic real biological systems in terms of topological properties observed in biological *in vivo* networks, such as modularity [76] and occurrences of network motifs [84]. They are also endowed with dynamical models of transcriptional regulation, thanks to the use of non-linear differential equations and the other approaches [23, 45, 77], and consider both transcription and translation processes in their dynamical models [45, 47, 77] using a thermodynamic approach. Expression data can be generated deterministically or stochastically and experimental noise, such as

the one observed in microarrays, can be added [89]. Credibility of simulated expression data strongly depends on the quality of the simulator. Therefore accurate inference methods should perform well on both *in vivo* and *in silico* benchmarks.

Several popular benchmark GRNs are used to evaluate the accuracy of the inference methods in this proposal. They are summarized in Table 2.1 and Table 2.2. The vast majority of them come from Dialogue for Reverse Engineering Assessments and Methods (DREAM) challenges. These challenges address not only the inference of transcriptional regulatory networks, but also other problems of systems biology, especially in the area of cellular network inference and quantitative model building. Their main objective is a rigorous performance assessment of the strengths and weaknesses of the inference methods, which are presented with the same input data and validated against the same gold standards. Performance profiling reveals different types of systematic prediction errors and indicate potential directions of improvement.

2.3 Techniques for evaluation of inference methods

An inferred GRN is commonly represented as a directed graph in form of an adjacency matrix. Values in that matrix are interpreted as probabilities of directed edges being true transcriptional regulations, according to a particular inference method. Such a representation has an advantage of being convenient to evaluate given that the gold standard network is known, because inferred edges can be classified into two categories: an edge is a regulatory interaction (positive result), or an edge is not a regulatory interaction (negative result). As a consequence, common metrics of evaluating inference methods are derived from classification theory: a confusion matrix is created for a given acceptance threshold t. All the edges in adjacency matrix with the confidence higher or equal than t are considered regulatory interactions (positive results). All the other edges are considered not

being regulatory interactions (negative results). A confusion matrix for a given threshold t contains four values: a number of true positives (TP), a number of false positives (FP), a number true negatives (TN), and a number of false negatives (FN). Many methods of evaluating GRN inference methods have been proposed based on the above numbers, yet the most popular ones are the following:

1. Precision for the top ranked edges.

2. Analysis of Precision-Recall (PR) curve.

3. Analysis of Receiver Operating Characteristics (ROC) curve.

Above methods compare inferred adjacency matrix with the gold standard one. The first method gives a single number, which is a ratio of how many TP edges were found among top ranked N edges, as shown in Formula 2.9:

$$\text{precision}(t) = \text{PPV}(t) = \frac{\text{TP}(t)}{\text{TP}(t) + \text{FP}(t)}. \tag{2.9}$$

Precision is also known as Positive Predictive Value (PPV). In general $N = TP + FP$ can be any number, but commonly it is a number of all regulatory interactions known for a particular network. In other words a confusion matrix is calculated once for an acceptance threshold t, such that exactly N edges are predicted as regulatory interactions. Note that usually number N is much lower than the number of all the edges in a network.

On the other hand, methods based on a precision-recall or ROC curves calculate not only one but many confusion matrices for different acceptance thresholds t. A method based on precision-recall curve concentrates on the precision of an inference method with increasing range of accepted edges. A precision-recall curve is a plot of precision, see Formula 2.9, in function of recall. Recall, also known as True Positive Rate (TPR), or sensitivity, is defined

in Formula 2.10:

$$\text{recall}(t) = \text{TPR}(t) = \frac{\text{TP}(t)}{\text{TP}(t) + \text{FN}(t)}. \tag{2.10}$$

The area under precision-recall (AUPR) curve is bounded by 0 and 1, AUPR $\in (0, 1)$. AUPR close to 0 indicates that the prediction is poor for any given range of accepted edges. AUPR close to 1 indicates that the prediction is accurate even for a wide range of accepted edges.

A method based on ROC curve accents how much an inference method is sensitive to changes of t in terms of a True Positive Rate (TPR) and a False Positive Rate (FPR). True Positive Rate (TPR), or sensitivity, is measured in the same way as recall, as given in Formula 2.10, whereas False Positive Rate (FPR) is defined as shown in Formula 2.11:

$$\text{FPR}(t) = \frac{\text{FP}(t)}{\text{FP}(t) + \text{TN}(t)}. \tag{2.11}$$

False Positive Rate is related to specificity: $FPR = 1 - \text{specificity}$. ROC curve is a plot of the TPR, in function of FPR. The area under ROC curve is bounded by 0.5 and 1, AUROC $\in (0.5, 1)$. AUROC close to 0.5 indicates that the prediction is not better than the random guess. AUROC close to 1 indicates a highly accurate prediction.

In addition to the above metrics, a statistical evaluation of a prediction can be calculated by computing corresponding p-values: p_{aupr} and p_{auroc}. These are the probabilities that a random list of edge predictions would obtain the same or better area under precision-recall curve, and area under ROC curve, respectively. In order to obtain proper distributions for AUPR and AUROC an estimation is made from many instances of random lists of edge predictions. For example in DREAM challenges, described later in details in Sections 5.1, 5.2, and 5.3, distributions for AUPR and AUROC were estimated from 100 000 instances of random lists of edge predictions. The overall p-values: $\overline{p}_{\text{aupr}}$ and $\overline{p}_{\text{auroc}}$ of the

five networks constituting each DREAM3 and DREAM4 subchallenge were defined as the geometric mean of the individual p-values, as shown in Equation 2.12:

$$\overline{p} = \sqrt[5]{p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_5}. \tag{2.12}$$

The *Overall* score for each method was the log-transformed geometric mean of the overall AUROC p-value and the overall AUPR p-value, as shown in Equation 2.13:

$$\text{Overall} = -\frac{1}{2} \cdot \log_{10}(\overline{p}_{\text{aupr}} \cdot \overline{p}_{\text{auroc}}). \tag{2.13}$$

A definition of the overall score for DREAM5 network, described later in details in Section 5.3, was identical to the one for DREAM3 and DREAM4, but the overall p-values were averaged over only 3 individual p-values.

A drawback of the metric used to calculate the overall score in DREAM challenges is that if for at least one network the individual p-value is lower than the smallest positive real number available for a given machine precision, and therefore rounded to zero, then the overall score for the whole subchallenge is equal to infinity regardless of how well the other networks were predicted. This is because the overall p-value is equal to zero if at least one of the individual p-value is equal to zero, see Formula 2.12. As a result *Overall* score in Equation 2.13 is equal to infinity.

2.4 Variety of mRNA expression data

Network inference process highly relies on a type of expression data provided as input. Two main groups of expression profiles are: the one with known, and the one with unknown initial perturbation state of the expression of genes in the underlying network of regulatory interactions. Knockout, and knockdown data are provided with the additional meta-data,

Table 2.2: Different types of expression data provided in popular data sets: WT- Wildtype, KO- Knockouts, KD- Knockdowns, MF- Multifactorial, TS- Time series, • Available, ◦ Unavailable.
⋆) Even though all the data types are available, they are all considered MF in this study.

Data set	WT	KO	KD	MF	TS
DREAM3 size 10	•	•	•	◦	•
DREAM3 size 50	•	•	•	◦	•
DREAM3 size 100	•	•	•	◦	•
DREAM4 size 10	•	•	•	•	•
DREAM4 size 100	•	•	•	◦	•
DREAM4 size 100 MF	◦	◦	◦	•	◦
DREAM5⋆	•	•	•	•	•
Expression profiles of *E. coli*⋆	•	•	•	•	•

which describe which genes were initially perturbed. On the other hand, multifactorial and time series data are expression profiles of unknown initial state of genes.

Moreover, there are steady or unsteady state expression profiles, i.e. gene expression levels in homeostasis or changing dynamically with time respectively. Wildtype, knockout, knockdown, and multifactorial data describe the expression of initially perturbed genes, which are however in a steady state at the time of a measurement, where time series data describe the dynamics of the expression levels of initially perturbed genes. The types of data available in popular data sets are summarized in Table 2.2.

Some data sets might be more suitable for the GRN inference problem because they provide additional knowledge about initial perturbations of genes. However, discovering regulatory interactions in such data sets requires integrating different types of expression profiles and meta-data provided with them. We distinguish several possible scenarios:

1. We are given a wildtype expression profile in form of a single vector, one expression value for each gene. This vector might be useful to determine an expression level of unperturbed genes in a network of regulatory interactions, and thus could serve as a reference vector for expression profiles of other types. By itself however, it does not

give too much information about the underlying network of regulatory interactions. We operate on a wildtype expression vector WT:

$$\text{WT} = \{e_1, e_2, ..., e_i, ..., e_P\},$$

where e_i is expression value of i-th gene in the unperturbed network.

2. We are given expression profiles from null-mutant knockout or heterozygous knockdown experiments, one known gene initially perturbed in each experiment. As an input, we are given P expression profiles, each of them measuring mRNA expression levels of P transcripts. One known gene is knocked out at a time. We operate on $P \times P$ knockout expression matrix KO:

$$\text{KO} = \begin{pmatrix} e_{1,1} & e_{1,2} & . & . & e_{1,P} \\ e_{2,1} & . & & & \\ . & & e_{i,j} & & \\ . & & & . & \\ e_{P,1} & & & & e_{P,P} \end{pmatrix},$$

where $e_{i,j}$ is expression value of j-th gene in i-th sample. Columns of matrix KO correspond to genes, rows correspond to experiments. For our convenience we order KO matrix in a way that expression values on the diagonal correspond to initially perturbed genes. In other words, the first gene is knocked out in the first experiment (first row), the second gene in the second experiment (second row), and so on, up until the last gene is knocked out in the last experiment (last row). In a similar way we build $P \times P$ knockdown expression matrix KD:

$$\mathrm{KD} = \begin{pmatrix} e_{1,1} & e_{1,2} & . & . & e_{1,P} \\ e_{2,1} & . & & & \\ . & & e_{i,j} & & \\ . & & & . & \\ e_{P,1} & & & & e_{P,P} \end{pmatrix}.$$

This time, however, $e_{i,j}$ is expression value of j-th gene in i-th sample of knockdown experiment.

3. We are given expression profiles from diversely conditioned experiments of unknown initial perturbation state of genes. These could come from different drug treatments, stress, overexpression, RNA interference, etc. As an input, we are given N_{MF} expression profiles, each of them measuring mRNA expression levels of P transcripts. However, we relate to the columns of MF matrix as to genes, not transcripts, to keep the set of columns of all the data tables consistent.

   We operate on $N_{MF} \times P$ expression matrix MF:

$$\mathrm{MF} = \begin{pmatrix} e_{1,1} & e_{1,2} & . & . & e_{1,P} \\ e_{2,1} & . & & & \\ . & & e_{i,j} & & \\ . & & & . & \\ e_{N_{MF},1} & & & & e_{N_{MF},P} \end{pmatrix},$$

   where $e_{i,j}$ is expression value of j-th gene in i-th sample. Columns of matrix MF correspond to transcripts (genes), rows correspond to samples. Because of the known initial perturbation state- one known gene perturbed in each experiment, previous scenario seems to be much more suitable for the network inference problem. However, it requires knocking out each transcription factor, and therefore is much more expensive, and consequently less feasible in practice.

4. We are given timestamped expression profiles from diversely conditioned experiments of unknown initial perturbation of genes. As an input, we are given N_{TS} sets of expression profiles, each of them measuring mRNA expression levels of P transcripts at different times t, for m-th set $t \in \{t_1, t_2, ..., t_{N_m}\}$.

   In m-th set we operate on $N_m \times P$ timestamped expression matrix TS^m:

$$\mathrm{TS}^m = \begin{pmatrix} e_{1,t_1} & e_{2,t_1} & . & . & e_{P,t_1} \\ e_{1,t_2} & . & & & \\ . & & e_{i,t_k} & & \\ . & & & . & \\ e_{1,t_{N_m}} & & & & e_{P,t_{N_m}} \end{pmatrix},$$

   where e_{i,t_k} is expression value of i-th transcript (gene) at the time t_k. Columns of matrix TS^m correspond to genes, rows correspond to different snapshots in m-th set. Samples timestamped with low values of t_k correspond to the early stages of the experiments, whereas these timestamped with high values of t_k describe mRNA expression of experiments in their late stages, where genes reach homeostasis.

Despite the variety of the data types, the inference method is based on the same principle regardless of the given input. We score transcription factors in a gene selection problem, once per each possible target transcript. At the end of this procedure we calculate the final network of regulatory interactions in a form of the adjacency matrix V:

$$\mathrm{V} = \begin{pmatrix} v_{1,1} & v_{1,2} & . & . & v_{1,P} \\ v_{2,1} & . & & & \\ . & & v_{i,j} & & \\ . & & & . & \\ v_{P,1} & & & & v_{P,P} \end{pmatrix},$$

where $v_{i,j}$ is the confidence that i-th transcription factor regulates j-th transcript. More details on that process is given later in Chapter 4.

Chapter 3

Tree ensembles

In this chapter we describe working principles of a popular machine learning technique, an ensemble of trees. It is a powerful modeling tool originally motivated from the perspective of classification, regression, and feature selection theory. Ensembles became popular in many scientific fields, including genetics and bioinformatics, for assessing the importance of predictor variables and predicting the response variable in high-dimensional problems. This brought promising new possibilities to the field of inferring biological networks. Indeed, in this study we exploit the relative variable importance measure of the ensemble of trees as the indication of the presence or absence of a regulatory interaction between products of genes.

Presenting high-dimensional data to the learning algorithm is always a challenge, because the resulting model must be fairly accurate, i.e. give low prediction error, and reasonably simple at the same time. The second demand does not only come from the practical perspective, i.e. to lower the computational overhead, but also from what is known in machine learning as a *bias-variance* dilemma. Because expression data are undersampled, a training data set, which we present to the learning algorithm, can only be assumed a fraction of an abstract complete set of expression profiles. Theoretical studies [9, 36] have shown that

there are two competing forces, which govern the predictive ability of any learning model: *bias* and *variance*. The expected prediction error is always composed of the irreducible error, a bias, and a variance components. The first component comes from the variance of unseen data and therefore is beyond our control. Bias represents the extent to which the average prediction over all training sets differs from the desired optimal model. It is an estimation of how much our prediction differs from the optimal solution given a single training set which we present to the algorithm. Variance represents the extent to which the solutions for different training sets vary around their average, and therefore how much is the learning procedure sensitive to the particular choice of the training set. Generally, as the model complexity increases, the variance tends to increase and bias tends to decrease. The opposite happens as the model complexity is decreased. Typically we would like to choose our model complexity to trade bias off with variance in such a way that our fine-tuned learning procedure minimizes the prediction error. An ensemble of trees is an example of bias and variance reducing procedure.

Below we describe how ensembles of trees are built. Essentially, they are predictors, which combine a set of simple models, also called base learners, and use their weighted outcome as a final prediction of a target variable. In other words ensemble methods combine outputs from multiple base learners to form a compound with improved performance. There are two major approaches to construct such an ensemble:

1. A **parallel ensemble**, which combines independently constructed base learners. It is a variance-reduction technique commonly applied to unstable, high-variance algorithms, such as Classification and regression trees (CART) [16, 88]. An example of a parallel ensemble is *Random Forest* algorithm introduced by Breiman [15].

2. A **serial ensemble**, which combines base learners dependent on each others to reduce both variance and bias. An algorithm of combining base learners is more sophisticated than the one in parallel ensembles, like in *Adaboost* algorithm, introduced by Freund

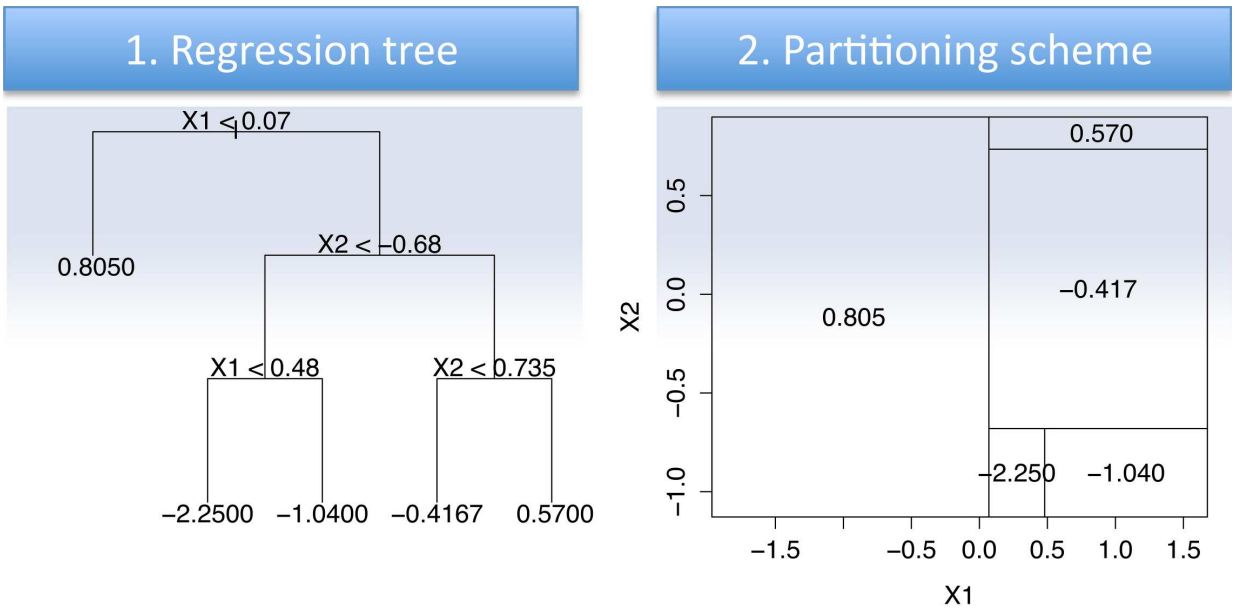

Figure 3.1: Partitions of a single regression tree. Right panel shows the partitioning of a two-dimensional feature space by recursive binary splitting applied to arbitrary data. Left panel shows the tree corresponding to the partitioning in the right panel.

and Schapire [31, 32], or in *Gradient Boosting Machine* algorithm, introduced by Breiman and Friedman [12, 34, 37].

Both approaches, and their working principles, will be described later in this chapter.

3.1 Parallel ensembles

Random Forest (RF) is the most commonly used example of a parallel ensemble. It is an improvement over *Bagging* method [11] and an extension of *Random Subspace* method [48]. In essence, the main loop of Random Forest algorithm repeats the following steps:

1. Select with replacement a sample from the training set. A number of samples to select is given as a parameter. Data selected in that process are called training samples. The rest of data are called out-of-bag (OOB) samples. Note that for each resampling, OOB sample size is around one-third of the number of all samples.

2. Select at random a small subset of predictors for each sample. The size of this subset is given as a parameter.

3. Grow each base learner, in this case a classification or regression tree (CART) [16], to a maximum depth on a sample using the best splits only from a subset of predictors.

After all the trees are grown, a community prediction is made based on individual predictions of base learners. The out-of-bag samples can serve as a test set for the tree grown on the training data. Moreover, it is empirically proved [14] that the OOB error estimates are as accurate as if using a test set of the same size as the training set. There is another use for OOB samples, especially important in this proposal. Namely, they rank predictors according to their relative importance in predicting the target variable, which helps solving a feature selection problem, as described in Section 4.3.

To fully comprehend parallel tree-based ensembles, particularly Random Forest algorithm, it is important to first focus on their building blocks: regression and classification trees (CART). They partition the space of all joint predictor variable values into disjoint regions $R_j, j = \{1, ..., J\}$, represented by the terminal nodes, as shown in Figure 3.1. Trees can be formally expressed as shown in Equation 3.1:

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j), \tag{3.1}$$

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$. I is the identity function, which returns numerical 1 for logical true, and numerical 0 for logical false. A constant γ_j is assigned to each region R_j to form a predictor:

$$x \in R_j \Rightarrow \widehat{f}(x) = \gamma_j.$$

In this proposal we focus on regression trees because expression data is numerical, not categorical. However, in practical applications, classification trees are at least as common

as regression trees.

A single regression tree uses a greedy, top-down recursive partitioning strategy to split the feature space into a set of regions R_j, and then fit a simple model in each of them, as shown in Figure 3.1. Formally, parameters $\widehat{\Theta}$ are found by minimizing the empirical risk:

$$\widehat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j), \tag{3.2}$$

where $L(y_i, \gamma_j)$ is a loss function, usually squared error. In practice, we can only find approximate suboptimal solutions to the optimization problem given in Equation 3.2. There are two levels of optimization:

1. **Finding γ_j given R_j:** Estimating γ_j given regions R_j is typically done as follows: $\widehat{\gamma_j} = \overline{y}_j$. In other words, the mean of the y_i falling in region R_j predicts output variable in that region.

2. **Finding R_j:** The exact solution is difficult to find, therefore typically a greedy, top-down recursive partitioning algorithm is used, as shown in Figure 3.1.

Let us now focus on the partitioning strategy used by CART, and analyze Figure 3.1. Note that all splits are parallel to the coordinate axes. In each region the output variable y is modeled with different constants γ_j. The first split partitions the whole x_1-x_2 domain into two regions. After the first partitioning is done, the process is recursively continued in both regions. The first region is represented by its average of output y variable, the second region undergoes further recursive binary splitting procedure. The process is continued until some stopping rule is applied. Eventually the corresponding regression model predicts y with a constant γ_j in region R_j, as follows:

$$\widehat{f}(x) = \sum_{j=1}^{J} \gamma_j I\{x \in R_j\}, \tag{3.3}$$

where J is the number of regions, and I is the identity function, which returns numerical 1 for logical $true$, and numerical 0 for logical $false$. At every step of the partitioning procedure an exhaustive search is used to test all available variables and split points to achieve the maximum reduction in a node impurity. A split point is simply a threshold used to split a variable (feature) into two parts: p_L and p_R. All the samples with y_i lower than the threshold are assigned to p_L part, all the samples with y_i higher or equal than the threshold are assigned to p_R part. Note that sometimes, like in Random Forest algorithm, a subset of available variables to split is smaller or equal than a set of all variables. For regression problem, variance is used as a measure of a node impurity, as given in Formula 3.4:

$$I(t) = \frac{1}{N(t)} \sum_{i \in t} (y_i - \overline{y})^2, \tag{3.4}$$

where the sum and mean are taken over all observations i in node t, and $N(t)$ is the number of observations in node t. Note that split points are found among available predictor variables, but node impurity is calculated with respect to the target variable.

The other metric is commonly used for classification trees, namely *Gini* index. It is equal to zero when all observations at node t belong to the same class, and it is maximal when classes are perfectly mixed. However, there is a reason of using variance as a node impurity measure for regression trees. It is the mean square error of the best constant predictor of a target variable in a node, which is the average of values of the predicted variable. In other words, if variance of a target variable y in node t was zero, no more splits would be necessary to perfectly model y.

The decrease in impurity $\Delta I(x_i, t)$ is calculated as deviation of the impurity at the node t and weighted average of impurities at each child node of t, as given in Formula 3.5:

$$\Delta I(x_i, t) = I(t) - p_L I(t_L) - p_R I(t_R), \tag{3.5}$$

where the weights p_L and p_R are proportional to the number of observations that are assigned to each child from the split at node t. In other words, a split which maximizes the decrease in node impurity, as given by Formula 3.5, is found by exhaustively examining all the possible split points. Impurity reduction due to a split on a specific variable indicates the relative importance of that variable to the tree model [16]. Moreover, the measure of a variable importance can be improved if out-of-bag samples were used. The split is calculated on the training data, but the variable importance measure is calculated from only the OOB samples. This provides more accurate and unbiased estimate of variable importance in each tree. Therefore, for a single regression tree T_i the measure of variable importance is given in Formula 3.6:

$$VI(x_i, T) = \sum_{t \in T} \Delta I(x_i, t), \tag{3.6}$$

where $\Delta I(x_i, t)$ is the decrease in impurity due to a split on variable x_i at a node t of the optimally pruned tree T, calculated on OOB samples. For ensembles, the metric is averaged over the collection of all the base learners, therefore the final relative importance measure of variable x_i in ensemble of M trees is given by Formula 3.7:

$$VI(x_i) = \frac{1}{M} \sum_{m=1}^{M} VI(x_i, T_m). \tag{3.7}$$

Due to the stabilizing effect of averaging, this measure turns out to be more reliable than the importance of a single regression tree, as given by Formula 3.6.

3.2 Serial ensembles

As opposed to parallel ensembles, serial ensembles combine base learners sequentially. Every following base model relies on previously built preceding models. This approach is also known as a boosting technique or additive modeling. It is one of the most powerful

learning methods, originally designed for classification problems but it can be extended to regression as well. In fact, Leo Breiman referred to the algorithm built on the ground of this methodology as the "best off-the-shelf classifier in the world" (NIPS Workshop, 1996). Both Adaboost algorithm, and Gradient Boosting Machine algorithms are forms of a gradient optimization algorithm in functional space [36]. Similar to parallel ensembles, serial ensembles combine the outputs of many weak learners to produce a powerful committee. However, they are fundamentally different, which shall be explained below in this section.

Algorithm 1 Adaboost

Require:
training set $\{(x_i, y_i)\}_{i=1}^{N}$,
base learner $h(x)$,
number of iterations M.

Ensure: classification model $\widehat{f}(x)$

1: Initialize the observation weights:
$d_0 = N^{-1}, i \in (1, ..., N)$.
2: **for** $m \leftarrow \{1, ..., M\}$ **do**
3: Fit a weak classifier $h_m(x)$ to the training data using weights d_{m-1}.
4: Calculate classification error with respect to d_{m-1}:
$\epsilon_m \leftarrow \frac{\sum_{i=1}^{N} d_{m-1} I(y_i \neq h_m(x_i))}{\sum_{i=1}^{N} d_{m-1}}$.
5: Calculate scaling coefficient:
$\alpha_m \leftarrow \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$.
6: Update the observation weights:
$d_m \leftarrow d_{m-1} \cdot e^{\alpha_m \cdot I(y_i \neq h_m(x_i))}$, $i \in (1, ..., N)$.
7: **end for**
8: Output $\widehat{f}(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m h_m(x))$.

The most well-known boosting procedure is Adaboost algorithm, introduced by Freund and Schapire [33]. The key point of the algorithm is to sequentially apply the base classification model, also called a weak learner, to repeatedly modified versions of the data, and this way producing a sequence of weak classifiers. The predictions from all the weak classifiers are then combined through a weighted majority vote to produce the final prediction, as shown in Equation 3.8:

$$H(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right), \tag{3.8}$$

where $\{\alpha_1, ..., \alpha_M\}$ are computed by the boosting algorithm, as shown in Algorithm 1, and scale the contribution of each weak learner $h_m(x)$. They are designed to give higher influence to the more accurate weak classifiers in the sequence. A similar reasoning gave foundations for the ADANET algorithm [85], described in details in Chapter 4, which exploits α_m coefficients as a relative importance measure of variables in the training set. The data modifications at each step of Algorithm 1 consist of applying weights $w_i, i \in (1, ..., N)$ to each of the training observations $\{(x_i, y_i)\}_{i=1}^{N}$. In the first line all the weights are set to $d_i = \frac{1}{N}$, so that initially all the samples are equally important. For each successive iteration the observation weights are individually modified, see line 6 of Algorithm 1, and the weak learner is invoked with respect to the updated weights. The key idea behind Adaboost algorithm is that at step m, those observations that were misclassified by the classifier $h_{m-1}(x)$, invoked at the previous step, have their weights increased, whereas the weights are decreased for those which were previously classified correctly. Therefore, observations that are difficult to correctly classify are becoming more important, and each successive classifier is forced to concentrate on them to minimize prediction error ϵ_m, calculated as shown in line 4 of Algorithm 1.

One-level classification trees [49], also known as decision stumps, are commonly used as weak classifiers $h_m(x)$ for Adaboost algorithm. Unlike fully grown decision trees, decision stumps do not calculate relative importance measure of variables in the training set. Instead, they exhaustively search for the optimal threshold χ_φ on the variable x_φ, which gives the lowest error ϵ_m predicting the class labels y, with regard to the observation weights d, as shown in Figure 3.2. As previously stated, in Adaboost algorithm α_m coefficients are designed to give higher influence to the more accurate weak classifiers in each step of boosting procedure. This way α_m coefficient can serve as a relative importance of a feature x_φ selected by decision stump to split. This reasoning gave foundations for the ADANET algorithm [85], described in details in Chapter 4, which exploits α_m coefficients as a relative

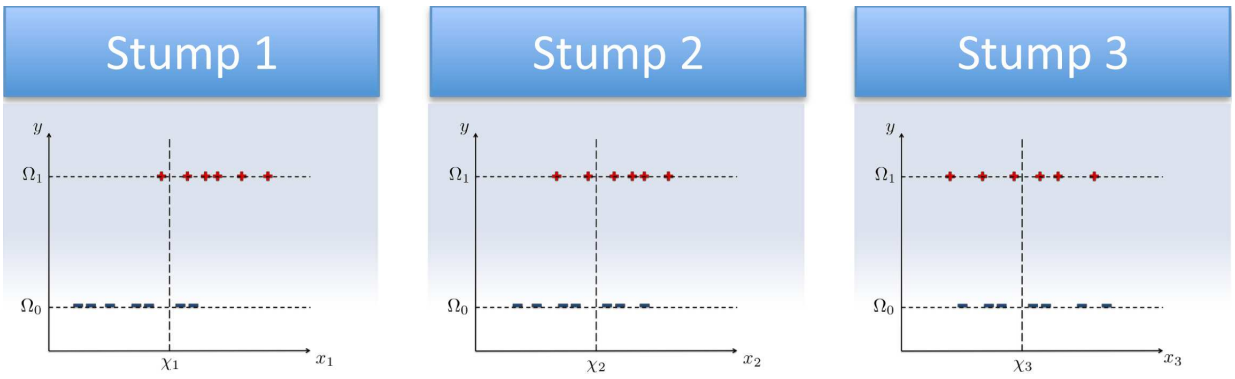

Figure 3.2: Solving classification problem by decision stumps. An arbitrary training set consists of 13 samples: $\{(x_i, y_i)\}_{i=1}^{13}$. Predictor variable x is 3-dimensional. Optimal thresholds: χ_1, χ_2, and χ_3 are found with respect to each variable. An optimal pair (φ, χ_φ) is the one, which gives the minimal prediction error. If the weights of every 13 observations are equal, then the misclassification error for the first pair $(1, \chi_1)$ is 0.23, for the second pair $(2, \chi_2)$ it is 0.38, and for the third pair $(3, \chi_3)$ it is 0.46. Therefore the first pair is selected as the optimal discriminating feature of the output residual y, and the corresponding threshold (split point).

importance measure of variables selected as the best discriminating variables for a target residuals.

It has been shown [36] that Adaboost algorithm is equivalent to a forward stage-wise additive modeling with the exponential loss function, as given by Formula 3.9:

$$L(y, f(x)) = \exp(-yf(x)). \tag{3.9}$$

It was indeed a remarkable discovery, since originally Adaboost algorithm was motivated from a very different perspective. Moreover, it places Adaboost algorithm in a broader group of optimization methods called boosting models. A generic algorithm for Gradient Boosting Machine is shown in Algorithm 2. A specific algorithm is obtained by inserting loss function $L(y, f(x))$ and base learner $h(x)$. Popular loss functions for regression, and their gradients, are summarized in Table 3.1. Gradient boosting is typically done with CART trees of a fixed size as base learners $h(x)$.

Algorithm 2 Generic Gradient Boosting Machine

Require:
training set $\{(x_i, y_i)\}_{i=1}^N$,
base learner $h(x)$,
a differentiable loss function $L(y, F(x))$,
number of iterations M, shrinkage factor ν.

Ensure: regression model $\widehat{f}(x)$

1: Initialize model with a constant value:
$f_0(x) \leftarrow \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2: **for** $m \leftarrow \{1, ..., M\}$ **do**
3: Calculate so-called pseudo-residuals:
$\forall i \in \{1, ..., N\}\ r_{im} \leftarrow -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$.
4: Fit a base learner $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^N$.
5: Calculate multiplier γ_m by solving the following one-dimensional optimization problem:
$\gamma_m \leftarrow \arg\min_\gamma \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i))$.
6: Update the model:
$f_m(x) \leftarrow f_{m-1}(x) + \nu \cdot \gamma_m h_m(x_i)$.
7: **end for**
8: Output $\widehat{f}(x) \leftarrow f_M(x)$.

The first line of Algorithm 2 initializes f_0 to be an optimal constant model, which is a single node terminal tree. In other words, f_0 is initialized to an average of y. Components r_{im}, which are calculated in line 3, are called pseudo-residuals. As opposed to Random Forest algorithm, gradient boosting procedure fits base learners not to y_i residuals, but to pseudo-residuals successively updated in each iteration of the algorithm. The exact definition of the negative gradient of loss function, which is used to calculate r_{im}, depends on the definition of the selected loss function. For squared error, pseudo-residuals are calculated simply as $r_{im} = y_i - f_{m-1}(x_i)$, see Table 3.1. If CART regression trees are used as base learners in generic Gradient Boosting Machine, then the algorithm is better known as Multiple Additive Regression Trees (MART). At m-th step MART algorithm fits a regression

tree as a base learner to pseudo residuals:

$$h_m(x) = \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm}). \tag{3.10}$$

MART algorithm applies a slight modification to line 5 of Algorithm 2. Instead of calculating a single multiplier γ_m for the whole tree, separate γ_{jm} are found for each region R_{jm}. Therefore line 5 becomes the following:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i)), \tag{3.11}$$

and the final updating rule from line 6 becomes the following:

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm}) \tag{3.12}$$

The parameter ν, also known as a shrinkage factor, is used to scale the contribution of each tree by a factor $\nu \in (0, 1)$ when it is added to the current approximation. In other words, ν controls the learning rate of the boosting procedure. Shrinkage techniques are also commonly used in neural networks. Smaller values of ν result in larger training risk for the same number of iterations M. However, it has been found [37] that smaller values of ν reduce test error, and require correspondingly larger values of M, which results in higher computational overhead. There is a trade-off between these two parameters.

Similar to parallel ensembles, serial ensembles are also linear combinations of base learners, see Equations 3.8 and 3.12, therefore for tree-based weak learners the relative importance $VI(x_i)$ is calculated in the same way in both approaches, as defined in Equation 3.7. Moreover, if only a fraction of the training observations is randomly selected in MART algorithm to propose the next tree in the expansion, then the relative importance of predictor variables could be improved in the same way as in Random Forest algorithm: trees are grown

Table 3.1: Commonly used loss functions for regression, and their gradients.

Loss function $L(y_i, f(x_i))$	
squared error	$\frac{1}{2}(y_i - f(x_i))^2$
absolute error	$\lvert y_i - f(x_i)\rvert$
Huber	$\begin{cases} (y - f(x))^2 & \text{if } \lvert y - f(x)\rvert \le \delta \\ \delta(\lvert y - f(x)\rvert - \frac{\delta}{2}) & \text{otherwise} \end{cases}$
gradient $-\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}$	
squared error	$y_i - f(x_i)$
absolute error	$\text{sign}(y_i - f(x_i))$
Huber	$\begin{cases} y_i - f(x_i) & \text{if } \lvert y_i - f(x_i)\rvert \le \delta_m \\ \delta_m \text{sign}(y_i - f(x_i)) & \text{otherwise} \end{cases}$, where $\gamma_m = \alpha\text{th-quantile}(\lvert y_i - f(x_i)\rvert)$

on training samples, but the variable importance measure is calculated from only the OOB samples. This provides more accurate and unbiased estimates, just like in Random Forest algorithm. However, because the process of growing trees in the two algorithms is not identical, relative importance estimates are quantitatively different.

Chapter 4

Proposed Methods

4.1 Overview of the proposed methods

We have proposed two methods for reverse-engineering gene regulatory networks from expression data, ADANET and ENNET, described in details in Sections 4.3.1 and 4.3.2, respectively. We have also proposed two methods for refining predictions of the above algorithms, described in Section 4.4. In this section we outline the basic steps of the proposed algorithms leading to the calculation of a directed graph of regulatory interactions between genes, in form of adjacency matrix V. As an input we are given different types of expression data, namely: wildtype, knockouts, knockdown, multifactorial, or/and time series expression data, as described in Section 2.4. The methods we propose execute the following steps:

1. **Collecting and pre-processing all the data tables**. Usually, raw expression data need to be pre-processed before any inference method could be applied to reverse-engineer a GRN. Pre-processing has a range of meanings, here it is regarded as a process of reducing variations or artifacts, which are not of biological origin, especially when expression is measured with multiple high-density microarrays [10]. From a practical point of view, it means that concentration levels of transcripts must be

adjusted and the entire distribution of adjusted values aligned with a normal distribution. Normalization of expression data is outside of the scope of our work and the data we obtain from databases are already normalized using state-of-the-art methods, such as RMA [10, 61]. We scale normalized expression data to zero mean and unit standard deviation.

2. **Decomposing the problem**. We decompose the problem of inferring incoming edges to all P genes to P independent subproblems. In each subproblem incoming edges from transcription factors to a single transcript are discovered. The process of decomposition is shown in Figure 4.1. We build a data matrix from a subset of genes, i.e. selected transcription factors, and score them independently for each target gene in the following steps of the method.

3. **Creating gene selection problem**. For k-th decomposed subproblem we create a target expression vector Y_k and a feature expression matrix X_{-k}. Columns of X_{-k} matrix constitute a set of possible transcription factors. Vector Y_k corresponds to a transcript, which is possibly regulated by transcription factors from X_{-k}. Creating gene selection problem is described in details in Section 4.2.

4. **Solving gene selection problem**. Once the target expression vector Y_k, and the feature expression matrix X_{-k} are created, we solve k-th gene selection problem as a feature selection problem. We use two different feature selectors as defined in ADANET and ENNET algorithms. The theoretical foundations for the above measures are described in Chapter 3. Practical considerations of solving a gene selection problem are described in details in Section 4.3.

5. **Merging solutions of gene selection problems**. Once the solutions of independent gene selection problems are calculated, we compose an adjacency matrix V representing a graph of inferred regulatory interactions. Each of the solutions constitutes a single column-vector, therefore we obtain V matrix by binding all the partial

solutions column-wise.

6. **Refining the network using the variance in rows of V.** Once the network of regulatory interactions is inferred, we apply a re-evaluation algorithm to achieve an improved final result. This step does not require any additional data to operate other than adjacency matrix V, which is obtained in the previous step. It exploits variance of edge probabilities in rows, i.e. edges outgoing from a single transcription factor, as a measure of the effect of a transcriptional regulation. It is further described in Section 4.4.1.

7. **Refining the network from the knockout experiments**. This step is only applied if knockout expression data is available. It analyzes z-score transformed KO matrix, as defined in Section 2.4, as a more accurate measure of the effect of a transcriptional regulation originated from a single transcription factor. This step is further described in Section 4.4.2.

4.2 Creating a gene selection problem

In a gene selection problem we decide which TFs contribute to the target gene expression across all the valid experiments. In order to justify all the possible interactions we need to solve a gene selection problem for each target gene. For example, if a regulatory network consists of four genes ($P = 4$), we need to solve four gene selection problems, as shown in Figure 4.1. In k-th problem, $k \in \{1, 2, 3, 4\}$, we find which TFs regulate k-th target gene. In other words, we calculate k-th column of V matrix.

At the beginning of the process of creating a gene selection problem we scale input data, regardless of its type, so that each column of expression matrix is zero mean and unit standard deviation. In principle we build a target expression vector Y_k, and the corresponding

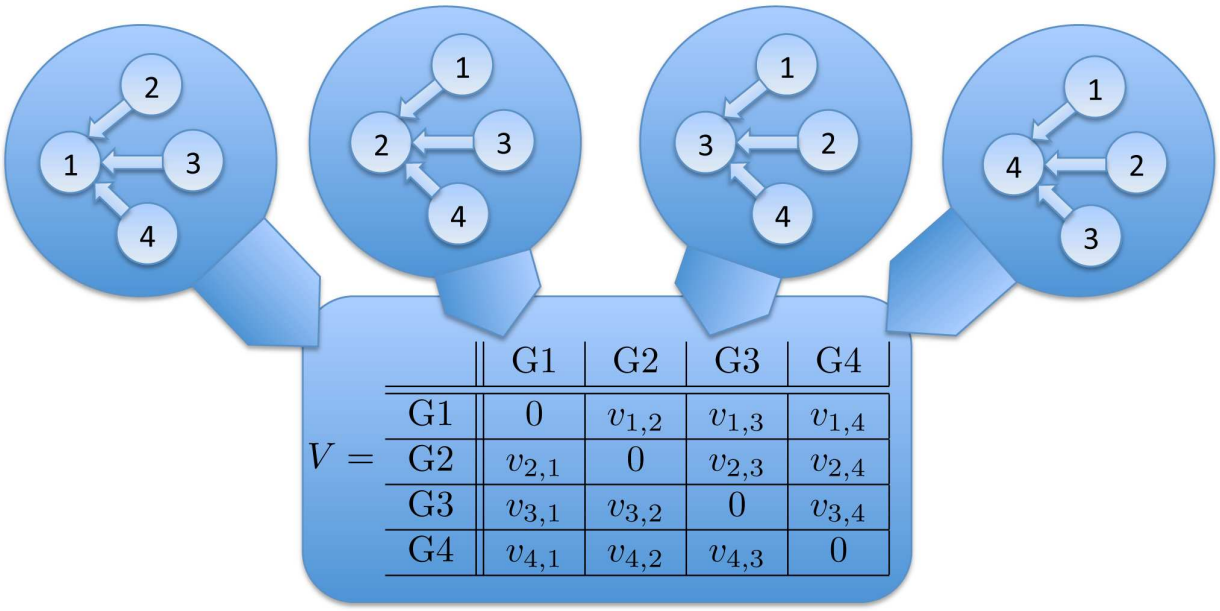

Figure 4.1: Decomposition of the network inference problem. An arbitrary problem of inferring a regulatory network of four genes is decomposed to four independent subproblems. In each subproblem incoming edges from transcription factors to a single gene are discovered. In the first problem the first gene is a target gene, the rest of genes are potential transcription factors. A solution of this subproblem contributes to the final solution as the first column of adjacency matrix V. All the other columns of V are calculated in a similar way.

feature expression matrix X_{-k}. Y_k is a column vector describing expression of the k-th target gene across all the valid experiments. X_{-k} is a matrix describing expression of all the possible TFs of k-th gene in the corresponding set of experiments. Columns of matrix X_{-k} correspond to TFs, rows of X_k and Y_{-k} are aligned with each other, and correspond to the valid samples. Columns of X_{-k} correspond to all the possible TFs, but if a target gene k is also a transcription factor, it is excluded from X_{-k}. We do not consider a situation in which a transcription factor would have a regulatory interaction with itself. The way we create X_k and Y_{-k}, and which experiments we consider valid in particular, depends on the input data we are given. We distinguish several possible scenarios:

1. We are given wildtype expression data: WT vector. When building target vector Y_k corresponding to k-th target gene, $k \in \{1, ..., P\}$, we consider WT vector as one valid

experiment. We reason that expression values in WT vector are determined by the underlying network of regulatory interactions just like in the other experiments in which genes are initially perturbed. However, WT vector by itself gives us only one sample, i.e. one row in a feature expression matrix, therefore it only makes sense to add it to the other experiments described below.

2. We are given known initial state expression profiles: KO or KD matrix. When building target vector Y_k corresponding to k-th target gene, $k \in \{1, ..., P\}$, we consider all the experiments valid except the one in which k-th gene was initially perturbed. We reason that the expression value of k-th gene in that experiment is not determined by its TFs, but by the external factor. Therefore we exclude k-th experiment (k-th row) from both Y_k and X_{-k}. The rest of experiments are aligned so that each row in Y_k vector is aligned with a corresponding row in X_{-k} matrix. Note that the other popular inference methods [73, 95], which are using knockout expression profiles, estimate edges outgoing from k-th transcription factor by looking at k-th row of KO matrix. This approach is however limited to the networks, for which a knowledge of the initial perturbation of a single transcription factor is provided, therefore unsuitable for the other data sets.

3. We are given unknown initial state expression profiles: MF matrix. When building a target vector Y_k and a feature matrix X_{-k} we consider all the N_{MF} experiments valid. We do not know how much the expression of a target gene depends on the regulatory interactions of its TFs, and how much on the external factor. This is a possible source of error in prediction. All the experiments are organized in a way that each row in Y_k vector is aligned with a corresponding row in X_{-k} matrix.

4. We are given timestamped unknown initial state expression profiles: a set of TS^m matrices, $m \in \{1, 2, ..., N_{TS}\}$. Like previously, we consider all the experiments valid because we can not confidently remove invalid experiments based on the knowledge

of the initial perturbations. However, unlike in previous cases, we do not align rows of Y_k vector and X_{-k} matrix to always reflect the same measurements. Rows in X_{-k} matrix are shifted backwards by Δt, like shown in Figure 4.2. We reason that expression values of TFs in X_{-k} matrix regulate their targets only after Δt. In this proposal we use Δt of the finest resolution of time series data. In other words, if expression is measured every 10 minutes, then $\Delta t = 10\text{min}$.

Once we collected all the expression data, scaled it, and excluded invalid experiments, we build Y_k and X_{-k} by aligning experiments as described above, and excluding the k-th gene from X_{-k} columns if it was a transcription factor. If we are given more than one type of data we reason that although different data come from different type of experiments, they all reflect the same underlying network of regulatory interactions, and thus we bind them all together row-wise as different samples in Y_k and X_{-k}, as shown in Figure 4.2.

Usually, different approaches are used to infer GRNs from different data, as described in Section 2.1. For example ordinary differential equations are commonly used to calculate the network from time series data, whereas information theoretic methods work best for multifactorial expression data. However, a method described in this work applies the same approach of finding true transcription factors on the integrated community of data of different types. Traditional methods operate on a pre-defined type of expression data, therefore are biased towards predicting edges that are exposed more clearly by a specific type of experiments. Our proposed approach reduces that bias, because all the samples are processed identically, regardless of their type.

Moreover, gene selection problems that we solve are high-dimensional, i.e. the number of known transcription factors is in range of hundreds. When the dimensionality of a problem increases, the volume of the space in feature domain increases so much that the available data become sparse. It is a problem for methods that require statistical signif-

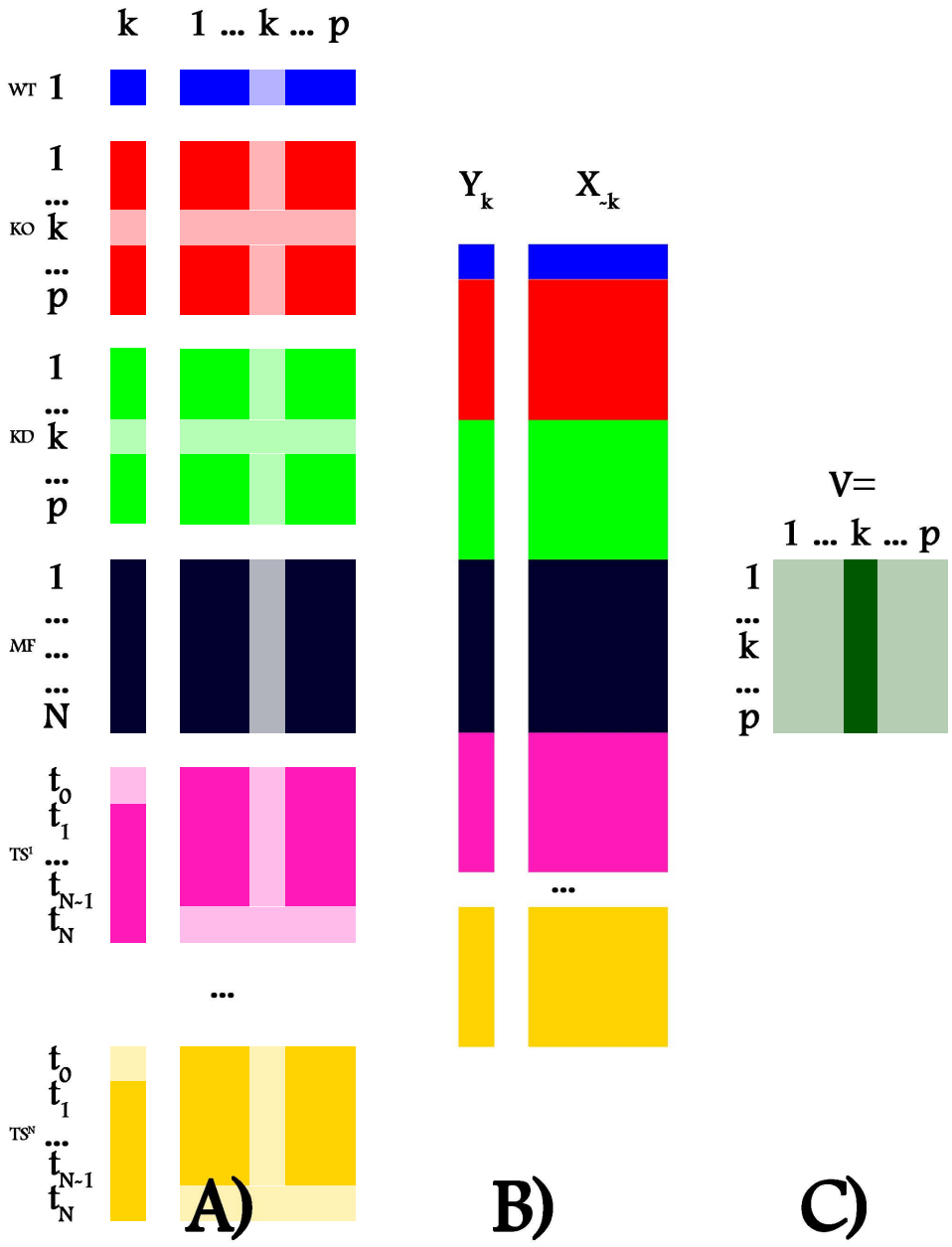

Figure 4.2: Solving GRN inference problem in three steps. A) Collecting input expression data of different types and removing invalid experiments with regard to k-th target gene, as described in Section 4.2. Briefly, rows correspond to experiments, and columns to genes/TFs. B) Y_k vector and X_{-k} matrix after aligning corresponding experiments. C) A result of predicting transcription factors of k-th gene is k-th column of V matrix.

icance. Therefore, predictions supported by a large set of samples are statistically well founded. Building algorithms that integrate all available expression data, i.e. increase the size of data but keeps the dimensionality of the problem, is the simplest way of reducing possible effects of that phenomenon, also known as the curse of dimensionality.

4.3 Solving a gene selection problem

We want to find which TFs from X_{-k} regulate a target gene from Y_k, as described above in Section 4.2. We search for the subset of columns in matrix X_{-k}, which are related to the target vector Y_k by a function f_k, as shown in Equation 4.1.

$$\forall k \in \{1, ..., P\}, \quad \exists f_k : Y_k = f_k(X_{-k}) + \epsilon_k, \tag{4.1}$$

where ϵ_k is a random noise with zero mean. In this problem both function f_k and ϵ_k are unknown. Function f_k represents a pattern of regulatory interactions, which drive the expression of k-th gene. We want the function f_k to rely only on a small number of genes, those which are the true regulators of gene k. In machine learning this problem is known as a feature selection problem [90, 91]. However, in literature related to GRN inference problem, it is also called a gene selection problem [51, 85]. Fundamentally, the goal of gene selection is to model target response Y_k, with an optimal subset of the important predictor variables, i.e. a subset of columns of X_{-k} matrix. A more relaxed objective of gene selection is a variable ranking, where relative relevance for all input columns of X_{-k} matrix is obtained with respect to the target vector Y_k. The higher a specific column is in that ranking, the higher is the confidence that a corresponding gene is in regulatory interaction with a gene corresponding to a target variable. We use the following gene selection methods:

1. Variable importance based on boosted ensemble of decision stumps, ADANET [85]. As input we use Y_k vector and X_{-k} matrix as described in Section 4.2. Below are the

default parameters of ADANET:

- Number of classification problems to create: 30.
- Number of stumps in the expansion: $\lfloor\sqrt{P-1}\rfloor$.
- Lower bound for classification ψ: 0.25.
- Upper bound for classification ξ: 0.75.
- Margin width: 0.05.
- Sampling rate: 0.67.

2. Variable importance based on boosted ensemble of regression stumps, ENNET [86]. As input we use the same data as for ADANET. Below are the default parameters of ENNET:

   - Sampling rate of samples s_s: 1.
   - Sampling rate of features s_f: 0.3.
   - Number of stumps in the expansion T: 5000.
   - Learning rate ν: 0.001.

As a comparison to the gene selection methods listed above we use popular inference methods to reverse-engineer GRN from expression data:

1. GENIE3 algorithm [51]. We use a Matlab implementation provided by the authors of that method. We use the default protocol with default parameters.

2. Information theoretic methods: C3NET [3, 4], MRNET [68], ARACNE [64]. We use *minet*: a popular R implementation of the methods. We use default parameters for all the three methods.

3. CLR method [29]. Implementation was downloaded from [68] and used with the default parameters.

We acknowledge that the above methods were initially designed to infer networks using MF matrix as the only input and not for the other types of data, specifically not for TS or KO/KD data. However, we reason that the expression matrix created as described in Section 4.2 might be considered as a special case of MF matrix. Moreover, for the benchmark GRN, where only MF data is available, the above methods work as designed by their creators.

4.3.1 ADANET Algorithm

Below we introduce ADANET algorithm that we proposed for reverse-engineering gene regulatory networks [85], and describe its working principles in two parts. In each of them we describe one functionally distinct module of our proposed algorithm.

4.3.1.1 Solving a gene selection problem by ADANET

In this section we describe the working principle of the method $PredictRegulators()$, as shown in Algorithm 3, which we invoke to decide which genes are related to the target gene k. Note that Y_k and X_{-k} provided to the method at $k-$th iteration as inputs are called Y and X respectively. Algorithm 3 returns feature importance vector V that constitutes $k-$th column of adjacency matrix V.

In the first phase we initialize feature importance vector $V, V \in [0,1]^{P-1}$ with zeros.

In the second phase we create C classification problems. We illustrate how we create a single classification problem in Figure 4.3. Each time we select a threshold point m on the target variable Y_k from a range bounded by ψ and ξ, $m \in [\psi, \xi]$. A threshold point m helps to define the percentage of samples which will be considered as "low expression of gene k" class, and "high expression of gene k" class. In the first classification problem m is equal to ψ, in the following problems it increases by a certain amount, until in the last one m is

Algorithm 3 PredictRegulators

Require:
$Y \in \mathbb{R}^N, X \in \mathbb{R}^{N \times P}, I \in \mathbb{N}^{N \times P}$
$C \in \mathbb{N}, s \in [0,1], \delta \in [0,1],$
$\psi \in [0,1], \xi \in [0,1], \psi \leq \xi$
Ensure: $V \in [0,1]^P$
1: $V \leftarrow 0$
2: **for** $c \leftarrow \{1, ..., C\}$ **do**
3: $m \leftarrow \psi + \frac{(\xi - \psi)c}{C}$
4: $Y_{\{0,1\}} \leftarrow Build(Y, I, m, \delta, s)$
5: $V_c \leftarrow AdaScoreRegulators(Y_{\{0,1\}}, X, I)$
6: $V \leftarrow V + V_c$
7: **end for**
8: $\forall k \in \{1, ..., P-1\} V(k) \leftarrow \frac{V(k)}{C}$

equal to ξ, i.e. the value of m is progressively increasing across the problems, but always limited by ψ and ξ. In this way, some classification problems are focused on separating very low expression of gene k from medium and high expression, others ask for separation of low from high expression, and others still prescribe discrimination of very high from medium and low expression of gene k.

For each classification problem we build a binary vector $Y_{\{0,1\}}$, $Y_{\{0,1\}} \in \{-1, 1\}^{N_s}$, $N_s \leq N$, which defines class labels. Samples, which belong to class Ω_0 and Ω_1 are labeled -1 and 1 respectively. We create class Ω_0 by randomly sub-sampling (by selecting without replacement with sampling rate s) all the samples that have expression value of gene k not higher than Percentile$_{m-\delta}(Y)$. Accordingly, we create class Ω_1 by sub-sampling all the samples that have expression value of gene k not lower than Percentile$_{m+\delta}(Y)$. From the training set we exclude all the samples, which have expression value of gene k between Percentile$_{m-\delta}(Y)$ and Percentile$_{m+\delta}(Y)$. For example, for $m = 0.6$, and $\delta = 0.1$, class Ω_0 contains part of the samples with expression value below the median of Y, and class Ω_1 contains part of the samples with expression value above 70% percentile of Y_k.

In the third phase we invoke method $AdaScoreRegulators()$ to test which genes are identi-

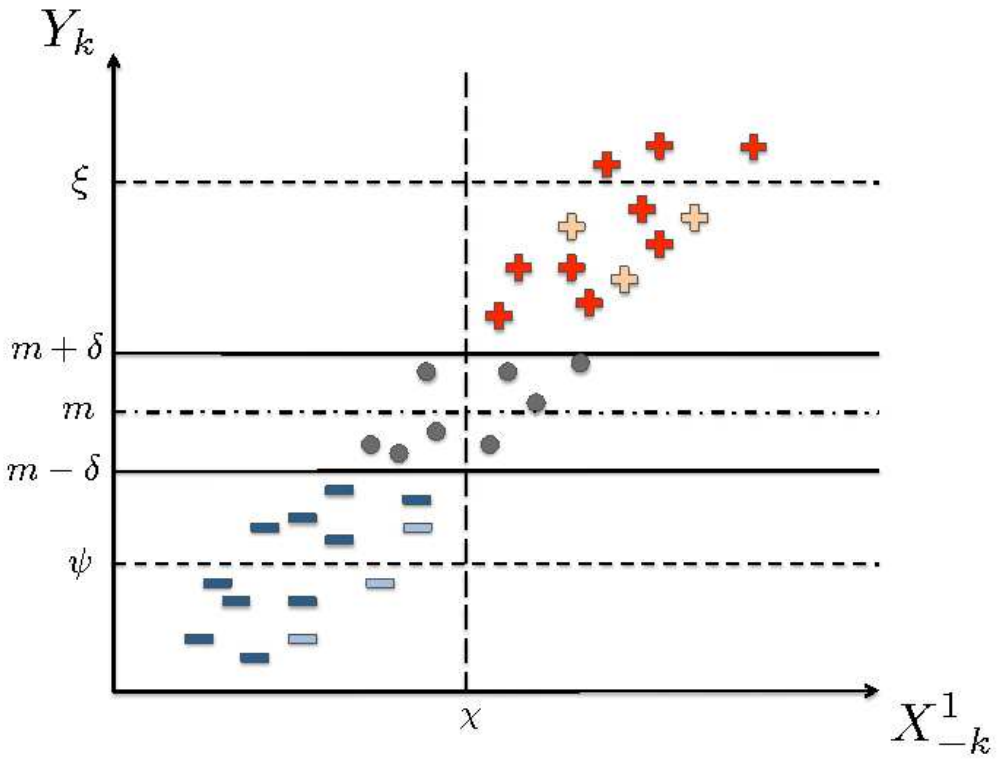

Figure 4.3: Creating a classification problem for Y_k. Samples with low (–) and high (+) expression of gene k are placed in class Ω_0 and Ω_1, respectively. Part of the samples is selected for the training set, according to the sampling rate s. Circles are representing samples within the margin, which are excluded from the training set. A gene corresponding to X^1_{-k} is a perfect discriminator for Y_k, because a threshold χ is found, such that all the training samples having $X^1_{-k} < \chi$ are classified as Ω_0, whereas all the training samples having $X^1_{-k} > \chi$ are classified as Ω_1. The training classification error is thus $\epsilon = 0$.

Algorithm 4 AdaScoreRegulators

Require:
$Y \in \{-1, 1\}^{N_s}$,
$X \in \mathbb{R}^{N_s \times P-1}$,
$T \in \mathbb{N}$, $I \in \mathbb{N}^{N \times P}$
Ensure: $W \in [0, 1]^{N_s}$

$W \leftarrow 0$
$\forall i \in \{1, ..., N_s\} : D_1(i) \leftarrow Initialize(Y)$
for $t \leftarrow \{1, ..., T\}$ **do**
  $h_t \leftarrow DecisionStump(X, Y, I, D_t)$
  $\epsilon_t \leftarrow GetError(h_t, X, Y, D_t)$
  $\varphi_t \leftarrow GetFeature(h_t)$
  if $\epsilon_t > 0$ **then**
    $\alpha_t \leftarrow \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$
    $\forall i \in \{1, ..., N_s\} :$
    $D_{t+1}(i) \leftarrow D_t(i) \exp(-\alpha_t Y(i) h_t(i))$
    $\forall i \in \{1, ..., N_s\} : D_{t+1}(i) \leftarrow \frac{D_{t+1}(i)}{\sum_{j=1}^{N_s} D_{t+1}(j)}$
    $W(\varphi_t) \leftarrow W(\varphi_t) + \alpha_t$
  else
    $W(\varphi_t) \leftarrow W(\varphi_t) + 1$
    $break$
  end if
end for
$\forall k \in \{1, ..., P-1\} W(k) \leftarrow \frac{W(k)}{\sum_{l=1}^{P-1} W(l)}$

fied as important discriminators for each of the classification problems described above. In other words we search for genes, which could discriminate samples belonging to class Ω_0, from those belonging to class Ω_1. In Figure 4.3 a gene corresponding to the single column X^1_{-k} would be given high importance score as a perfect discriminator for Y_k, because a threshold χ can be found, such that all the training samples having $X^1_{-k} < \chi$ are assigned to class Ω_0, and all the training samples having $X^1_{-k} > \chi$ are assigned to class Ω_1. We describe the mechanism of scoring the discriminative genes in details later in Section 4.3.1.2.

Finally we average the importance score over all classification problems.

4.3.1.2 AdaScoreRegulators Method

As described in Section 4.3.1.1 we invoke *AdaScoreRegulators*() method to decide which genes are related to the target gene k in each of the classification problems. In this section we describe the working principle of the method *AdaScoreRegulators*(), as shown in Algorithm 4.

We use a general scheme of *AdaBoost* meta-classifier, introduced by Freund and Schapire [31]. In the original definition it was used to solve classification problems. However, with slight modifications, we adopted the algorithm to solve gene selection problems. We use a basic concept, that genes which solve a classification problem, as described in Section 4.3.1.1, are more important than the others. In other words, all the genes that are selected as discriminators for the classification problem are given positive importance score. All the other genes are given zero importance score.

In the first phase of Algorithm 4 we initialize sample weight vector $D_t, t = 1$. A sample weight vector $D_t, D_t \in \mathbb{R}^{N_s}$ is interpreted as an importance measure of samples. Samples having higher weights are more important than those having lower weights, i.e. an error of prediction of a class label of a sample with a high weight is penalized more than the one of a sample with a low weight. At each step of the algorithm, D_t is normalized as shown in Equation 4.2.

$$\forall t \in \{1, ..., T\} : \sum_{j=1}^{N_s} D_t(j) = 1 \tag{4.2}$$

We initialize $D_t, t = 1$ to be normalized according to the Equation 4.2 and to correct the possible imbalances in the sizes of the Ω_0 and Ω_1 classes. Samples in each class are weighted uniformly, and the value of the weight is chosen to make the total sum of weights for points in the class equal to 0.5. In this way, the total importance of each class is the same. Specifically, an initial importance of samples belonging to class Ω_0 is given as follows:

$$D_1^{\Omega_0} = \frac{1}{2 \cdot |\Omega_0|}$$

An initial importance of samples belonging to class Ω_1 is given in a similar way as follows:

$$D_1^{\Omega_1} = \frac{1}{2 \cdot |\Omega_1|}$$

In the second phase we run T iterations of the main loop of Algorithm 4. In each loop we first create a weak hypothesis using Decision Stumps, described in [49]. Referring to Figure 4.3, a weak hypothesis is a threshold χ_t on the expression values of gene φ_t. A pair $(\varphi_t, \chi_t)_{optimal}$, which gives the lowest error ϵ_t on predicting the class labels, is found with regard to the sample importance D_t.

In the third phase of Algorithm 4, after the weak hypothesis $(\varphi_t, \chi_t)_{optimal}$ is found, if the error ϵ_t is greater than 0, we derive importance weights D_{t+1} with respect to the error ϵ_t, and update gene φ_t importance score $W(\varphi_t)$. On the other hand, if the error ϵ_t is equal to 0, i.e. a perfect weak hypothesis is found and no more iterations are needed, we update the only one gene φ_t importance score $W(\varphi_t)$ and break from the main loop. In order to derive the importance weights D_{t+1}, an updating coefficient α_t is calculated, as shown in Algorithm 4. In the original formulation of AdaBoost, α_t coefficient not only was used to update the importance weights D_{t+1}, but also to construct the final classifier, also called a strong hypothesis. However, to solve a gene selection problem, an explicit definition of the classification model is unnecessary, i.e. we do not need to know the explicit definition of f_k function, as described in Section 4.3.1. Despite, we use α_t coefficient to update gene φ_t importance score $W(\varphi_t)$.

In the last phase of Algorithm 4 we normalize gene importance score W.

4.3.2 ENNET algorithm

Below we introduce ENNET algorithm that we proposed for reverse-engineering gene regulatory networks [86], and describe its working principles. In essence, it uses boosted ensemble of regression stumps to score important transcription factors for each target gene independently. Like ADANET algorithm, for $k-$th target gene it returns a vector of feature importance constituting $k-$th column of the final adjacency matrix V.

4.3.2.1 Solving a gene selection problem by ENNET

As in ADANET, our solution to the variable ranking involves ensemble learning. However, instead of using boosted ensemble of decision stumps, we use an iterative regression method, which in each iteration chooses one transcription factor based on an optimality criterion, and adds it to the non-linear regression ensemble. The main body of our method, presented in Figure 4.4, is based on Gradient Boosting Machine [37] with a squared error loss function. The first line of ENNET initializes f_0 to be an optimal constant model, without selecting any transcription factor. In other words, f_0 is initialized to an average of Y_k. At each next t-th step the algorithm creates an updated model f_t, by fitting a base learner h_t and adding it to the previous model f_{t-1}. The base learner is fitted to a sample of pseudo residuals, with respect to a sample of transcription factors, and thus is expected to reduce the error of the model. Pseudo-residuals are re-calculated at the beginning of each iteration with respect to the current approximation f_t. As a base learner, we use regression stumps, which select a single TF that best fits pseudo residuals. A regression stump $h_t(x)$ partitions the expression values x of a candidate TF into two disjoint regions R_l and R_r, where $R_r = \mathbb{R} - R_l$, and returns values γ_l and γ_r, respectively, for those regions, as shown in Equation 4.3,

$$h_t(x) = \gamma_l I(x \in R_l) + \gamma_r I(x \in R_r), \tag{4.3}$$

Require:
expression of TFs: $x \in \mathbb{R}^{N \times P}$, expression of target gene: $y \in \mathbb{R}^N$,
number of iterations: $T \in \mathbb{N}$, shrinkage factor: $\nu \in [0, 1]$,
sampling rate for observations: $s_s \in [0, 1]$,
sampling rate for TFs: $s_f \in [0, 1]$.
Ensure: importance of TFs: $I^2 \in \mathbb{R}^P$

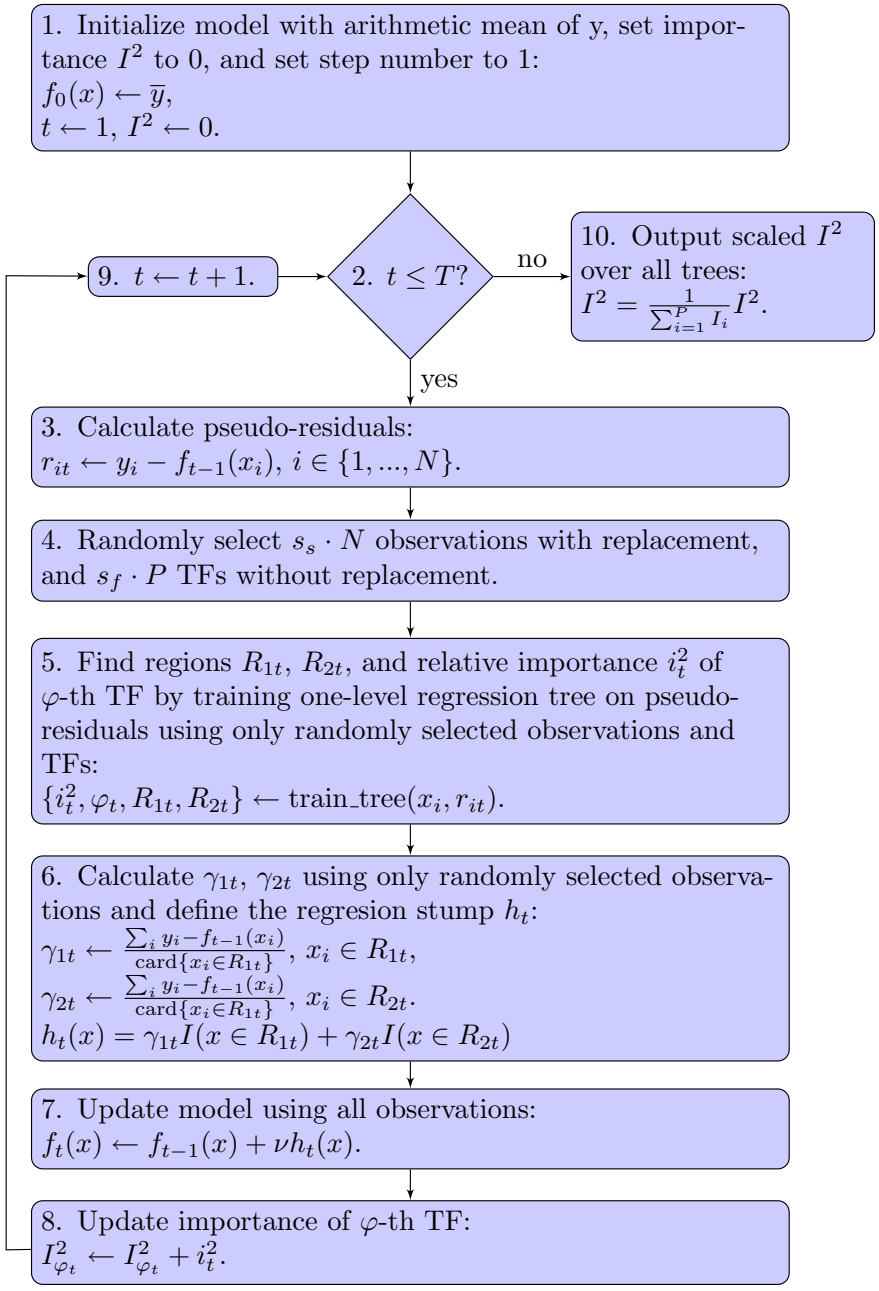

Figure 4.4: ENNET algorithm is a modification of a Gradient Boosting Machine algorithm, with a squared error loss function and a regression stump base learner. The algorithm calculates a vector of importance scores of transcription factors, which can possibly regulate a target gene. It is invoked P times in a problem of inferring a P-gene network, i.e. a P-column adjacency matrix V.

where I is the identity function returning the numerical 1 for the logical true, and the numerical 0 for the logical false. Regions R_l, R_r are induced such that the least-squares improvement criterion is maximized:

$$i^2(R_l, R_r) = \frac{w_l w_r}{w_l + w_r}(\overline{y}_l - \overline{y}_r)^2, \tag{4.4}$$

where w_l, w_r are proportional to the number of observations in regions R_l, R_r respectively, and $\overline{y}_l$, $\overline{y}_r$ are corresponding response means. That is, $\overline{y}_l$ is the average of the values from the vector Y_k for those samples where an expression of the chosen TF falls into the region R_l. The value of $\overline{y}_r$ is defined in an analogous way. The averages $\overline{y}_l$ and $\overline{y}_r$ are used as the regression output values γ_l and γ_r for regions R_l and R_r, respectively. The criterion in Equation 4.4 is evaluated for each TF, and the transcription factor with the highest improvement is selected. In each t-th step, we only use a random portion of rows and columns of X_{-k}, sampled according to the observation sampling rate s_s, and the TF sampling rate s_f.

The procedure outlined above creates a non-linear regression model of the target gene expression based on the expression of transcription factors. However, in the network inference, we are interested not in the regression model as a whole, but only in the selected transcription factors. At each t-th step of the ENNET algorithm, only one TF is selected as the optimal predictor. The details of the regression model can be used to rank the selected TFs by their importance. Specifically, if a transcription factor φ_t is selected in an iteration t, an improvement i_t^2 serves as an importance score $I_{\varphi_t}^2$ for that φ_t-th TF. If the same TF is selected multiple times at different iterations, its final importance score is a sum of the individual scores.

In the training of the regression model, the parameter ν, known as a shrinkage factor, is used to scale a contribution of each tree by a factor $\nu \in (0, 1)$ when it is added to the

current approximation. In other words, ν controls the learning rate of the boosting procedure. Shrinkage techniques are also commonly used in neural networks. Smaller values of ν result in a larger training risk for the same number of iterations T. However, it has been found [37] that smaller values of ν reduce the test error, and require correspondingly larger values of T, which results in a higher computational overhead. There is a trade-off between these two parameters.

4.4 Refining the prediction

Once the network of regulatory interactions is inferred, using either ADANET or ENNET, we apply a post-processing step to achieve an improved final result. We propose two steps: the first step does not require any additional data to operate other than adjacency matrix V, which is obtained in the previous step. It exploits variance of edge probabilities in rows, i.e. edges outgoing from a single transcription factor, as a measure of the effect of a transcriptional regulation. The second step is only possible if knockout expression data are available. It analyzes z-score transformed KO matrix, as defined in Section 2.4, as a more accurate measure of the effect of a transcriptional regulation originated from a single transcription factor.

4.4.1 Exploiting variance of the edge predictions

In this phase we score transcription factors based on the effect of a possible regulation of the target transcripts. We assume that the effect of the transcriptional regulation on a directly regulated transcript is stronger than the one of the regulation on indirectly regulated transcripts, i.e. transcripts regulated through another transcription factor. Otherwise, knocking out a single gene in a strongly connected component in a network of regulatory interactions would cause the same rate of perturbation of the expression level of all the transcripts in that component. As a measure of that effect we use previously calculated adjacency matrix

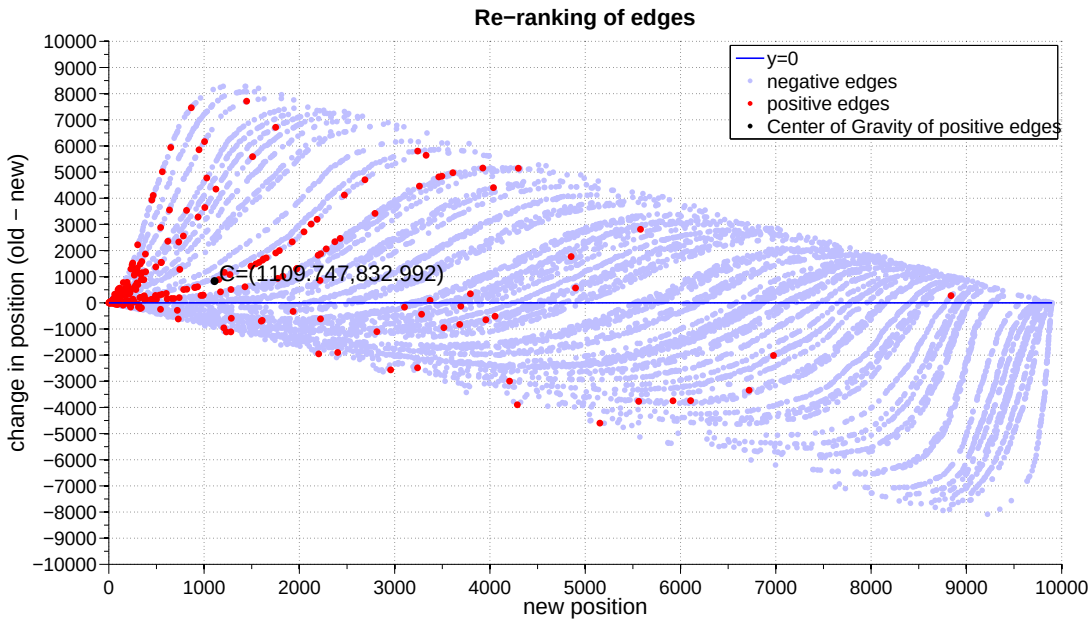

Figure 4.5: The effect of refining the edges calculated by ENNET algorithm. Plot compares ranked positions of edges in the old adjacency matrix V and in the new refined adjacency matrix V^1 for an exemplary prediction of DREAM4 challenge 100, network #2, see Table 2.1. Lighter points are the negative edges, red points are the positive edges (true regulators). Center of gravity of all the positive edges is a highlighted black point C. Ideally, we want all red points to have positive change in position.

V and multiply each row of V matrix by its variance σ_i^2. An updated adjacency matrix V^1 is given by Formula 4.5:

$$\forall(i,j) : v^1_{i,j} = \sigma_i^2 \cdot v_{i,j}, \tag{4.5}$$

where σ_i^2 is a variance in i-th row of V. Note that V matrix is built column-wise, i.e. a single column of V contains relative importance scores of all the transcription factors averaged over all the base learners with respect to a single target transcript. On the other hand, rows of V matrix are calculated independently in different subproblems of the proposed inference method. Each row of V contains relative importance scores with respect to a different target transcript. We reason that if a transcription factor regulates many

target transcripts, i.e. a transcription factor is a hub node, the variance in a row of V corresponding to that transcription factor is inclined. In that sense it is a way of estimating the number of target transcripts regulated by a transcription factor.

All the edges outgoing from a transcription factor, which was found to be a hub node, are promoted, as shown in Figure 4.5. An exemplary plot was calculated for Network #2 from DREAM4 size 100 (see Table 2.1). All the edges in an adjacency matrix can be ranked by their confidences. In such a ranking, the first edge is the one with the highest confidence, the last edge is the one with the lowest confidence. Figure 4.5 compares the two rankings: the one derived from V (the old ranking resulting from ENNET), and the one derived from V^1 (the new ranking). More precisely, it shows how a change in position in a ranking (promotion or degradation) depends on the position in the new ranking. Each edge is represented as a point. All the edges with positive second coordinate are promoted, i.e. their position in the new ranking is higher than in the old one. Visible "stripes" correspond to the transcription factors (rows of V and V^1). Additionally, center of gravity of all the true edges is depicted as point C. The first coordinate of C is an average position of a true edge in the new ranking. The second coordinate of C is an average number of positions that a true edge was promoted (or degraded if negative). Note that V^1 is an improvement over V if the second coordinate of C is positive. Analyzing Figure 4.5, we observe that the most of the true regulators have high positions in the new ranking, see the first coordinate of the Center of Gravity point C, and that the most of the beneficial promotions, i.e. positive changes in a position of a true positive edge, are visible a the beginning of the new ranking. It is especially important in practical applications of the inference algorithm, because we want to make sure that it is accurate for the top-ranked edges.

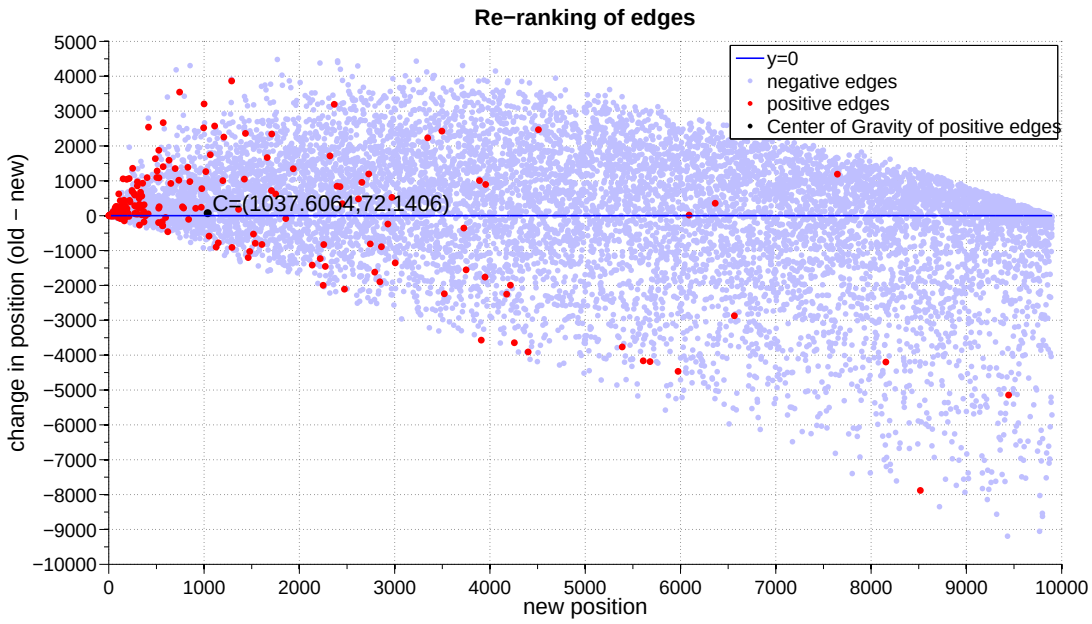

Figure 4.6: The effect of refining the edges calculated by ENNET algorithm. Plot compares ranked positions of edges in the old adjacency matrix V^1 and in the new adjacency matrix V^2 for an exemplary prediction of DREAM4 challenge 100, network #2, see Table 2.1. Lighter points are the negative edges, red points are the positive edges (true regulators). Center of gravity of all the positive edges is a highlighted black point C.

4.4.2 Exploiting the effects of a possible regulation

In this phase we score transcription factors based on the effect of a possible regulation of the target transcripts. Unlike in the second phase, the third phase requires additional data, namely knockout expression data. We reason that the effect of a transcriptional regulation on a directly regulated transcript is the easiest to capture from knockout expression data. A similar reasoning gave foundations for the null-mutant z-score method [74] of reverse-engineering GRN. However, in the proposed method this step is only applied if knockout expression profiles are available. In this step we calculate an adjacency matrix V^2, which is an update to already derived adjacency matrix V^1, as shown in Formula 4.6:

$$\begin{gathered}\forall(i,j) : v^2_{i,j} = |\frac{\overline{e_{\alpha,j}} - \overline{e_{\beta,j}}}{\sigma_j}| \cdot v^1_{i,j}, \\ \alpha \in \{\alpha : k_{\alpha,j} \neq 0\}, \beta \in \{\beta : k_{\beta,j} = 0\},\end{gathered} \tag{4.6}$$

where $\overline{e_{\alpha,j}}$ is an average expression value of the j-th transcript in all the experiments in which the i-th gene was knocked-out, as defined by K matrix, $\overline{e_{\beta,j}}$ is the mean expression value for that transcript across all the other knockout experiments, and σ_j is the standard deviation of the expression value of that transcript in all the knockout experiments. We reason that $|\frac{\overline{e_{\alpha,j}}-\overline{e_{\beta,j}}}{\sigma_j}|$ coefficient shows how many standard deviations was that typical expression of the j-th transcript different from the average expression in the experiment in which its potential i-th transcription factor was knocked-out. Such a measure of deviation of an observation from sample mean is called z-score in statistics. Here it serves as an update to already calculated V^1 matrix, as shown in Figure 4.6. It is the same kind of plot as in Figure 4.5. However this time it captures an improvement of V^2 over V^1. Both plots were derived from results of ENNET on network #2 of DREAM4 challenge 100 benchmark. Note that in Section 4.4.1 we calculate a single updating coefficient for the whole row of V, whereas here we compute a single updating coefficient for every element in V^1. That is why the "stripes" from Figure 4.5 are no longer visible in Figure 4.6. Center of gravity point C is above $y = 0$ line in both plots, which means that V^2 is an improvement over V^1, and V^1 is an improvement over V. This phenomenon is also observed in the other benchmark networks, which will be later discussed in Chapter 5 and 6.

Chapter 5

Results

In this chapter we quantitatively assess the performance of the proposed inference algorithms, and compare them to the state-of-the-art methods on universally recognized benchmark networks. All of the networks come from DREAM initiative, introduced in Section 2.2. For large networks of size 100 genes and more, as examples of genome-wide transcriptional regulation systems, we also provide Precision-Recall analysis.

5.1 DREAM3

DREAM3 (Dialogue for Reverse Engineering Assessments and Methods) GRN inference challenge [62, 63, 74] consisted of 3 subchallenges: DREAM3 size 10, DREAM3 size 50, and DREAM3 size 100. For each subchallenge, the topology of two benchmark networks were derived from *E. coli* transcriptional regulatory system, whereas *S. cerevisiae* genetic interaction network gave foundations for the remaining three networks. *In silico* networks and expression data were simulated using an open-source software: GeneNetWeaver (GNW). Benchmark networks were derived as subnetworks of regulatory interactions from known model organisms, however genes in expression data were labeled with concealed names: G1, G2, G3, etc. In other words, the identity of genes was hidden from participants of the challenge. This way no other sources than the knowledge learned from experimental

Table 5.1: Results of different inference methods on DREAM3 networks, challenge size 10. Area under ROC curve (AUROC) and area under precision-recall curve (AUPR) are given for each network respectively. Overall score for all the networks is given in the last column, as described in Chapter 5.1. The best result for each column is in bold.
Numbers in *Experimental results* part of the table were collected after running the algorithms with default sets of parameters on pre-processed data, as described in Section 4.2. However, GENIE3, C3NET, CLR, MRNET, and ARACNE methods, as they are originally defined, take MF matrix as input, which is unavailable in this challenge. Therefore they are marked N/A.
Numbers in *Winner of the challenge* part of the table correspond to the best methods participating in the challenge.

Method	Network (AUPR/AUROC respectively)										Overall
	1		2		3		4		5		
Experimental results											
ENNET	0.470	0.804	**0.768**	**0.919**	**0.983**	**0.998**	0.447	0.636	0.537	0.637	4.478
ADANET	0.635	0.808	0.399	0.745	0.432	0.901	0.303	0.549	0.349	0.591	2.464
GENIE3	N/A										
C3NET	N/A										
CLR	N/A										
MRNET	N/A										
ARACNE	N/A										
Winner of the challenge											
Yip et al.	**0.710**	**0.928**	0.713	0.912	0.897	0.949	**0.541**	**0.747**	**0.627**	**0.714**	**5.124**
2nd	0.544	0.794	0.748	0.856	0.771	0.944	0.352	0.590	0.493	0.715	3.821
3nd	0.193	0.697	0.377	0.791	0.468	0.909	0.332	0.554	0.388	0.658	2.188

expression data could have been used to reverse-engineer regulatory interactions. The size of each network in DREAM3 challenge is reported in Table 2.1. The type of expression data available in each of DREAM3 subchallenges is reported in Table 2.2.

Multifactorial data in DREAM3 size 10 subchallenge were not available, therefore GENIE3, C3NET, CLR, MRNET, and ARACNE methods were excluded from the comparison. The results of the other methods are summarized in Table 5.1. The best performer in this subchallenge was Yip et al. [95] method. However, it is believed from the analysis of the later challenges [78] that the method made a strong assumption on the Gaussian type of measurement noise, which was used in DREAM3, but was no longer used in later DREAM challenges. For example, in DREAM4 challenge Yip et al. method was ranked 7th. Our

Table 5.2: Results of different inference methods on DREAM3 networks, challenge size 50. Area under ROC curve (AUROC) and area under precision-recall curve (AUPR) are given for each network respectively. Overall score for all the networks is given in the last column, as described in Chapter 5. The best results for each column are in bold.
Numbers in *Experimental results* part of the table were collected after running the algorithms with default sets of parameters on pre-processed data, as described in Section 4.2. However, GENIE3, C3NET, CLR, MRNET, and ARACNE methods, as they are originally defined, take MF matrix as input, which is unavailable in this challenge. Therefore they are marked N/A.
Numbers in *Winner of the challenge* part of the table correspond to the best method participating in the challenge.

Method	Network (AUPR/AUROC respectively)										Overall
	1		2		3		4		5		
Experimental results											
ENNET	0.703	0.917	0.723	0.899	**0.727**	**0.939**	**0.492**	**0.806**	**0.481**	0.790	**41.930**
ADANET	0.024	0.599	0.020	0.627	0.032	0.558	0.052	0.566	0.050	0.472	3.241
GENIE3	N/A										
C3NET	N/A										
CLR	N/A										
MRNET	N/A										
ARACNE	N/A										
Winner of the challenge											
Yip et al.	**0.734**	**0.930**	**0.778**	**0.924**	0.579	0.917	**0.429**	0.792	0.424	**0.805**	39.828
2nd	0.671	0.862	0.672	0.842	0.486	0.836	0.367	0.688	0.381	0.728	31.341
3nd	0.245	0.694	0.296	0.778	0.384	0.823	0.263	0.672	0.284	0.676	17.927

proposed method ENNET achieved a competitive Overall Score and the best prediction accuracy for two out of five networks constituting this challenge. Yip et al. method outperformed ENNET in the remaining three networks and achieved the best Overall Score.

In DREAM3 size 50 subchallenge multifactorial data were also not available, therefore GENIE3, C3NET, CLR, MRNET, and ARACNE methods were excluded from the comparison. The results of the other methods are summarized in Table 5.2. Again Yip et al. and ENNET methods achieved the best prediction accuracy in this subchallenge. ENNET achieved the best Overall Score and outperformed Yip et al. method for two out of five networks constituting this challenge. Yip et al. method achieved the best prediction accuracy for two other networks and achieved a very competitive Overall Score, compared with

Table 5.3: Results of different inference methods on DREAM3 networks, challenge size 100. Area under ROC curve (AUROC) and area under precision-recall curve (AUPR) are given for each network respectively. Overall score for all the networks is given in the last column, as described in Chapter 5. The best results for each column are in bold.
Numbers in *Experimental results* part of the table were collected after running the algorithms with default sets of parameters on pre-processed data, as described in Section 4.2. However, GENIE3, C3NET, CLR, MRNET, and ARACNE methods, as they are originally defined, take MF matrix as input, which is unavailable in this challenge. Therefore they are marked N/A.
Numbers in *Winner of the challenge* part of the table correspond to the best method participating in the challenge. Yip et al. method achieved overall score of infinity, details are given in Section 2.3.

Method	Network (AUPR/AUROC respectively)										Overall
	1		2		3		4		5		
Experimental results											
ENNET	0.627	0.901	**0.865**	**0.963**	**0.568**	0.892	**0.522**	0.842	0.384	0.765	**>300**
ADANET	0.288	0.753	0.415	0.829	0.223	0.733	0.257	0.709	0.214	0.677	57.059
GENIE3	N/A										
C3NET	N/A										
CLR	N/A										
MRNET	N/A										
ARACNE	N/A										
Winner of the challenge											
Yip et al.	**0.694**	**0.948**	0.806	0.960	0.493	**0.915**	0.469	**0.856**	**0.433**	**0.783**	**>300**
2nd	0.209	0.854	0.249	0.845	0.184	0.783	0.192	0.750	0.161	0.667	45.443
3nd	0.132	0.835	0.154	0.879	0.189	0.839	0.179	0.738	0.164	0.667	42.240

ENNET algorithm. On the other hand, ADANET algorithm did not calculate accurate predictions in this subchallenge.

Similar outcomes could be observed with respect to the biggest DREAM3 subchallenge: DREAM3 size 100. Again, multifactorial data were not available, therefore GENIE3, C3NET, CLR, MRNET, and ARACNE methods were excluded from the comparison. The results of the other methods are summarized in Table 5.3. Yip et al. and ENNET methods achieved the best Overall Scores for that subchallenge. The two methods achieved the best scores for all the individual networks in that subchallenge: Yip et al. method achieved 6 of them, ENNET achieved the remaining 4 scores. Moreover, both methods were evalu-

ated with the Overall Score of infinity, which indicates that they achieved the individual p-value lower than the smallest positive real number available for the authors of DREAM3 challenge (see Section 2.3 for details), for at least one network in that subchallenge, which is a remarkable result, especially for a moderately large network of 100 genes like the ones in DREAM3 size 100 subchallenge.

5.2 DREAM4

DREAM4 challenge [62, 63, 74] was posted one year after DREAM3 challenge. It consisted of 3 subchallenges: DREAM4 size 10, DREAM4 size 100, and DREAM4 size 100 multifactorial. For each subchallenge, the topology of the benchmark networks were derived from *E. coli* and *S. cerevisiae* transcriptional regulatory system. The size of each network in DREAM3 challenge is reported in Table 2.1. The type of expression data available in each of DREAM3 subchallenges is reported in Table 2.2.

In DREAM4 size 10 subchallenge all the data types listed in Table 2.2 were available, therefore GENIE3, C3NET, CLR, MRNET, and ARACNE methods were included in the comparison, and run as originally designed on multifactorial data. The results of all the methods are summarized in Table 5.4. Both ADANET and ENNET algorithms outperformed GENIE3 algorithm and MI algorithms: C3NET, CLR, MRNET, and ARACNE. It is especially interesting to compare GENIE3 with ENNET, which both work based on a similar principle of solving gene selection problem using ensemble of regression trees. However, by design ENNET uses all available expression data, whereas GENIE3 only takes multifactorial data as input. Therefore ENNET method is able to build regression models supported by a greater amount of available data.

Table 5.4: Results of different inference methods on DREAM4 networks, challenge size 10. Area under ROC curve (AUROC) and area under precision-recall curve (AUPR) are given for each network respectively. Overall score for all the networks is given in the last column, as described in Chapter 5. The best results for each column are in bold.
Numbers in *Experimental results* part of the table were collected after running the algorithms with default sets of parameters on pre-processed data, as described in Section 4.2.
Numbers in *Winner of the challenge* part of the table correspond to the best method participating in the challenge.

Method	Network (AUPR/AUROC respectively)										Overall
	1		2		3		4		5		
Experimental results											
ENNET	0.669	0.832	0.220	0.593	0.703	0.843	0.744	0.904	0.554	0.786	4.145
ADANET	0.509	0.823	0.534	0.728	0.641	0.771	0.617	0.826	0.545	0.769	3.894
GENIE3	0.304	0.676	0.492	0.697	0.312	0.626	0.187	0.363	0.269	0.718	1.593
C3NET	0.458	0.644	0.304	0.637	0.202	0.551	0.186	0.584	0.171	0.519	1.075
CLR	0.276	0.613	0.413	0.712	0.257	0.650	0.221	0.501	0.343	0.662	1.462
MRNET	0.379	0.577	0.404	0.747	0.290	0.691	0.226	0.549	0.293	0.693	1.690
ARACNE	0.434	0.651	0.399	0.733	0.255	0.611	0.231	0.553	0.291	0.635	1.602
Winner of the challenge											
1st	**0.916**	**0.972**	**0.547**	**0.841**	**0.968**	**0.990**	**0.852**	**0.954**	**0.761**	**0.928**	**7.127**
2nd	0.881	0.967	0.382	0.796	0.682	0.916	0.698	0.902	0.424	0.822	5.290
3rd	0.623	0.864	0.301	0.567	0.646	0.824	0.693	0.820	0.673	0.776	3.968

In DREAM4 size 100 subchallenge all the data types listed in Table 2.2 were available except from multifactorial, therefore GENIE3, C3NET, CLR, MRNET, and ARACNE methods were excluded from the comparison. The results of all the methods are summarized in Table 5.5. ENNET clearly outperformed all the others and achieved consistently high scores across all the benchmark networks.

In DREAM4 size 100 multifactorial challenge only multifactorial data were available, therefore GENIE3, C3NET, CLR, MRNET, and ARACNE methods were included in the comparison, and run as originally designed. However, without knockout expression data refining the edges as described in Section 4.4.2 could not be applied. The results of all the methods are summarized in Table 5.6. ADANET and GENIE3 methods performed similarly but ENNET ennet algorithm significantly outperformed them in this comparison. Note that

Table 5.5: Results of different inference methods on DREAM4 networks, challenge size 100. Area under ROC curve (AUROC) and area under precision-recall curve (AUPR) are given for each network respectively. Overall score for all the networks is given in the last column, as described in Chapter 5. The best results for each column are in bold.
Numbers in *Experimental results* part of the table were collected after running the algorithms with default sets of parameters on pre-processed data, as described in Section 4.2. However, GENIE3, C3NET, CLR, MRNET, and ARACNE methods, as they are originally defined, take MF matrix as input, which is unavailable in this challenge. Therefore they are marked N/A.
Numbers in *Winner of the challenge* part of the table correspond to the best method participating in the challenge.

Method	Network (AUPR/AUROC respectively)										Overall
	1		2		3		4		5		
Experimental results											
ENNET	**0.604**	0.893	**0.456**	**0.856**	**0.421**	**0.865**	**0.506**	**0.878**	**0.264**	**0.828**	**87.738**
ADANET	0.441	0.811	0.146	0.648	0.086	0.643	0.065	0.633	0.064	0.645	25.913
GENIE3	N/A										
C3NET	N/A										
CLR	N/A										
MRNET	N/A										
ARACNE	N/A										
Winner of the challenge											
Pinna et al.	0.536	**0.914**	0.377	0.801	0.390	0.833	0.349	0.842	0.213	0.759	71.589
2nd	0.512	0.908	0.396	0.797	0.380	0.829	0.372	0.844	0.178	0.763	71.297
3rd	0.490	0.870	0.327	0.773	0.326	0.844	0.400	0.827	0.159	0.758	64.715

GENIE3 is listed in Table 5.6 twice: once in *Experimental results* part, and once in *Winner of the challenge* part. This is because the evaluation of the results achieved by the creators of GENIE3 algorithm were published as well as the source code of the method. Results from the first part of the table come from an experimental run of GENIE3 method and match with results achieved in the actual challenge by the authors. ENNET achieved the best scores for almost all the individual networks.

5.3 DREAM5

Unlike the other DREAM challenges described in this chapter, DREAM5 [61] was not divided into subchallenges. The 3 benchmark networks in DREAM5 were each of a different

Table 5.6: Results of different inference methods on DREAM4 networks, challenge size 100 multifactorial. Area under ROC curve (AUROC) and area under precision-recall curve (AUPR) are given for each network respectively. Overall score for all the networks is given in the last column, as described in Chapter 5. The best results for each column are in bold. Numbers in *Experimental results* part of the table were collected after running the algorithms with default sets of parameters on pre-processed data, as described in Section 4.2. Numbers in *Winner of competition* part of the table correspond to the best method participating in the challenge.

Method	Network (AUPR/AUROC respectively)										Overall
	1		2		3		4		5		
Experimental results											
ENNET	**0.184**	0.731	**0.261**	**0.807**	**0.289**	**0.813**	**0.291**	**0.822**	**0.286**	**0.829**	**52.839**
ADANET	0.125	0.659	0.205	0.709	0.207	0.739	0.225	0.750	0.255	0.740	36.464
GENIE3	0.158	**0.747**	0.154	0.726	0.232	0.777	0.210	0.795	0.204	0.792	37.669
C3NET	0.077	0.562	0.095	0.588	0.126	0.621	0.113	0.687	0.110	0.607	15.015
CLR	0.142	0.695	0.118	0.700	0.178	0.746	0.174	0.748	0.174	0.722	28.806
MRNET	0.138	0.679	0.128	0.698	0.204	0.755	0.178	0.748	0.187	0.725	30.259
ARACNE	0.123	0.606	0.102	0.603	0.192	0.686	0.159	0.713	0.166	0.659	22.744
Winner of the challenge											
GENIE3	0.154	0.745	0.155	0.733	0.231	0.775	0.208	0.791	0.197	0.798	37.428
2nd	0.108	0.739	0.147	0.694	0.185	0.748	0.161	0.736	0.111	0.745	28.165
3rd	0.140	0.658	0.098	0.626	0.215	0.717	0.201	0.693	0.194	0.719	27.053

size, and structured with respect to the other model organism. However, this time expression data of only one network was simulated *in silico*, the two other sets of expression data were measured in real experiments *in vivo*. Like in all DREAM challenges, *in silico* expression data were simulated using open-source GeneNetWeaver simulator [63]. Additionally, a set of decoy genes (around 5% of the compendium) were introduced to a list of genes by randomly selecting gene expression values from the compendium. Moreover, this was the first DREAM challenge where participants were asked to infer GRNs on a genomic scale, i.e. the size of networks reflected a real-life size of biological networks: thousands of target genes, and hundreds of known transcription factors, see Table 2.1. In addition to the gene expression data, descriptive features were supplied for each microarray experiment. Based on that additional information participants could determine if a specific sample was part of a gene deletion experiment, or time series experiment. In other words, with the use of an additional meta-data, participants could build WT, KO, KD, MF, and TS matrices.

Table 5.7: Results of different inference methods on DREAM5 networks. Area under ROC curve (AUROC) and area under precision-recall curve (AUPR) are given for each network respectively. Overall score for all the networks is given in the last column, as described in Chapter 5.3. The best results for each column are in bold.
Numbers in *Experimental results* part of the table were collected after running the algorithms with default sets of parameters on pre-processed data, as described in Section 4.2. Numbers in *Winner of the challenge* part of the table correspond to the best method participating in the challenge.

Method	Network (AUPR/AUROC respectively)						Overall
	1		3		4		
Experimental results							
ENNET	**0.423**	**0.867**	0.069	**0.642**	0.021	**0.532**	**>300**
ADANET	0.263	0.760	0.064	0.596	0.019	0.512	18.716
GENIE3	0.291	0.814	**0.094**	0.619	0.021	0.517	40.335
C3NET	0.080	0.529	0.026	0.506	0.018	0.501	0.000
CLR	0.217	0.666	0.050	0.538	0.019	0.505	4.928
MRNET	0.194	0.668	0.041	0.525	0.018	0.501	2.534
ARACNE	0.099	0.545	0.029	0.512	0.017	0.500	0.000
Winner of the challenge							
GENIE3	0.291	0.815	0.093	0.617	**0.021**	**0.518**	40.279
2nd	0.245	0.780	0.119	0.671	**0.022**	0.519	34.023
3rd	0.301	0.782	0.069	0.595	0.020	0.517	31.099

Additionally, a list of known transcription factors were provided together with expression data. This could help participants to narrow the range of possible regulatory interactions. A similar form of anonymization like in the other DREAM challenges was used: all genes were assigned arbitrary identifiers.

Only expression data for the first network in DREAM5 challenge were generated *in silico*. The second network was originally designed to represent a compendium of microarray data from *S. aureus*. However, eventually it was not used for evaluation and no gold standard was constructed. The third network represented a compendium of microarray data from *E. coli*. A list of known transcription factors was obtained from two sources: RegulonDB database [40], and Gene Ontology (GO) annotations [5]. The last network represented a compendium of microarray data from *S. cerevisiae*. A list of known transcription factors was obtained from [99] and Gene Ontology annotations. Despite all the effort in construct-

Table 5.8: Precision of ADANET and the other GRN inference methods. The methods were run on a data set described in Section 5.4. Parameters of the methods were set as reported in Section 5.4. Precision for the best n edges is a fraction of the number of the true edges (TP) to the number of all the best ranked n edges. Area under precision-recall curve is denoted as AUPR. The best results for each of the columns are in bold.

Method	Precision for the best n edges					AUPR
	50	100	200	500	1000	
ADANET	**0.90**	**0.79**	0.55	**0.33**	**0.22**	0.0729
GENIE3	0.80	0.68	**0.56**	**0.33**	**0.22**	**0.0732**
C3NET	0.58	0.40	0.29	0.12	0.06	0.0254
CLR	0.44	0.32	0.24	0.14	0.09	0.0275
MRNET	0.24	0.18	0.17	0.11	0.08	0.0232
ARACNE	0.26	0.17	0.13	0.09	0.05	0.0173
ADANET-L	0.88	0.75	0.56	0.33	0.22	0.0741
ADANET-XL	0.88	*0.80*	*0.60*	0.33	*0.24*	*0.0785*

ing gold standards for *in vivo* networks, they contain only a subset of the true interactions, e.g. they may contain many false negatives (FN). Therefore, predicted interactions that are not part of the gold standard can be either truly incorrect or newly discovered interactions, as described in Section 2.2.

The results of all the inference methods for DREAM5 expression data are summarized in Table 5.7. Clearly all the participating methods achieved better scores for an *in silico* network than for either one of *in vivo* networks. ENNET achieved overall score of infinity for this challenge, which is a good result with respect to the *in silico* network, which was modeled in a genome-wide scale. It means that for that network the proposed methods achieved p-value lower than the smallest positive real number available for our computer architecture, which is 2.2251e-308. However, none of the proposed inference algorithms nor the other reviewed methods did give accurate predictions for *in vivo* networks. The results were especially disappointing with respect to *S. cerevisiae*.

5.4 Expression profiles of *E. coli*

We applied ADANET to expression data from *E. coli*. From Many Microbe Microarrays (M^{3D}) database [28] (version 4 build 6) we downloaded a dataset containing 466 microarrays of 4297 genes. From [40] we downloaded manually assembled RegulonDB Release 7.4 reference network as a gold standard.

From the gold standard network we excluded all the cases, in which a gene was regulating itself. Following standard practice [4, 29, 51], we identified 1443 genes with known regulatory interactions, i.e. genes having at least one incoming or outgoing edge, and 160 transcription factors, i.e. genes having at least one outgoing edge. Considering the gold standard, there were 2873 true edges. We bounded the domain of all possibly adjacent genes to 160 transcription factors and 1443 regulated genes.

We limited ADANET in the problem decomposition phase, as described in Section 4.3.1, i.e. we decomposed the GRN inference problem to 1443 feature selection problems, each of them of size 160. We used the following parameters for ADANET: $C = 30$, $T = \lceil \sqrt{P} \rceil = 13$, $\psi = 0.25$, $\xi = 0.75$, $\delta = 0.05$, and $s = 0.67$, as discussed later in Section 6.1.

We compared the results of ADANET with the other GRN inference algorithms:

1. Popular Mutual Information-based (MI) network inference methods: C3NET, ARACNE, CLR, and MRNET. We downloaded an implementation of: ARACNE, CLR, and MRNET from [69], and C3NET from [4]. For C3NET we created MI matrix as proposed in [4], and set the statistical significance threshold to the default 0.01. For ARACNE, CLR, and MRNET we created MI matrix as proposed in [69]. For ARACNE we set the threshold used when removing an edge to the default 0.

2. GENIE3 algorithm, which was proved to work well on a synthetic dataset DREAM4

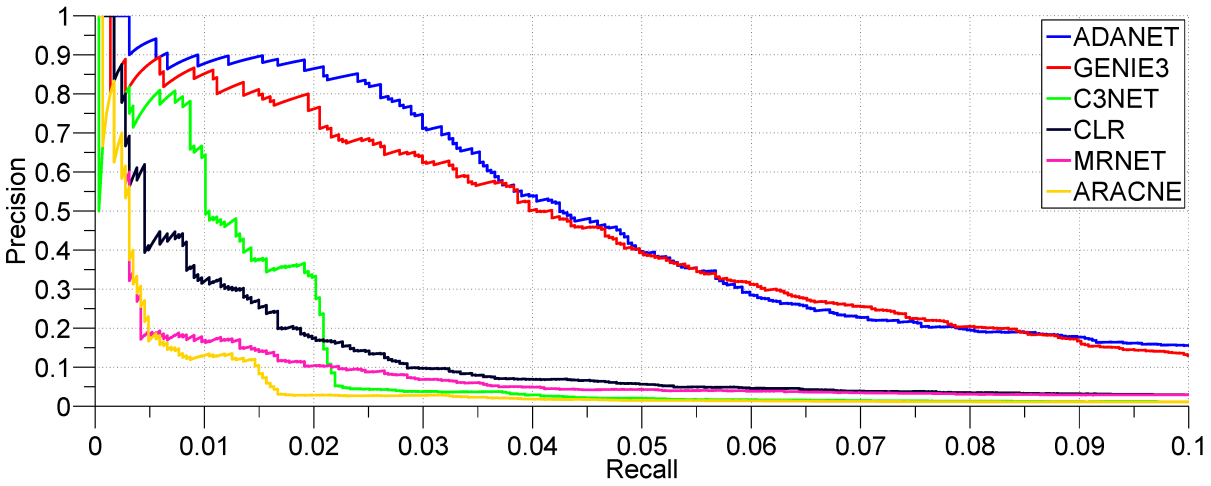

Figure 5.1: Precision-recall dependency of ADANET and the other GRN inference algorithms on *E. coli* expression profiles. The parameters of the methods were set as reported in Section 5.4, that is, for ADANET, we used: $C = 30$, $T = \lceil\sqrt{P}\rceil = 13$, $\psi = 0.25$, $\xi = 0.75$, $\delta = 0.05$, and $s = 0.67$. The settings of the parameters are discussed in Section 6.1. Our method has precision higher by around 10% than the second best performer in the most crucial part of the plot, where the high precision of the methods promises an accurate inferred network.

in [51]. For *E. coli* dataset, as a parameter of GENIE3, we used the default Random Forest of 1000 trees. Because ADANET and GENIE3 share the same problem decomposition principle, we decided to limit the feature selection space for GENIE3 in the same way as for ADANET, as described above in this section.

In Table 5.8 we show precision of ADANET, and compare it to the precision of the other GRN inference algorithms. Because all the methods rank predicted edges, we measure precision as a ratio of the true edges (TP) predicted by the method to the number of all the predicted edges (TP+FP) for a given number of the best ranked edges TP+FP. Additionally, we report the area under precision-recall curve for all the methods.

We compare comprehensively the GRN inference algorithms in Figure 5.1. For each of the method we present precision-recall curve for the most interesting part of the plot, where recall is lower or equal than 0.1, and the high precision of the methods promises

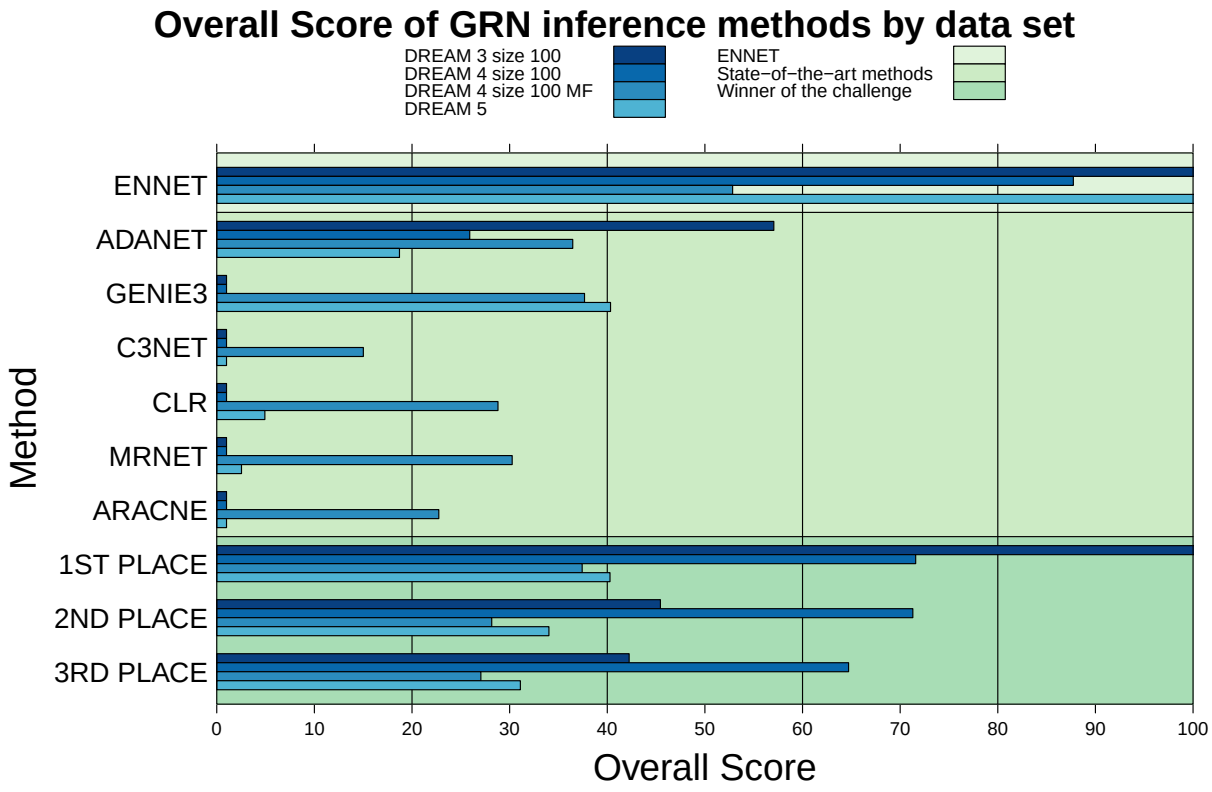

Figure 5.2: Results of the different inference methods on DREAM challenges. Results of the state-of-the-art methods were collected after running the algorithms with the default sets of parameters on pre-processed data. Results in the "Winner of the challenge" part of the figure correspond to the best methods participating in the challenge.

an accurate inferred network. In the most crucial part of the plot, from a practical point of view, our method has precision higher by around 10% than the second best performer. From Table 5.8, and Figure 5.1, we observe a high precision of ADANET, relatively to the other methods.

5.5 Analysis of Results for Large Regulatory Networks

We assessed the performance of the proposed inference algorithms: ENNET and ADANET on large Gene Regulatory Networks of 100 and more genes, and compared them to the state-of-the-art methods in Figure 5.2. We summarize the results of running different inference

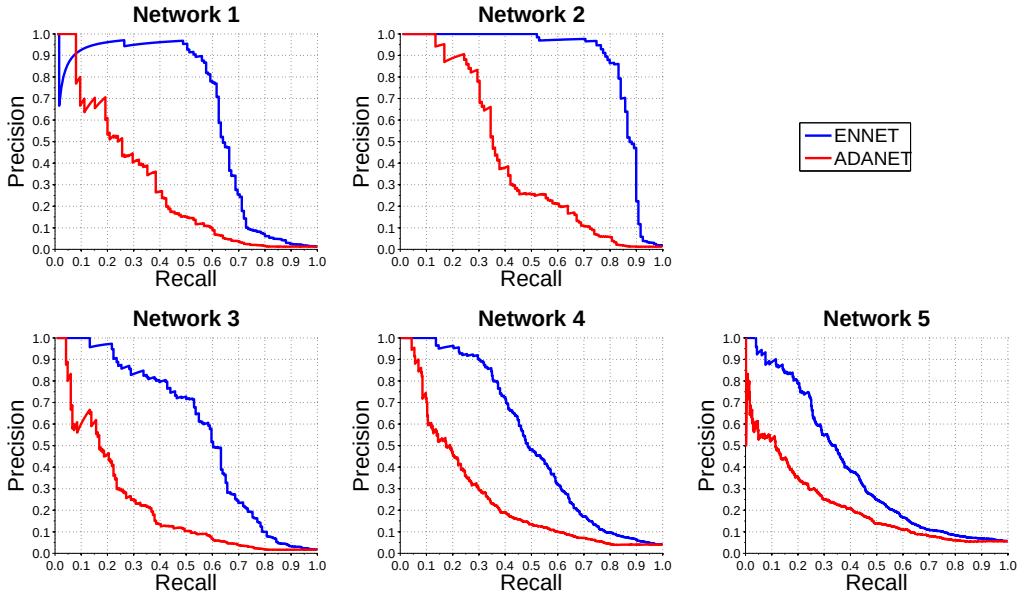

Figure 5.3: Precision-Recall dependency of different inference methods on DREAM3 networks, challenge size 100. Each of the benchmark networks consists of 100 genes, as shown in Table 2.1. Participating methods are described in details in Section 4.3. Methods, which do not accept input expression profiles other than MF table, as described in Section 2.4, are excluded from the comparison. Validation based on Precision-Recall curve is described in Section 2.3. Methods from the "Winner of the challenge" part of Figure 5.2 are excluded from this comparison due to the fact that Precision-Recall dependency of these method was not published.

methods presented above in Tables 5.3, 5.5, 5.6, 5.7. For a comparison we selected a range of established methods from the literature: ARACNE, CLR, and MRNET as implemented in the *minet* R package [69], GENIE3 and C3NET as implemented by their respective authors, and the top three performers in each of the three DREAM challenges as listed on the DREAM web site. Some of the methods were designed for use with knockout data, while others were developed with multifactorial data in mind, where no information was given about the nature of the perturbations. Therefore, depending on the type of the data available in the particular DREAM data set, only the suitable group of methods was used for the comparison. Note that methods in the "Winner of the challenge" part of Figure 5.2 correspond to different inference methods, e.g., the "1ST PLACE" method in DREAM 3

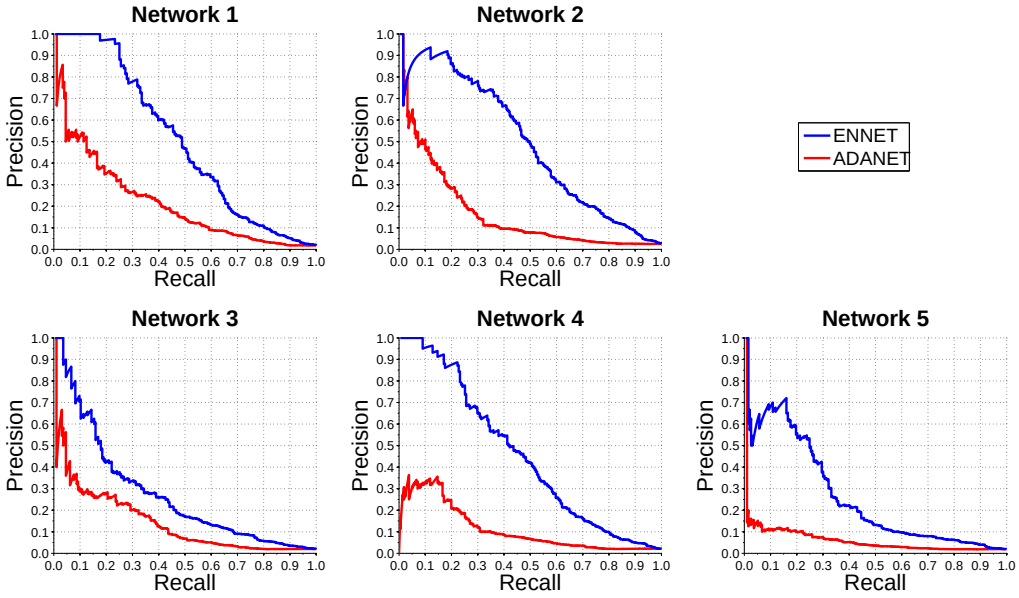

Figure 5.4: Precision-recall dependency of different inference methods on DREAM4 networks, challenge size 100. Each of the benchmark networks consists of 100 genes, as shown in Table 2.1. Participating methods are described in details in Section 4.3. Methods, which do not accept input expression profiles other than MF table, as described in Section 2.4, are excluded from the comparison. Validation based on Precision-Recall curve is described in Section 2.3. Methods from the "Winner of the challenge" part of Figure 5.2 are excluded from this comparison due to the fact that Precision-Recall dependency of these method was not published.

size 100 challenge was different than the "1ST PLACE" method in DREAM 5 challenge. Our proposed algorithm ENNET was the only one that performed robustly across all the large networks.

Precision-Recall dependencies of the inference methods ENNET and ADANET for DREAM 3 size 100 and DREAM 4 size 100 are presented in Figures 5.3 and 5.4, respectively. ENNET algorithm calculated consistently better predictions than ADANET both for DREAM 3 size 100 and for DREAM 4 size 100 challenges. ENNET achieved both higher area under the curve, see Table 5.3 and 5.5, and a favorable shape of the plot, i.e. from the practical point of view a rapid drop in Precision for a higher values of Recall is proffered over a steady drop of Precision in the full range of Recall. Note that only methods that accept

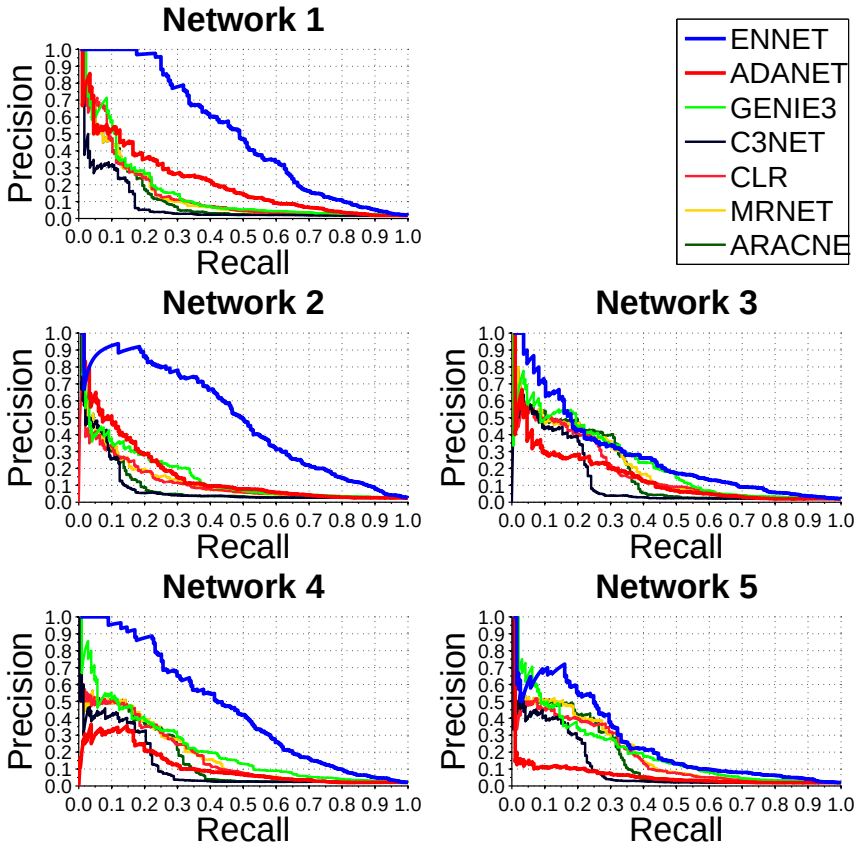

Figure 5.5: Precision-recall dependency of different inference methods on DREAM4 networks, challenge size 100 multifactorial. Each of the benchmark networks consists of 100 genes, as shown in Table 2.1. Participating methods are described in details in Sections 2.1 and 4.3. Validation based on Precision-Recall curve is described in Section 2.3.

expression profiles different than MF table, as described in Section 2.4, were included in this comparison.

Precision-Recall dependency of the inference methods, with respect to DREAM4 size 100 multifactorial challenge, is presented in Figure 5.5. Because MF table was available for this challenge, see Section 2.4, all the methods discussed previously were included in the comparison. However, refining the edges in ENNET and ADANET, as described in Section 4.4.2 could not be applied, since knockout data were not available. Methods which exploit

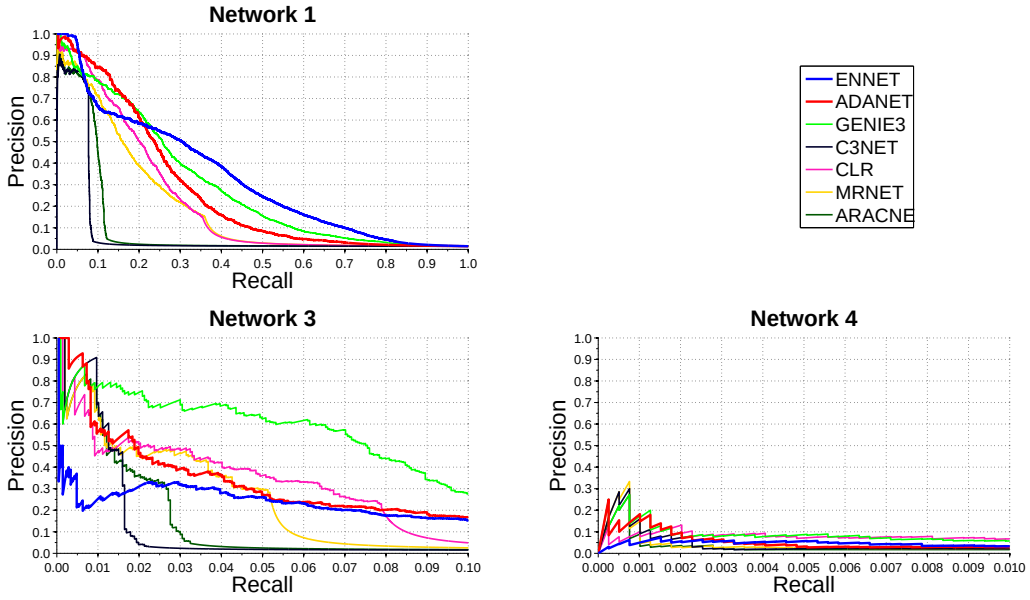

Figure 5.6: Precision-recall dependency of different inference methods on DREAM5 networks. Each of the benchmark networks consists of different number of known transcription factors and regulated genes, as shown in Table 2.1. Participating methods are described in details in Sections 2.1 and 4.3. Validation based on Precision-Recall curve is described in Section 2.3. Note that for Network 3 only Recall $\in (0, 0.1)$ part of the plot is shown, and for Network 4 only Recall $\in (0, 0.01)$. Prediction accuracy for these two *in vivo* networks is poor.

variance of the edge predictions achieved favorable trajectory of Precision-Recall curve over the competitors. However, due to the lack expression profiles of known initial perturbation state, i.e. knockout or knockdown data, neither of the methods achieved accuracy as high as in the DREAM4 size 100 challenge.

Precision-recall dependency of the inference methods, with respect to DREAM5 challenge, is presented in Figure 5.6. None of the methods calculated an accurate prediction of Networks 3 and 4, i.e. networks from expression profiles collected *in vivo* from *E. coli* and *S. cerevisiae* organisms. Clearly, GENIE3 method calculated the best prediction for *E. coli* organism. However, reverse-engineering gene networks for eukaryotes, such as *S. cerevisiae*, becomes a challenge for all the methods described in this study.

Chapter 6

Discussion

In this chapter we discuss the results that we achieved on benchmark data sets, as well as the general properties of our proposed methods. First, we present the default set of parameters of ADANET algorithm, and show its computational complexity. Later, we discuss the same aspects of ENNET. Moreover, we discuss important aspects of the analysis of a Precision-Recall dependency of our inference models.

6.1 Setting parameters of ADANET

The accuracy of ADANET algorithm might depend on a set of parameters for a given data set. In this section we discuss the importance of the parameters controlling ADANET, running the algorithm on a data set described in Section 5.4. There are the following parameters to set: $C \in \mathbb{N}$, $T \in \mathbb{N}$, $\psi \in [0, 1]$, $\xi \in [0, 1]$, $\delta \in [0, 1]$, and $s \in [0, 1]$.

In Figure 6.1 we show how the accuracy of ADANET depends on a choice of the number of classification problems C, and the number of importance scores T, which together control the number of iterations of the main loop of ADANET. The accuracy is significantly decreased for low ranges of iterations: $(C, T) \in [1, 10] \times [1, 10]$, and reaches maximal values, and flats out in a range of $(C, T) \in [30, 100] \times [15, 30]$. However, an interesting fact

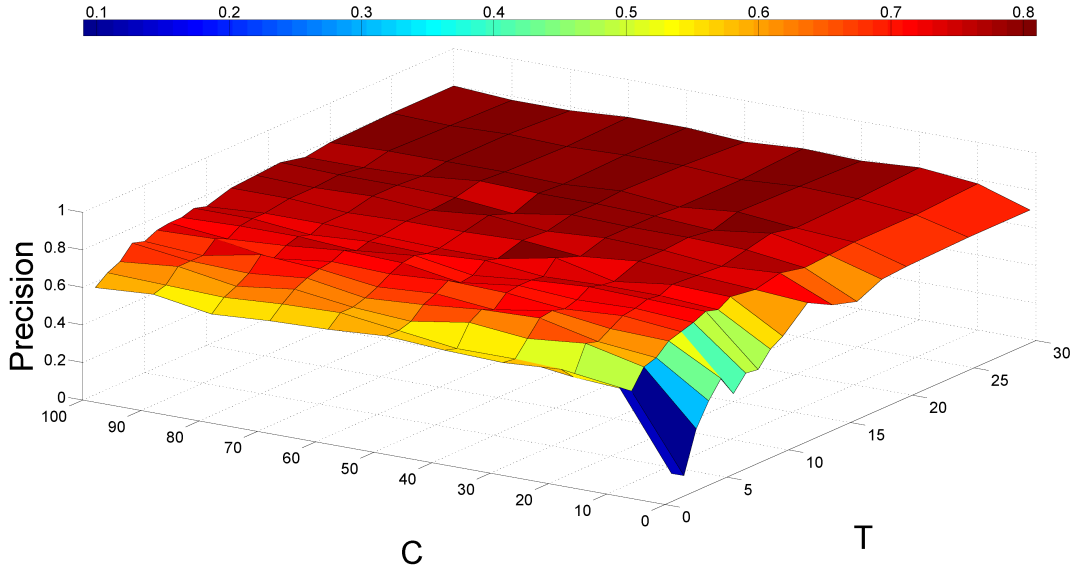

Figure 6.1: Precision of ADANET with respect to parameters C, and T. Precision was measured for the top ranked 100 edges, as described in Section 5.4. Other parameters set to: $\psi = 0.25$, $\xi = 0.75$, $\delta = 0.05$, $s = 0.67$.

is worth noticing, that for any number of classification problems C, the accuracy does not seem to decrease in function of T, i.e. the algorithm does not overtrain, which is a common observation when applying AdaBoost meta-classifier to classification problems.

For further analysis, we selected three sets of parameters with increasing C, and T: the default number of iterations, denoted simply as ADANET: $C = 30$, $T = 13$, elevated number of iterations, denoted as ADANET-L: $C = 60$, $T = 16$, and high number of iterations, denoted as ADANET-XL: $C = 100$, $T = 30$. The rest of the parameters were set to: $\psi = 0.25$, $\xi = 0.75$, $\delta = 0.05$, and $s = 0.67$. In Table 5.8 we report the precision of ADANET for all the three sets of parameters, as well as the area under precision-recall curve. Moreover, in Figure 6.2 we show precision-recall curves, which we obtained running ADANET for all the three sets of parameters, for recall lower or equal than 0.1. We do not observe a substantial improvement in accuracy for increasing number of iterations of the main loop of ADANET. However, especially for higher values of recall, for the elevated

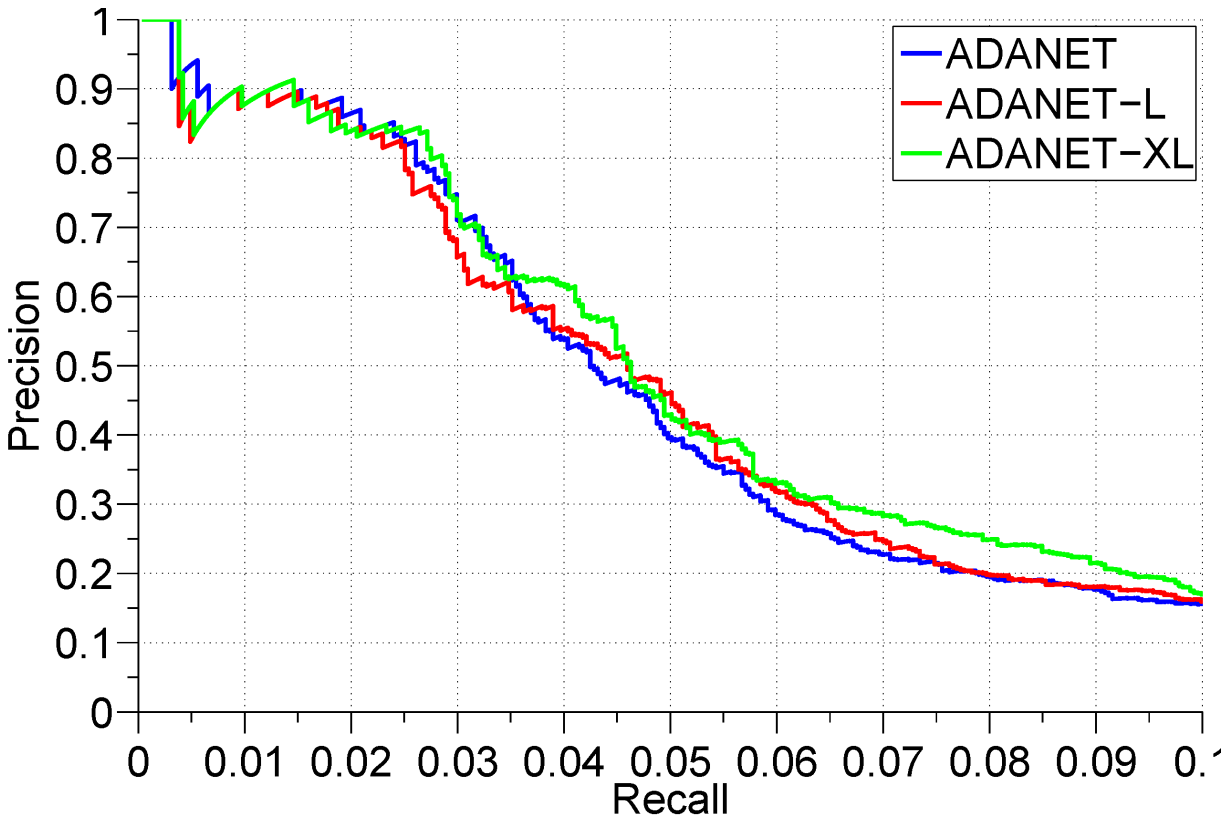

Figure 6.2: Precision-recall dependency of ADANET with respect to different parameters C, and T. ADANET: $C = 30$, $T = 13$, ADANET-L: $C = 60$, $T = 16$, ADANET-XL: $C = 100$, $T = 30$. Other parameters set to: $\psi = 0.25$, $\xi = 0.75$, $\delta = 0.05$, $s = 0.67$.

and high values of parameters C, and T, we observe a slight improvement in the precision of ADANET, which also reflects in a higher area under precision-recall curve.

In Figure 6.3 we show how the accuracy of ADANET depends on parameters ψ, and ξ. Maximal accuracy is reached for parameters ψ and ξ skewed towards medium and high ranges of expression values, i.e. $(\psi, \xi) \in [0.3, 0.5] \times [0.7, 0.8]$. The diagonal of the map again indicates that, for our method, the variability in medium and high ranges of expression values is the most informative.

In Figure 6.4 we show how the accuracy of ADANET depends on a margin width δ, and

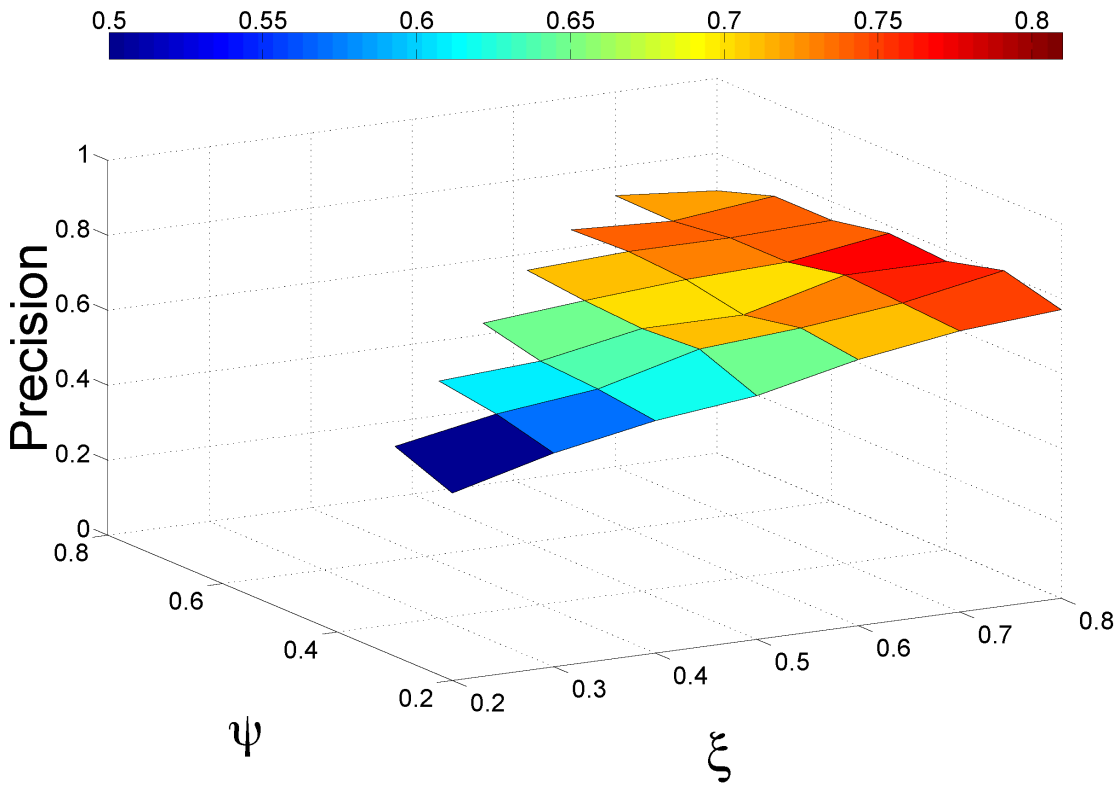

Figure 6.3: Precision of ADANET with respect to parameters ψ, and ξ. Precision was measured for the top ranked 100 edges, as described in Section 5.4. Other parameters set to: $C = 30$, $T = 13$, $\delta = 0.05$, $s = 0.67$.

sampling rate s. The best accuracy is achieved for low δ, and high s. This allows the algorithm to operate on higher number of training samples in each iteration.

With regard to the above results, we recommend using the following default set of parameters for ADANET: $C \geq 30$, $T \geq \lceil\sqrt{P}\rceil$, $\psi = 0.25$, $\xi = 0.75$, $s = 0.67$, and $\delta = 0.05$.

6.2 Computational Complexity of ADANET

In this section we present computational complexity of ADANET and compare it to the other GRN inference methods in Table 6.1.

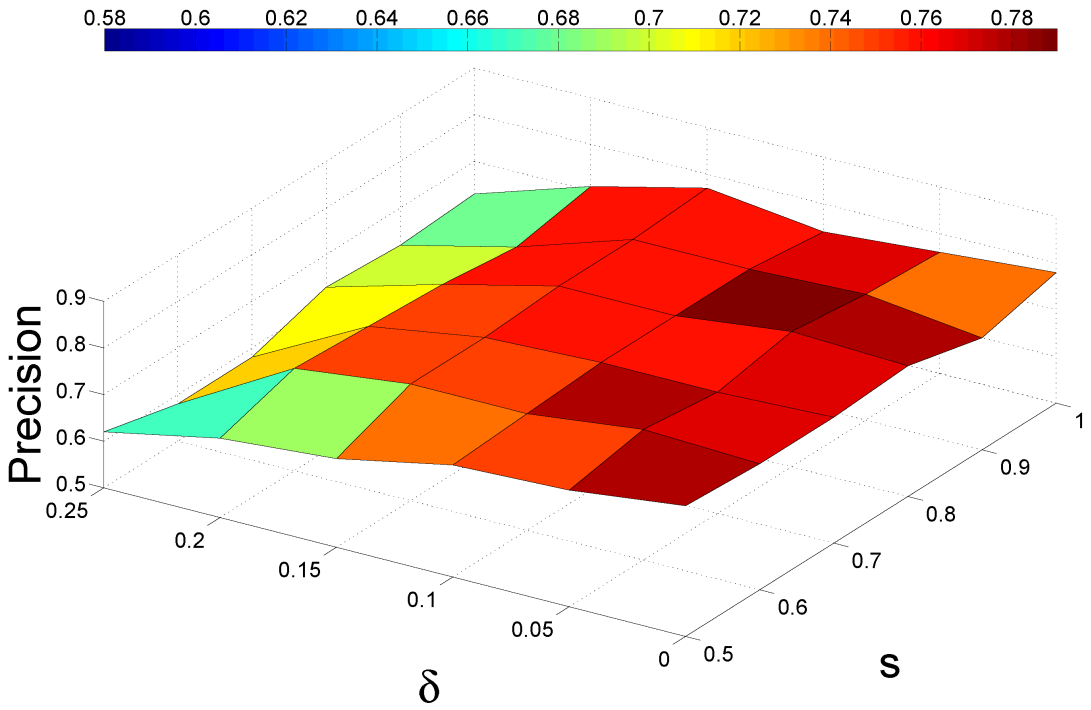

Figure 6.4: Precision of ADANET with respect to parameters δ, and s. Precision was measured for the top ranked 100 edges, as described in Section 5.4. Other parameters set to: $C = 30$, $T = 13$, $\psi = 0.25$, $\xi = 0.75$.

We sort E with complexity $O(PN \log N)$, and call P times Algorithm 3. In Algorithm 3 we call C times Algorithm 4. Therefore, the computational complexity of ADANET depends mainly on the computational complexity of Algorithm 4. In Algorithm 4 we call T times Decision Stump algorithm. Given sorted input, Decision Stump algorithm is $O(PN)$ complex. The computational complexity of the whole method is thus $O(PN \log N + CTP^2N)$. Because, in practice, the dominating part of the sum is CTP^2N, we finally report computational complexity of ADANET as $O(CTP^2N)$.

With our MATLAB implementation of ADANET, it took 56 minutes to infer the network using the default parameters described in Section 5.4. It is around 5 times faster than

Table 6.1: Computational complexity of ADANET and the other GRN inference methods. Running times were measured for the network described in Section 5.4, on a 16GB RAM, Intel®Core™i7-870 Processor (8M Cache, 2.93GHz) computer.

Method	Complexity	Running time
ENNET	$O(TP^2N)$ $T = 5000$	113 min
ADANET	$O(CTP^2N)$ $C = 30$, $T = \lceil\sqrt{P}\rceil$	56 min
GENIE3	$O(TKPN \log N)$ $T = 1000$, $K = \lceil\sqrt{P}\rceil$	4h 55min
C3NET	$O(P^2)$	7 sec
CLR	$O(P^2)$	7 sec
MRNET	$O(fP^2)$, $f \in [1, P]$	25 sec
ARACNE	$O(P^3)$	13 sec

the other machine learning-based method we tested, GENIE3, with default parameters. Execution time was measured on a computer described in Table 6.1. Our implementation of ADANET was serial, i.e. we used only a single processor in runtime. However, because the gene selection problems are independent of each other, it is feasible to implement efficient parallel code of ADANET.

6.3 Setting parameters of ENNET

ENNET algorithm is controlled by four parameters: the two sampling rates s_s and s_f, the number of iterations T and the learning rate ν. The sampling rate of samples s_s and the sampling rate of transcription factors s_f govern the level of randomness when selecting rows and columns of the expression matrix to fit a regression model, respectively. The default choice of the value of s_s is 1, e.g. we select with replacement a bootstrap sample of observations of the same size as an original training set at each iteration. Because some observations are selected more than once, around 0.37 of random training samples are out of bag in each iteration. It is more difficult to choose an optimal value of s_f, which governs how many transcription factors are used to fit each base learner. Setting this parameter to a low

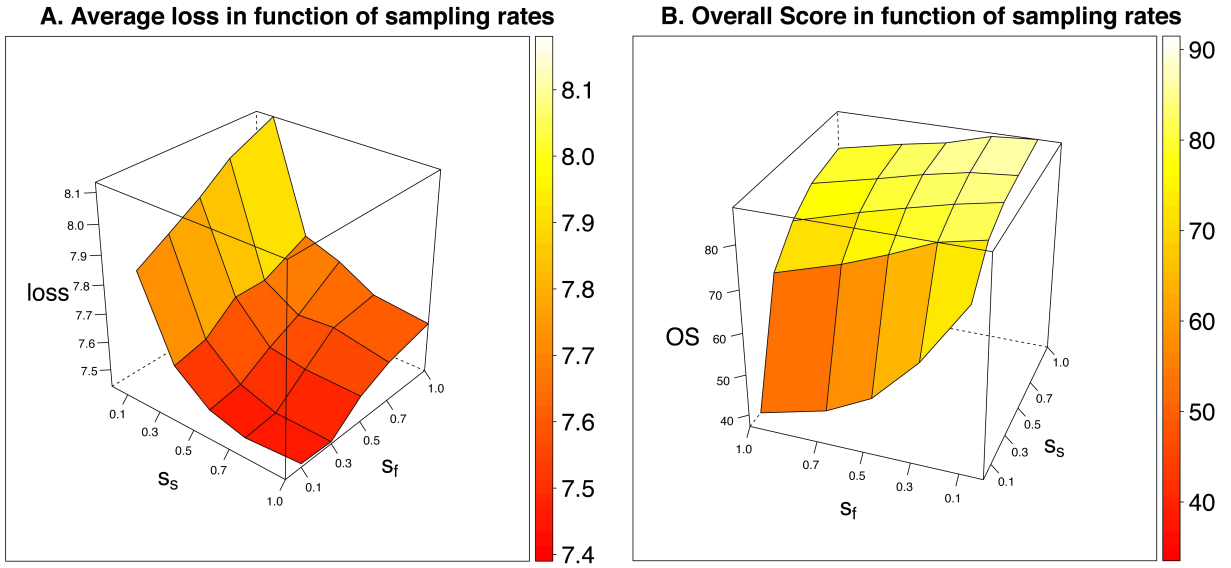

Figure 6.5: The analysis of the sampling rates s_s and s_f for the DREAM 4 size 100 challenge. **A:** For each set of parameters $(s_s, s_f, M, \nu) \in \{0.1, 0.3, 0.5, 0.7, 1\} \times \{0.1, 0.3, 0.5, 0.7, 1\} \times \{5000\} \times \{0.001\}$ we analyzed an average 5-fold cross-validated loss over all the observations (across all gene selection problems) from all 5 networks. The minimal average loss was achieved for high values of $s_s = 1$ and low values of $s_f = 0.3$. **B:** We also compared the measure based on an average loss with the original Overall Score, as proposed by the authors of the DREAM challenge. The results were consistent across the two measures, e.g. a selection of parameters, which gave a low average loss, led to accurate network predictions (a high Overall Score).

value forces ENNET to score transcription factors, even if their improvement criterion, as shown in Formula 4.4, would not have promoted them in a pure greedy search, e.g. $s_f = 1$. However, if a chance of selecting a true transcription factor as a feature is too low, ENNET will suffer from selecting random genes as true regulators. Even though reverse-engineering of GRNs does not explicitly target a problem of predicting gene expression, we choose the values of sampling rates such that the squared-error loss of a prediction of the target gene expression as given by f_T (see Figure 4.4) is minimal. This is done without looking at the ground truth of regulatory connections. For each benchmark challenge we performed a grid search over $(s_s, s_f) \in \{0.1, 0.3, 0.5, 0.7, 1\} \times \{0.1, 0.3, 0.5, 0.7, 1\}$ with fixed $\nu = 0.001$, $T = 5000$. For each specific set of parameters we analyzed an average 5-fold cross-validated loss over all the observations (across all gene selection problems). We further analyze our

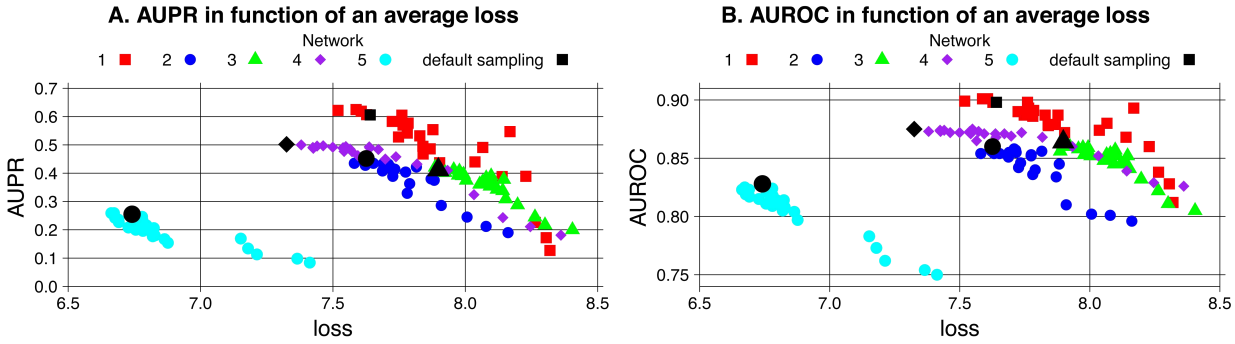

Figure 6.6: The analysis of the sampling rates s_s and s_f for DREAM 4 size 100 challenge. **A:** For each set of parameters $(s_s, s_f, M, \nu) \in \{0.1, 0.3, 0.5, 0.7, 1\} \times \{0.1, 0.3, 0.5, 0.7, 1\} \times \{5000\} \times \{0.001\}$ we analyzed an area under the Precision-Recall curve (AUPR) in function of an average 5-fold cross-validated loss over all the observations (across all gene selection problems) from all 5 networks. For each network AUPR is decreasing in a function of a loss. For each network a point corresponding to the default set of parameters is highlighted, e.g. $(s_s, s_f, M, \nu) = (1, 0.3, 5000, 0.001)$. Usually, the default set of parameters gives the minimal loss (maximal AUPR). **B:** By analogy, different choices of parameters lead to a different area under the ROC curve (AUROC). The two measures are consistent with each other.

approach with respect to one of the challenges: DREAM4 size 100, as shown in Figure 6.5. The minimal average loss was achieved for $s_s = 1$ and $s_f = 0.3$, which is consistent with the default parameters proposed for Random Forest algorithm [51], see Figure 6.5 A. We also compared the measure based on an average loss with the original Overall Score, as proposed by the authors of DREAM challenge. The results were consistent across the two measures, e.g. a selection of parameters that gave a low average loss led to the accurate network predictions, see Figure 6.5 B. An advantage of the average loss measure is a fact that the gold standard network is not used to tune parameters. In Figure 6.6 we present a detailed analysis of the accuracy of the GRN inference across different networks of a DREAM4 size 100 challenge. Each point on both Figure 6.6 A and Figure 6.6 B is a result of running ENNET with different parameters: $(s_s, s_f) \in \{0.1, 0.3, 0.5, 0.7, 1\} \times \{0.1, 0.3, 0.5, 0.7, 1\}$ with fixed $\nu = 0.001$, $T = 5000$. The highlighted points are corresponding to $s_s = 1$, $s_f = 0.3$, $\nu = 0.001$, $T = 5000$. An area under the Precision-Recall curve and an area under the ROC curve are two different measures of the accuracy of an inferred network,

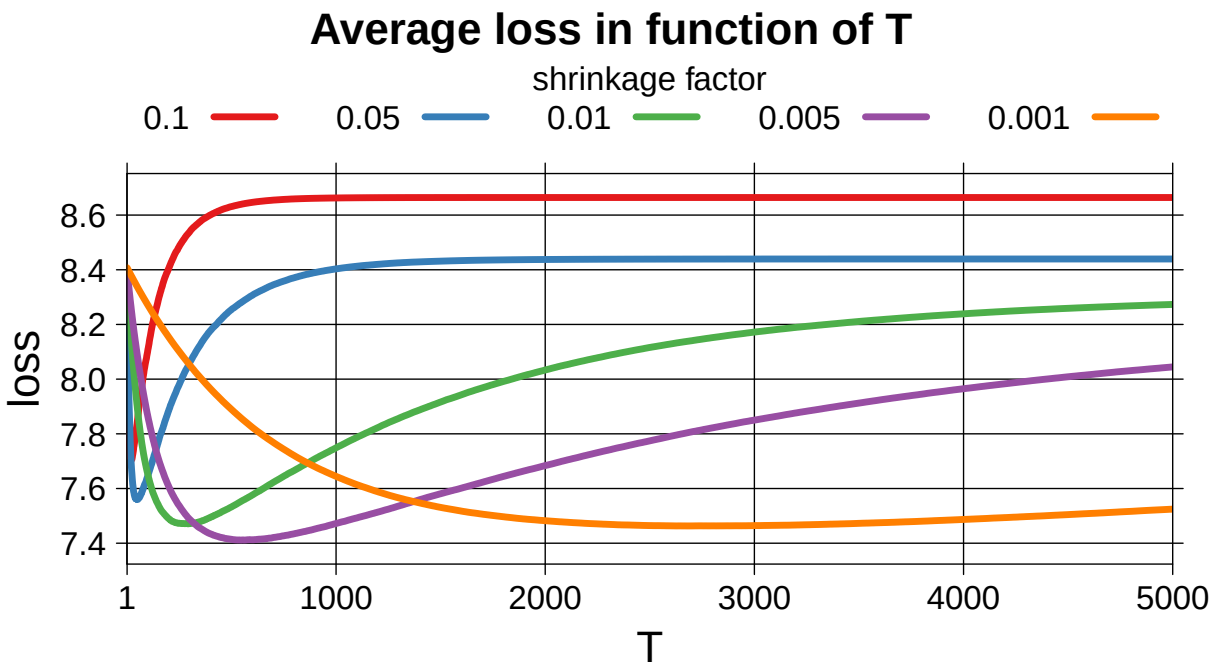

Figure 6.7: The analysis of the number of iterations T and the shrinkage factor ν for DREAM 4 size 100 challenge. These two parameters are closely coupled: the lower is the shrinkage parameter ν, the more iterations T are needed to train the model such that it achieves the minimal loss.

which are well preserved across the five networks: for each separate network we observe that AUPR and AUROC decreases in a function of an average loss. As the Overall Score is closely related to AUPR and AUROC, the results shown in Figure 6.6 explain the shape of a surface shown in Figure 6.5. As ENNET uses boosting, it needs a careful tuning of the number of iterations T and the learning rate ν. It has been shown [37] that parameters T and ν are closely coupled. Usually the best prediction results are achieved when ν is fixed to a small positive number, e.g. $\nu \leq 0.001$, and the optimal value of T is found in a process of cross-validation. As described above, we reason that the choice of parameters, which gives a low average loss on a cross-validated test set, leads to an accurate network prediction. Therefore in Figure 6.7 we present how an average loss depends on $T \in \{1, ..., 5000\}$ for different values of $\nu \in \{0.001, 0.005, 0.01, 0.05, 0.1\}$, with fixed $s_s = 1$, $s_f = 0.3$. Each of the line shows how much ENNET overtrains the data for a given T and ν. Finally,

the optimal choice of parameters for DREAM4 size 100 challenge is $s_s = 1$, $s_f = 0.3$, $T = 5000$, $\nu = 0.001$. Following the same practice, we used this default set of parameters: $s_s = 1$, $s_f = 0.3$, $T = 5000$, $\nu = 0.001$ to evaluate ENNET algorithm on all the benchmark networks using ground truth, e.g. for calculating the Overall Score and comparing it to the other algorithms.

6.4 Computational complexity of ENNET

Computational complexity of ENNET depends mainly on the computational complexity of the regression stump base learner, which is used in the main loop of the algorithm. As shown in Figure 4.4, we call the regression stump algorithm T times for each k-th target gene, $k \in \{1, ..., P\}$. Given a sorted input, a regression stump is $O(PN)$ complex. We sort the expression matrix in an $O(PN \log N)$ time. All the other instructions in the main loop of ENNET are at most $O(N)$. The computational complexity of the whole method is thus $O(PN \log N + TP^2N + TPN)$. Because, in practice, the dominating part of the sum is TP^2N, we report a final computational complexity of ENNET as $O(TP^2N)$, and compare it to the other inference methods in Table 6.1. Note that the measure for the information-theoretic methods: CLR, MRNET, and ARACNE does not include a calculation of the mutual information matrix.

When implementing ENNET algorithm we took advantage of the fact that gene selection problems are independent of each other. Our implementation of the algorithm is able to calculate them in parallel, given that multiple processors are available. User can choose from variety of parallel backends including multicore package for a single computer and parallelization based on Message Passing Interface for a cluster of computers. The biggest data we provided as input were *in vivo* expression profiles of *S. cerevisiae* from DREAM 5 challenge. These are genome-wide expression profiles of size 5950 genes (333 of them are

known transcription factors) and 536 experiments. It took 113 minutes and 30 seconds to calculate the network on a standard desktop workstation with one Intel®Core™i7-870 processor with 4 cores and two threads per core (in total 8 logical processors), and 16 GB RAM. However, it took only 16 minutes and 40 seconds to calculate the same network on a machine with four AMD Opteron™6282 SE processors, each with 8 cores and two threads per core (in total 64 logical processors), and 256 GB RAM. All the data sets from DREAM 3 and DREAM 4 challenges were considerably smaller, up to 100 genes. It took less than one minute to calculate each of these networks on a desktop machine.

6.5 Stability of ENNET

Because ENNET uses random sampling of samples and features at each iteration of the main loop, as shown in Figure 4.4, it may calculate two different networks for two different executions. With the default choice of parameters, e.g. $s_s = 1$, $s_f = 0.3$, $T = 5000$, $\nu = 0.001$, we expect numerous random resamplings, and therefore we need to know if a GRN calculated by ENNET is stable between different experiments. We applied ENNET to the 5 networks that form DREAM 4 size 100 benchmark, repeating the inference calculations independently ten times for each network. Then, for each network, we calculated a Spearman's rank correlation between all pairs among the ten independent runs. The lowest correlation coefficient we obtained was $\rho > 0.975$, with p-value $< 2.2e-16$, indicating that the networks that result from independent runs are very similar. This proves that ENNET, despite being a randomized algorithm, finds a stable solution to the inference problem.

Chapter 7

Conclusions

We have proposed a set of methods for reverse-engineering of Gene Regulatory Networks. They use variety of types of expression data as input, and show robust performance across different benchmark networks. Moreover, the algorithms do not assume a specific mathematical model of a regulatory interaction, e.g., a linear model, and do not require fine-tuning of their parameters, which promises accurate predictions for the unknown networks. Also, processing large, genome-scale expression profiles is feasible: including up to few hundreds transcription factors, and up to few thousands regulated genes. As shown in this study, the proposed methods compare favorably to the state-of-the-art algorithms on the universally recognized expression data sets. However, predictions for *in vivo* expression profiles are inaccurate. One of the reasons for a poor performance of the inference methods for such expression profiles is the fact that experimentally confirmed pathways, and consequently gold standards derived from them, can not be assumed complete, regardless of how well is a model organism known. In other words, only a subset of the existing regulatory interactions is known. On top of that, predictors struggle to process undersampled and noisy data sets. Additionally, there are regulators of gene expression other than transcription factors, such as miRNA, and siRNA. In other words, systems of transcriptional regulations in living organisms could be far more sophisticated than what synthetic benchmarks assume. As

shown in this study, *in silico* expression profiles provide enough information to confidently reverse-engineer their underlying structure, whereas *in vivo* data hide much more complex system of regulatory interactions.

List of Terms

ADANET	ADAptive boosting of decision stump-based gene regulatory NETwork inference algorithm
ARACNE	Algorithm for the Reconstruction of Accurate Cellular Networks
AUPR	Area under Precision-Recall curve
AUROC	Area under ROC curve
BN	Bayesian Network
C3NET	Conservative Causal Core Network
CART	Classification and Regression Trees
CLR	Context Likelihood of Relatedness
DNA	Deoxyribonucleic Acid
DPI	Data Processing Inequality
DREAM	Dialogue for Reverse Engineering Assessments and Methods
FN	False Negatives
FP	False Positives
GBM	Gradient Boosting Machine
GENIE3	GEne Network Inference with Ensemble of trees
GO	Gene Ontology
GRN	Gene Regulatory Network
KD	Knockdown expression data
KO	Knockout expression data
M^{3D}	Many Microbe Microarrays
MART	Multiple Additive Regression Trees
MF	Multifactorial expression data
MI	Mutual Information
miRNA	Micro RNA

MN	Markov Network
MRMR	Maximum Relevance/Minimum Redundancy
mRNA	Messenger RNA
NIPS	Neural Information Processing Systems Foundation
ODE	Ordinary Differential Equation
OOB	Out-of-bag samples
PDB	Protein Data Bank
PPV	Positive Predictive Value
PR	Precision-Recall
RF	Random Forest
RNA	Ribonucleic Acid
ROC	Receiver Operating Characteristics
siRNA	Small Interfering RNA
TF	Transcription Factor
TN	True Negatives
TP	True Positives
TPR	True Positive Rate
TRN	Transcriptional Regulatory Network
TS	Time Series expression data
WT	Wildtype expression data

Bibliography

[1] T. Akutsu, S. Miyano, and S. Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16(8):727–734, 2000.

[2] T. Akutsu, S. Miyano, S. Kuhara, et al. Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. In *Pacific Symposium on Biocomputing*, volume 4, pages 17–28. World Scientific Maui, Hawaii, 1999.

[3] G. Altay and F. Emmert-Streib. Inferring the conservative causal core of gene regulatory networks. *BMC Systems Biology*, 4(1):132, 2010.

[4] G. Altay and F. Emmert-Streib. Structural influence of gene networks on their inference: analysis of C3NET. *Biology Direct*, 6(1):31, 2011.

[5] M. Ashburner, C. Ball, J. Blake, D. Botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J. Eppig, et al. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25, 2000.

[6] M. Bansal, V. Belcastro, A. Ambesi-Impiombato, and D. di Bernardo. How to infer gene networks from expression profiles. *Molecular Systems Biology*, 3:78, 2007.

[7] D. Bernardo, T. Gardner, and J. Collins. Robust identification of large genetic networks. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 486–497, 2004.

[8] F. C. Bernstein, T. F. Koetzle, G. J. Williams, E. F. M. Jr., M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The Protein Data Bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, 112(3):535 – 542, 1977.

[9] C. M. Bishop et al. *Pattern Recognition and Machine Learning*, volume 4. Springer New York, 2006.

[10] B. Bolstad, R. Irizarry, M. Åstrand, and T. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185–193, 2003.

[11] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[12] L. Breiman. Arcing Classifier. *The Annals of Statistics*, 26(3):801–849, 1998.

[13] L. Breiman. Prediction Games and Arcing Algorithms. *Neural Computation*, 11(7):1493–1517, 1999.

[14] L. Breiman. Randomizing Outputs to Increase Prediction Accuracy. *Machine Learning*, 40(3):229–242, 2000.

[15] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[16] L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

[17] A. Butte and I. Kohane. Mutual information relevance networks: functional genomic clustering using pairwise entropy measurements. In *Pacific Symposium on Biocomputing*, volume 5, pages 418–429, 2000.

[18] D. Camacho and J. Collins. Systems biology strikes gold. *Cell*, 137(1):24, 2009.

[19] I. Cantone, L. Marucci, F. Iorio, M. Ricci, V. Belcastro, M. Bansal, S. Santini, M. di Bernardo, D. di Bernardo, M. Cosma, et al. A Yeast Synthetic Network for In Vivo Assessment of Reverse-Engineering and Modeling Approaches. *Cell*, 137(1):172, 2009.

[20] T. Chen, H. He, G. Church, et al. Modeling gene expression with differential equations. In *Pacific Symposium on Biocomputing*, volume 4, page 4, 1999.

[21] K. Cios, W. Pedrycz, R. Swiniarski, and L. Kurgan. *Data Mining: A Knowledge Discovery Approach*. Springer Publishing Company, Incorporated, 2010.

[22] F. Crick et al. Central Dogma of Molecular Biology. *Nature*, 227(5258):561–563, 1970.

[23] B. Di Camillo, G. Toffolo, and C. Cobelli. A Gene Network Simulator to Assess Reverse Engineering Algorithms. *Annals of the New York Academy of Sciences*, 1158(1):125–142, 2009.

[24] C. Ding and H. Peng. Minimum redundancy feature selection from microarray gene expression data. In *Bioinformatics Conference, 2003. CSB 2003. Proceedings of the 2003 IEEE*, pages 523–528. IEEE, 2003.

[25] P. D'haeseleer, S. Liang, and R. Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. *Bioinformatics*, 16(8):707–726, 2000.

[26] P. D'haeseleer, X. Wen, S. Fuhrman, R. Somogyi, et al. Linear modeling of mRNA expression levels during CNS development and injury. In *Pacific Symposium on Biocomputing*, volume 4, pages 41–52. Singapore: World Scientific Press, 1999.

[27] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863, 1998.

[28] J. Faith, M. Driscoll, V. Fusaro, E. Cosgrove, B. Hayete, F. Juhn, S. Schneider, and T. Gardner. Many Microbe Microarrays Database: uniformly normalized Affymetrix compendia with structured experimental metadata. *Nucleic Acids Research*, 36(suppl 1):D866–D870, 2008.

[29] J. Faith, B. Hayete, J. Thaden, I. Mogno, J. Wierzbowski, G. Cottarel, S. Kasif, J. Collins, and T. Gardner. Large-Scale Mapping and Validation of Escherichia coli Transcriptional Regulation from a Compendium of Expression Profiles. *PLoS Biology*, 5(1):e8, 2007.

[30] Y. Freund. Boosting a weak learning algorithm by majority. In *Computational Learning Theory*, pages 202–216, 1990.

[31] Y. Freund, R. Schapire, and N. Abe. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(771-780):1612, 1999.

[32] Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[33] Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[34] J. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[35] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–407, 2000.

[36] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer Series in Statistics, 2001.

[37] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):pp. 1189–1232, 2001.

[38] N. Friedman. Inferring Cellular Networks Using Probabilistic Graphical Models. *Science*, 303(5659):799–805, 2004.

[39] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian Networks to Analyze Expression Data. *Journal of Computational Biology*, 7(3-4):601–620, 2000.

[40] S. Gama-Castro, H. Salgado, M. Peralta-Gil, A. Santos-Zavaleta, L. Muñiz-Rascado, H. Solano-Lira, V. Jimenez-Jacinto, V. Weiss, J. García-Sotelo, A. López-Fuentes, et al. RegulonDB version 7.0: transcriptional regulation of Escherichia coli K-12 integrated within genetic sensory response units (Gensor Units). *Nucleic Acids Research*, 39(suppl 1):D98–D105, 2011.

[41] T. Gardner and J. Faith. Reverse-engineering transcription control networks. *Physics of Life Reviews*, 2(1):65–88, 2005.

[42] T. S. Gardner, D. di Bernardo, D. Lorenz, and J. J. Collins. Inferring Genetic Networks and Identifying Compound Mode of Action via Expression Profiling. *Science*, 301(5629):102–105, 2003.

[43] A. Greenfield, C. Hafemeister, and R. Bonneau. Robust data-driven incorporation of prior knowledge into the inference of dynamic regulatory networks. *Bioinformatics*, 29(8):1060–1067, 2013.

[44] R. Gutierrez-Rios, D. Rosenblueth, J. Loza, A. Huerta, J. Glasner, F. Blattner, and J. Collado-Vides. Regulatory Network of Escherichia coli: Consistency Between Literature Knowledge and Microarray Profiles. *Genome Research*, 13(11):2435–2443, 2003.

[45] H. Hache, C. Wierling, H. Lehrach, and R. Herwig. GeNGe: systematic generation of gene regulatory networks. *Bioinformatics*, 25(9):1205–1207, 2009.

[46] A.-C. Haury, F. Mordelet, P. Vera-Licona, and J.-P. Vert. TIGRESS: trustful inference of gene regulation using stability selection. *BMC systems biology*, 6(1):145, 2012.

[47] B. Haynes and M. Brent. Benchmarking regulatory network reconstruction with GRENDEL. *Bioinformatics*, 25(6):801–807, 2009.

[48] T. K. Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832 –844, 1998.

[49] W. Iba and P. Langley. Induction of One-Level Decision Trees. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240, 1992.

[50] T. Ideker, V. Thorsson, and R. Karp. Discovery of regulatory interactions through perturbation: inference and experimental design. In *Pacific Symposium on Biocomputing*, volume 5, pages 302–313. World Scientific Maui, Hawaii, 2000.

[51] A. Irrthum, L. Wehenkel, P. Geurts, et al. Inferring Regulatory Networks from Expression Data Using Tree-Based Methods. *PLoS One*, 5(9):e12776, 2010.

[52] S. Kim, S. Imoto, and S. Miyano. Inferring gene networks from time series microarray data using dynamic Bayesian networks. *Briefings in Bioinformatics*, 4(3):228–235, 2003.

[53] A. Kremling, S. Fischer, K. Gadkar, F. Doyle, T. Sauter, E. Bullinger, F. Allgöwer, and E. Gilles. A Benchmark for Methods in Reverse Engineering and Model Discrimination: Problem Formulation and Solutions. *Genome Research*, 14(9):1773–1785, 2004.

[54] R. Küffner, T. Petri, P. Tavakkolkhah, L. Windhager, and R. Zimmer. Inferring gene regulatory networks by ANOVA. *Bioinformatics*, 28(10):1376–1382, 2012.

[55] K. Kyoda, M. Morohashi, S. Onami, H. Kitano, et al. A Gene Network Inference Method from Continuous-Value Gene Expression Data of Wild-Type and Mutants. *Genome Informatics*, pages 196–204, 2000.

[56] P. Lecca and C. Priami. Biological network inference for drug discovery. *Drug Discovery Today*, (0):–, 2012.

[57] W. Lee and W. Tzou. Computational methods for discovering gene networks from expression data. *Briefings in Bioinformatics*, 10(4):408–423, 2009.

[58] S. Liang, S. Fuhrman, R. Somogyi, et al. REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. In *Pacific Symposium on Biocomputing*, volume 3, page 2, 1998.

[59] N. Lim, Y. Şenbabaoğlu, G. Michailidis, and F. d'Alché Buc. OKVAR-Boost: a novel boosting algorithm to infer nonlinear dynamics and interactions in gene regulatory networks. *Bioinformatics*, 29(11):1416–1423, 2013.

[60] H. Lodish, A. Berk, S. Zipursky, P. Matsudaira, D. Baltimore, and J. Darnell. Molecular Cell Biology. *New York*, 2000.

[61] D. Marbach, J. Costello, R. Küffner, N. Vega, R. Prill, D. Camacho, K. Allison, M. Kellis, J. Collins, G. Stolovitzky, et al. Wisdom of crowds for robust gene network inference. *Nature Methods*, 2012.

[62] D. Marbach, R. Prill, T. Schaffter, C. Mattiussi, D. Floreano, and G. Stolovitzky. Revealing strengths and weaknesses of methods for gene network inference. *Proceedings of the National Academy of Sciences*, 107(14):6286–6291, 2010.

[63] D. Marbach, T. Schaffter, C. Mattiussi, and D. Floreano. Generating Realistic In Silico Gene Networks for Performance Assessment of Reverse Engineering Methods. *Journal of Computational Biology*, 16(2):229–239, 2009.

[64] A. Margolin, I. Nemenman, K. Basso, C. Wiggins, G. Stolovitzky, R. Favera, and A. Califano. ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context. *BMC Bioinformatics*, 7(Suppl 1):S7, 2006.

[65] A. Margolin, K. Wang, W. Lim, M. Kustagi, I. Nemenman, and A. Califano. Reverse engineering cellular networks. *Nature Protocols*, 1(2):662–671, 2006.

[66] F. Markowetz and R. Spang. Inferring cellular networks–a review. *BMC Bioinformatics*, 8(Suppl 6):S5, 2007.

[67] P. Mendes, W. Sha, and K. Ye. Artificial gene networks for objective comparison of analysis algorithms. *Bioinformatics*, 19(suppl 2):ii122–ii129, 2003.

[68] P. Meyer, K. Kontos, F. Lafitte, and G. Bontempi. Information-Theoretic Inference of Large Transcriptional Regulatory Networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007:8–8, 2007.

[69] P. Meyer, F. Lafitte, and G. Bontempi. minet: A R/Bioconductor Package for Inferring Large Transcriptional Networks Using Mutual Information. *BMC Bioinformatics*, 9(1):461, 2008.

[70] I. Nachman, A. Regev, and N. Friedman. Inferring quantitative models of regulatory networks from expression data. *Bioinformatics*, 20(suppl 1):i248–i256, 2004.

[71] R. Neapolitan. *Learning Bayesian Networks*. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[72] B.-E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. d'Alche Buc. Gene networks inference using dynamic Bayesian networks. *Bioinformatics*, 19(suppl 2):ii138–ii148, 2003.

[73] A. Pinna, N. Soranzo, and A. de la Fuente. From Knockouts to Networks: Establishing Direct Cause-Effect Relationships through Graph Analysis. *PloS One*, 5(10):e12912, 2010.

[74] R. Prill, D. Marbach, J. Saez-Rodriguez, P. Sorger, L. Alexopoulos, X. Xue, N. Clarke, G. Altan-Bonnet, and G. Stolovitzky. Towards a Rigorous Assessment of Systems Biology Models: The DREAM3 Challenges. *PloS One*, 5(2):e9202, 2010.

[75] J. Qi and T. Michoel. Context-specific transcriptional regulatory network inference from global gene expression maps using double two-way t-tests. *Bioinformatics*, 28(18):2325–2332, 2012.

[76] E. Ravasz, A. Somera, D. Mongru, Z. Oltvai, and A. Barabási. Hierarchical Organization of Modularity in Metabolic Networks. *Science*, 297(5586):1551–1555, 2002.

[77] S. Roy, M. Werner-Washburne, and T. Lane. A system for generating transcription regulatory networks with combinatorial control of transcription. *Bioinformatics*, 24(10):1318–1320, 2008.

[78] T. Schaffter, D. Marbach, and D. Floreano. GeneNetWeaver: In silico benchmark generation and performance profiling of network inference methods. *Bioinformatics*, 27(16):2263–2270, 2011.

[79] R. E. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5:197–227, 1990.

[80] R. E. Schapire and Y. Singer. Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning*, 37(3):297–336, 1999.

[81] E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature Genetics*, 34(2):166–176, 2003.

[82] E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Rich probabilistic models for gene expression. *Bioinformatics*, 17(suppl 1):S243–S252, 2001.

[83] E. Segal, H. Wang, and D. Koller. Discovering molecular pathways from protein interaction and gene expression data. *Bioinformatics*, 19(suppl 1):i264–i272, 2003.

[84] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of Escherichia coli. *Nature Genetics*, 31(1):64–68, 2002.

[85] J. Sławek and T. Arodź. ADANET: Inferring Gene Regulatory Networks using Ensemble Classifiers. In *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, pages 434–441. ACM, 2012.

[86] J. Sławek and T. Arodź. ENNET: inferring large gene regulatory networks from expression data using gradient boosting. *BMC systems biology*, 7(1):106, 2013.

[87] E. Someren, L. Wessels, E. Backer, and M. Reinders. Genetic network modeling. *Pharmacogenomics*, 3(4):507–525, 2002.

[88] D. Steinberg and P. Colla. CART: Classification and Regression Trees. *The Top Ten Algorithms in Data Mining, Chapman & Hall/CRC data mining and knowledge discovery series*, pages 179–201, 1997.

[89] G. Stolovitzky, A. Kundaje, G. Held, K. Duggar, C. Haudenschild, D. Zhou, T. Vasicek, K. Smith, A. Aderem, and J. Roach. Statistical analysis of MPSS measurements: Application to the study of LPS-activated macrophage gene expression. *Proceedings of the National Academy of Sciences of the United States of America*, 102(5):1402–1407, 2005.

[90] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Elsevier/Academic Press, 2006.

[91] E. Tuv, A. Borisov, G. Runger, and K. Torkkola. Feature Selection with Ensembles, Artificial Variables, and Redundancy Elimination. *The Journal of Machine Learning Research*, 10:1341–1366, 2009.

[92] T. Van den Bulcke, K. Van Leemput, B. Naudts, P. Van Remortel, H. Ma, A. Verschoren, B. De Moor, and K. Marchal. SynTReN: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. *BMC Bioinformatics*, 7(1):43, 2006.

[93] E. Van Someren, L. Wessels, M. Reinders, and E. Backer. Searching for Limited Connectivity in Genetic Network Models. In *Proceedings of the Second International Conference on Systems Biology*, pages 222–230, 2001.

[94] D. Veiga, G. Balázsi, et al. Network inference and network response identification: moving genome-scale data to the next level of biological discovery. *Molecular BioSystems*, 6(3):469–480, 2010.

[95] K. Yip, R. Alexander, K. Yan, and M. Gerstein. Improved Reconstruction of In Silico Gene Regulatory Networks by Integrating Knockout and Perturbation Data. *PloS One*, 5(1):e8121, 2010.

[96] J. Yu, V. Smith, P. Wang, A. Hartemink, and E. Jarvis. Advances to Bayesian network inference for generating causal networks from observational biological data. *Bioinformatics*, 20(18):3594–3603, 2004.

[97] C. Yuh, H. Bolouri, and E. Davidson. Genomic Cis-Regulatory Logic: Experimental and Computational Analysis of a Sea Urchin Gene. *Science*, 279(5358):1896–1902, 1998.

[98] X. Zhang, K. Liu, Z.-P. Liu, B. Duval, J.-M. Richer, X.-M. Zhao, J.-K. Hao, and L. Chen. NARROMI: a noise and redundancy reduction technique improves accuracy of gene regulatory network inference. *Bioinformatics*, 29(1):106–113, 2013.

[99] C. Zhu, K. Byers, R. McCord, Z. Shi, M. Berger, D. Newburger, K. Saulrieta, Z. Smith, M. Shah, M. Radhakrishnan, et al. High-resolution DNA-binding specificity analysis of yeast transcription factors. *Genome Research*, 19(4):556–566, 2009.

VITAE

Janusz Sławek was born on May 11th, 1986, in Wadowice, Poland, and is a Polish citizen. He graduated from Marcin Wadowita 1st High School in Wadowice, Poland in 2005. He received his Bachelor of Science and Master of Science in Computer Science from AGH University of Science and Technology, Cracow, Poland in 2010. He joined Doctor of Philosophy program at Virginia Commonwealth University in Richmond, Virginia in 2011. He has professional experience from AzimuthIT − Smart4Aviation Group, and Softeco Sismat S.p.A., where he worked in the field of software engineering and database management for three years between 2007 and 2010. His research is oriented towards computational analysis of genome-wide systems of transcriptional regulations.