UNIVERSITA' DEGLI STUDI DI MILANO BICOCCA
Facolta' di Scienze Matematiche, Fisiche e Naturali

COMPUTATIONAL INTELLIGENCE APPROACHES: FROM TIME SERIES TO DATA DRIVEN GENE REGULATORY NETWORK

PhD Dissertation of:
Antonella FARINACCIO

Supervisors
Prof. Giancarlo Mauri
Prof. Leonardo Vanneschi

Tutor
Prof.ssa Paola Bonizzoni

PhD Coordinator

Prof.ssa Stefania Bandini

XXIII Phd Cycle

To my parents

Abstract

For the past decade or so, Computational Intelligence has been an extremely hot topic among researchers working in the fields of biomedicine and bioinformatics. There are many successful applications of Computational Intelligence in such areas as computational genomics, prediction of gene expression, protein structure, and protein-protein interactions, modeling of evolution, or neuronal systems modeling and analysis. However, there are still many problems in biomedicine and bioinformatics that are in desperate need of advanced and efficient computational methodologies to deal with the tremendous amounts of data so prevalent in those kinds of research pursuits.

In an attempt to fill this gap, in the last decade many tools of Systems Biology have been developed to elaborate the large quantity of data generated by high-throughput experimental techniques with the increasingly sophisticated range of mathematical modelling techniques. The aim of systems biology is to integrate models at multiple biological scales and investigate system-level properties of biological organisms. This aim includes understanding at four levels:
(a) the structure of biological interaction networks;
(b) their dynamics, how states change over time in different conditions;
(c) the methods biological systems use to control the state of a cell;
(d) the design of systems, including both how they have evolved and how they may potentially be artificially constructed.

A key feature of systems biology is the integration of both theoretical modelling and empirical investigation, in which current biological knowledge informs the development of models and the analysis of these models produces a set of predictions that may then be tested in the laboratory.

Many models have been proposed to describe the network, one of the most extensively used is Boolean Network, that notwithstanding its numerous successes, in some cases could suffer from being too coarse.

Another widely studied candidate is the system of differential equations,

which is a very powerful and flexible model to describe complex relations among components. But it is not necessarily easy to determine the suitable form of equations which represent the network. Thus, the form of the differential equations had been fixed during the learning phase in previous studies. As a result, their goal was to simply optimize parameters, i.e., coefficients in the fixed equations.

In the analysis of time series of gene expression data presented in this thesis, a mathematical model has been identified and a system for the reconstruction of a Gene Regulatory Network Driven from Data has been implemented. Based on Genetic Programming, its target is to extractes knowledge and properties from data and so to generate the network that underlies the behaviour of genes. For this reason the system is called Data Driven Gene Regulatory Network Generator.

Planning to individualize the mutual interactions between genes, a Genetic Programming application for the extraction of the best activation function of the genes has also been developed. In order to test such a system, it has been applied to a serial temporal dataset of microarray gene expression data of breast cancer, while a study aimed at predicting the survival of a set of cancer patients has also been performed. This study has led to the definition of a Medical Decision Support System.

The activation functions of genes performed by this system have been successively used to reconstruct the gene regulatory network that underlies the development, response and regulation of the biological system. With the intent to test it, a reverse engineering of a synthetic gene regulatory network has been made and a dynamic symulation has been performed allowing for the related time series reconstruction. The gene regulatory network used for the reverse engineering has been the recently published IRMA network, a yeast synthetic network for the assessment of reverse engineering networks and modelling approaches.

Finally, in order to apply this system to a realistic gene regulatory network composed by thousands of genes, a new cluster kernel method has been identified and a framework driven by it has been developed. It is based on Gene Ontology to facilitate the detection of similar patterns of interacting genes, with the aim of reducing the dimension of the related serial temporal data.

Acknowledgments

First and foremost, I would like to thank my Supervisors, Prof. Giancarlo Mauri and Prof. Leonardo Vanneschi. Their expertise, excellent understanding and patient guidance had a major influence on this thesis, and this research work would not have been possible without their fundamental support.

I gratefully acknowledge Prof.ssa Bonizzoni, I enjoyed her interest in my research as well as the fruitful advises and support.

I owe special gratitude to Prof. Marco Antoniotti, Prof. Italo Zoppis, Prof. Mario Giacobini, Prof. Paolo Provero and Dott. Daniele Merico, for their crucial contributions that have been of great value in this study.

I want to thank all members of the Bimib group, for providing an excellent and inspiring working atmosphere.

I was delighted to interact with Prof.ssa Raffaella Rizzi. Most important, she became a friend.

My sincerely thanks go to my fellow labmates, Fabrizio and Simone. I enjoyed the atmosphere, their friendship and their support.

My deepest gratitude goes to my parents for their unflagging love and support throughout my life; this dissertation would have been simply impossible without them.

Contents

Chapter 1

Introduction

The past few decades have seen a huge growth in biological information gathered by the related scientific communities. A fload of such information coming in the form of genomes, protein sequences, gene expression data and so on have led to the absolute need for effective and efficient computational tools to store, analyze and interpret such multiformed data. Techniques including applied mathematics, informatics, statistics, computer science, artificial intelligence, chemistry, and biochemistry have been involved to solve biological problems usually at the molecular level, flooding in Bioinformatic and Computational Biology.
Research in computational biology often overlaps with systems biology. Major research efforts in the field include sequence alignment, gene finding, genome assembly, protein structure alignment, protein structure prediction, prediction of gene expression and protein-protein interactions, and the modeling of evolution. Hence, in other words, bioinformatics can be described as "the application of computational methods to make biological discoveries" [Baldi and Brunak, 1998]. The ultimate attempt of the field is to develop new insights into the science of life as well as creating a global perspective, from which the unifying principles of biology can be derived [Altman et al., 2001].

In the year 2006, there were at least 26 billion base pairs (bp) representing the various genomes available on the server of the National Center for Biotechnology Information (NCBI) [Ezziane, 2006]. Besides the human genome with about 3 billion bp, many other species have their complete genome available there.

Both bioinformatics and computational biology are concerned with the use of computation to understand biological phenomena and to acquire and exploit biological data, increasingly large-scale data [Gusfield, 2004]. Methods from bioinformatics and computational biology are increasingly used to augment or leverage traditional laboratory and observation-based biology.

These methods have become critical in Biology due to recent changes in our ability and determination to acquire massive biological data sets, and due to the ubiquitous, successful biological insights that have come from the exploitation of those data. This transformation from a data-poor to a data-rich field began with DNA sequence data, but is now occurring in many other areas of biology [Ezziane, 2006]. So bioinformatic and computational biology have emerged as a strategic frontier between biology and computer science.

Computational Intelligence(CI) is placed in this context. It is a well-established paradigm, where new theories with a sound biological understanding have been evolving. The current experimental systems have many of the characteristics of biological computers (brains) and are beginning to be built to perform a variety of tasks that are difficult or impossible to do with conventional computation paradigms. Computational intelligence methods are now being applied to problems in molecular biology and bioinformatics [Mitra and Hayashi, 2006]. Reader may refer to [Kelemen et al., 2008, Cios et al., 2005] for an extensive review of various computational intelligence techniques applied to different bioinformatics problems.

Defining computational intelligence is not an easy task. In a nutshell, which becomes quite apparent in light of the current research line, the area is heterogeneous, with a combination of technologies such as neural networks, fuzzy systems, evolutionary computation, swarm intelligence, and probabilistic reasoning [Hassanien et al., 2008].
The recent trend is to integrate different components to take advantage of complementary features and to develop a synergistic system [Kelemen et al., 2008]. Hybrid architectures like neuro-fuzzy systems, evolutionary-fuzzy systems, evolutionary neural networks, evolutionary neuro-fuzzy systems, rough-neural, rough-fuzzy, etc. are widely applied for real world problem solving [Abraham, 2002, Abraham, 2005, Hunga et al., 2006].

The work presented in this thesis is based on this line of research. An integrated stochastic approach is used in order to reconstruct a gene regulatory network. The building process is guided by the phenomena under investigation, attempting to extract the knowledge that underlies the gene expression data and using it to reconstruct the network that regulate the genes mutually interactions. The process is driven from time series gene ex-

pression microarray data and is performed using regression through Genetic Programming(GP).

1.1 Preliminary Concepts

1.1.1 Computational Intelligence in Gene Expression

Gene expression refers to a process through which the coded information of a gene is converted into structures operating in the cell. It provides the physical evidence that a gene has been turned on or activated. Expressed genes include those that are transcribed into mRNA and then translated into proteins and those that are transcribed into RNA but not translated into proteins (e.g., transfer and ribosomal RNAs) [Luscombe et al., 2001, Mitra et al., 2007]. The expression levels of thousands of genes can be measured at the same time using the modern microarray technology [Quackenbush, 2001].

DNA microarrays usually consist of thin glass or nylon substrates containing specific DNA gene samples spotted in an array by a robotic printing device. Researchers spread fluorescently labeled mRNA from an experimental condition onto the DNA gene samples in the array. This mRNA binds (hybridizes) strongly with some DNA gene samples and weakly with others, depending on the inherent double helical characteristics. A laser scans the array and sensors to detect the fluorescence levels (using red and green dyes), indicating the strength with which the sample expresses each gene. The logarithmic ratio between the two intensities of each dye is used as the gene expression data [Hassanien et al., 2008].

In order to understand better these concepts, the Chapter 2 is devoted to a brief description of cell biology. In it some elements of molecular biology have been exposed, such as organisms and cells, genes and genomes, gene regulatory network, gene expression and microarray.

1.1.2 Machine Learning and Statistical Data Analysis

Machine learning techniques are increasingly being used to address problems in computational biology, bioinformatics and computational intelligence.

Machine learning consists in programming computers to optimize a performance criterion by using example data or past experience. The optimized criterion can be the accuracy provided by a predictive model in a modelling

problem, and/or the value of a fitness or evaluation function in an optimization problem. In a modelling problem, the term ”learning” refers to running a computer program to induce a model by using training data or past experience.

The exponential growth of the amount of biological data available (see Fig. 1.1) raises at least two problems: need of capable information storage and expert management and, secondly, the intelligent extraction of useful information from these data. The second problem is one of the main challenges in computational biology. It requires the development of tools and methods capable of transforming all these heterogeneous data into biological knowledge about the underlying mechanism. These tools and methods should allow one to go beyond a mere description of the data and provide knowledge in the form of testable models. Predictions of the system should be obtained by this simplifying abstraction that constitutes a model.

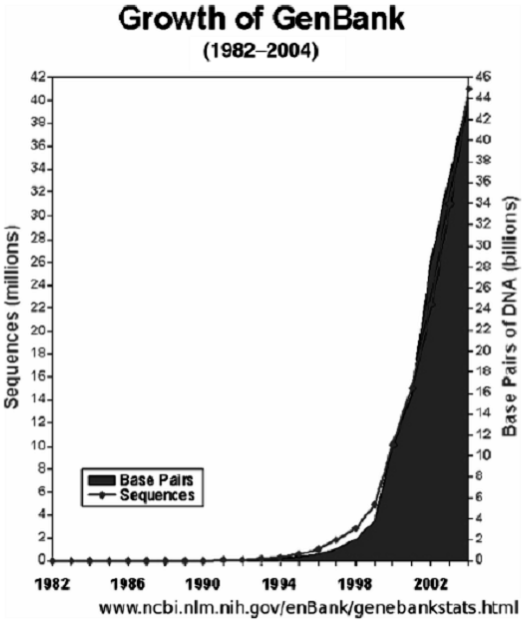


Figure 1.1: Evolution of the GenBank database size

Machine learning approaches (e.g., genetic programming, neural networks, hidden Markov models and belief networks) are ideally suited for areas where there is a lot of data but little theory and this is exactly the

situation in molecular biology. As with its predecessor, statistical model fitting, the goal in machine learning is to extract useful information from a body of data by building good probabilistic models. The particular twist behind machine learning, however, is to automate the process as much as possible [Zhang and Rajapakse, 2008].

There are several biological domains where machine learning techniques are applied for knowledge extraction from data, as showed in Fig. 1.2. The main are genomics, proteomics, microarrays, systems biology, evolution and text mining.

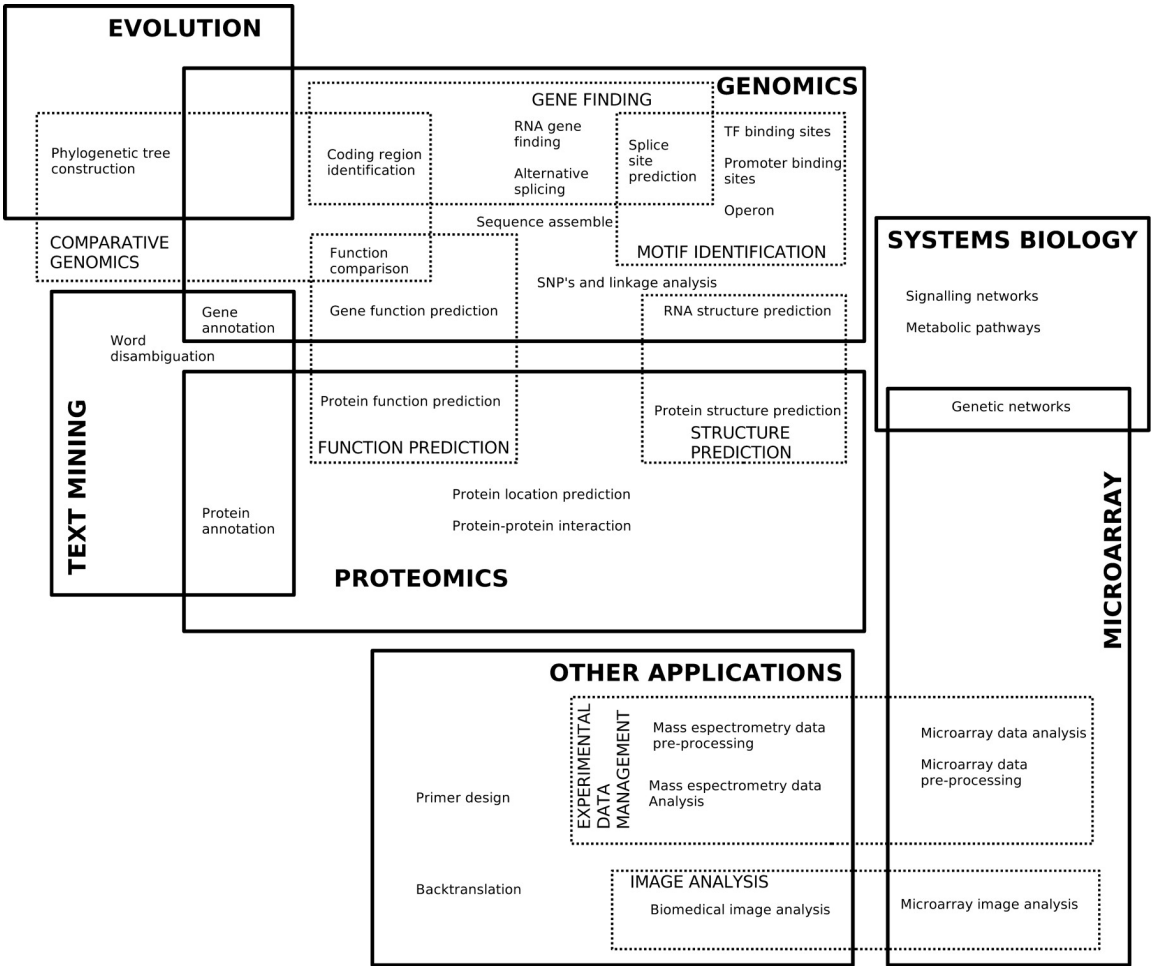


Figure 1.2: Classification of the biology-related topics where machine learning methods are applied - Picture taken from [Larranaga et al., 2006]

Microarray essays are the best known (but not the only) domain where

this kind of data is collected.

Complex experimental data raise two different problems. First, data need to be preprocessed, i.e. modified to be suitably used by machine learning algorithms. Second, the analysis of the data, which depends on what we are looking for. In the case of microarray data, the most typical applications are expression pattern identification, classification and genetic network induction.

Systems biology and computational intelligence is another domain where biology and machine learning work together. It is very complex to model the life processes that take place inside the cell. Thus, computational techniques are extremely helpful when modelling biological networks, especially signal transduction networks, metabolic pathways and genetic networks.

Moreover, machine learning often uses statistical methods when building computational models, since the objective is to make inferences from a sample, and/or when making use of stochastic methods. Furthermore, it uses statistical theory to evaluate the outcomes. The statistical methods used in this thesis for the evaluation of the results obtained are described in Chapter 3.

The two main steps in the machine learning process are to induce the model by processing the huge amount of data as well as to represent the model and make inferences efficiently. It must be noticed that the efficiency of the learning and inference algorithms, as well as their space and time complexity and their transparency and interpretability, can be as important as their predictive accuracy. Note that the process of transforming data into knowledge may be both iterative and interactive. The iterative phase consists of several steps. In the first step, we need to integrate and merge the different sources of information into a single format. By using data warehouse techniques, the detection and resolution of outliers and inconsistencies are solved. In the second step, it is necessary to select, clean and transform the data. To carry out this step, we need to eliminate or correct the incorrect data and instances.

Approximate algorithms always output a candidate solution, but it is not guaranteed to be the optimal one. Optimization is also a fundamental task when modelling from data. In fact, the process of learning from data can be regarded as searching for the model that best fits the data [Larranaga et al., 2006]. Therefore, an optimization method based

on Genetic Programming(GP) is used in this thesis as a fundamental ingredient of modelling. In Chapter 4 GP is described in depth.

The analysis of gene expression data achieved in this thesis has been performed using GP and subsequently compared with other techniques of Machine Learning, such as Support Vector Machines, Multilayered Perceptron, Random Forest, Radial Basis Function Network and Voted Perceptron. So, a brief description of these methods is given in Chapter 3, where the basic concepts of Machine Learning and Data Mining are introduced, and, as stated above, the statistical methods used to evaluate and validate the results obtained by the developed systems are also described.

1.2 Motivations and Goals

1.2.1 Medical Decision Support System For Survival Prediction in Breast Cancer

The advent of high-throughput techniques to measure gene expression led in the last decade to a large body of research on gene expression as just stated, and especially in cancer, in particular in view of the possibility of using gene expression data to improve cancer patient classification.

The ability to accurately classify cancer patients into risk classes, i.e. to predict the outcome of the pathology on an individual basis, is a key ingredient in making therapeutic decisions, especially for cancer therapies, due to their serious side effects. Therefore the classification of cancer patients into risk classes is a very active field of research, with direct clinical applications.

A *gene signature* is a set of genes whose levels of expression can be used to predict a biological state (see [Nevins and Potti, 2007]): in the case of cancer, gene signatures have been developed both to distinguish cancerous from non-cancerous conditions and to classify cancer patients based on the aggressiveness of the tumor, as measured for example by the probability of relapsing within a given time.

While many studies have been devoted to the identification of gene signatures in various types of cancer, the question about which the algorithms should be used to maximize the predictive power of a gene signature has received less attention.

In this thesis, the intent is to investigate this issue systematically, considering one of the best established gene signatures, the 70-gene signature

for breast cancer [van't Veer et al., 2002]. In that work, a set of microarray features were selected based on the correlation with survival, on which the molecular prognostic test for breast cancer "MammaPrint"^{TM}is based.

We aim at developing a Medical Decision Support System (GP-MDSS) in order to perform a survival prediction of breast cancer giving also the possibility of generating biological insight and hypotheses on the influence of the various gene expression levels on the pathology.

1.2.2 Generator of Gene Regulatory Network and Dynamic Simulator System

The principal purpose of this thesis is to investigate the functions of genes and their mutual interactions. These functions can be better understood if they are not only studied as isolated entities, but if their reciprocal relationships are also investigated.

An ensemble approach which seems very promising is the study of *Gene Regulatory Networks*. It is based on the hypothesis that genes that have similar expression profiles should also have similar regulation mechanisms as there must be a reason why their expression is similar under a variety of conditions. Therefore, if we cluster the genes by similarities in their expression profiles and take sets of promoter sequences from genes in such clusters, some of these sets of sequences may contain a "signal" as a specific sequence pattern such as a particular substring, which is relevant to the regulation of these genes.

However, Gene Regulatory Networks, far from being as straightforward as they may seem, are extremely complex systems, comprising genes, proteins and other interacting molecules. So a field of systems biology is emerging [Hasty and McMillen, 2002, Hayete et al., 2007, Sprinzak and Elowitz, 2005] that is aimed at a formal understanding of the biological processes caused by the numerous regulatory, signaling and metabolic interactions between the different components and their coordinated action. This is usually done by developing quantitative mathematical models, able to describe changes in concentration of each gene transcript and protein in a network as a function of their regulatory interactions (Gene Regulatory Network).

The formalisms to model biological networks defined so far are numerous (see for instance [Di Ventura et al., 2006, Szallasi et al., 2006]). A widely

used one is the Random Boolean Network (RBN) that, notwithstanding its numerous successes, in some cases could suffer from being too coarse.

Another commonly used model is based on a system of differential equations. It is a very powerful and flexible model to describe complex relations among components, but it is not necessarily easy to determine the suitable form of equations which represent the network. Thus, the form of the differential equations was fixed during the learning phase in previous studies. As a result, their goal was to simply optimize parameters, i.e., coefficients in the fixed equations.

The aim of this thesis is to conceive a new model of Gene Regulatory Network driven from data, to overcome some of the limitations of RBN's. So the goal is to develop a system that, extracting both the topology and the activation functions directly from the time series gene expression microarray data by means of GP, is able to generate the gene regulatory network that underlies the data, to simulate the dynamics of the network and to reconstruct the underlying time series.

1.2.3 An Application of Kernel Methods to Gene Cluster Temporal Meta-Analysis

As just stated, microarrays are one of the most successful and widespread technologies in the field of gene expression studies, enabling the parallel measurement of thousands of transcripts for a given cellular extract [Eisen et al., 1998, Lockhart and Winzeler, 2000], as described in more details in Chapter 2. But, after the generation of raw signals, the data need to undergo a sophisticated process of statistical analysis [Speed, 2003].

So, the intent of this work is to address a more sophisticated design, such as the profiling of different cellular systems (e.g. their response to different stimuli), or the same system over a time-course (e.g. the cell cycle).
The principal purpose is to perform a dimensional reduction of data in order to apply it to a study of a realistic genetic network using the Data Driven Gene Regulatory Network developed in this work and described in Chapter 7.

The state-of-the-art approach is to perform *clustering*, in order to group together genes with similar expression profiles across experiments, and then summarize the clusters by identifying the most prominent functional groups.

If the same experiment is sampled using different technological platforms, or if different experiments generate highly correlated data, this is informative

to track similarities among different clusters. The most elementary solution to evaluate cluster similarity is to count the number of overlapping genes; however, this strategy is suboptimal when the clusters are generated from partially or non-overlapping gene sets, as can happen when different technological platforms are used. A more general and high-level approach is to compute similarity according to the functional profiles of the clusters; this approach is usually called *cluster meta-analysis*.

A specific case of cluster meta-analysis is encountered when analyzing time-course gene expression experiments. In this case the temporal dimension can be exploited to gather information about the dynamics of the biological system under observation. Most analysis setups for time-course experiments analyze the time-span of the measurements globally, [Bar-Joseph, 2004, Ernst and Bar-Joseph, 2006], and may generate clusters spanning the full length of the time-course; consequently, genes are associated into a cluster only if their expression is coordinated for the entire duration of the experiment.

In this thesis a different approach is followed where gene expression experiments are analyzed by splitting the time course into shorter *time-windows*.

The key observation is that temporally localized (i.e., within a limited sequence of time-steps) relationships among genes are worth detecting.

If the time-course is split into shorter time-windows, and clusters are generated separately within each window, it is possible to concentrate on temporally localized gene relationships. In this method a cluster meta-analysis method is required to detect further relationships among clusters belonging to consecutive time-windows. The result of this meta-analysis will be the *reconnection* of the clusters from different windows, in order to better visualize the temporal evolution of the system.

Clustering meta-analysis is a valuable resource for the understanding of complex microarray data-sets. In particular, segmenting time-course experiments enables to identify local patterns; clustering meta-analysis enables to track the differences and similarities among such patterns. A typical question answered by the proposed framework is the following: are two functional groups of genes constantly co-expressed along a time-course, or in related yet different experiments?

PART I

BIOLOGICAL AND COMPUTATIONAL BACKGROUND

Chapter 2

Elements of Molecular Biology

2.1 Organisms and cells

In this chapter well known concepts are described that are useful for the comprehension of this dissertation. Contents of molecular cell biology are based on [Albert et al., 2010, Lodish et al., 2008, Watson et al., 2004, Nelson and Cox, 2000, Muller-Esterl et al., 2004], knowledge of microarray analysis is acquired from [Amaratunga and Cabrera, 2004, Knudsen, 2002, Gentleman et al., 2005, Emmert-Streib and Dehmer, 2008] and many pictures are taken from [Farabee,].

With the invention of the microscope, it became clear that all living organisms are composed of small *cells*, that cells can also exist as independent organisms, and that individual cells are living in the sense that they can grow, reproduce, convert energy from one form into another, control their internal working, respond to their environment, and so on [Lodish et al., 2008, Watson et al., 2004, Muller-Esterl et al., 2004, Nelson and Cox, 2000]. Each cell is a complex system of many different membrane-enclosed units filled with a concentrated acqueous solution of chemicals and provided with the astonishing ability to create copies of themselves by growing and dividing in two. The simplest forms of life are solitary cells, they are unicellular organisms, consisting of only one cell, like bacteria, amoeba and baker's yeast. Each of these kinds of cell is able to survive and multiply independently in an appropriate environment. Higher organisms, including humans, are communities of cells derived by growth and division from a single founder cell: each animal, plant, or fungus is a vast colony of individual cells that perform specialized functions coordinated by complex systems of communication.

Cell biologists often speak of "the cell" without specifying any particular cell. But cells are not all alike; in fact they can be widely different. It is estimated that there are at least 10 million of distinct species of living

things in the world and about 320 different types in the $6x10^{13}$ cells of human body. For instances there are several types of skin cells, muscle cells, brain cells (neurons), among many others. The number of cell types is not well-defined and depends on the similarity threshold (the level of detail we would like to use to distinguish the cell types; for instance, it is unlikely that we could find two identical cells in an organism if we counted the number of their molecules). But what does a bacterium have in common with the cells of a butterfly, what do the cells of a rose have in common with those of a dolphin? And in what ways do they differ?
Cells vary widely in size, shape and functions, and in chemical requirements and activities. For example, some cells need oxygen to live, while for others it is deadly. Some consume little more than air, sunlight and water, as their raw materials; other need a complex mixture of molecules produced by other cells. Some appear to be specialized factories for the production of particular substances, such as hormones, starch, fat, latex, or pigments. Some are engines, like muscles, burning fuel to do mechanical work, other are electricity generators.
Although they are infinitely varied when viewed from the outside, all living things are fundamentally similar inside. Nowadays we know that cells resemble one another to a surprising degree in the details of their chemistry, sharing the same machinery for the most basic functions. All cells are composed of the same sorts of molecules that partecipate in the same types of chemical reactions.
Traditionally, cell biologists divided the organisms into two big groups and two types of cells respectively: organisms whose cells have a nucleus are called *eukaryotes* (from the Greek word *eu*, meaning "well" or "truly", and *karion*, a "kernel" or "nucleus"), instead organisms whose cells do not have a nucleus are called *prokaryotes* (from *pro*, meaning "before"). However, most organisms which we can see, such as trees, grass, flowers, weeds, worms, flies, mice, cats, dogs, humans, mushrooms and yeast are eukaryotes, whereas bacteria and archea belong to the procaryotes. Nowaday molecular studies reveal that there is an enormous difference within the class of procaryotes, dividing it into two distinct domains called the *eubacteria* and the *archea* [Albert et al., 2010, Lodish et al., 2008]. For our purpose, it is enough to distinguish between eukaryotes and prokaryotes.

Prokaryotic cells are smaller than eukaryotic cells (Fig. 2.1): a typical

size of a prokaryotic cell is about 1 micron in diameter and has a simpler structure (e.g., it does not have any inner cellular membranes that are always present in Eukaryotes, see below). Prokaryotes are single cellular organisms, but the opposite is not true: if an organism is a single cell it is not necessarily a prokaryote. They are sometimes also known as microbes.

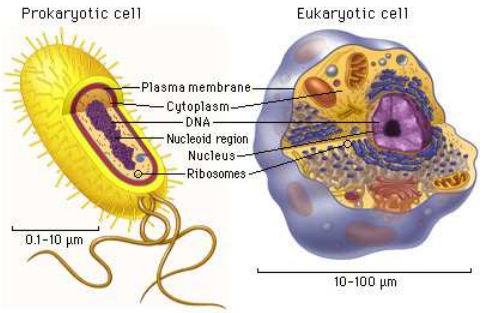


Figure 2.1: A figurative comparison of eukaryotic and procariotic cell - Picture taken from On-Line Biology Book [Farabee,]

Eukaryotic cells, in general, are bigger and more elaborate than bacteria and archea. Some live independent lives as single-celled organisms, such as amoeba and yeast; others live in multicellular assembly. Such a cell has a nucleus, which is separated from the rest of it by a membrane. The nucleus contains chromosomes, which are the carrier of the genetic material. There are internal membrane enclosed compartments within eukaryotic cells, called organelles, e.g., centrioles, lysosomes, golgi complexes, mitochondria, among others (Fig. 2.2), which are specialised for particular biological processes. The area of the cell outside the nucleus and the organelles is called the cytoplasm. Membranes are complex structures which provide an effective barrier to the environment, and regulate the flow of food, energy and information in and out of the cell.

An essential feature of most living cells, prokaryote and eukaryote, is their ability to create copies of themselves by growing and dividing into two. The growth of a single cell and its subsequent division is called the *cell cycle*. However, not all cells continually grow and divide, for example neurons only undergo an initial growth phase. Prokaryotes, particularly bacteria, are extremely successful at multiplying. Multicellular organisms typically begin life as a single cell, usually as a result of fusion of a male and a female sex cell (gametes). The single cell has to grow, divide and differentiate

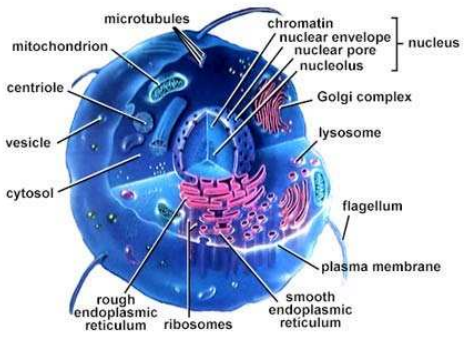


Figure 2.2: A model of eukaryotic cell - Picture taken from On-Line Biology Book [Farabee,]

into different cell types to produce tissues and, in higher eukarotyes, organs. Cell division and differentiation need to be controlled. Cancerous cells grow without control and can go on to form tumours. Development of single cells into complex organisms is in itself an area of study called *developmental biology*. It has a relevant effect on the modern biology, in fact the Nobel Prize for Physiology or Medicine in 2001 has been awarded to scientists for the discoveries of key regulators of the cell cycle. In 2002, it has been awarded to scientists for their discoveries concerning 'genetic regulation of organ development and programmed cell death', and in 2006 to scientists for their discovery of RNA interference - gene silencing by double-stranded RNA.

2.2 Molecules of life

Cells consist of molecules. There are four basic kinds of molecules implied in life: (1) *small molecules*, (2) *proteins*, (3) *DNA* (4) *RNA*.
Proteins, DNA and RNA are known collectively as *macromolecules*.

2.2.1 Small molecules

Small molecules can be the building blocks of the macromolecules or can have independent roles, such as signal trasmission or being a source of energy or material for cells. Some relevant examples besides *water* are *sugar, fatty acids, amino acids* and *nucleotides*. These small organic molecules account for 75-80 % of living material by weight, of which water is by far the most

abundant. The rest of living material consists of macromolecules, including proteins, DNA and RNA (Fig. 2.3).

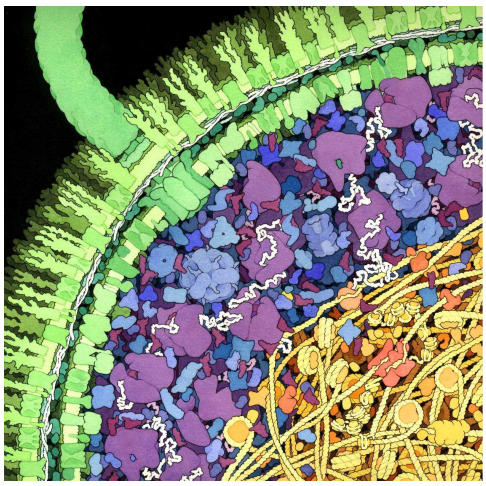


Figure 2.3: This illustration shows a cross-section of a small portion of an Escherichia coli cell. The cell wall, with two concentric membranes studded with transmembrane proteins, is shown in green. A large flagellar motor crosses the entire wall, turning the flagellum that extends upwards from the surface. The cytoplasmic area is colored in blue and purple. The large purple molecules are ribosomes and the small, L-shaped maroon molecules are tRNA, while the white strands are mRNA. Enzymes are shown in blue. The nucleoid region is shown in yellow and orange, with the long DNA circle shown in yellow, wrapped around the HU protein (bacterial nucleosomes). [Illustration by D.Goodsell [Goodsell, 2009]]

Cells acquire and use these two size classes of molecules (small molecules and macromolecules) in essentially different ways. Ions, water, and many small organic molecules are imported into the cell. Cells also make and alter many small organic molecules by a series of different chemical reactions.
In contrast, cells can obtain macromolecules only by making them. Their synthesis entails linking together a specific set of small molecules (monomers) to form polymers through the repetition of a single type of chemical-linkage reaction.

For example the protein macromolecules are built of amino-acids molecules. There are 20 different amino acid molecules. They differ by the R side chains which determine their properties while the order of these different amino acids within the protein determines the three-dimensional structure of the protein, some of which are shown in Fig. 2.4. There is a

convention that each amino acid is denoted by a letter in Latin alphabet, for instances Arginine is denoted by R, Histidine by H, Lysine by L, and so there are 20 such letters.

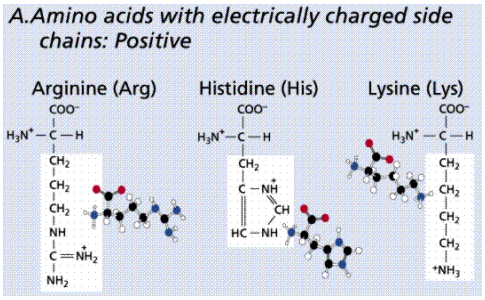


Figure 2.4: The image shows the basic composition of three amino acids: Arginine, Histidine and Lysine

Some small molecules function as precursors for the synthesis of macromolecules, and the cell provides the appropriate mix of small molecules needed. Small molecules also store and distribute the energy for all cellular processes; they are broken down to extract this chemical energy, as when sugar is degraded to carbon dioxide and water with the release of the energy bound up in the molecule. Other small molecules (e.g., hormones and growth factors) act as signals that direct the activities of cells, and nerve cells communicate with one another by releasing and sensing certain small signaling molecules.

2.2.2 Proteins

Proteins have complex three-dimensional (3D) structure (Fig. 2.5). They are the workhorses of the cell and are the most abundant and functionally versatile of the cellular macromolecules. Among others, there are:

- *Transmembrane proteins* they play a key role in maintenance of the cellular environment, as communication between cells, communication between organells and cytosol, nutrient transport, receptor for viruses, regulating cell volume, extraction and concentration of small molecules from the extracellular environment, and generation of ionic gradients essential for muscle and nerve cell function.
- *Structural proteins*, which are the main basic building blocks of the organism. An example is elastin that is a critical components of con-

nective tissue. It allows many tissues in the body to resume their shape after stretching or contracting and helps skin to return to its original position when it is poked or pinched.

- *Enzymes*, which increase the rate of chemical reactions (catalyze) such as altering, joining together or chopping up other molecules. Like all catalysts, enzymes work by lowering the activation energy for a reaction, thus dramatically increasing the rate of the reaction. Most enzyme reaction rates are millions of times faster than those of comparable un-catalyzed reactions. Together these reactions and the pathways they make up is called *metabolism*. Usually enzymes are very specific and catalyze only a single type of reaction, however the same enzyme can play role in more than one pathway.

Figure 2.5: 3D structure of Human Hemoglobin protein

2.2.3 DNA

Deoxyribonucleic acid (DNA) is an informational molecule that contains, in the sequence of its nucleotides, the information required to build all proteins of an organism, and hence the cells and tissues of that organism . It is ideally suited to perform this function on a molecular level (Fig. 2.7). Its chemical structure is finely stable under most terrestrial conditions, as exemplified by

the ability to recover DNA sequence from fossils that are tens of thousands of year old. For this characteristic of long-term storage of information it is often compared to a set of blueprints or a recipe, or a code, since it contains the instructions needed to construct other components of cells, such as proteins and RNA molecules. The DNA segments that carry this genetic information are called *genes*, but other DNA sequences have structural purposes or are involved in regulating the use of this genetic information. Genes are the core ingredient of this thesis, they are the start and the end of the study presented here. In fact, the long term target of the scientific line in which this thesis is inserted is the investigation of genes functionalities in a biological organism, through the exploration of their mutual interaction. In order to approach this aim, the relations between genes and diseases are investigated through a knowledge extraction from time series of gene expression data. To this end a Medical Decision Support System for the survival prediction in breast cancer has been developed, and is described in depth in Chapter 5 and Chapter 6. Morever, with the purpose to investigate on the genes and their mutual interaction, a mathematical model has been identified and a system for the reconstruction of the control network that involves interaction between genes has been developed; it is named Gene Regulatory Network Generator and Simulator is described in Chapter 7. Furthermore a method useful for its application is showed in Chapter 8.
So, for the comprehension of the following systems, this entire chapter is dedicated to the description of the basic concepts about genes and related notions.

From the chemical point of view, a molecule of DNA consists of two long *polynucleotide chains*. Each of these *DNA chains*, or *DNA strands*, is composed of four types of *nucleotide subunits*, and the two chains are held together by hydrogen bonds between the base portions of nucleotides. The members of each pair can fit together within the double helix because the two strands of the helix run antiparallel to each other, that is, they are oriented with opposite polarities (Fig. 2.6).

Attached to each sugar is one of four types of molecules called *bases*, (which are, in fact, the only distinguishing elements between different nucleotides, (Fig. 2.8) and are denoted by their initial letters, A,C ,G and T.

The information is encoded according to the sequence of these four

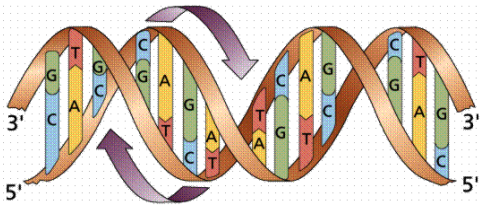


Figure 2.6: Rendering of two complementary bases on a DNA molecule (picture taken from On-Line Biology Book)

bases along the backbone. Different nucleotides can be linked together in any order to form a polynucleotide, for instance, like this:
A-G-T-C-C-A-A-G-C-T-T
Polynucleotides can be of any length and can have any sequence. The two ends of this molecule are chemically different, i.e., the sequence has a directionality, like this:
A->G->T->C->C->A->A->G->C->T->T
The end of the polynucleotide are marked either 5' and 3' (this has chemical reasons in the numbering of the -OH groups of the sugar ring); by convention DNA is usually written with 5' left and 3' right, with the coding strand at top. Two such strands are termed complementary, if one can be obtained from the other by mutually exchanging A with T and C with G, and inverting the direction of the molecule. For instance:
<-T<-C<-A<-G<-G<-T<-T<-C<-G<-A<-A
is complementary to the polynucleotide given above.
Specific pairs of nucleotides can form weak bonds between them. A binds to T, C binds to G (to be more precise, two hydrogen bonds can be formed between each A-T pair, and three hydrogen bonds between each C-G pair). Although such interactions are individually weak, when two longer complementary polynucleotide chains meet, they tend to stick together, as shown in Fig. 2.9.

Vertical lines between two strands represent the forces between them (to be more accurate we could draw triple lines between each C and G and double lines between A and T) as shown in Fig. 2.10. The A-T and G-C pairs are called base-pairs (bp). The length of a DNA molecule is usually measured in base-pairs or nucleotides (nt), which in this context is the same thing.

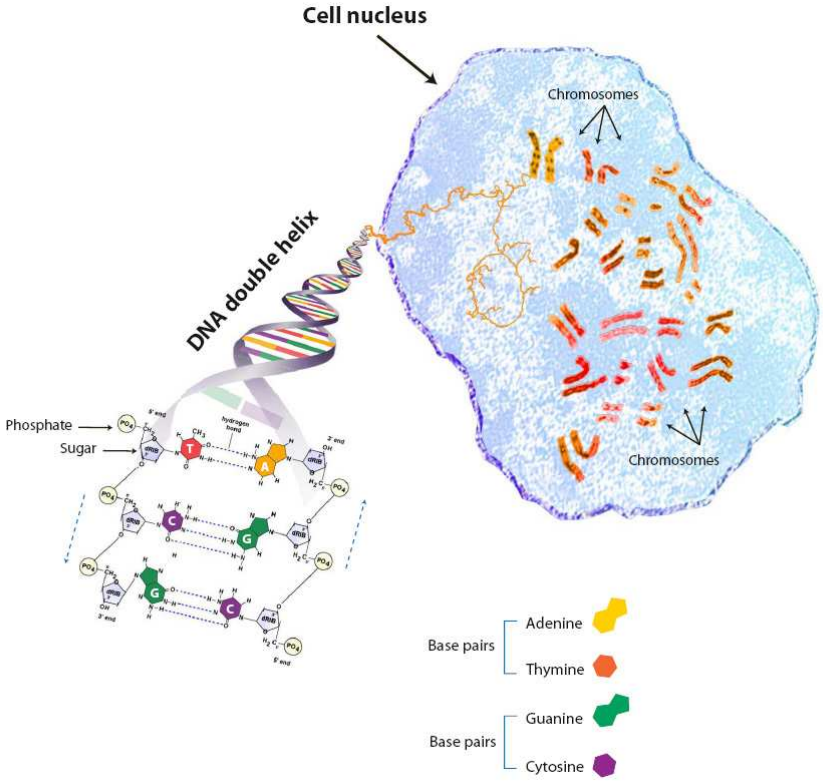


Figure 2.7: DNA molecules inside the cell's nucleus tightly packed into chromosome

The antiparallel sugar-phosphate strands twist around each other to form a double helix containing 10 base pairs per helical turn, which is about 3.4 nm long (Fig. 2.11, Fig. 2.12).

It is noticeable that two complementary DNA polypeptides form a stable double helix almost regardless of the sequence of the nucleotides. This makes the DNA molecule a perfect medium for information storage. A consequence of the double helix base-pairing requirements is that each strand of a DNA molecule contains a sequence of nucleotides that is exactly complementary to the nucleotide sequence of its partner strand - an A always matches a T on the opposite strand, and a C always matches a G - therefore for the information extraction it is enough to give only one strand of the genome molecules. Thus, for many information related purposes, the molecule used on the example above can be represented as just the sequence CGATTCAACGATGC. The maximal amount of information that can be

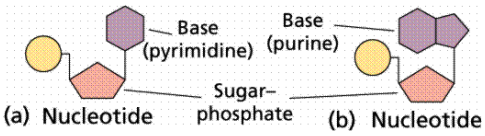


Figure 2.8: Two nucleotides that differes from bases (picture taken from On-Line Biology Book)

```
5'  C-G-A-T-T-G-C-A-A-C-G-A-T-G-C  3'
    | | | | | | | | | | | | | | |
3'  G-C-T-A-A-C-G-T-T-G-C-T-A-C-G  5'
```

Figure 2.9: Two complementary polynucleotides sticked together

encoded in such a molecule is therefore 2 bits times the length of the sequence. Noting that the distance between nucleotide pairs in DNA is about 0.34 nm, we can calculate that the linear information storage density in DNA is about $6x10^8$ bits/cm, which is approximately 75 GB or 125 CD-ROMs per cm.

The complementarity of two strands in the DNA is exploited for copying (multiplying) DNA molecules in a process known as the *DNA replication*, in which one double stranded DNA is replicated into two identical ones. The DNA double helix unwinds and forks during the process, and a new complementary strand is synthesized by a specific molecular procedure on each branch of the fork. After the process is finished there are two DNA molecules identical to the original one. In a cell this happens during cell division and a copy identical to the original goes to each of the new cells.

It is remarkable that mismatched components between polynucleotide strands are possible, if the total sum of weak forces between the complementary nucleotides are strong enough. So the molecules shown in Fig. 2.13 are chemically possible, though they may be rare in a living cell. More bonds, i.e., more complementary pairs, make the molecule more stable. If there are not enough bonds, the two stranded molecular structure may become weak and the strands may be separated. DNA which is no longer in the helical form is said to be *denatured*.

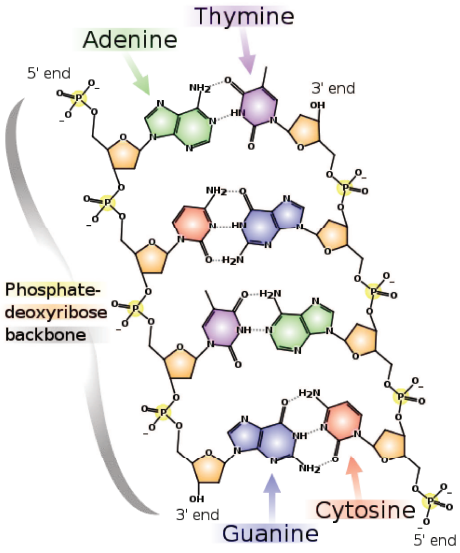


Figure 2.10: Chemical structure of DNA

2.2.4 RNA

Ribonucleic acid (RNA) was discovered after DNA. DNA, with exceptions in chloroplasts and mitochondria, is restricted to the nucleus (in eukaryotes, to the nucleoid region in prokaryotes); instead, RNA occurs in the nucleus as well as in the cytoplasm. RNA is constructed from nucleotides, as DNA, but differs in few very important details: instead of the thymine (T), it has an alternative base, the uracil (U), which is not found in DNA. Because of this minor difference, RNA does not form a double helix, instead usually it is single stranded, but may have complex spatial structure due to complementary links between the parts of the same strand. RNA has various functions in a cell required for protein synthesis.

RNA can bind complementarily to a single strand of a DNA molecule, even though T is replaced by U, so molecules like that shown in Fig. 2.14 are possible and, in fact, play an important role in life processes and in biotechnology.

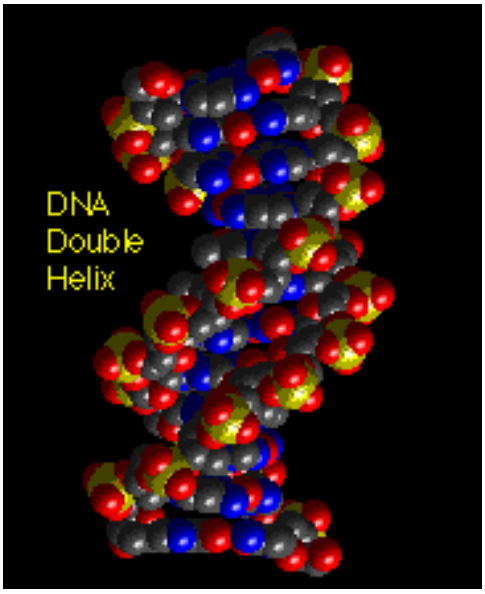


Figure 2.11: A space-filling model shows the conformation of the DNA double helix. The two strands of DNA wind around each other to form a right-handed helix with 10 bases per turn.

2.3 Genes and Genomes

2.3.1 Chromosomes, Genomes and Sequencing

Large amounts of DNA are required to encode all the information needed to make even a single-celled bacterium, and far more DNA is needed to encode the instructions for the development of multicellular organisms like humans. Each human cell contains about 2 m of DNA; yet the cell nucleus is only 5-8 μm in diameter. Tucking all this material into such a small space is the equivalent of trying to fold 40 km (24 miles) of extremely fine thread into a tennis ball.

In eucaryotic cells, very long double-stranded DNA molecules are packaged into structures called *chromosomes*, which not only fit readily inside the nucleus but can be easily apportioned between the two daughter cells at each cell division. The complex task of packaging DNA is accomplished by specialized proteins that bind to and fold the DNA, generating a series of coils and loops that provide increasingly higher levels of organization and prevent the DNA from becoming an unmanageable tangle. Amazingly, the DNA is compacted in a way that allows it to remain accessible to all of the enzymes

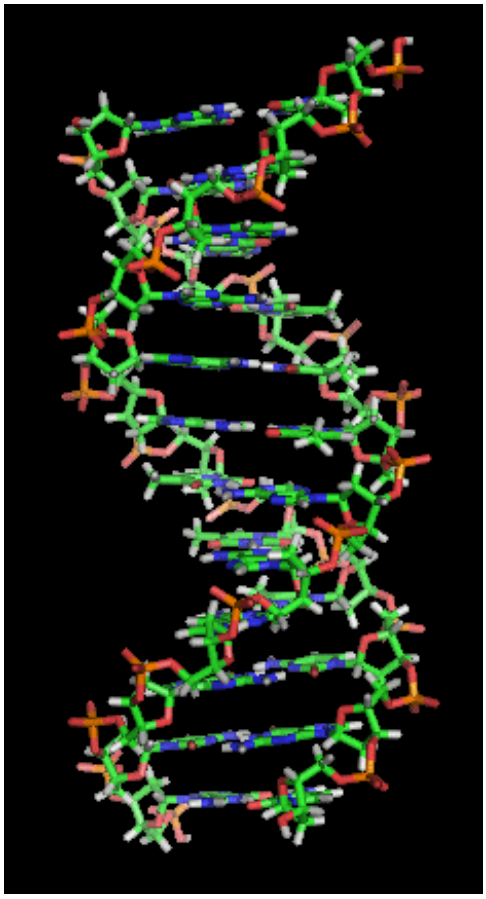


Figure 2.12: A section of DNA. The bases lie horizontally between the two spiraling strands

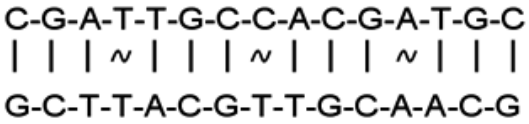


Figure 2.13: Molecules with mismatched components between polynucleotide strand

and other proteins that replicate it, repair it, and direct the expression of its genes.

There is a molecular machinery in cells, which keeps both DNA strands intact and complementary (i.e., if one strand is damaged, it is repaired using

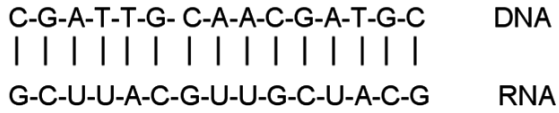


Figure 2.14: A complementary bind between RNA and a single strand of DNA

the second as a template). This is important as DNA damage (caused by environmental factors like radiation) can result in breaks in one or both strands, or mispairing of the bases, which would disrupt DNA replication among other things. If damaged DNA is not repaired the result can be cell death or tumours. Changes in genomic DNA are known as *mutations*. In this context the developed Medical Decision Support System for the survival prediction in breast cancer is inserted, which is described in depth in Chapter 5 and Chapter 6.

Determining the four-letter sequence for a given DNA molecule is known as the *DNA sequencing*. The first full genome for a bacterium was sequenced in 1995. The yeast (Saccharomyces cerevisiae) genome was sequenced in 1997, worm (nematode Caenorhabditis elegans) in 1999, fly (Drosophila melanogaster) in 2000, and weed (Arabidopsis thaliana) in 2001. The sequencing of the human genome was completed in 2003.

2.3.2 Genes and Protein Synthesis

Genomes contain genes, most of which encode proteins.
There are many discussions between biologists aimed at finding a comprehensive definition of a gene, which is not easy, if possible at all.

The definition given by Lodish et al. in [Lodish et al., 2008] is:
" A *Gene* is the entire nucleic sequence that is necessary for the synthesis of a functional polypeptide."
To better understand this definition, it is necessary to describe the molecular machinery making proteins based on the information encoded in genes. This process is called *protein synthesis* and has three essential stages: (1) transcription, (2) splicing, and (3) translation.

I. In the ***transcription*** phase the two-stranded DNA double helix is unwound and information is read only from one strand. The protein

complex, called RNA polymerase II, copies one strand of the DNA molecule into a complementary strand called *pre-mRNA* (pre stands for preliminary and m for messenger).

II. The genomic DNA that corresponds to the coding part of genes is not continuous, but consists of *exons* and *introns*. Exons are the part of the gene that encode proteins; they are interspersed with non-coding introns which must be removed (Fig. 2.15).

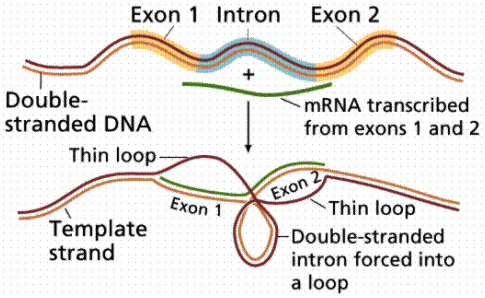


Figure 2.15: Introns and exons in a double strand of DNA

Splicing is the stage that removes the introns and joins together the remaining sections of the pre mRNA, the exons (Fig. 2.16). The number and size of introns and exons differs considerably between genes and also between species. Prokaryote genes do not have introns and the splicing step is not present. The result of splicing is *mRNA*. Many eukaryote genes are known to have different alternative splice variants, i.e. the same pre-mRNA producing different mRNAs, known as *alternative splicing*.

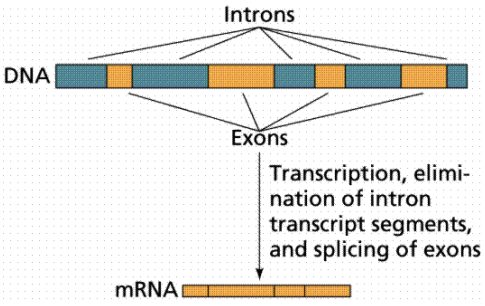


Figure 2.16: A scheme of transcription phase of DNA into mRNA

III. ***Translation*** is the process of making proteins by joining together

amino-acids in the order encoded in the mRNA. The order of the amino acids is determined by 3 adjacent nucleotides in the DNA. This is known as the triplet or genetic code. Each triplet is called a *codon* and codes for one amino-acid. As there are 64 codons and only 20 amino-acids the code is redundant, for example histidine is encoded by CAT and CAC.

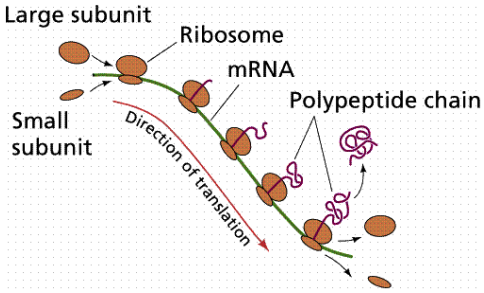


Figure 2.17: Polysome forms: a structure in which many ribosomes read the same message

In cytoplasm the mRNA forms a complex with ribosomes, which are large complex of proteins and RNA molecules. Each different transfer (or tRNA molecule) carries one specific amino acid to the ribosome and specifically recognises one codon on the mRNA. The amino acid carried by the tRNA is added to the nascent (growing) protein, Fig. 2.17.
The end of translation is the final part of gene expression and the final product is a protein, the sequence of which corresponds to the sequence encoded by the mRNA. Anyway, the translation is a complex process and not all the details are yet understood.

Moreover, proteins can be modified post-translationally, e.g., by adding of sugars or cleavage (chopping), and this affects their location and function.
Biologists used to believe in the paradigm ”one gene - one protein”. Now this is known not to be true, since, due to alternative splicing and post-translational modifications, one gene can produce a variety of proteins. There are also genes that do not encode proteins but encode RNA (for instance tRNA and ribosomal RNA); for a more detailed explanation see [Albert et al., 2010] .

2.3.3 Gene Prediction, Counting, Annotation

Given the genomic DNA sequence, the detection of the exact location of the genes is yet an attractive challenge. The current methodologies and technologies are able to achieve this purpose [Polymeropoulos et al., 1996], although the accuracy of such predictions is not very high. Most of the know-how needed to make these predictions comes from experimentally identified genes. It is an important problem for computational biology and there are a number of applications approaching this issue, including DNA sequencing, DNA fragment assembly, multiple sequence alignment, protein function prediction, gene expression, gene selection and cancer classification [Vanneschi et al., 2010, Farinaccio et al., 2010, Hassanien et al., 2008, Altman et al., 2001, Cios et al., 2005, Ezziane, 2006]. The first two references are related with a part of the work presented in this thesis, in Chapter 5 and Chapter 6.

A related issue is the detection of when and where genes are expressed in the cell, or in the organism as a whole. Determining the pattern and timing of a gene's expression can be accomplished by associating the regulatory region of the gene under study to a *reporter gene*, whose activity can be monitored. Gene expression is controlled by regulatory DNA sequences, usually located upstream of the coding region, that are not trascribed themselves. These regulatory sequences, which control which cells will express a gene and under what conditions, can also be made to control the expression of a reporter gene. The level, timing, and cell specificity of reporter protein production will reflect the function of the original gene as well as the action of the regulatory sequences that belong to it.

Moreover, one gene can affect the expression of another gene by binding of the gene product of one gene to the promoter region of another gene. Looking at more than two genes, we refer to the *regulatory network* as the regulatory interactions between the genes. If we have a large number of measurements of the expression level of a number of genes, we should be able to model or reverse engineer the regulatory network that controls their expression level.
To this purpose, in the second phase of this thesis a system for a gene regulatory network reconstruction has been developed, which is described in Chapter 7.
No matter which organism is analysed, there is a large fraction of genes that

is not yet functionally characterized. They have been predicted either by a match to homologous gene in another organism, or by *gene finding* in the genomic sequence.
Gene finding uses computer software to predict the structure of genes based on DNA sequence alone [Guigo et al., 1992]. Hopefully they are marked as 'hypothetical' genes by annotator. For certain purposes, for example when designing a chip to measure all genes of a new microrganism, we are not able to rely exclusively on functionally characterized genes and genes identified by homology. To get a better coverage of genes in the organism we may have to include those predicted by gene finding.

So it is important to measure the quality of the gene finding methods and approaches that have been used. While expression analysis may be considered a good method for experimental verification of predicted genes (if we find expression of the predicted gene it confirms the prediction), this method can become a costly verification if there are hundreds of false positive predictions that all have to be tested. A study of Skovgaard [Skovgaard et al., 2001], showed that for *Escherichia Coli* the predicted number of 4300 genes probably contains 500 false positive predictions. The most extreme case is the *Archea Aeropyrum Pernix* where all open reading frames longer than 100 triplets were annotated as genes. Half of these predictions are probably false [Skovgaard et al., 2001].

Thus, for all these reasons, it is not yet entirely obvious how to count genes in the genome. Due to the existence of overlapping genes and splice variants, it is difficult to define which parts of the DNA should be regarded as the same or several different genes. In a rough manner, allowing for some experimental error described above, we can count how many genes an organism has. Some of the results of the counting of predicted genes have turned out to be quite surprising. For instance, the number of genes in a human genome, compared to the genome of a worm, is relatively small 2.1.

Organism	*Number of predicted genes*	*Percentage of genes that encode proteins (exons)*
E.Coli (bacteria)	5000	90%
Yeast	6000	70%
Worm	18000	27%
Fly	14000	20%
Weed	25500	20%
Human	30000	$< 5\%$

Table 2.1: Percentage of exons in E.Coli, Yeast, Worm, Fly, Weed and Human

The presence of 95% of non-coding DNA in the human genome (sometimes called the *junk DNA*) remains not understood at all.

2.3.4 Genome Similarity and Genetic Variation

Actually it is well-known that there is no people having the same genome as another one, with the exception of identical twins. When the same region of the genome from two different humans is compared, the nucleotide sequences typically differ by about 0.1% . That may seem an insignificant degree of variation, but considering the size of the human genome, that amounts to some 3 million genetic differences in each maternal or paternal chromosome set between one person and the next.
Most of the genetic variation in the human genome takes the form of single-base changes called *single-nucleotide polymorphisms* (SNPs.) These polymorphisms are DNA sequence mutations which occur when a single nucleotide (A,C,G or T) is altered so that different individuals may have different nucleotide in these positions.
Two human genomes chosen at random from the world's population will differ by approximately $2.5x10^6$ SNPs that are scattered throughout the genome. Since SNPs are present in such quantities, they provide useful markers in genetic analyses in which one tries to link a particular property (such as disease susceptibility) with a specific pattern of SNPs. For example, there is evidence that certain combinations of SNPs occur in individuals with Alzheimer's disease. This type of analysis should lead to improvements in healthcare by allowing specialists to determine whether an individual is vulnerable to a disease, such as heart disease, breast cancer, etc. So the subject person can change behavior to help prevent the disease before it arises or becomes irreversible. The Medical Decision Support System for survival prediction in breast cancer, presented in Chapter 5 and Chapter 6 is developed to this purpose.

Some SNPs vary in frequency between different populations, so SNPs analysis can also be used in population genetics studies. There are about 3 million SNPs collected in public SNP databases.

In addition to the SNPs, there are several different sources of variation inherited from our ancestors, like the duplication and deletion of large blocks of DNA and like repetitive nucleotide sequences that are particularly prone to new mutations. CA repeaters, for example, are ubiquitous in the human

genome. Nucleotide sequences containing large numbers of CA repeats are often replicated inaccurately; hence the exact number of repeats can vary widely from one individual to another. Since they show such exceptional variability, they make ideal markers for differences between individual humans. In fact, at present, differences in the numbers of CA and other types of repeats at different positions in the genome are used to identify individuals by *DNA fingerprinting* in crime investigations, paternity suits, and other forensic investigations.

Most of the variations in the human genome sequence are genetically silent, as they fall within DNA sequences in noncritical regions of the genome. Such variations have no effect on how we look and how our cells function. This means that only a small subset of the variation we observe in our DNA is responsible for the heritable differences from one human to the next. A major challenge in human genetics is to learn to recognize those relatively few variations that are functionally important against the large background of neutral variation.

2.4 Functional Genomics

Even with the human genome sequence in hand, many questions will continue to challenge cell biologists throughout the next century. Functional genomics can be roughly defined as using the emerging knowledge about genomes to answer these questions, understanding the gene and their product functions and interactions, and most importantly of all, comprehending how all this makes organisms function the way they do.

2.4.1 Gene Regulatory Networks

Traditionally molecular biology has followed the so-called reductionist approach mostly concentrating on the study of a single or very few genes in any particular research project. There is a proverb: ”one gene at a time” or ”one post-dot, one gene”. With genomes being sequenced, this is now changing into the so-called *systems approach*. Almost all cells in a particular organism have an identical genome, so we can begin asking questions such as how many genes are expressed in different cell types, which genes are expressed in all cell types, what are the functional roles of these genes, how big is the gene function universe, how many genes are needed for life, how it can be

that a worm has more genes than a fly, and the human only a bit more than a worm, and, of course the most perplexing questions about human genome: given that a human, a chimp and a mouse contain essentially the same genes, and therefore the same proteins, what makes these creatures so different from each other?

The answer, it seems, will come in large part from studies of *gene regulatory network*. The protein encoded in the genome are like the components of a construction kit. By assembling the components in different combinations and in different orders, many different things can be built with the same kit. Anyway, the overall shape of the final object is determined by the instructions that prescribe how the components are to be put together.

To a large extent, the instructions needed to produce a multicellular animal are contained in the noncoding regulatory DNA that is associated with each gene. This DNA contains sparse within it, dozen of separate regulatory elements, short DNA segments that are used as binding sites for specific transcription regulators. This regulatory DNA can be said to define the sequential program of development of the rules that cells follow as they proliferate, assess their positions in the embryo, and switch on new sets of genes accordingly.

Altough comparison among many different species are a powerful way to locate key regulatory sequences in a vast excess of irrilevant DNA, we still do not know how to read these sequences accurately. For example, different transcription regulators can bind to the same short strech of DNA, so that knowing the DNA sequence is not enough to specify which protein or proteins might really regulate the gene. In addition, the fact that control of gene expression occur in complex and combinatorial ways complicates our attemps to decode when in development and in which type of cell each gene is expressed.

Another challenge in interpreting the information encoded in the human genome comes from the prevalence of *alternative splicing*. We know that most human genes are subjected to alternative splicing, allowing cells to produce a range of related but distinct proteins from a single gene. Often this splicing is regulated, so that one form of the protein is produced in one type of tissue, while other forms are produced preferentially in other tissues. Thus an organism can produce far more protein products than it has genes. We do not know enough yet about the biology of alternative splicing to

predict exactly which human genes are subject to this process, and when and where during development such regulation might occur. However, it does seem that alternative splicing is especially prevalent in the developing brain.

Another doubt in interpreting the human genome concerns the exact roles of *microRNAs*. Discovered relatively recently, these short RNAs regulate as many as one-third of all human genes, yet few of them have been studied in any detail. Finally, although an estimated 1.5% of the human genome codes for protein, an additional 3.5% is highly conserved when compared with that of other mammalian genomes, and is therefore presumed to be considerable. Some of this *conserved DNA* produces RNA molecules of known function and some is regulatory DNA; the rest remains a mistery [Albert et al., 2010].

Microarrays (see next section) and *computational methods* are playing a major role in attempts to reverse-engineering gene networks from various observations. Note that, in reality, the gene regulation is likely to be a stochastic and not a deterministic process. For this reason an attempt to reconstruct and simulate it is performed in this thesis using a stochastic approach and implemented through Genetic Programming, a non deterministic algorithm-based methodology.

2.5 Microarrays and Gene Expression

As previously described, proteins in a cell are synthesised from genes and their life cycle can be roughly described as synthesis, functionality and degradation. Nobody really knows how many different proteins are synthesized from the estimated 36.000 genes in a human cell [Lodish et al., 2008, Albert et al., 2010].
The protein abundance may depend on many factors such as whether the respective gene is expressed (i.e., is actively transcribed) or not, how intensively (how fast) it is expressed, whether and how fast it is spliced, translated and modified, how long a half-life the mRNA and the protein have, and whether it is actively degraded at a given moment. Direct experimental studies of protein abundance are technically difficult at present. However, thanks to the microarray technology (see the next section), it is possible to measure the mRNA abundance (gene expression) for tens of thousands

of genes in parallel in a single experiment. The correlation between gene expression and the presence of the respective proteins in the cell is not straightforward, still in many cases some estimates about the proteins can be made from gene expression.

Microarray technology makes use of the sequence resources created by the genome projects and other sequencing efforts to answer the question, what genes are expressed in a particular cell type of an organism, at a particular time, under particular conditions. For instance, they allow comparison of gene expression between normal and diseased cells (e.g., cancerous cells), between cancerous and non-cancerous conditions, allowing also a prediction on patients survival status, as the case presented in Chapter 5 and Chapter 6.

2.5.1 Microarrays Technology and Applications

The DNA microarrays (*microarray* or *array* for short) were developed as an extensive mean of monitoring the expression patterns (or more precisely the transcription patterns) of large numbers of genes (sometimes even entire genomes) at once, exploiting the preferential binding of complementary single-stranded nucleic acid sequences. So they induced an enormous improvement over the classical "one gene per experiment" paradigm that dominated until then.

A typical DNA microarray experiment proceeds as follows: a small slide of glass (or of some other material) is taken. The surface of the slide has been divided into a series of imaginary square cells, called *spots*, to form a rectangular grid [Amaratunga and Cabrera, 2004]. There may be tens of thousands of spots on an array, each containing a huge number of identical DNA molecules; These should ideally identify one gene or one exon in the genome, even if this is not so straightforward and may not even be always possible due to families of similar genes in a genome. The spots are either printed on the microarrays by a robot, or synthesized by photo-lithography (similar as in computer chip productions) or by ink-jet printing. A typical dimension of such an array is about 1 inch or less, the spot diameter is of the order of 0.1 mm, although for some microarray types it can be even smaller. Separately, a solution that contains a mixture of mRNAs whose sequences are unknown, is prepared. A substance that fluoresces when excited by light is added to this solution. The solution then is poured onto the slide. From that moment on, the mRNA molecules will diffuse over the slide and,

wherever they find a matching DNA sequence, such as the one taken from the gene from which the mRNA was transcribed, they will hybridize to each other, i.e. join their complementary strands of DNA, and the solution will stick to the slide. Without a match the solution will not stick to the slide and can be washed away. Finally, a laser scanner is used to detect and measure the fluorescent signal being emitted to each cell.

So, summarizing, the five basic steps of a typical basic microarray experiment are [Amaratunga and Cabrera, 2004]:

I. Preparing the microarray

II. Preparing the labeled sample

III. Hybridizing the labeled sample to the microarray and washing the microarray

IV. Scanning the microarray

V. Interpreting the scanned image

There are different ways microarrays can be used to measure the gene expression levels. One of the most popular micorarray applications allows to compare gene expression levels in two different samples, e.g., the same cell type in a healthy and diseased state.

It is easier to explain the principle behind this kind of microarray experiments with a very simple hypothetical example. Suppose that we have obtained some cancerous liver tissue and some normal liver tissue from a liver cancer patient and that we want to know which genes are expressed differently in the two. We will begin by extracting mRNA from each tissue so that we have two mRNA samples. Note that in each sample only mRNA corresponding to any genes that were expressed (i.e. transcribed) would be present. We will reverse-transcribe the mRNA to cDNA (complementary DNA) and label these two samples with two different fluorescent dye: for example a green dye for cells obtained from the cancerous liver tissue and a red dye for the cells obtained from the normal liver tissue. These two labeled samples are sometimes called *target*, sometimes are called *probes*, because they are used to probe the collection of spots on a microarray, and sometimes are simply called *labeled samples*.

Now suppose we have prepared a DNA microarray containing the entire human genome. Suppose there are 36.000 genes. A DNA microarray for this experiment would be a tiny glass slide on which the 36.000 genes are printed in, say a 300 × 120 rectangular array of spots, one gene per spot.

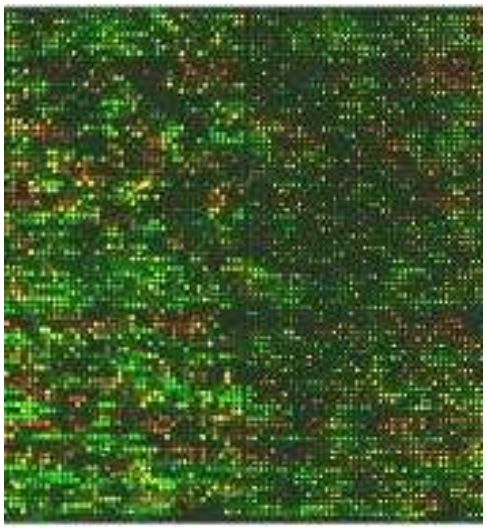


Figure 2.18: An illuminated microarray (enlarged)

We will now flood the microarray with the labeled sample from the cancerous tissue and then with the other labeled sample from the normal tissue. We allow enough time for any cDNA in the samples to recognize and hybridize to its complementary sequence in the spots of microarray due to the preferential binding. Complementary single stranded nucleic acid sequences tend to attract each other and the longer the complementary parts, the stronger the attraction. (see Fig. 2.19).

The dyes enable the amount of sample bound to a spot to be measured by the level of fluorescence emitted when it is excited by a laser. If the mRNA from the sample in cancerous condition is in abundance, the spot will be green, if the mRNA from the sample in normal condition is in abundance, it will be red. If both are equal, the spot will be yellow, while if neither are present it will not fluoresce and appear black (Fig. 2.18). Thus, from the fluorescence intensities and colours for each spot, the relative expression levels of the genes in both samples can be estimated.

The raw data that are produced from microarray experiments are the hybridised microarray images. To obtain information about gene expression

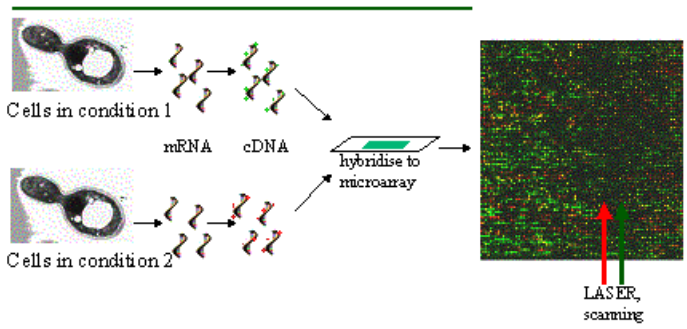


Figure 2.19: Microarray Technology

levels, these images should be analysed, each spot on the array identified, its intensity measured and compared to the background. This is called *image quantitation*.

Image quantitation is done by image analysis software. To obtain the final gene expression matrix from spot quantitations, all the quantities related to some gene (either on the same array or on arrays measuring the same conditions in repeated experiments) have to be combined and the entire matrix has to be scaled to make different arrays comparable.

2.5.2 Gene Expression Data Analysis and Expression Profiler

Capturing and storage of microarray data is not a goal in itself. The amounts of data from even a single microarray experiment are so large, that software tools have to be used to make any sense out of it. This PhD work is developed in this direction. Clustering and classification are typical methods currently used in gene expression data analysis.

An example of such a research is an approach to reverse engineering of gene regulatory networks, which is based on the hypothesis that genes that have similar expression profiles (i.e., similar rows in the gene expression matrix) should also have similar regulation mechanisms as there must

be a reason why their expression is similar under a variety of conditions. Therefore, if we cluster the genes by similarities in their expression profiles and take sets of promoter sequences from genes in such clusters, some of these sets of sequences may contain a ”signal” as a specific sequence pattern such as a particular substring which is relevant to regulation of these genes.

Chapter 3

Machine Learning and Statistical Data Analysis

In this chapter the basic concepts of Machine Learning (ML) and Statistical Methods are introduced. The purpose is not to describe the entire field of ML exhaustively, but to describe some fundamental concepts useful for the comprehension of the Medical Decision Support System and the Data Driven Gene Regulatory Network developed in this thesis and described in the following chapters. This introduction is widely inspired by the books [Mitchell, 1996, Alpaydin, 2010, Banzhaf et al., 1998] to which the reader is referred for a deeper description.

3.1 Overview of Machine Learning

Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviours based on empirical data.
With the development of the computer science the natural behaviour and evolution inspired scientists to better understand them and teach their processes to the computers so that they could improve automatically with experience. Following this dream, in 1958 and 1959 Friedberg attempted to solve fairly simple problems by teaching a computer to write computer programs [Friedberg, 1958, Friedberg et al., 1959].

> If we are ever to make a machine that will speak, understand or translate human languages, solve mathematical problems with imagination, practice a profession or direct an organization, either we must reduce these activities to a science so exact that we can tell a machine precisely how to go about doing them or we must develop a machine that can do things without being told precisely how... . The machine might be designed to grav-

> itate toward those procedures which most often elicit from us a favorable response. We could teach this machine to perform a task even though we could not describe a precise method for performing it, provided only that we understood the task well enough to be able to ascertain whether or not it had been done successfully. ...In short, although it might learn to perform a task without being told precisely how to perform it, it would still have to be told precisely how to learn.

Friedberg's analysis anticipated the coming split between artificial intelligence (with its emphasis on expert *knowledge*) and machine learning (with its emphasis on *learning*) [Banzhaf et al., 1998]. During the 1960s and 1970s, the study of domain knowledge and expert knowledge systems, named as *artificial intelligence* (AI), was the dominant form of computational intelligence. These systems generally encoded human knowledge. For example, an expert system might be developed by consulting human experts about how they make particular kinds of decisions. Then, the results of that interview would be encoded into the expert system for use in making real-world decision. The type of intelligence represented by such expert systems was a static comprehension because it did not learn from experience, so it was quite different from *machine learning* (ML). Along the time, expert systems have turned out to be fragile and to have difficulty handling inputs that are different or noisy. So, in the 1970s the focus on machine learning reappeared. Interest switched from the static question of how to represent knowledge to the dynamic question of how to acquire it. The search began in earnest to find a way, in Friedberg's words, to tell a computer "precisely how to learn". By the early 1980s, machine learning was recognized as a distinct scientific discipline [Banzhaf et al., 1998].
Today in the field known as data mining, machine learning algorithms are being used widely to discover valuable knowledge from large commercial databases containing equipment maintenance records, loan applications, financial transactions, medical records, to perform time series prediction, industrial process control, prediction of creditworthiness, pattern recognition such as optical character recognition and voice recognition.
For the purposes of this work, learning is defined roughly, to include any computer program that improves its performance at some task through experience. Put more precisely, as stated in [Mitchell, 1996]:

Definition 3.1. A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Machine learning systems are usually applied to a "learning domain". A learning domain is any problem or search of facts where the researcher can identify "features" of the domain that may be measured, and can discover a result or results (usually organized as "classes") that he would like to predict. Simply speaking, machine learning is a process that begins with the identification of the learning domain and ends with testing and using the results of the learning. An example could be the case studied in this work and discussed in more details in Chapter 5 and in Chapter 6 . In this case, the microarray of the gene expression value of a set of breast cancer patients has been chosen as domain. The genes are the features of the domain selected and the survival status of the patients at a fixed timepoint are the results that have to be predicted. Of course, the features, i.e. the input (genes, in the example) ought to be related in some manner to the desired result, that is the output (survival status of the patients). Otherwise, a machine learning system based on this features will have little predictive power. Once the features are chosen from the learning domain, they define the overall dimensions of the environment that will train the ML system and from which it will, hopefully, learn.
But the selection of features (inputs) does not completely define the environment from which the system will learn. The researcher must also choose specific examples from the learning domain. Each example should contain data that represent one instance of the relationship between the choosen features (inputs) and the classes (output). These examples are often referred to as "training cases", or "training instances" or "training set". Once the training set is selected, the learning environment of the system has been defined. Machine learning occurs by training. An ML system goes through the training set and attempts to learn from the examples [Banzhaf et al., 1998]. Finally, the researcher must estimate the quality of the learning that has taken place. One way to estimate the quality of learning is to test the ability of the best solution of the ML system to predict outputs from a "test set". A test set is comprised of inputs and outputs from the same domain the system was trained upon. Although from the same domain, the test set contains different examples than the training set. The ability of a system

to predict the outputs of the test set is often referred to as ”generalization”, that is, it answers the question: could the learned solution *generalize* to new data or has it just *memorized* the existing training set?

3.2 Machine Learning Methods

This section contains a brief description of some ML methods, which were used to compare the performance of the binary and floating point version of the Medical Decision Support System developed.
It is painted with a very broad brush and is not a complete discussion of them; a deeper description is referred to in the bibliography.

3.2.1 Support Vector Machines

Support Vector Machines (SVM) are a set of related supervised learning methods used for classification and regression. They were originally introduced in [Vapnik, 1998]. Their aim is to devise a computationally efficient way of identifying separating hyperplanes in a high dimensional feature space. In particular, the method seeks separating hyperplanes maximizing the margin between sets of data. This should ensure a good generalization ability of the method, under the hypothesis of consistent target function between training and testing data. To calculate the margin between data belonging to two different classes, two parallel hyperplanes are constructed, one on each side of the separating hyperplane, which are “pushed up against” the two data sets. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the neighboring data points of both classes, since in general the larger the margin the lower the generalization error of the classifier. The parameters of the maximum-margin hyperplane are derived by solving large quadratic programming (QP) optimization problems.

3.2.2 Multilayer Perceptron

The Multilayered Perceptron is a feed-forward artificial neural network model [Haykin, 1999]. It is a modification of the standard linear perceptron in that it uses three or more layers of neurons (nodes) with nonlinear activation functions, and is more powerful than the simple perceptron in that it can distinguish data that are not linearly separable, or separable

by a hyperplane. It consists of an input and an output layer with one or more hidden layers of nonlinearly-activating nodes. Each node in one layer connects with a certain weight to every other node in the following layer.

3.2.3 Random Forests

Random Forests denotes an improved Classification and Regression Trees method [Breiman et al., 1984]. It works by creating a large number of classification trees or regression trees. Every tree is built using a deterministic algorithm and the trees are different owing to two factors. First, at each node, a best split is chosen from a random subset of the predictors rather than from all of them. Secondly, every tree is built using a bootstrap sample of the observations. The out-of-bag data, approximately one-third of the observations, are then used to estimate the prediction accuracy. Unlike other tree algorithms, no pruning or trimming of the fully grown tree is involved.

3.2.4 Radial Basis Function Network

A Radial Basis Function Network (RBFN) is an artificial neural network that uses radial basis functions as activation. RBFNs are embedded in a two-layer neural network, where each hidden unit implements a radial activated function. The output units implement a weighted sum of hidden unit outputs. Their excellent approximation capabilities have been studied in [Park and Sandberg, 1991, Poggio and Girosi, 1990]. Due to their nonlinear approximation properties, RBFNs are able to model complex mappings, which generally perceptron neural networks can only model by means of multiple intermediary layers [Haykin, 1999].

3.2.5 Voted Perceptron

Voted Perceptron [Freund and Schapire, 1998] is an artificial neural network which combines the Rosenblatt's Perceptron algorithm [Haykin, 1999] with Helmbold and Warmuth's [Helmbold and Warmuth, 1995] leave-one-out method. Like Vapnik's maximal-margin classifier [Vapnik, 1998], this method takes advantage of data that are linearly separable with large margins. Compared to Vapnik's algorithm, however, it is simpler to implement, and usually more efficient in terms of computation time. It can also be efficiently used in very high dimensional spaces using kernel

functions. In particular, compared to the standard Rosenblatt's perceptron algorithm, Voted Perceptron stores more information during training and then uses this elaborate information to generate better predictions on the test data. The information maintained during training is the list of all prediction vectors that were generated after each and every mistake. For each such vector, the number of iterations it survives until the next mistake is made is counted (this count is often referred to as the weight of the prediction vector). To perform a classification, the binary prediction of each prediction vector is computed and all these classifications are combined by a weighted majority vote. The weights used are the survival times described above. This makes intuitive sense as good prediction vectors tend to survive for a long time and thus have a larger weight in the majority vote. For a more detailed exposition see [Helmbold and Warmuth, 1995].

Another technique widely used in ML is Genetic Programming. It is intensely applied in the Medical Decision Support System and Data Driven Gene Regulatory Network developed in this thesis, so an entire chapter, the next one, is devoted to its description.

3.3 Statistical Methods for Resampling, Validation and Evaluation

The use of statistic methods is helpful to analyzing machine learning results in such a way that whatever conclusion we get is not subjective or due to chance. If, for example, we want to answer the question "Is A a more accurate algorithm than B?", we can transform it in the following statistical hypothesis test : "Can we say that the average error of learners trained by A is significantly lower than the average error of learners trained by B?" In this manner we can apply the statistical methods to generalize the results.

3.3.1 Resampling Methods with Cross-Validation

We need several replications to draw objective conclusions analyzing data with statistical methods. The problem is that, for replication purposes, it is necessary to get a number of training and test set pairs from a dataset X. To get them, if the sample X is large enough, we can randomly divide it into K parts, then randomly divide each part into two and use one half for

training and the other half for validation. K is generally 10 or 30, sometimes 50. Unfortunately, datasets are never large enough to do so. So we must do our best with small datasets. This is done by repeated use of the same data, split differently; this is called *cross-validation*. So, given a dataset X, we create from this dataset K training/test set pairs, $\{T_i, V_i\}_{i=1}^{K}$.
There are several Cross-Validation methods, as K-Fold Cross-Validation, 5x2 Cross-Validation and Bootstrapping. Just for example, the K-Fold Cross-Validation is described in more details in the next section.

K-Fold Cross-Validation

In K-fold cross-validation, the dataset X is divided randomly into K equal sized parts, $X_i, i = 1,...,K$. To generate each pair, we keep one of the K parts out as the test set and use the remaining $K-1$ parts as the training set. Doing this K times, each time leaving out a different part out of the K parts, we get K pairs:

$$V_1 = X_1 \; T_1 = X_2 \bigcup X_3 \bigcup ... \bigcup X_K$$
$$V_2 = X_2 \; T_2 = X_1 \bigcup X_3 \bigcup ... \bigcup X_K$$
$$\vdots$$
$$V_K = X_K \; T_K = X_1 \bigcup X_2 \bigcup ... \bigcup X_{K-1}$$

There are two crucial points with this. First, to keep the training set large, we allow test sets that are small. Second, the training sets overlap considerably, in fact any two training sets share $K-2$ parts. As K increases, the percentage of training instances increases and we get more robust estimators, but the test set becomes smaller. Furthermore, there is the cost of training the classifier K times, which increases as K is increased. As N, the cardinality of the dataset, increases, K can be smaller; if N is small, K should be large to allow large enough training sets. One extreme case of K-fold cross-validation is the leave-one-out case where given a dataset of N instances, only one instance is left out as the validation set (instance) and training uses the other $N-1$ instances. We then get N separate pairs by leaving out a different instance at each iteration.

3.3.2 Evaluation of Classifier Performance

In order to evaluate classifier performance, it is necessary to define a measure. Several types have been proposed. One of them, the most used, is the percentage of the *true positives*, *true negatives*, *false positives* and *false negatives* (see the Table 3.1).

	PREDICTED CLASS		
TRUE CLASS	**POSITIVE**	**NEGATIVE**	**TOTAL**
Positive	*tp:* true positive	*fn:* false negative	p
Negative	*fp:* false positive	*tn:* true negative	n
Total	p'	n'	N

Table 3.1: Confusion matrix for two classes

For a positive instance, if the prediction is also positive, this is called a true positive; if the prediction is negative for a positive example, this is called a false negative. For a negative instance, if the prediction is also negative, we have a true negative, and we have a false positive if we predict a negative instance as positive.
In some two-class problems, we make a distinction between the two classes and hence the two types of errors, false positives and false negatives.
Let us image an application which classifies, for example, the survival status of a cohort of cancer patients, the application studied in this work and described in more details in Chapters 5 and Chapter 6. A *false negative* is a patient wrongly classified as negative to the disease, therefore classified as healthy, while he is actually affected by cancer. A *false positive*, instead, is a healthy patient wrongly classified as diseased. It is clear that the two type of errors are not equally bad: the former is much worse.

From another perspective, but with the same goal, there are the two measures of *sensitivity* and *specificity*. Sensitivity measures which percentage of positive cases have been correctly detected. Specificity measures how well we detect the negatives, and is defined as the number of true negatives divided by the total number of negatives.

Interval Estimation

Let us now take a quick glance of the *interval estimation* that we will use in the hypothesis testing for the comparison of two classification algorithms.

A point estimator, for example, the maximum likelihood estimator, specifies a value for a certain parameter ϑ of the problem analyzed. Interval estimation, instead, specifies an interval within which ϑ lies with a certain degree of confidence. To obtain such an interval estimator, we make use of the probability distribution of the point estimator.

Interval Estimation using normal distribution

In order to make the explanation clearer, we first introduce firstly an example.

Let us say we are trying to estimate the mean μ of a normal density function from a sample $X = \{x^t\}_{t=1}^N$, where N is the sample size. Also $m = \sum_t x^t/N$ is the sample average and is the point estimator to the mean; m is a sum of normals, and therefore is also normal, $m \sim \mathcal{N}(\mu, \sigma^2/N)$, σ^2 is the variance of the distribution.

We define the statistic with a *unit normal distribution*:

$$\frac{(m-\mu)}{\sigma/\sqrt{N}} \sim \mathcal{Z} \tag{3.1}$$

We know that 95 percent of $\mathcal{Z}$ lies in (-1.96, 1.96), namely, $P\{-1.96 < \mathcal{Z} < 1.96\} = 0.95$, and we can write

$$P\{-1.96 < \sqrt{N}\frac{(m-\mu)}{\sigma} < 1.96\} = 0.95 \tag{3.2}$$

or equivalently

$$P\{m - 1.96\frac{\sigma}{\sqrt{N}} < \mu < m + 1.96\frac{\sigma}{\sqrt{N}}\} = 0.95 \tag{3.3}$$

That is "with 95 percent confidence", μ will lie within $1.96\sigma/\sqrt{N}$ units of the sample average (see Fig. 3.1).

This is a *two-sided confidence interval*.

If we want more confidence, the interval gets larger, in fact, in the previous example the confidence with which μ lies in $(m - 2.58\sigma/\sqrt{N}, m + 2.58\sigma/\sqrt{N})$ is 99 percent. Note that the interval gets smaller as N increases. This can be generalized for any required confidence as follows.

Let us denote z_α such that

$$P\{\mathcal{Z} > z_\alpha\} = \alpha,\ 0 < \alpha < 1 \tag{3.4}$$

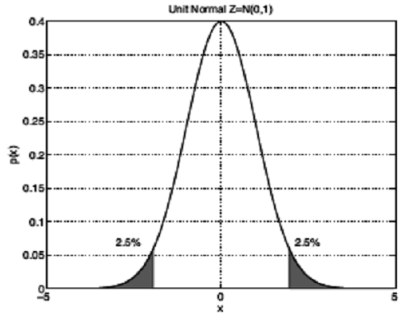


Figure 3.1: 95 percent of the unit normal distribution lies between -1.96 and 1.96

Because $\mathcal{Z}$ is symmetric around the mean (zero), $z_{1-\alpha/2} = -z_{\alpha/2}$, and $P\{X < -z_{\alpha/2}\} = P\{X > z_{\alpha/2}\} = \alpha/2$. Hence for any specified level of confidence $1-\alpha$, we have

$$P\{-z_{\alpha/2} < \mathcal{Z} < z_{\alpha/2}\} = 1-\alpha \tag{3.5}$$

and

$$P\{-z_{\alpha/2} < \sqrt{N}\frac{(m-\mu)}{\sigma} < z_{\alpha/2}\} = 1-\alpha \tag{3.6}$$

or

$$P\{m - z_{\alpha/2}\frac{\sigma}{\sqrt{N}} < \mu < m + z_{\alpha/2}\frac{\sigma}{\sqrt{N}}\} = 1-\alpha \tag{3.7}$$

Hence a $100(1-\alpha)$ percent two-sided confidence interval for μ can be computed for any α.

Similarly for the *one-sided upper confidence interval for* μ. Knowing, for example, that $P\{\mathcal{Z} < 1.64\} = 0.95$, we have

$$P\{\sqrt{N}\frac{(m-\mu)}{\sigma} < 1.64\} = 0.95 \tag{3.8}$$

or

$$P\{m - 1.64\frac{\sigma}{\sqrt{N}} < \mu\} = 0.95 \tag{3.9}$$

and $(m - 1.64\sigma/\sqrt{N}, \infty)$ is a 95 percent one-sided upper confidence interval for μ, which is defined by its lower bound (see Fig. 3.2).

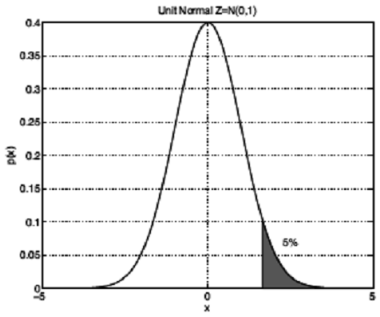


Figure 3.2: 95 percent of the unit normal distribution lies before 1.64

Generalizing, a $100(1-\alpha)$ percent one-sided confidence interval for μ can be computed from

$$P\{m - z_\alpha \frac{\sigma}{\sqrt{N}} < \mu\} = 1 - \alpha \tag{3.10}$$

Similarly, the *one-sided lower confidence interval* that defines an upper bound can also be calculated.

Interval Estimation using the t-distribution

In the previous intervals, we used σ, so we assumed that the variance is known. If it is not, one can plug the sample variance into the equation

$$S^2 = \sum_t (x^t - m)^2/(N-1) \tag{3.11}$$

instead of σ^2.

We know that when $x^t \sim \mathcal{N}(\mu, \sigma^2)$, $(N-1)S^2/\sigma^2$ follows a distribution χ^2 with $N-1$ degrees of freedom. We also know that m and S^2 are independent. Then, $\sqrt{N}(m-\mu)/S$ is t-distributed with $N-1$ degrees of freedom, denoted as

$$\frac{\sqrt{N}(m-\mu)}{S} \sim t_{N-1} \tag{3.12}$$

Hence for any $\alpha \in (0, 1/2)$, we can define an interval, using the values specified by the *t-distribution*, instead of the unit normal $\mathcal{Z}$

$$P\{t_{1-\alpha/2,N-1} < \sqrt{N}\frac{(m-\mu)}{S} < t_{\alpha/2,N-1}\} = 1-\alpha \tag{3.13}$$

or using $t_{1-\alpha/2,N-1} = -t_{\alpha/2,N-1}$, we can write

$$P\{m - t_{1-\alpha/2,N-1}\frac{S}{\sqrt{N}} < \mu < m + t_{\alpha/2,N-1}\frac{S}{\sqrt{N}}\} = 1-\alpha \tag{3.14}$$

Similarly, the one-sided confidence intervals can be defined.
The t-distribution has larger spread (longer tails) than the unit normal distribution, and generally the interval given by the t is larger; this should be expected since additional uncertainty exists due to the unknown variance.

Hypothesis Testing

Instead of explicitly estimating some parameters, in certain applications we may want to use the sample to test some particular hypothesis concerning the parameters. For example, instead of estimating the mean, we may want to test whether the mean is less than 0.02. If the random sample is consistent with the hypothesis under consideration, we "fail to reject" the hypothesis; otherwise, we say that it is "rejected". But when we make such a decision, we are not really saying that it is true or false but rather that the sample data appears to be consistent with it to a given degree of confidence.
Formally, the approach is as follows. We define a statistic that obeys a certain distribution if the hypothesis is correct. If the statistic calculated from the sample has very low probability of being drawn from this distribution, then we reject the hypothesis; otherwise, we fail to reject it.

Hypothesis Testing with normal distribution

Let us say we have a sample from a normal distribution with unknown mean μ and known variance σ^2, and we want to test a specific hypothesis about μ, for example, whether it is equal to a specified constant μ_0.
It is denoted as H_0 and is called the *null hypothesis*

$$H_0 : \mu = \mu_0 \tag{3.15}$$

against the alternative hypothesis

$$H_1 : \mu \neq \mu_0 \tag{3.16}$$

We denote m as the point estimate of μ, and it is reasonable to reject H_0 if m is too far from μ_0. This is where the interval estimate is used. We fail to reject the hypothesis with *level of significance* α if μ_0 lies in the $100(1-\alpha)$ percent confidence interval, namely, if

$$\frac{\sqrt{N}(m-\mu_0)}{\sigma} \in (-z_{\alpha/2}, z_{\alpha/2}) \tag{3.17}$$

We reject the null hypothesis if it falls outside, on either side. This is a *two-sided hypothesis test.*

If we reject H_0 when the hypothesis is correct, this is called *type I error* and thus α, set before the test, defines how much type I error we can tolerate, typical values being $\alpha = 0.1, 0.05, 0.01$ (see table 3.2). A *type II error* is if we fail to reject the null hypothesis when the true mean μ is unequal to μ_0.

	DECISION	
Truth	***Fail to reject***	***Reject***
True	Correct	Type I error
False	Type II error	Correct

Table 3.2: Type I error and Type II error of the Hypothesis Test

The probability that H_0 is not rejected when the true mean is μ is a function of μ and is given as

$$\beta(\mu) = P_\mu\{-z_{\alpha/2} \leq \frac{(m-\mu_0)}{\sigma/\sqrt{N}} \leq z_{\alpha/2}\} \tag{3.18}$$

The function $1-\beta(\mu)$ is called the *power function* of the test and is equal to the probability of rejection when μ is the true value. Type II error probability increases as μ and μ_0 get closer, and we can calculate how large a sample we need for us to be able to detect a difference $\delta = |\mu - \mu_0|$ with sufficient power.

One can also have a *one-sided hypothesis test.* The α level of significance in it defines the $100(1-\alpha)$ confidence interval bounded on one side in which m should lie for the hypothesis not to be rejected.
It has the form:

$$H_0 : \mu \leq \mu_0 \quad vs \quad H_1 : \mu > \mu_0 \tag{3.19}$$

as opposed to the two-sided test when the alternative hypothesis is $\mu \neq \mu_0$. We fail to reject if

$$\frac{\sqrt{N}}{\sigma}(m-\mu_0) \in (-\infty, z_\alpha) \tag{3.20}$$

and reject if it is outside such an interval.

Note that the null hypothesis H_0 also allows equality, which means that we get ordering information only if the test rejects. This tells us which of the two one-sided tests we should use. Whatever claim we have should be in H_1 so that rejection of the test would support our claim.

Hypothesis Testing with t-distribution

If the variance is unknown, just as we did in the interval estimates, we use the sample variance instead of the population variance as well as the assertion that

$$\frac{\sqrt{N}(m-\mu_0)}{S} \sim t_{N-1} \tag{3.21}$$

For example, for $H_0 : \mu = \mu_0$ *vs* $H_1 : \mu \neq \mu_0$, we fail to reject at significance level α if

$$\frac{\sqrt{N}(m-\mu_0)}{S} \in (-t_{(\alpha/2),N-1}, t_{(\alpha/2),N-1}) \tag{3.22}$$

which is known as the *two-sided t test.*

A *one-sided t test* can be defined similarly.

t-Testing

The two tests we discussed earlier use a single validation set. If we run the algorithm K times, on K training/test set pairs, we get K error percentages, p_i, with $i = 1, ..., K$ on the K test sets. Let x_i^t be 1 if the classifier trained on $\mathcal{T}_i$ makes a misclassification error on instance t of $\mathcal{V}_i$, the ith test set; otherwise let x_i^t be equal to 0. Then

$$p_i = \frac{\sum_{t=1}^{N} x_i^t}{N} \tag{3.23}$$

Given that

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K}(p_i - m)^2}{K-1} \tag{3.24}$$

we know that we have

$$\frac{\sqrt{N}(m-p_0)}{S} \sim t_{K-1} \tag{3.25}$$

and the *t-test* rejects the null hypothesis that the classification algorithm has p_0 or less error percentage at significance level α if this value is greater than $t_{\alpha,K-1}$.

3.4 Statistical Methods for Comparison

3.4.1 Comparing Two Classification Algorithms: K-Fold Cross-Validated Paired t-Test

We use the two classification algorithms to train on the training sets $\mathcal{T}_i, i = 1,...,K$, and test on the test sets $\mathcal{V}_i$. The error percentages of the classifiers on the test sets are recorded as p_i^1 and p_i^2.
If the two classification algorithms have the same error rate, then we expect them to have the same mean, or equivalently, that the difference of their means is 0. The difference in error rates on fold i is $p_i = p_i^1 - p_i^2$.
This is a *paired test*; that is, for each i, both algorithms see the same training and test sets. When this is done K times, we have a distribution of p_i containing K points. Given that p_i^1 and p_i^2 are both (approximately) normal, their difference p_i is also normal. The null hypothesis is that this distribution has 0 mean:

$$H_0 : \mu = 0 \quad vs \quad H_1 : \mu \neq 0 \tag{3.26}$$

So, defining

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K-1} \tag{3.27}$$

under the null hypothesis that $\mu = 0$, we have a statistic that is t-distributed with $K-1$ degrees of freedom:

$$\frac{\sqrt{N}(m-0)}{S} = \frac{\sqrt{N}m}{S} \sim t_{K-1} \tag{3.28}$$

Thus the $K-fold\ cv\ paired\ t-test$ rejects the hypothesis that two classification algorithms have the same error rate at significance level α if this value is outside the interval $(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1})$.

If we want to test whether the first algorithm yields less error than the second, we need a one-sided hypothesis and use a one-tailed test:

$$H_0 : \mu \geq 0 \quad vs \quad H_1 : \mu < 0 \tag{3.29}$$

If the test rejects, our claim that the first one has significantly less error is supported.

3.4.2 Comparing Multiple Algorithms: Analysis of Variance(ANOVA)

In many cases, as in the study of a Medical Decision Support System performed in this thesis and described in the following, we have more than two algorithms, and we would like to compare their expected error.
Given L algorithms, we train them on K training sets, induce K classifiers with each algorithm, and then test them on K validation sets and record their error rates. This gives us L groups of K values. The problem then is the comparison of these L samples for statistically significant difference. This is an experiment with a single factor with L levels, the learning algorithms, and there are K replications for each level.
In *Analysis of Variance* (ANOVA), we consider L independent samples, each of size K, composed of normal random variables of unknown mean μ_j and unknown common variance σ^2:

$$X_{ij} \sim \mathcal{N}(\mu_j, \sigma^2), \quad j = 1, ..., L, \quad i = 1, ..., K \tag{3.30}$$

We are interested in testing the hypothesis H_0 that all means are equal:

$$H_0 : \mu_1 = \mu_2 = ... = \mu_L \quad vs. \quad H_1 : \mu_r \neq \mu_s \tag{3.31}$$

for at least one pair (r, s)

The comparison of error rates of multiple classification algorithms fits this scheme. We have L classification algorithms, and we have their error rates on K validation folds. X_{ij} is the number of validation errors made by the classifier, which is trained by classification algorithm j on fold i. Each X_{ij} is binomial and approximately normal. If H_0 is not rejected, we fail to find a significant error difference among the error rates of the L classification algorithms. This is therefore a generalization of the tests we saw in the previous section that compared the error rates of two classification algorithms.

The L classification algorithms may be different or may use different hyperparameters, for example, number of hidden units in a multilayer perceptron, number of neighbors in k-nn, and so forth. The approach in ANOVA is to derive two estimators of σ^2. One estimator is designed such that it is true only when H_0 is true, and the second is always a valid estimator, regardless of whether H_0 is true or not. ANOVA then rejects H_0, namely, that the L samples are drawn from the same population, if the two estimators differ significantly. Our first estimator to σ^2 is valid only if the hypothesis is true, namely, $\mu_j = \mu, j = 1,...,L$. If $X_{ij} \sim \mathcal{N}(\mu, \sigma^2)$, then the group average

$$m_j = \sum_{i=1}^{K} \frac{X_{i,j}}{K} \tag{3.32}$$

is also normal with mean μ and variance σ^2/K. If the hypothesis is true, then $m_j, j = 1,...,L$ are L instances drawn from $\mathcal{N}(\mu, \sigma^2/K)$. Then their mean and variance are

$$m = \frac{\sum_{j=1}^{L} m_j}{L}, \quad S^2 = \frac{\sum_j (m_j - m)^2}{L-1} \tag{3.33}$$

Thus an estimator of σ^2 is $K \cdot S^2$, namely,

$$\hat{\sigma}_b^2 = K \sum_{j=1}^{L} \frac{(m_j - m)^2}{L-1} \tag{3.34}$$

Each of m_j is normal and $(L-1)S^2/(\sigma^2/K)$ is χ^2 with $(L-1)$ degrees of freedom. Then, we have

$$\sum_j \frac{(m_j - m)^2}{\sigma^2/K} \sim \mathcal{X}^2_{(L-1)} \tag{3.35}$$

We define SS_b, the between-group sum of squares, as

$$SS_b \equiv K \sum_j (m_j - m)^2 \tag{3.36}$$

So, when H_0 is true, we have

$$\frac{SS_b}{\sigma^2} \sim \mathcal{X}^2_{(L-1)} \tag{3.37}$$

Our second estimator of σ^2 is the average of group variances, S_j^2 defined as

$$S_j^2 = \frac{\sum_{i=1}^{k}(X_{ij} - m_j)^2}{K-1} \tag{3.38}$$

and their average is

$$\hat{\sigma}_w^2 = \sum_{j=1}^{L} \frac{S_j^2}{L} = \sum_j \sum i \frac{(X_{ij} - m_j)^2}{K-1} \tag{3.39}$$

We define SS_w , the within-group sum of squares:

$$SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2 \tag{3.40}$$

Remembering that for a normal sample, we have

$$(k-1)\frac{S_j^2}{\sigma^2} \sim \mathcal{X}_{k-1}^2 \tag{3.41}$$

and that the sum of chi-squares is also a chi-square, we have

$$(k-1)\sum_{j=1}^{L} \frac{S_j^2}{\sigma^2} \sim \mathcal{X}_{L(k-1)}^2 \tag{3.42}$$

So

$$\frac{SS_w}{\sigma^2} \sim \mathcal{X}_{L(k-1)}^2 \tag{3.43}$$

Then we have the task of comparing two variances for equality, which we can do by checking whether their ratio is close to 1. The ratio of two independent chi-square random variables divided by their respective degrees of freedom is a random variable that is F-distributed, and hence when H_0 is true, we have

$$F_0 = \frac{\left(\frac{SS_b/\sigma^2}{L-1}\right)}{\left(\frac{SS_w/\sigma^2}{L(K-1)}\right)} = \frac{SS_b/(L-1)}{SS_w/(L(K-1))} = \frac{\hat{\sigma}_b^2}{\hat{\sigma}_w^2} \sim F_{L-1,L(K-1)} \tag{3.44}$$

For any given significance value α, the hypothesis that the L classification algorithms have the same expected error rate is rejected if this statistic is greater than $F_{\alpha,L-1,L(K-1)}$. Note that we are rejecting H_0 if the two estimators disagree significantly. If H_0 is not true, then the variance of m_j around m will be larger than what we would normally have if H_0 were true, and hence if H_0 is not true, the first estimator $\hat{\sigma}_b^2$ will overestimate σ^2, and the ratio will be greater than 1. For $\alpha = 0.05, L = 5$ and $K = 10, F_{0.05,4,45} = 2.6$. If X_{ij}

varies around m with a variance of σ^2, then if H_0 is true, m_j varies around m by σ^2/K. If it seems to vary more, then H_0 should be rejected because the displacement of m_j around m is more than what can be explained by some constant added noise. The name *analysis of variance* is derived from a partitioning of the total variability in the data into its components.

$$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2 \tag{3.45}$$

SS_T divided by its degree of freedom, namely, $K \cdot L - 1$ (there are $K \cdot L$ data points, and we lose one degree of freedom because m is fixed), gives us the sample variance of X_{ij} . It can be shown that the total sum of squares can be split into between-group sum of squares and within-group sum of squares

$$SS_T = SS_b + SS_w \tag{3.46}$$

Results of ANOVA are reported in an ANOVA table as shown in Tab. 3.3 This is the basic *one-way* analysis of variance where there is a single factor, for example, the learning algorithm.

Source of variation	***Sum of squares***	***Degrees of freedom***	***Mean square***	F_0
Between groups	$SS_b \equiv K\sum_j (m_j - m)^2$	$L-1$	$MS_b = \frac{SS_b}{L-1}$	$\frac{MS_b}{MS_w}$
Within groups	$SS_w \equiv K\sum_j \sum_i (X_{ij} - m_j)^2$	$L(K-1)$	$MS_w = \frac{SS_w}{L(K-1)}$	
Total	$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2$	$L \cdot K - 1$		

Table 3.3: The analysis of variance (ANOVA) table for a single factor model

We may consider experiments with multiple factors, for example, we can have one factor for classification algorithms and another factor for feature extraction algorithms used before, and this will be a *two-factor experiment with interaction*. If the hypothesis is rejected, we only know that there is some difference between the L groups but we do not know where. For this, we do the so called *posthoc testing*, that is, an additional set of tests involving subsets of groups, for example, pairs.
Fisher' s *least square difference test* (LSD) compares groups in a pairwise manner. For each group, we have $m_i \sim \mathcal{N}(\mu_i, \sigma_w^2 = MS_w/K)$ and $m_i - m_j \sim \mathcal{N}(\mu_i - \mu_j, 2\sigma_w^2)$. Then, under the null hypothesis that $H_0 : \mu_i = \mu_j$, we have

$$t = \frac{m_i - m_j}{\sqrt{2}\sigma_w} \sim t_{L(K-1)} \tag{3.47}$$

We reject H_0 in favor of the alternative hypothesis $H_1 : \mu_1 \neq \mu_2 \quad if \quad |t| > t_{\alpha/2,L(K-1)}$. Similarly, one-sided tests can be defined to find pairwise orderings.

When we do a number of tests to draw one conclusion, this is called *multiple comparisons*, and we need to keep in mind that if T hypotheses comparisons are to be tested, each at significance level α, then the probability that at least one hypothesis is incorrectly rejected is at most $T\alpha$.

For example, the probability that six confidence intervals, each calculated at 95 percent individual confidence intervals, will simultaneously be correct is at least 70 percent. Thus to ensure that the overall confidence interval is at least $100(1-\alpha)$, each confidence interval should be set at $100(1-\alpha/T)$. This is called a *Bonferroni correction.*

Note that sometimes it may happen that ANOVA rejects H_0 and none of the posthoc pairwise tests finds a significant difference.

Chapter 4

Genetic Programming

4.1 A Brief Introduction to Evolutionary Algorithms

> "A process which led from amoeba to man appeared to philosophers to be obviously a progress, though whether the amoeba would agree with this opinion is not known "
>
> B.Russel, 1914.

The dream of evolving computer programs to execute an automatic programming was born together with the computer itself. In the early 1950s, the father of computer science, Alan Turing, already imaged it, when this area of research had just a vague form. Some years later the idea rose to evolve computer programs on the basis of natural evolution and there were many efforts in this direction, noteworthy Friedberg's works [Friedberg, 1958, Friedberg et al., 1959], that can be considered an embryonic form of program evolution, or genetic programming.

More recently, the general term *Evolutionary Algorithms (EA)* has emerged to denote all the techniques tuned up for computer simulation of evolution [Banzhaf et al., 1998].
Evolutionary algorithms define a goal expressed as a quality criterion, successively they use this goal to measure and compare solution candidates in a step-by-step refinement of a set of data structures. After some iterations, if EA has success, it will return an optimal or near optimal solution.

"In this sense, the algorithms are more similar to breeding off dogs than to natural selection, since breeding also works with a well-defined quality criterion. When dogs are bred to have, for example, long hair and short legs, the breeder selects - from a group of individuals - the best individuals for reproduction according to this quality criterion. In this case, he or she

selects the ones with the longest hair and shortest legs for mating. The process is repeated with the offspring over many generations of dogs until a satisfying individual is found - and a Syberian Husky has been turned into an Angora Dachshund.... [Banzhaf et al., 1998] "

So the term *selection* indicates the process of selecting the best individuals for mating. Of course a quality criterion is necessary to determine which individuals shall be selected, and this criterion is often realized by means of a user-defined function called *fitness* in the EAs. Furthermore a technique is also needed to simulate the mating and the reproduction. Note that the variation is important for the evolution process, so a mechanism is also needed to generate a variation to the genetic material and to make sure that children do not become identical copies of their parents, to achieve the improvements. Among several variation operators in EAs, the two main ones, as in nature, are mutation and crossover of genetic material between individuals. Mutation changes a small part of an individual, while crossover (recombination or sexual reproduction) exchanges genetic material between two individuals, to create an offspring that is a combination of its parents. A scheme of a basic EA can be seen in Fig. 4.1

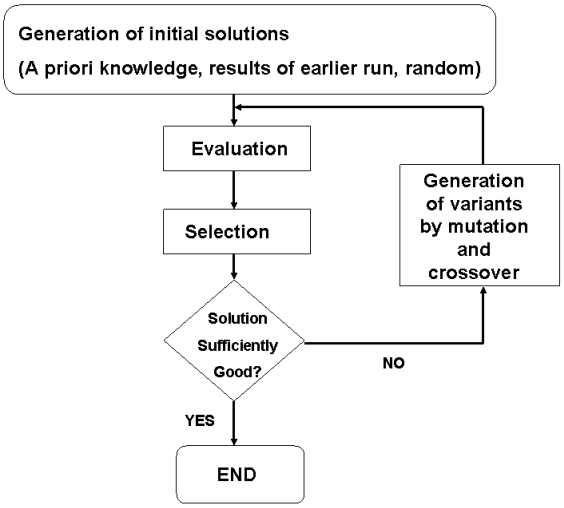


Figure 4.1: Evolutionary Algorithm. (Scheme taken from [Banzhaf et al., 1998])

4.2 Genetic Programming - basic concepts

As stated previously, in EA there is a population of individuals. The ones that survive and produce children are those that fit better the chosen quality criterion (fitness). These children are placed in the new population, instead older individuals die and are removed from it. The population is usually of fixed size and each new child replaces an existing member of the population. The individual that dies may be chosen from the relatively unfit individuals (e.g. the worst) in the population, or chosen at random or selected from the child's parents.
Most of the computer resources are devoted to deciding which individuals will have children [Langdon and Poli, 2002].

In Genetic Programming (GP), one of the EA paradigms, the evolving individuals are themselves computer programs.

Since the publication of Koza's 1992 book, there was a fast growth of GP studies, and almost six thousand GP papers have been published up to now. Researchers in the field have devised many different systems: systems that use tree, linear and graph structures, systems that use high crossover rates, systems that use high mutation rates, etc. All of these systems may reasonably be called Genetic Programming.
The purpose of the description of GP in the present work, principally due to W. Banzhaf's description [Banzhaf et al., 1998], is to give a several descriptions of these varieties by their essential common properties.
The essential characteristics shared by most GP systems are:

- **Stochastic decision making** GP emulates the randomness hypothesis of natural selection making random choices and using probability and pseudo-random numbers.

- **Program structures** GP assembles variable length program structures from basic elements called *functions* (e.g. plus, minus, sin, arctan, turn left, turn right, etc.) and *terminals* (e.g. the indipendent variables of the problem, zero-argument functions and random constants).

- **Genetic operators** GP iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. These operations include

crossover, mutation, reproduction, duplication and deletion.

- **Simulated evolution of a population by means of fitness-based selection** The process of population evolution in GP is driven by a *fitness-based selection* that determines which programs are selected for further improvements.

4.3 Preliminary Steps of Genetic Programming

Genetic programming starts from a high-level description of a problem and, transforming the problem's requirements into an appropriate language, attemps to produce a computer program that solves it.
Figure 4.2 shows the five substantial preparatory steps for the basic version of GP [Koza et al., 2003].

The first two preliminary steps specify the basic components that are available to create the computer programs. A run of GP is a competitive search among a population of possibily different programs constructed using the available functions and terminals.

The identification of the function set and terminal set for a particular problem is a simple process in most cases. In some situations, the function set may consists of merely the arithmetic functions as addition, subtraction, multiplication, and protected division. The terminal set may consist of the program's external inputs (indipendent variables), numeric constants and zero-argument functions. For many other programs, the ingredients include specialized functions and terminals, as 'turn', 'stop', 'read sensor', ecc.

The third preliminary step is preparatory of the fitness-based selection, so it indentifies the fitness measure of the problem. On the basis of this fitness measure, GP determines which programs are selected for further improvements.

The third and fourth preliminary steps are administrative. The fourth preliminary step specifies the control parameters for the run. The most influential control parameter is the population size, because it has a high effect on the result and the execution time.

The fifth preliminary step entails of specifying the termination criterion and the method of representing the result of run. The termination criterion may include a maximum number of generations to be run as well as a problem-specific success state. When the run is stopped, the single

best-so-far individual is picked up and designated as the result of the run [Koza et al., 2003].

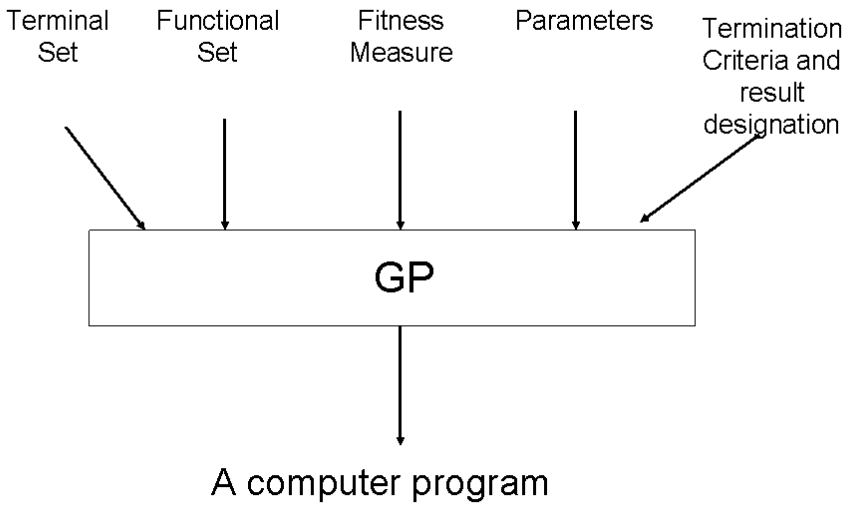


Figure 4.2: Five major GP preparatory steps

4.4 The Basic GP Algorithm

The execution steps of GP, as reported by J.Koza [Koza et al., 2003] are the following and are presented schematically in Fig. 4.3:

I. Randomly create an initial polulation (generation 0) of individuals (computer programs) created using the available functions and terminals.

II. Iteratively perform the following sub-steps (called a *generation*) on the population until the termination criterion is satisfied:

II. 1 Execute each program in the population and determine its fitness (explicitly or implicitly) using the problem's fitness measure.

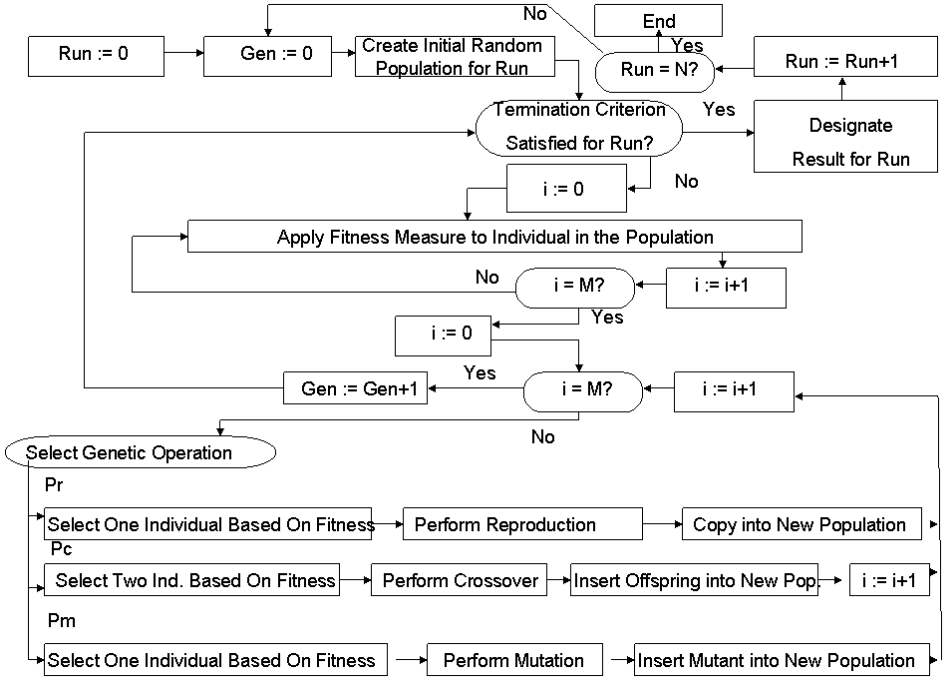


Figure 4.3: GP Flowchart. (Scheme inspired from [Koza et al., 2003])

II. 2 Select one or two individuals from the population with a probability based on fitness to participate in the genetic operations in (c).

II. 3 Create new individual program(s) for the population by applying the following genetic operations with specified probabilities.

i. *Reproduction*: Copy the selected individual to the new population.

ii. *Crossover*: Create new offspring program(s) for the new population by recombining randomly chosen parts from two selected programs.

iii. *Mutation*: Create one new offspring program for the new population by randomly mutating a randomly chosen part of one selected program.

III. After the termination criterion is satisfied, the single best program in the population during the run (the best-so-far individual) is picked up

and designated as the result of the run. If the run is successful, the result may be a solution (or approximate solution) to the problem.

4.5 Representation of GP Individuals: Terminals and Functions

The functions and terminals, as mentioned above, are the primitives with which a program in GP is built. They play different roles. Loosely speaking, terminals *provide* a value to the system and so they are usually either variables, constants or zero-argument functions, while functions *process* a value already in the system and so the set of functions which is chosen includes those functions which are a priori believed to be useful for the problem at hand [Banzhaf et al., 1998].

Moreover, functions and terminals are the primitives of GP but are not themselves programs. They must be assembled into a *structure* before they may be executed as programs. The three principal program structures used in GP are tree, linear and graph structures. The choice of a program structure in GP affects execution order, use and locality of memory, and the application of genetic operators to the program. Of the three fundamental structures, tree structures are the most common in GP. Therefore this representation is the one used in this work and GP using this representation is called *tree-based GP* from now on.

Hence, the set of all the possible structures that tree-based GP can generate is the set of all the possible trees that can be built recursively from a set of function symbols $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ (used to label internal tree nodes) and a set of terminal symbols $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ (used to label tree leaves). More formally:

Definition 4.1. *The **terminal set** is composed by the inputs to the GP programs, the constants supplied to the GP programs, and the zero-arguments functions with side-effects executed by the GP program.*

In practice, terminals are inputs to the programs, i.e. they are constants or functions without argument, so they return an actual numeric value without having to take an input. For this reason they are called *terminals*, because in tree-based GP they terminate a branch of a tree.
Note that in typical tree-based GP, a set of real-numbered constants is chosen for the entire population at the beginning of the run. These constants do

not change their value during the run. They are called *ephemeral random constants*, frequently represented by the symbol $\Re$. Other constants may be constructed within programs by combining ephemeral random constants using arithmetic functions.

Definition 4.2. *The **function set** is composed of the statements, operators and functions available to the GP system.*

Each function in the function set $\mathcal{F}$ takes a fixed number of arguments, specifying its *arity*.

Definition 4.3. *The **arity** of a function is the number of inputs to (arguments of) that function.*

The collection of available functions is very wide:

- **Boolean Functions** e.g. *and, or, not, xor*
- **Arithmetic Functions** e.g. *plus, minus, multiply, divide*
- **Trascendental Functions** e.g. *trigonometric* and *logarithmic functions*
- **Variable Assignment Functions** e.g. Let a be a variable available to the GP system, $a := 1$ would be a variable assignment function in a register machine code approach. The same function would appear in a tree-based system with an expression that somehow lookes like this: (ASSIGN a 1), where 1 is an input to the ASSIGN node. Of course, there should be a corresponding READ node, which would read whatever value was stored in a and pass it along as its output.
- **Index Memory Functions** Some GP systems use indexed memory, i.e. access to memory cells via an index.
- **Conditional Statements** e.g. *if-then-else; case, switch*
- **Control Transfer Statements** e.g. *go to, call, jump*
- **Loop Statements** e.g. *while...do, repeat...until, for...do*
- **Subroutines**

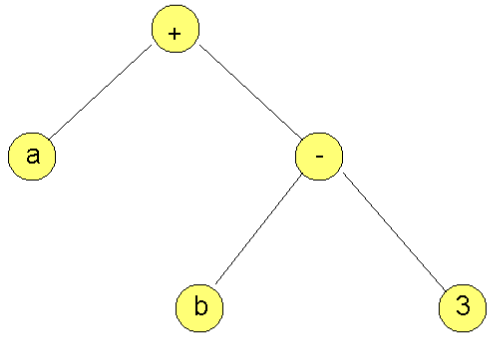


Figure 4.4: A tree that can be built with the sets $\mathcal{F} = \{+, -\}$ and $\mathcal{T} = \{a, b, 3\}$.

For example, given the following sets of functions and terminals:

$$\mathcal{F} = \{+, -\}, \quad \mathcal{T} = \{a, b, 3\}$$

a valid GP individual is represented in Fig. 4.4.
This tree can also be represented by the following LISP-like S-expression (for a definition of LISP S-expressions see, for instance, [Koza, 1992]):

$$(+\ a\ (-\ b\ \ 3))$$

Figure 4.4 shows the graphical representation of a tree-based GP individual. It has some different symbols that could be executed in any order. But there is a convention for executing the tree structure.
The standard convention for tree execution is that it proceeds by repeatedly evaluating the leftmost node for which all inputs are available. This order of execution is referred to as *postfix order* because the operators appear after the operands. Another convention for execution is called *prefix order*. It is the opposite of postfix order and executes the nodes close to the root of the tree before it executes the terminal nodes. The advantage of prefix ordering is that a tree containing nodes like *if-then* branches can often save execution time by evaluating first whether the *then* tree must be evaluated.

4.5.1 The Choice of Functions and Terminals

The set of functions may be much wider then the previous list. It may use any programming construct that is available in any programming language.

However, as stated above, GP assembles variable length program structures from basic components (functions and terminals) using randomness in several phases of the process. So, without any restriction, the syntax of most languages is such that GP operators would create a large percentage of syntactically incorrect programs.

For this reason, Koza chose a syntax in prefix form analogous to LISP and a restricted language with an appropriate number of variables, constants and operators defined to fit the problem to be solved [Koza, 1992]. The restricted language, as said, is formed by a user-defined *function set* F and *terminal set* T. The function chosen are those a priori believed to be useful for the problem at hand, and the terminals are usually either variables or constants. In addition, each function in the function set must be able to accept as arguments any other function return value and any data type in the terminal set T. Thus the space of possible programs is constituted by the set of all possible compositions of functions that can be recursively formed from the elements of F and T. This property is called *syntactic closure*. In such way syntax constraints are respected and the program search space is limited [Tettamanzi and Tomassini, 2001].

The function set usually contains math and logical functions such as **plus, minus, sin, cos, and, or, etc.**, as said, but it is usefull to remark that it may be also application-specific and be selected according to the problem domain. Any function that a programmer can dream of may become part of the function set in GP. For example, in a robotic application, primitives could be created by the programmer that are specific to the problem, such as **read sensor, turn left, turn right, move ahead**, ecc. Each of those primitives would become part of the function set or of the terminal set, if its arity were 0. The choice to fit tightly the function set in GP often reduces the need for pre- and post- processing.

Anyway, even if the functions and terminals used for GP should be powerful enough to be able to represent a solution to the problem, often it is not convenient to use too large function sets. This enlarges the search space and can sometimes make the search for a solution more difficult. A good starting point for a function set might be the set of arithmetic and logic functions (+,

-, *, /, OR, AND, XOR,...). The number of problems that can be solved with these functions is surprising. Effective solutions using only this function set have been obtained on several different classification problems, robotics control problems and symbolic regression problems [Banzhaf et al., 1998] (for more details on symbolic regression see the paragraph 4.8.1). A parsimonious approach to choosing a function and terminal set is often reasonable. The practice often shows that GP is very ingenious at taking simple functions and creating what it needs by combining them. It frequently ignores the more sophisticated functions in favor of the primitives during evolution [Banzhaf et al., 1998].

4.6 Inizialization of a GP Population

The first step in actually performing a GP run is to initialize the population. It means creating a variety of program structures for successive evolution. One of the main parameters of a GP run is the maximum size permitted for a program. For tree-based GP, that parameter is expressed as the maximum depth of a tree or the maximum total number of nodes in the tree.

Definition 4.4. *The **depth** of a node is the minimal number of nodes that must be traversed to get from the root node of the tree to the selected node.*

Consequently, the MDP(Maximum Depth Parameter) indicates the largest depth that will be permitted between the root node and the outermost terminals in an individual.

4.6.1 Initializing Tree Structure

The most common initialization methods in tree-based GP are the *grow* method, the *full* method and the *ramped half and half* method [Koza, 1992]. As stated above, trees are built from basic units, called functions and terminals. We shall assume, now, that the functions and terminals allowable in the program trees have been already selected:

$$\mathcal{F} = \{+, -, *\}, \quad \mathcal{T} = \{a, b, c, d, e\}$$

so that we shall use them to describe the most frequently used tree initialization methods.

Grow Initialization

In the grow initialization of a tree, nodes are selected randomly from the function and the terminal set throughout the entire tree (except the root node which uses only the function set). Once a branch contains a terminal node, that branch is complete. The steps of grow initialization algorithm are:

- a random symbol is selected with uniform probability from $\mathcal{F}$ to be the tree root;
- let n be the arity of the selected function symbol. Then n nodes are selected randomly with uniform probability from the set $\mathcal{F} \cup \mathcal{T}$ to be its children;
- for each function symbol among these n nodes, the method is recursively applied, i.e. its children are selected randomly from the set $\mathcal{F} \cup \mathcal{T}$, unless this symbol has a depth equal to $d-1$, in that case its children are selected from $\mathcal{T}$.

Note that this method produces trees of irregular shape, as the tree in Fig. 4.5 . Nodes with depth between 1 and $d-1$ are selected randomly with uniform probability from $\mathcal{F} \cup \mathcal{T}$, but once a branch contains a terminal node, that branch is complete, even if the maximum depth d has not been reached. In fact, in the tree of Fig. 4.5 some branches (a, b and e), have a depth of only three, while some others (c and d) have a depth of four. Moreover it is noteworthy that the root is selected with uniform probability from $\mathcal{F}$ and not from $\mathcal{F} \cup \mathcal{T}$, to avoid creating trees composed by a single node.
If the number of nodes is used as a size measure, growth stops when the tree has reached the preset maximum size, forcing the next nodes to be all terminals.

Full Initialization

Instead of selecting nodes randomly from the union of the function and terminal sets, as the grow initialization method, the full method chooses only function symbols until the maximum depth is reached. Then it chooses only terminals. The result is that every branch of the tree goes to the

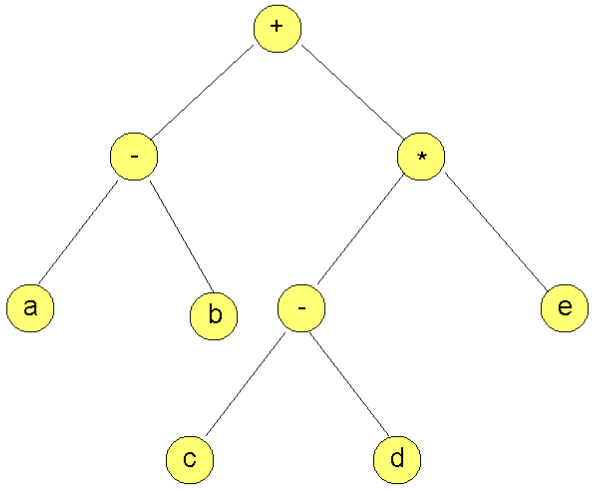


Figure 4.5: Example of maximum depth tree initialized with Grow Method

full maximum depth, as we can see in the tree of Fig. 4.6, that has been initialized with the full method having a maximum depth of three.

Ramped Half-and-Half Initialization

Diversity is useful in GP populations. As first noted by Koza [Koza, 1992], each of the above methods, by itself, could result in a set of similar structures in the initial population, because the routine is the same for all individuals. So the ramped half-and-half technique has been conceived in order to enhance population diversity from the outset.
Let d be the maximum depth parameter. The population is divided equally among individuals to be initialized with trees having depths equal to 1, 2, ..., $d-1$, d. For each depth group, half of the trees are initialized with the full technique and half with the grow technique.

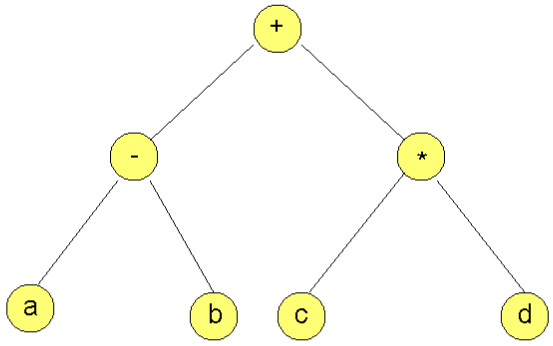


Figure 4.6: Example of tree initialized with Full Method

4.7 Genetic Operators of GP

The individuals of an initial population usually have very low fitness. Evolution proceeds by transforming the initial population by the use of genetic operators.
The three principal ones, are:

- Crossover
- Mutation
- Reproduction

4.7.1 Crossover

The crossover operator produces new offspring creating variation in the population. It combines the component of two parents by swapping a part of one parent with a part of the other. More specifically, standard tree-based GP crossover proceeds by the following steps:

I. Choose two individuals as parents. An example of two parents is shown in Figure 4.7(a)

II. Select one random point in each parent, which takes the name of *crossover point* for the parents. The selection of subtrees can be biased so that subtrees constituting terminals are selected with lower probability than other subtrees, so usually it is used a probability of 0.1 for terminals and 0.9 for the other nodes.

III. Swap the selected subtrees between the two parents. Figure 4.7(b) showes the two resulting children trees.

Because entire subtrees are swapped and because of the closure property of the functions, crossover always produces syntactically legal programs.
It is important to remark that in cases where a terminal and/or the root of one parent are located at the crossover point, generated offspring could have considerable depths. This may be one possible cause for the phenomenon of *bloat* [Vanneschi, 2004], which is a progressive growth of the code size of individuals in the population without a corresponding improvement in fitness. For this reason, many variants of the standard GP crossover have been proposed in literature. The most common ones select the root with lower probability than the leaves, for the reason described above, to limit the occurence of such degenerative phenomena. Another kind of GP crossover is *one-point crossover*, introduced in [Poli and Langdon, 1997, Poli and Langdon, 1998] and not described in this work, for brevity reasons.

4.7.2 Mutation

Mutation operates on only one individual. Standard GP mutation is often called *subtree mutation*. When an individual has been selected for mutation, a point in the tree is randomly choosen with a uniform probability distribution, and the existing subtree at that point is replaced by a new randomly-generated subtree. The new randomly-generated subtree is created in the same way and is subject to the same limitations, on depth or size, as programs in the initial random population. The altered individual is then placed back into the population (see Figure 4.8)

As for crossover, researchers have developed many alternatives of standard GP mutation. The most commonly used are the ones aimed at limiting

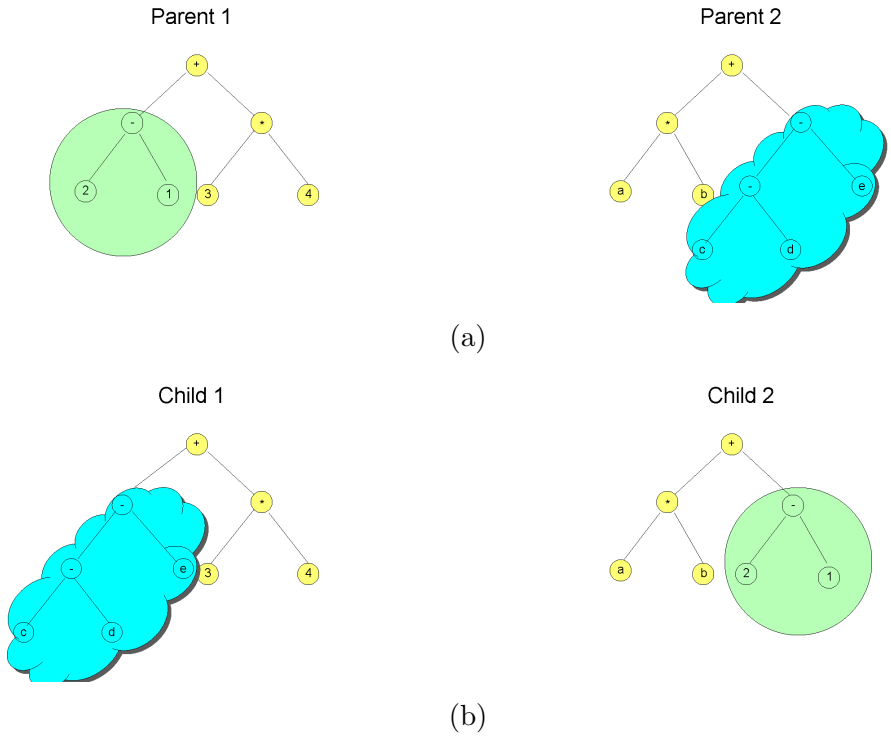


Figure 4.7: Example of crossover operator, with parents on the first level (a) and children on the second level (b)

the probability of selecting the root and/or the leaves of the parent as mutation points. A special mention is deserved by the *point mutation*, that exchanges a single node with a random node of the same arity, for the importance it has had in GP theory [Poli and Langdon, 1997]. A new kind of GP mutation, similar to, but more general than one-point mutation, called *structural mutation* has been introduced in chapter 4 of [Vanneschi, 2004]. Other variants of GP mutation that are often used are *permutation* (also named *swap* mutation) and *shrink mutation*. The first one exchanges two arguments of a node and the second one generates a new individual from a parent's subtree.

4.7.3 Reproduction

The reproduction operator is very simple. It selects an individual, makes a copy of it and places the copy into the new population.

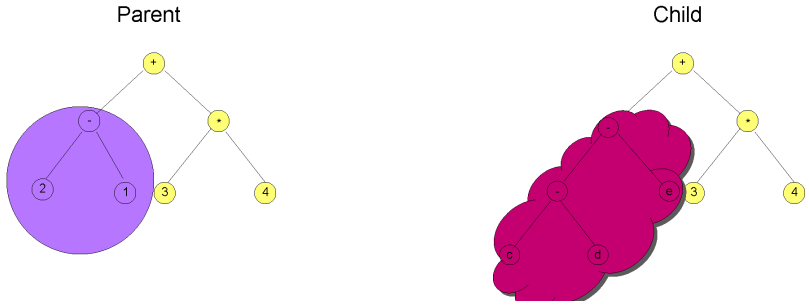


Figure 4.8: Example of mutation operator

4.8 Fitness in GP

In order to create variability and evolution in the population, as mentioned before, GP chooses some members of the population that will be subject to the genetic operators (crossover, mutation and reproduction). In this phase, one of the most important parts of GP's model of evolution is implemented: the *fitness based selection* [Banzhaf et al., 1998]. GP's evaluation metric is called a *fitness function* and the way in which the fitness function is used in the selection of individuals for genetic operators may be referred to as the GP *selection algorithm*.
The aim of performing a fitness evaluation is to have feedback to the learning algorithm regarding which individuals should have higher probability of being allowed to multiply and reproduce and which individuals should have higher probability of being removed from the population. So, each program in the population is assigned a fitness value, representing its ability to solve the problem. The two fitness measures most commonly used in GP are raw fitness and standardized fitness. They are described below.

Definition 4.5. ***Raw Fitness*** *is the measurement of fitness that is stated in the natural terminology of the problem itself*

This definition is stated by [Koza, 1992]. In other words, raw fitness is the most simple and natural way to calculate the ability of a program to solve a problem.

Often raw fitness is calculated over a set of *fitness cases*. A fitness case represents an element of the learning domain for which the ability of the program to evaluate it can be estimated.
The fitness cases are typically a small sample of the entire domain space

and they form the basis for generalizing the results obtained to the entire domain space.
In order to define raw fitness more formally, let the output of the ith fitness case from the learning domain be o_i. Let the output of a GP program p applied to the ith fitness case from the learning domain be p_i. In this case for a learning domain of n examples the raw fitness f_p of program p would be:

$$f_p = \sum_{i=1}^{n} |p_i - o_i|^k \tag{4.1}$$

Because raw fitness is stated in the natural terminology of the problem, the better value may be either smaller (as when raw fitness is error) or larger (as when raw fitness is benefit achieved) [Koza, 1992].
Anyway, in some cases, it could be useful to have the best value of fitness always equal to zero. This kind of fitness is known as *standardized fitness*.

Definition 4.6. ***Standardized Fitness*** *is a fitness function or a transformed fitness function for which zero is the best possible value.*

If the fitness chosen for a specific problem has not this characteristic, it can be obtained by subtracting or adding a constant. If a greater value of raw fitness corresponds to a better individual (maximization problem), and the maximum possible value of fitness f_{max} is known, standardized fitness f^S of an individual i can be defined as:

$$f_i^S = f_{max} - f_i \tag{4.2}$$

where f_i is the fitness of i.

Instead, if a lesser value of raw fitness is better (minimization problem), and the minimum possible value of fitness f_{min} is known and is grater than zero, in this case standardized fitness f^S of an individual i can be defined as:

$$f_i^S = f_i - f_{min} \tag{4.3}$$

where f_i is the fitness of i.

In some cases it could be useful also to have the values of the fitness function comprised in a predetermined interval, for instance the interval between zero and one. In these cases it is enough to normalize the fitness function.

Definition 4.7. ***Normalized Fitness*** *is a fitness function or a transformed fitness function where fitness is always between zero and one.*

4.8.1 Symbolic Regression

Where the learning domain is composed by numeric inputs and outputs, the process of approximating the function that, with the given inputs, returns the desired corresponding outputs is called *symbolic regression*. Many GP applications can be reformulated as instances of symbolic regression, as the survival prediction in breast cancer and the reverse engineering of gene regulatory networks, cases studied in this PhD thesis and described in the last chapters.

For example, suppose that GP is used to find a function satisfying the fitness cases in Table 4.1 - that is, a program that could predict the output column from the values in the input column.
This example is very simple, in fact a program representing the function $f(x) = x^2 + 1$ would be a perfect match on this learning domain.

	INPUT	OUTPUT
Fitness case 1	1	2
Fitness case 2	3	10
Fitness case 3	4	17
Fitness case 4	9	82
Fitness case 5	12	145

Table 4.1: Example of a set of possible fitness cases for a one-dimensional symbolic regression problem

Let the output of the ith example from the training set be o_i. Let the output of a GP program p applied to the ith example of the training set be p_i. For this domain of 5 examples the fitness f_p of program p would be:

$$f_p = \sum_{i=1}^{5} |p_i - o_i|^k \tag{4.4}$$

One of the most used version of this fitness function is to calculate the sum of the squared differences between p_i and o_i, called *squared error* and

stated in the equation 4.5 for a generic domain of n examples.

$$f_p = \sum_{i=1}^{n} (p_i - o_i)^2 \tag{4.5}$$

Note that all these fitness functions are continous. As p_i gets a little closer to o_i, the fitness gets a little better. They are also standardized because the fitness of a perfect solution, like $f(x) = x^2 + 1$ in the above example, would be equal to zero.

4.9 Selection in GP

After the quality of an individual has been evaluated using a fitness function, it is necessary to decide whether to apply genetic operators to that individual and whether to keep it in the population or replace it. This process is called *selection* and is performed by a proper operator, the selection operator. This step is determinant in the GP process because it is responsible for the speed of evolution and is often cited as the main cause for premature convergence, which may prevent the success of an evolutionary algorithm. For this reason, many selection algorithms have been developed.

4.9.1 Roulette-Wheel or Fitness Proportional Selection

Fitness-proportional selection is employed in a GP scenario for generational selection and specifies probabilities for individuals to be given a chance to pass offspring into the next generation. Let N be the number of individuals belonging to a population P and $\{f_1, f_2, ..., f_N\}$ be the set of their fitness values. Each individual i in P is given a probability of:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{4.6}$$

to pass on traits by being applied genetic operators. While candidate solutions with a higher fitness will be more likely to be selected, there is still a chance that they are not. This contrasts with a less sophisticated selection algorithm, such as truncation selection, which "eliminates" a fixed percentage of the weakest candidates. So, with fitness proportionate selection, there is a chance some weaker solutions may survive the selection process; this is

an advantage as, although a solution may be weak, it may include some component which could prove useful following the recombination process.

The analogy to a roulette wheel can be visualised by imagine a roulette wheel in which each candidate solution represents a pocket on the wheel; the size of the pockets are proportionate to the probability of selection of the individual to which they are associated. So, selecting N individuals from the population is equivalent to playing N games on the roulette wheel, as each candidate is drawn independently. The resulting population is called *mating pool*.

This selection algorithm, although widely used in GP, has been criticized for attaching differential probabilities to the absolute values of fitness [Blickle and Thiele, 1995]: if the difference between individuals with better fitness and those with worse fitness in a population is high, it is possible that the first ones will be selected in many copies, which increases the *selection pressure*, decreasing the population diversity. Early remedies for this situation were introduced through fitness scaling, a method by which absolute fitness values were adapted to the population average [Grefenstette and Baker, 1989], and several other methods; for details see [Koza, 1992].

4.9.2 Ranking selection

Ranking selection [Grefenstette and Baker, 1989] was proposed to reduce the possible predominant effects of high fitness individuals in the fitness proportionate selection.
It is based on fitness order, by which individuals are sorted. The selection probability is assigned to individuals as a function of their position in the ordered population. Types of ranking used for the ordering are principally linear and exponential ranking.
Let be p^-/N the probability of the worst individual being selected, and p^+/N the probability of the best individual being selected ($p^- + p^+ = 2$ in order for the population size to stay constant, in the case of the selection of the survival individuals). Then the probability for *linear ranking* is a linear function of the rank:

$$p_i = \frac{1}{N}\left[p^- + (p^+ - p^-)\frac{i-1}{N-1}\right] \tag{4.7}$$

Instead the probability for the exponential ranking can be computed using a selection bias constant c, with $0 < c < 1$

$$p_i = \frac{c-1}{c^{N-1}} c^N - i \tag{4.8}$$

These are the most commonly used ranking functions.

The drawback of this technique is the opposite as the one of fitness proportionate selection: it amplifies the difference between closely clustered fitness values so that the better ones can be sampled more frequently [Whitley, 1989].

4.9.3 Tournament selection

Tournament selection is not based on competition within the full generation but in several subsets of the population. A number of individuals, called *tournament size*, is selected randomly and the ones with better fitness are selected. The traits of the better individuals in the tournament are then allowed to replace those of the worse individuals. In the smallest possible tournament, two individuals compete. The better of the two is allowed to reproduce with mutation. The result of that reproduction is returned to the population, replacing the loser to the tournament. The tournament size allows practitioners to adjust selection pressure. A small tournament size causes a low selection pressure, and a large tournament size causes high pressure [Banzhaf et al., 1998]. This method is widely used in GP mainly because it does not require a centralized fitness comparison between all individuals. This saves computational time and also provides an easy way to parallelize the algorithm.

4.10 GP Parameters

In paragraph 4.3 we have stated that one of the preliminary steps in a GP run is the definition of control parameters. The fundamental ones are:

- Population size
- Maximum individual size
- Initialization of population
- Selection method

- Crossover probability
- Mutation type and rate
- Termination criterion

The setting of each of these parameters represents in general a crucial choice for the performance of the GP system. Much of what GP researchers know about them is empirical and based on experience. Following in this work some choices are presented for two problem-specific implementations, a survival prediction in breast cancer and a reverse engineering of gene regulatory networks.

PART II

MEDICAL DECISION SUPPORT SYSTEM

Chapter 5

Medical Decision Support System For Survival Prediction in Breast Cancer using a binary dataset

5.1 A first glance

Current cancer therapies have serious side effects: ideally type and dosage of the therapy should be matched to each individual patient based on his/her risk of relapse. Therefore the classification of cancer patients into risk classes is a very active field of research, with direct clinical applications. Until recently patient classification was based on a series of clinical and histological parameters. The advent of high-throughput techniques to measure gene expression led, in the last decade, to a large body of research on gene expression in cancer, and in particular on the possibility of using gene expression data to improve patient classification. A *gene signature* is a set of genes whose levels of expression can be used to predict a biological state (see [Nevins and Potti, 2007]): in the case of cancer, gene signatures have been developed both to distinguish cancerous from non-cancerous conditions and to classify cancer patients based on the aggressiveness of the tumor, as measured for example by the probability of relapsing within a given time.

While many studies have been devoted to the identification of gene signatures in various types of cancer, the question of the algorithms to be used to maximize the predictive power of a gene signature has received less attention. To investigate this issue systematically, one of the best established gene signatures has been considered: the 70-gene signature for breast cancer [van't Veer et al., 2002]. The performance of four different machine learning algorithms in using this signature to predict the survival of a cohort of breast cancer patients is compared in the present chapter. The 70-gene signature is a set of microarray features selected in [van't Veer et al., 2002]

based on correlation with survival, on which the molecular prognostic test for breast cancer "MammaPrint"^{TM}is based. In the field of machine learning algorithms used to classify cancer samples based on gene expression data [Chu and Wang, 2005, Deb and Reddy, 2003, Deutsch, 2003, Langdon and Buxton, 2004, Paul and Iba, 2005, Yu et al., 2007], is inserted the Medical Decision Support System (GP-MDSS) presented in this thesis. It was developed with Genetic Programming using the GP toolbox GPLab [Silva, 2007]. Also a systematic comparison of its performance with three other machine learning algorithms is shown using the same features to predict the same classes. In such a comparison, feature selection is thus *not* explicitly performed as a pre-processing phase before executing the machine learning algorithms[1]. It is compared with Support Vector Machines, Multilayer Perceptrons and Random Forests, which are applied to the problem of using the 70-gene signature to predict the survival of the breast cancer patients included in the NKI dataset [van de Vijver et al., 2002]. This is considered one of the gold-standard datasets in the field, and the predictive power of the 70-gene signature on these patients was already shown in [van de Vijver et al., 2002]. In this preliminary study all the methods are used in an "out-of-the-box" version so as to obtain a first evaluation, as unbiased as possible, of the performance of the methods.

5.2 Previous and Related Work

Many different machine learning methods [Lu and Han, 2003] have already been applied for microarray data analysis, like k-nearest neighbors [Michie et al., 1994], hierarchical clustering [Alon et al., 1999], self-organizing maps [Hsu et al., 2003], Support Vector Machines [Guyon et al., 2002, Hernandez et al., 2007] or Bayesian networks [Friedman et al., 2000]. Furthermore, in the last few years Evolutionary Algorithms (EA) [Holland, 1975] have been used for solving both problems of feature selection and classification in gene expression data analysis. Genetic Algorithms (GAs) [Goldberg, 1989] have been employed for building selectors where each allele of the representation

[1] As discussed later, the system presented is the only method, among the ones studied in this chapter, that is able to perform automatically a further feature selection and thus identify small subsets of the original signature characterized by high predictive power. That happens thanks to the intrinsic characteristic of GP.

corresponds to one gene and its state denotes whether the gene is selected or not [Liu et al., 2005]. GP on the other hand has been shown to work well for recognition of structures in large data sets [Moore et al., 2001]. GP was applied to microarray data to generate programs that reliably predict the health/malignancy states of tissue, or classify different types of tissues. An intrinsic advantage of GP is that it automatically selects a small number of feature genes during evolution [Rosskopf et al., 2007]. The evolution of classifiers from the initial population seamlessly integrates the process of gene selection and classifier construction. In fact, in [Yu et al., 2007] GP was used on cancer expression profiling data to select potentially informative feature genes, build molecular classifiers by mathematical integration of these genes and classify tumour samples. Furthermore, GP has been shown to be a promising approach for discovering comprehensible rule-based classifiers from medical data [Bojarczuk et al., 2001] as well as gene expression profiling data [Hong and Cho, 2006]. The results presented in those contributions are encouraging and pave the way to a further investigation of GP for this kind of datasets, which is the goal of this part of the work.

5.3 Computational Methods

The machine learning methods considered in this work are roughly depicted in this section, whereas GP, the core of the system, is described in details in Chapter 4.

5.3.1 Genetic Programming

The most common version of GP, and also the one used here, considers individuals as *syntax tree* structures[2] that can be built recursively from a set of function symbols $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ (used to label internal tree nodes) and a set of terminal symbols $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ (used to label tree leaves). GP breeds these solutions to solve problems by executing an iterative process involving the probabilistic selection of the fittest solutions and their variation by means of a set of genetic operators.

In this work a tree-based GP configuration inspired by boolean problems and introduced in [Koza, 1992] is used: each feature in the dataset was

[2]Traditionally represented in Lisp notation.

represented as a boolean value and thus the set of terminals $\mathcal{T}$ was composed by 70 boolean variables (i.e. one for each feature of our dataset). Potential solutions (GP individuals) were built using the set of boolean functions $\mathcal{F} = \{AND, OR, NOT\}$. The fitness function is the number of incorrectly classified instances, which turns the problem into a minimization one (lower values are better)[3].

Finally no explicit feature selection strategy was employed, due to the GP's ability to automatically perform an implicit feature selection. The mechanism allowing GP to perform feature selection, is simple: GP searches over the space of all boolean expressions of 70 variables. This search space includes the expressions that use *all* the 70 variables, but also the ones that use a *smaller* number of variables. In principle there is no reason why an expression using a smaller number of variables could not have a better fitness value than an expression using all the 70 variables. If expressions using smaller number of variables have a better fitness, they survive and reproduce, given that fitness is the only principle used by GP for selecting genes.
This mechanism is already pointed out in [Vanneschi et al., 2010, Farinaccio et al., 2009, Archetti et al., 2006, Archetti et al., 2007a, Archetti et al., 2007b, Rosskopf et al., 2007]. The first two papers contain a first version of the work presented in this chapter.

The parameters used in this work are reported in Table 6.1, together with the parameters used by the other machine learning methods studied. There is no particular justification for the choice of those parameter values, if not the fact that they are standard for the GP Matlab tool [Silva, 2007] based on which GP-MDSS is constructed.

5.3.2 Support Vector Machines

In this work the Support Vector Machine implementation of John Platt [Platt, 1998] sequential minimal optimization (SMO) algorithm for training the support vector classifier is used. SMO works by breaking the large Quadratic Programming (QP) problem into a series of smaller 2-dimensional sub-problems that may be solved analytically, eliminating

[3]In case of minimization problems, the term "fitness" might be inappropriate, given that a fitness is usually a measure that has to be maximized. Nevertheless, this term is used for simplicity.

the need for numerical optimization algorithms such as conjugate gradient methods. The implementation used is the one contained in the Weka public domain software [Weka, 2006]. This implementation globally replaces all missing values and transforms nominal attributes into binary ones. It also normalizes all attributes by default (in that case the coefficients in the output are based on the normalized data, not the original data and this is important for interpreting the classifier). Neverthless, in this work this normalization is not used, because binary data and targets are directly passed to the Weka software, as explained in Section 5.4

The main parameter values used in this work are reported in Table 6.1. All these parameter values correspond to the standard values offered by the Weka software [Weka, 2006] and they are defined for instance in [Platt, 1998].

5.3.3 Multilayer Perceptron

The Multilayer Perceptron implementation adopted in this work is the one included in the Weka software distribution [Weka, 2006]. The Back-propagation learning algorithm [Haykin, 1999] was used and the values used for all the parameters are reported in Table 6.1. As for the previously discussed machine learning methods, also in the case of the Multilayer Perceptron it is important to point out that the parameter setting used is as standard as possible, without doing any fine parameter tuning for this particular application. The goal is, in fact, to compare different computational methods under standard conditions and not to solve in the best possible way the problem itself. In particular, all the values reported in Table 6.1 correspond to the default ones adopted by the Weka software.

5.3.4 Random Forests

In this work the Breiman model presented in [Breiman, 2001] and implemented in the Weka software [Weka, 2006] has been used. As can be seen from Table 6.1, this method, compared to the other ones, has the advantage of a smaller amount of parameter setting required. In order to allow a fair comparison with GP-MDSS, random forests composed by 2500 trees were considered (given that the GP population is composed by 500 trees and it runs for 5 generations, 2500 trees are globally inspected by GP too) such

that each node corresponds to exactly one feature (as it is for GP-MDSS). All the other parameters were set to the standard values used by the Weka software.

GP Parameters	
population size	500 individuals
population initialization	ramped half and half [Koza, 1992]
selection method	tournament (tournament size = 10)
crossover rate	0.9
mutation rate	0.1
maximum number of generations	5
algorithm	generational tree based GP with no elitism
SVM Parameters	
complexity parameter	0.1
size of the kernel cache	10^7
epsilon value for the round-off error	10^{-12}
exponent for the polynomial kernel	1.0
tolerance parameter	0.001
Multilayered Perceptron Parameters	
learning algorithm	Back-propagation
learning rate	0.03
activation function for all the neurons in the net	sigmoid
momentum	0.2 progressively decreasing until 0.0001
hidden layers	(number of attributes + number of classes) / 2
number of epochs of training	500
Random Forest Parameters	
number of trees	2500
number of attributes per node	1

Table 5.1: Parameters used in the experiments.

5.4 Validation Dataset

The NKI breast cancer dataset [van de Vijver et al., 2002] used in this work provides gene expression and survival data for 295 breast carcinoma patients. Only the expression data for the genes included in the "70-gene" signature [van't Veer et al., 2002] were considered.

Both survival and gene expression data were transformed into binary form. For the survival data, the outcome is defined as the survival status of the patient at time $t_{end} = 10.3$ years. By choosing this particular endpoint the number of dead and alive patients is balanced: out of 148 patients for which the status at t_{end} is known, 74 were dead and 74 were alive. Binary expression data were obtained by replacing all positive logarithmic fold changes in the

original dataset with 1 and all negative and missing ones with 0.

The dataset is a matrix $H = [H_{(i,j)}]$ of binary values composed by 148 rows (instances) and 71 columns (features), where each line i represents the gene signature of a patient whose binary target ($0 =$ survived after t_{end} years, $1 =$ dead for breast cancer before t_{end} years) has been placed at position $H_{(i,71)}$. In this way, the last column of matrix H represents all the known target values.

The task is now to generate a mapping F such that

$$F(H_{(i,1)}, H_{(i,2)}, ..., H_{(i,70)}) = H_{(i,71)}$$

for each line i in the dataset. Of course, we also want F to have a good generalization ability, i.e. to be able to assess the target value for *new* patients who have not been used in the training phase. For this reason, a set of machine learning techniques is used, as discussed in Section 5.3.

Each computational method was run 50 independent times. For each run the dataset was randomly partitioned into a training and a test set: 70% of the patients are randomly selected with uniform probability and inserted into the training set, while the remaining 30% form the test set.

5.5 Experimental Results

Table 5.2 summarizes the results returned by each machine learning method on the 50 runs. The first line indicates the different methods, the second line shows the best (i.e. lowest) value of the incorrectly classified instances obtained on the test set over the 50 runs, and the third line reports the mean performances of each group of 50 runs on their test sets, together with the corresponding standard error of mean (SEM).

As Table 5.2 clearly shows, the best solutions were found by GP-MDSS and Multilayer Perceptrons and the best average result was found by GP-MDSS. Moreover, statistical analysis indicates that GP-MDSS consistently outperforms the other methods except SVM using polynomial kernel with degree 1. In fact, as can be seen in Table 5.3, the difference between the various average results is statistically significant (P-value 3.05×10^{-5} for ANOVA test on the 4 samples of solutions found by each method). Finally, pairwise 2-tailed Student t-tests comparing GP-MDSS with each other method demonstrates its general better performance. These statistical tests were performed since there was no evidence of deviation from normality or

	GP-MDSS	**SVM-K1**	**SVM-K2**	**SVM-K3**	**MP**	**RF**
best	10	13	14	15	10	12
average (SEM)	16.40 (0.30)	18.32 (0.37)	16.76 (0.18)	17.62 (0.17)	18.08 (0.39)	17.60 (0.35)

Table 5.2: Experimental comparison between the number of incorrectly classified instances found on the test sets by the different machine learning methods. Each method was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method: Medical Decision Support System based on GP (GP-MDSS), Support Vector Machine with exponent for the polynomial kernel 1.0 (SVM-K1), 2.0 (SVM-K2), and 3.0 (SVM-K3), Multilayer Perceptrons (MP), and Random Forest (RF). The second line shows the best value of the incorrectly classified instances obtained on the test set over the 50 runs, and the third line reports the average performances of each group of 50 runs on their test sets (standard error of mean is shown in parentheses).

unequal variances.

ANOVA $P = 3.05 \times 10^{-5}$				
GP-MDSS vs. SVM-K1	GP-MDSS vs. SVM-K2	GP-MDSS vs. SVM-K3	GP-MDSS vs. MP	GP-MDSS vs. RF
$P = 0.0001$	$P = 0.3107$	$P = 0.0008$	$P = 0.0009$	$P = 0.0103$

Table 5.3: Statistical significance of the difference in performance between the methods. The first line shows ANOVA test on the 6 samples of solutions found by each method, while the second line depicts pairwise 2-tailed Student t-tests comparing GP-MDSS with each other method.

When using gene signatures to predict the survival of a cohort of breast cancer patients, one of the main goals in clinical applications is to minimize the number of false negative predictions. Table 5.4 summarizes the false negative predictions returned by each machine learning method on the 50 runs. The first line indicates the different methods, while the second and the third lines show the best (i.e. lowest) and mean performances (together with the corresponding SEM).

The best solutions were found by GP-MDSS, and statistical analysis indicates that GP-MDSS consistently outperforms the other five methods as can be seen in Table 5.6. The difference between the various average results is statistically significant (P-value 2.75×10^{-9} for ANOVA test on the 4 samples of solutions found by each method). Finally, pairwise 2-tailed Student t-tests comparing GP-MDSS with each other method demonstrates its better performance.

	GP-MDSS	SVM-K1	SVM-K2	SVM-K3	MP	RF
best	2	6	6	6	5	6
average (SEM)	9.82 (0.44)	13.26 (0.51)	12.60 (0.35)	14.08 (0.39)	12.88 (0.51)	13.38 (0.49)

Table 5.4: Experimental comparison between the number of false negatives found on the test sets by the different machine learning methods. Each method was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method: Medical Decision Support System based on GP (GP-MDSS), Support Vector Machine (SVM), Multilayer Perceptrons (MP), and Random Forest (RF). The second line shows the smallest number of false negatives obtained on the test set over the 50 runs, and the third line reports the average performances of each group of 50 runs on their test sets (standard error of mean is shown in parentheses).

The solutions found by GP-MDSS typically use a rather small number of genes (i.e. features, terminals). In fact, the solutions of the 50 GP-MDSS runs are functions of a number of gene that ranges from 1 to 23, with a median value of 4, and first and third quartiles of 2 and 7 respectively. Few of these features tend to recur in several solutions as can be seen in Table 5.5, where the gene symbol, the gene name of each feature, together with the number of solutions where the feature occurs are shown.

Accession ID	Gene name	Gene description	Solutions
NM_003981	PRC1	protein regulator of cytokinesis 1	48
NM_002916	RFC4	replication factor C (activator 1) 4, 37kDa	23
AI992158	-	-	16
AI554061	-	-	10
NM_006101	NDC80	NDC80 homolog, kinetochore complex component (S. cerevisiae)	9
NM_015984	UCHL5	ubiquitin carboxyl-terminal hydrolase L5	7
NM_020188	C16orf61	chromosome 16 open reading frame 61	6
NM_016448	DTL	denticleless homolog (Drosophila)	6
NM_014791	MELK	maternal embryonic leucine zipper kinase	6
NM_004702	-	-	6

Table 5.5: The 10 most recurring features in the solutions found by GP-MDSS. The four columns show: accession ID, gene name, gene description, and number of solutions where that feature occurs.

The authors of [van de Vijver et al., 2002] used the seventy-gene signature by computing the correlation coefficient between the expression profile of the patient (limited to the 70 genes of the signature) and a previously computed typical expression profile of a good prognosis patient. To compare the performance of the various machine learning algorithms with this scoring system the process was as follows:

- The prognostic score s of the patients was obtained (excluding the ones used to train the signature in [van't Veer et al., 2002]) from the Supplementary Material of [van de Vijver et al., 2002]; authors classified as good prognosis the patients with $s > 0.4$ and as bad prognosis the ones with s≤ 0.4. This is the cutoff used in [van de Vijver et al., 2002].
- 50 random lists of 50 patients from this set were generated, to match the statistic used for machine learning techniques, and for each list the number of false predictions given by the scoring method was computed.

The mean number of false predictions was 16.24, with a SEM of 0.37. Therefore the scoring method appears to be better than all machine learning algorithms other than GP-MDSS, and slightly better than GP-MDSS. The difference between the performances of GP-MDSS and the scoring method are not statistically significant ($P = 0.49$, 2-tailed Student t-test).
The original scoring method of [van't Veer et al., 2002, van de Vijver et al., 2002], and in particular the suggested cutoff of 0.4, was chosen to minimize the number of false negatives. Therefore it is not surprising that in this respect the scoring method is far superior to all machine learning methods, including GP-MDSS. Indeed the average number of false negatives given by the scoring method is 1.78, to be compared to the numbers reported in Table 5.6.

ANOVA $P = 2.75 \times 10^{-9}$				
GP-MDSS vs. SVM-K1 $P = 2.74 \times 10^{-6}$	GP-MDSS vs. SVM-K2 $P = 3.32 \times 10^{-6}$	GP-MDSS vs. SVM-K3 $P = 1.27 \times 10^{-10}$	GP-MDSS vs. MP $P = 8.53 \times 10^{-6}$	GP-MDSS vs. RF $P = 4.65 \times 10^{-7}$

Table 5.6: False negative prediction: statistical significance of the difference in performance between the methods. First line shows ANOVA test on the 6 samples of solutions found by each method, while second line reports pairwise 2-tailed Student t-tests comparing GP-MDSS with each other method.

5.6 Discussion

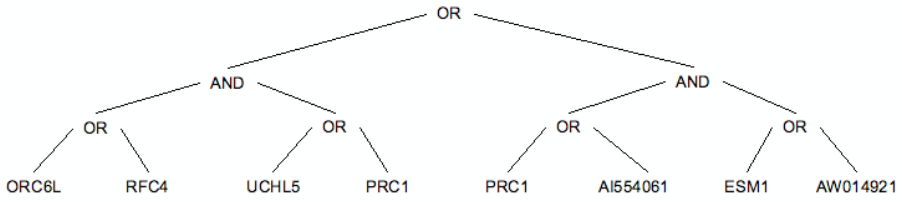
It is well known that the fitness function driving the evolutionary dynamics in a GP framework can be modified in order to let solutions with different characteristics emerge. The results presented and discussed in the previous section were obtained with the goal of minimizing all incorrectly classified instances, summing both false negative and false positive predictions obtained by the solutions. However, when using gene signatures to predict the survival of a cohort of breast cancer patients, minimizing the number of false negative predictions is recognized as one of the most important goals.

For all these reasons, the GP-MDSS fitness function was modified so that false negatives (positives) are penalized more than errors of the other type, hoping to tune the algorithm towards better sensitivity (sensibility). In particular, solutions with greater sensitivity can emerge if larger weights are assigned to false negatives compared to false positives. In general, the fitness function can be transformed in a weighted average of the form:

$$Fitness = 0.9 \times FalseNegative + 0.1 \times FalsePositive$$

The results of 50 runs of this new version of the GP-MDSS technique showed an average of 16.04 (with SEM $= 0.44$) incorrectly classified instances. Compared with the performances of the previous GP-MDSS algorithm, no statistically significant difference can be highlighted (Student t-test $P = 0.50$). When looking only at the number of false negative incorrectly classified instances, the average performance of 4.32 (SEM $= 0.346$) is better than the one of standard GP-MDSS reported in table 5.6 (Student t-test $P = 6.62 \times 10^{-16}$), even if it is still worse than that of the original scoring method.

Finally, it is noteworthy that GP-MDSS can potentially offer biological insight and generate hypotheses for experimental work (see also [Yu et al., 2007]). Indeed an important result of this analysis is that the trees produced by GP-MDSS tend to contain a limited number of features, and therefore are easily interpretable in biological terms. For example, the best-performing tree, shown in Fig. 5.1, includes only 7 genes (features).




```
(or (and (or ORC6L RFC4) (or UCHL5 PRC1))
    (and (or PRC1 AI554061) (or ESM1 AW014921)))
```

Figure 5.1: Tree representation and traditional Lisp representation of the model with the best fitness found by GP over the 50 independent runs.

Chapter 6

Medical Decision Support System For Breast Cancer Based On Floating Point Dataset

6.1 Classification of breast cancer patients into risk classes using floating point data

In the previous chapter the use of several machine learning schemes for the classification of cancer patients survival was investigated. In order to compare their performances, the well known "70-genes signature" dataset, a set of microarray features selected in [van't Veer et al., 2002] based on correlation with survival was used. This is the set of genes on which the molecular prognostic test for breast cancer "MammaPrint"^{TM}is based and has already been discussed in the previous chapter.

In such an investigation both the disease outcome (survival status) of the patients and the expression data were preprocessed and binarized. This choice was made in order to use GP in a rather simple and standard way, with logical operators for the functional nodes (as is usual in many standard GP benchmarks such as the even parity or the multiplexer problems [Koza, 1992]), allowing a simple and direct interpretation of the solutions found by GP-MDSS from the biological viewpoint. Using default implementations and parameters of all machine learning methods, so as to minimize the possible bias given, for instance, by a smart parameter setting in a computational method and a naive one in another, the results presented showed that GP-MDSS clearly outperforms all other techniques. Since all methods other than GP-MDSS had very similar performances, our conclusions indicated GP-MDSS as the most promising method. Finally the best solution found by GP-MDSS was compared to the original scoring method proposed in [van't Veer et al., 2002, van de Vijver et al., 2002] showing that

the difference in prediction of the two methods was rather small and not statistically significant. However, as expected, the scoring method was superior to all machine learning algorithms in minimizing false negatives, which is clearly a very important task in clinical applications.

Nevertheless, GP-MDSS showed some interesting characteristics. For instance, it produced solutions containing a limited number of features, thus performing an automatic feature selection of the relevant genes in the dataset. This added value of the GP-MDSS approach, together with the promising performances shown by GP-MDSS, motivated us in pursuing this research line, whose results shall be presented in this chapter.

First, to overcome the limitations of the analysis of the binarized dataset, the use of original floating point valued expression data was investigated, abandoning the binarization of the 70-genes dataset. Moreover, the fitness function was modified so that false negatives are penalized more than false positives, hoping to tune the classifiers towards better sensitivity.

6.2 Dataset and methods

6.2.1 The 70-Genes Signature

As stated above, the NKI breast cancer dataset [van de Vijver et al., 2002], providing gene expression and survival data for 295 young, systemically untreated breast carcinoma patients is used. As in the previous chapter, only the expression data for the genes included in the "70-genes signature" [van't Veer et al., 2002] were considered. Survival data (targets) were transformed into binary form: the outcome was defined as the survival status of the patient at time $t_{end} = 10.3$ years. By choosing this particular endpoint the number of dead and alive patients was balanced: out of 148 patients for which the status at t_{end} is known, 74 were dead and 74 were alive.

Thus, the dataset is a matrix $H = [H_{(i,j)}]$ composed by 148 rows (instances) and 71 columns (features). Each line i represents the gene signature of a patient and consists in a vector $[H_{(i,1)}\ H_{(i,2)}\ \ldots\ H_{(i,70)}]$ of 70 floating point numbers, and one binary number representing the target ($0 =$ survived after t_{end} years, $1 =$ dead for breast cancer before t_{end} years), that has been placed at position $H_{(i,71)}$. In this way, the last column of matrix H represents all the known target values.

As in the previous chapter, the task is to generate a binary classification

model F such that $F(H_{(i,1)}, H_{(i,2)}, ..., H_{(i,70)}) = H_{(i,71)}$ for each line i in the dataset. Each computational method used was run 50 independent times. For each run, a random splitting of the dataset was performed before model construction, by partitioning it into a training and a test set: 70% of the patients are randomly selected with uniform probability and inserted into the training set, while the remaining 30% form the test set.

6.2.2 Methods

The methods used are described in the previous chapters.

GP potential solutions (GP individuals) were built using the set of functions $\mathcal{F} = \{\texttt{plus}, \texttt{minus}, \texttt{mul}, \texttt{div}, \texttt{squaredsin}\}$, where `plus`, `minus` and `mul` are the usual binary addition, subtraction and multiplication operators, respectively; `div` is the protected division operator as in [Koza, 1992] and `squaredsin` is the square of the sine trigonometric function. GP individuals (which are expressions returning a floating point number) have been turned into binary classifiers by using a threshold value. The threshold used was the average between the maximum and minimum values in the whole dataset set.

For Support Vector Machines (SVM), Multilayer Perceptron (MP), Voted Perceptron (VP) and Radial Basis Function Network (RBFN), the Weka standard implementation has been used, using the Weka default parameters reported in Table 6.1. In this study with the original floating point valued expression data Support Vector Machine with polynomial kernel with degree 1 was used. This means that just the accurancy of linear SVM is evaluated, due to the little difference between different kernels demonstrated in this feature space.

Moreover, in order to use a RBFN, it is necessary to specify the hidden unit activation function, the number of processing units, a criterion for modeling a given task and a training algorithm for finding the parameters of the network. In this work the standard Weka implementation has been used. It consists in a normalized Gaussian radial function network. It uses the k-means clustering algorithm to provide the basis functions and learns a logistic regression (discrete class problems) on top of that. Symmetric multivariate Gaussians are fit to the data from each cluster. All the other parameters are specified in Table 6.1.

GP-MDSS Parameters	
population size	500 individuals
population initialization	ramped half and half [Koza, 1992]
selection method	tournament (tournament size = 10)
crossover rate	0.9
mutation rate	0.1
maximum number of generations	5
algorithm	generational tree based GP-MDSS with no elitism
SVM Parameters	
complexity parameter	0.1
size of the kernel cache	10^7
epsilon value for the round-off error	10^{-12}
exponent for the polynomial kernel	1.0
tolerance parameter	0.001
Multilayered Perceptron Parameters	
learning algorithm	Back-propagation
learning rate	0.03
activation function for all the neurons in the net	sigmoid
momentum	0.2 progressively decreasing to 0.0001
hidden layers	(number of attributes + number of classes) / 2
number of epochs of training	500
Voted Perceptron Parameters	
exponent for the polynomial kernel	1.0
maximum number of alterations to the perceptron	10000
Radial Basis Function Network Parameters	
minimum standard deviation for the clusters	0.1
number of clusters for K-Means	2
ridge value for the logistic	1.8×10^{-8}

Table 6.1: Parameters used in the experiments.

6.3 Experimental Results

The experiment focus is twofolds: first, to overcome the limitation of the study on the binarized expression data, the use of the original floating point valued expression data has been investigated, abandoning the binarization of the 70-genes dataset. The machine learning techniques are tested on it, and the results are discussed in Section 6.3.1. Second, the fitness function in the GP-MDSS runs was modified so that false negatives (positives) are penalized more than errors of the other type, so as to tune the algorithm towards better sensitivity (specificity). These experiments are presented and analyzed in Section 6.3.2. The best solutions found by GP-MDSS are discussed in Section 6.3.3

6.3.1 Comparing Different Algorithms

Table 6.2 summarizes the results of the 50 runs returned by each machine learning method introduced in Section 6.2.2 on the floating point valued 70-genes signature dataset introduced in Section 6.2.1. The first two lines indicate the different methods and the best (i.e. lowest) values of the incorrectly classified instances obtained on the test set over the 50 runs, respectively. Since every set of runs does not show any evidence of deviation from a normal distribution, the third line reports the mean performances of each group of 50 runs on their test sets, together with the corresponding standard error of mean (SEM from now on).

	GP-MDSS	MP	RBFN	SVM	VP
best	12	10	14	11	10
average (SEM)	16.56 (0.37)	15.70 (0.49)	17.82 (0.28)	14.72 (0.35)	14.74 (0.31)

Table 6.2: Experimental comparison between the number of *incorrectly classified instances* found on the test sets by the different machine learning methods. Each method was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method: Genetic Programming Medical Decision Support System (GP-MDSS), Multilayer Perceptron (MP), Radial Basis Function Network (RBFN), Support Vector Machines (SVM), and Voted Perceptron (VP). The second line shows the best value of the incorrectly classified instances obtained on the test set over the runs, and the third line reports the average performances of each group of runs on their test sets (standard error of mean is shown between parentheses).

These groups being similar in the large number of samples, an ANOVA test indicated a significant difference in their average performances (P-value 8.41×10^{-9}). Tukey's Honestly Significant Difference test evidenced a difference between RBFN and all other methods, together with an outperformance of both VP and SVM with respect to GP-MDSS. Finally, no statistically significant difference was outlined between VP and SVM, and between MP and VP, SVM, and GP-MDSS.

When using gene signatures to predict the survival of a cohort of breast cancer patients, one of the main goals in clinical applications is to minimize the number of false negative predictions. Table 6.3 summarizes the false negative predictions returned by each machine learning method on the 50 runs. The first two lines indicate the different methods and the best (i.e. lowest) values of the false negatives obtained on the test set over the 50 runs,

respectively. Since every set of runs does not show any evidence of deviation from a normal distribution, also in this case the third line reports the mean performances (together with the corresponding SEM) of each group of 50 runs on their test sets.

	GP-MDSS	MP	RBFN	SVM	VP
best	3	5	2	6	3
average (SEM)	10.66 (0.48)	11.56 (0.49)	10.34 (0.43)	10.34 (0.36)	9.18 (0.36)

Table 6.3: Experimental comparison between the number of *false negative* predictions found on the test sets by the different machine learning methods. Each method was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method: Genetic Programming Medical Decision Support System (GP-MDSS), Multilayer Perceptron (MP), Radial Basis Function Network (RBFN), Support Vector Machines (SVM), and Voted Perceptron (VP). The second line shows the best value of the false negatives obtained on the test set over the runs, and the third line reports the average performances of each group of runs on their test sets (standard error of mean is shown between parentheses).

If looking only at false negative predictions, the only difference evidenced by ANOVA (P-value of 4.16×10^{-4}) together with Tukey's Honestly Significant Difference test was the outperformance of VP with respect to all the other methods.

6.3.2 Towards GP-MDSS with Greater Sensitivity or Sensibility

Contrarily to what was observed with the binarization of the 70-genes signature dataset, when faced with the problem of classifying floating point valued expression data, GP-MDSS did not outperform the other machine learning techniques. On the other hand, it is well known that the fitness function driving the evolutionary dynamics in a GP system can be modified in order to let solutions with different characteristics emerge. The results presented and discussed in the previous section were obtained with the goal of minimizing all incorrectly classified instances, summing both false negative and false positive predictions obtained by the solutions. However, when using gene signatures to predict the survival of a cohort of breast cancer patients, minimizing the number of false negative predictions is recognized as one of the most important goals. In fact, the original scoring method

of [van't Veer et al., 2002, van de Vijver et al., 2002] was designed in such a way as to minimize the number of false negatives. It is therefore not surprising that in this respect the scoring method is far superior to all machine learning methods.

For all these reasons,the GP-MDSS fitness function was modified so that false negatives (positives) are penalized more than errors of the other type, hoping to tune the algorithm towards better sensitivity (sensibility). In particular, solutions with greater sensitivity can emerge if larger weights are assigned to false negatives compared to false positives. As stated previously, we can transform the fitness function in a weighted average of the form:

$$Fitness = \alpha \times FalseNegative + \beta \times FalsePositive$$

where α and β are two floating point values. In this work, has been decided to use $\alpha, \beta \in [0,1]$ and such that $\alpha + \beta = 1$, in order to facilitate the task of giving higher weights to false negatives or false positives.

The fitness function of the GP-MDSS, whose results have been presented in the previous section, can be expressed using $\alpha = \beta = 0.5$ (from now on, called $GP\text{-}MDSS_{5,5}$). In order to explore the possibility of tuning the fitness function, the evolutions of other four GP-MDSS variants, designed to find solutions from larger specificity to larger sensibility, have been performed. The variants are:

- $GP\text{-}MDSS_{1,9}$ where $\alpha = 0.1$ and $\beta = 0.9$;
- $GP\text{-}MDSS_{3,7}$ where $\alpha = 0.3$ and $\beta = 0.7$;
- $GP\text{-}MDSS_{7,3}$ where $\alpha = 0.7$ and $\beta = 0.3$;
- $GP\text{-}MDSS_{9,1}$ where $\alpha = 0.9$ and $\beta = 0.1$.

Table 6.4 summarizes the results of the 50 runs returned by each GP-MDSS variant. The first two lines indicate the different methods and the best (i.e. lowest) values of the incorrectly classified instances obtained on the test set over the 50 runs, respectively. Since every set of runs does not show any evidence of deviation from a normal distribution, the third line reports the mean performances of each group of 50 runs on their test sets, together with the corresponding SEM.

Statistical analysis has been performed comparing these results among one another them and with the machine learning method that showed the

	GP-MDSS$_{1,9}$	*GP-MDSS*$_{3,7}$	*GP-MDSS*$_{5,5}$	*GP-MDSS*$_{7,3}$	*GP-MDSS*$_{9,1}$
best	17	14	12	11	11
average (SEM)	24.28 (0.45)	20.58 (0.48)	16.56 (0.37)	15.20 (0.26)	16.02 (0.39)

Table 6.4: Experimental comparison between the number of *incorrectly classified instances* found on the test sets by *GP-MDSS*$_{1,9}$, *GP-MDSS*$_{3,7}$, *GP-MDSS*$_{5,5}$ (reported for completeness), *GP-MDSS*$_{7,3}$, and *GP-MDSS*$_{9,1}$. Each variant was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method, the second line shows the best value of the incorrectly classified instances obtained on the test set over the 50 runs, and the third line reports the average performances of each group of 50 runs on their test sets (SEM is shown between parentheses).

	GP-MDSS$_{1,9}$	*GP-MDSS*$_{3,7}$	*GP-MDSS*$_{5,5}$	*GP-MDSS*$_{7,3}$	*GP-MDSS*$_{9,1}$
best	12	5	3	2	0
average (SEM)	21.72 (0.55)	16.60 (0.73)	10.66 (0.48)	6.64 (0.37)	5.0 (0.37)

Table 6.5: Experimental comparison between the number of *false negative* predictions found on the test sets by *GP-MDSS*$_{1,9}$, *GP-MDSS*$_{3,7}$, *GP-MDSS*$_{5,5}$ (reported for completeness), *GP-MDSS*$_{7,3}$, and *GP-MDSS*$_{9,1}$. Each variant was independently run 50 times using each time a different training/test partition of the validation dataset (see text for details). The first line indicates the method, the second line shows the best value of the false negatives obtained on the test set over the 50 runs, and the third line reports the average performances of each group of 50 runs on their test sets (standard error of mean is shown between parentheses).

best performance on minimizing the total number of incorrectly classified instances, the Support Vector Machines, as discussed in Section 6.3.1. ANOVA test showed a significant (P-value of 8.74×2.26^{-58}) difference between these methods. Tukey's Honestly Significant Difference test highlighted an outperformance of SVM, *GP-MDSS*$_{9,1}$, *GP-MDSS*$_{7,3}$, and *GP-MDSS*$_{5,5}$ (that do not show any statistical difference among them) with respect to *GP-MDSS*$_{3,7}$, and of *GP-MDSS*$_{3,7}$ with respect to *GP-MDSS*$_{1,9}$.

If the GP-MDSS variants did not result in better performances on all incorrectly classified instances, this is not true when looking only at false negative predictions. Below this issue is approached. Table 6.5 summarizes the results of the 50 runs returned by each GP-MDSS variant. The first two lines indicate the different methods and the best (i.e. lowest) values of the false negatives obtained on the test set over the 50 runs, respectively. Since also in this case every set of runs does not show any evidence of deviation from a normal distribution, the third line reports the mean performances (together with the corresponding SEM) of each group of 50 runs on their

test sets.

Statistical analysis of the results summarized in Table 6.5 depicts the following ranking in the performance among the five GP-MDSS variants and the VP, the best performing among all other machine learning techniques in minimizing false negatives (as discussed in Section 6.3.1): *GP-MDSS*$_{9,1}$, *GP-MDSS*$_{7,3}$, *GP-MDSS*$_{5,5}$, *GP-MDSS*$_{3,7}$, *GP-MDSS*$_{1,9}$, with VP statistically indistinguishable from *GP-MDSS*$_{5,5}$, but *GP-MDSS*$_{9,1}$ outperforming *GP-MDSS*$_{7,3}$ in a statistically significant way and *GP-MDSS*$_{7,3}$ outperforming VP in a statistically significant way. This analysis clearly demonstrates the effectiveness of the modified fitness function when minimizing the false negative predictions.

The authors of Refs. [van't Veer et al., 2002, van de Vijver et al., 2002] used the seventy-gene signature by assigning a coefficient to each of the features and computing a score for each patient as the scalar product of these coefficients and the patient gene expression. The process used to compare the performance of the various machine learning algorithms is the same as the one used for the binarized dataset. The mean number of false predictions was 16.7, while the mean number of false negative predictions was 1.7.

This method's performances have been compared to those obtained by the other machine learning techniques on the binarized dataset. Concerning the total number of false predictions, the scoring method appeared to perform better than all machine learning algorithms other than GP-MDSS, and slightly worse than GP-MDSS. The difference between the performances of GP-MDSS and the scoring method were not statistically significant. On the other hand, not surprisingly, with respect to false negative predictions, the scoring method is far superior to all machine learning methods, including GP-MDSS.

When compared to the predictive power of the best performing GP-MDSS classifier presented in this work and in Refs. [Farinaccio et al., 2010], the comparison is interesting. If the scoring method still outperforms GP-MDSS in minimizing the number of false negative predictions, the opposite is true when looking at the total number of incorrectly classified instances.

6.3.3 Analysis of the best solutions found by GP-MDSS

Figure 6.1 reports the individual with the best value of the incorrectly classified instances found by *GP-MDSS*$_{1,9}$, represented both in a tree structure

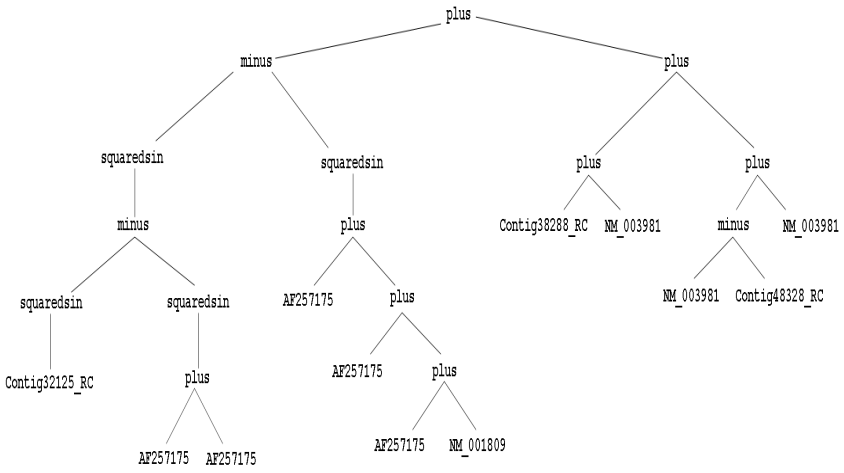
and in prefix expression form. From this figure, we can clearly see that GP-MDSS performed a strong implicit feature selection. In fact, of the total 70 available features, only 6 are used by this solution (some of them, like for instance `AF257175`, repeated several times in different points in the tree). Analogously, Figure 6.2 reports the individual with the best value of the false negatives found by $GP\text{-}MDSS_{1,9}$. In this case, the implicit feature selection performed by GP-MDSS is even stronger, given that only two features are used by this solution. Interestingly, one of those two features (`NM_001809`) was used also by the solution with the best value of the incorrectly classified instances. Another interesting characteristic shared by these two solutions is the frequent use of operators such as `squaredsin`, `plus` and `minus` and the absence of the multiplication and division operators.

In Table 6.6 a sketch of the features contained in the two solutions shown in Figures 6.1 and 6.2 is reported.

Accession ID	**Gene description**
`Contig32125_RC`	–
`AF257175`	Homo sapiens hepatocellular carcinoma-associated antigen 64 (HCA64) mRNA, complete cds.
`NM_001809`	Homo sapiens centromere protein A (CENPA), transcript variant 1, mRNA.
`Contig38288_RC`	ESTs, weakly similar to quiescin
`NM_003961`	Homo sapiens rhomboid, veinlet-like 1 (Drosophila) (RHBDL1), mRNA
`Contig48328_RC`	–
`NM_016448`	Homo sapiens denticleless homolog (Drosophila) (DTL), mRNA

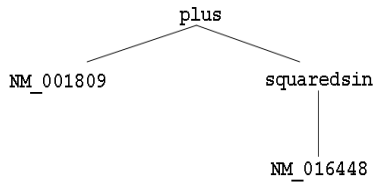
Table 6.6: Features used in the solutions found by GP-MDSS reported in Figures 6.1 and 6.2. The two columns show accession ID and gene description.

The two columns of this table show the gene accession IDs and an informal description of the genes, respectively. A further analysis of the semantics of these features and their reciprocal relationships, in order to verify their effective binding with breast cancer and to have a biological insight of the results found by GP-MDSS, is an important part of future work.




```
plus(minus
      (squaredsin(minus(squaredsin(Contig32125_RC),
       squaredsin(plus(AF257175,AF257175)))),
       squaredsin(plus(AF257175,plus(AF257175,plus(AF257175,NM_001809))))),
     plus
      (plus(Contig38288_RC,NM_003981),plus(minus(NM_003981,Contig48328_RC),
        NM_003981)))
```

Figure 6.1: Individual found by *GP-MDSS*$_{1,9}$ with the best value of the Incorrectly Classified Instances. Upper part: tree representation. Lower part: prefix expression representation.




```
(plus ( NM_001809, squaredsin ( NM_016448)))
```

Figure 6.2: Individual found by *GP-MDSS*$_{1,9}$ with the best value of the False Negatives. Upper part: tree representation. Lower part: prefix expression representation.

PART III

DATA DRIVEN GENE REGULATORY NETWORK

Chapter 7

Generator of Gene Regulatory Network and Dynamic Simulator System

7.1 Introduction

As stated in Chapter 1, the aim of this thesis is to investigate the functions of genes and their possible relations. These functions can be better understood if they are not only studied as isolated entities, but if their reciprocal relationships are also investigated.
One approach which seems very promising is the ensemble approach to the study of genetic networks, pioneered twenty years ago by S.A. Kaufmann [Kaufmann, 1971, Kaufmann, 1993]. According to this line of research, the interest is focused on the typical properties of networks which are supposed to capture some charachteristics of biological systems, like for instance gene regulatory networks.

Gene regulatory networks comprising genes, proteins and other interacting molecules, are extremely complex systems. The emerging field of system's biology [Hasty and McMillen, 2002, Hayete et al., 2007, Sprinzak and Elowitz, 2005] is aimed at a formal understanding of the biological processes caused by the numerous regulatory, signaling and metabolic interactions between the different components and their coordinated action. This is usually done by developing quantitative mathematical models, able to describe changes in concentration of each gene transcript and protein in a network as a function of their regulatory interactions (gene regulatory network).
The formalisms to model biological networks defined so far are numerous (see for instance [Di Ventura et al., 2006, Szallasi et al., 2006]); a widely used one is the system of differential equations. It is

a very powerful and flexible model to describe complex relations among components. Some researchers studied how to learn about the gene regulatory networks by using systems of differential equations [Chen et al., 1999, Tominaga et al., 2000, Sakamoto and Iba, 2000]. But it is not necessarily easy to determine the suitable form of equations which represent the network. Thus, the differential equations form had been fixed during the learning phase in previous studies. As a result, their goal was to simply optimize parameters, i.e., coefficients in the fixed equations.

Another technique most commonly used is the Random Boolean Networks, to which the system developed in this thesis and described in this chapter is partially inspired, so that it is briefly described in the next section.

7.1.1 Random Boolean Networks

Let us consider a network composed of N genes, or nodes, which can take either the value 0 (inactive) or 1 (active). Let $x_i(t) \in \{0,1\}$ be the activation value of node i at time t, and let $X_i(t) = [x_1(t), x_2(t), ..., x_N(t)]$ be the vector of activation values of all the genes. In a classical Random Boolean Network (RBN) each node has the same number of incoming connections k_{in}, and its k_{in} input nodes are chosen at random with uniform probability among the remaining N-1 nodes. The output (i.e. the new value of a node) corresponding to each set of values of the input nodes is determined by a boolean function, named *activation function*, which is associated to that node (see Fig.7.1), which is also chosen at random, according to some probability distribution [Kaufmann, 1993]; the simplest choice is that of a uniform distribution among all the possible boolean function of k_{in} arguments. So the RBNs have two degrees of randomness: the random topology and the random activation function for each node.

Even if this model is yet one of the main reference models for many studies of genetic networks [Kaufmann et al., 2003, Serra et al., 2008, Semeria et al., 2004], it has the weakness of the assumption of randomness. In fact, a careful analysis of some known real biological control circuits has shown that there is a strong bias in favour of the so-called *canalizing functions* [Harris et al., 2002]. A boolean function is said to be canalizing if at least one value of one of its input nodes uniquely determines its output, no matter what the other input values are, while both the topology and the

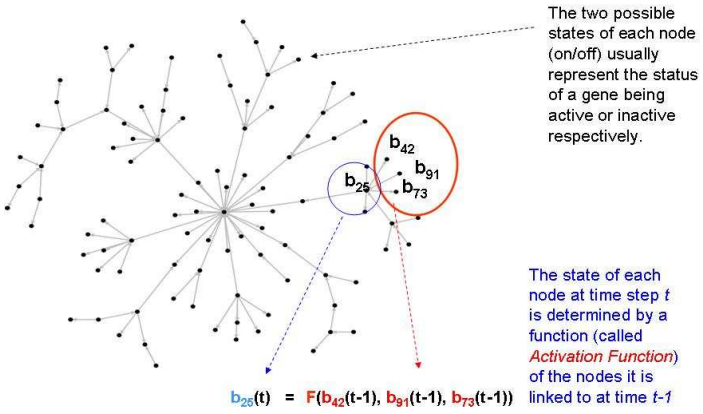


Figure 7.1: Example of Random Boolean Network and node's Activation Function

boolean function associated to each gene do not change in time.

In order to analyse the properties of an ensemble of random boolean networks, different networks are synthesized and their dynamical properties are examined. The ensemble differs mainly in the choice of the number of nodes N, the input connectivity k_{in} and the choice of the set of allowed boolean functions.

So the aim of this thesis is modelling a gene regulatory network with a data driven topology, instead of a random one. It is driven by Genetic Programming (GP) applied to serial temporal data first elaborated with a new cluster kernel method, in order to perform dimensional reduction of data. It is described in details in the next chapter; in addition the interested reader is referred to [Antoniotti et al., 2010].

7.2 Data Driven Gene Regulatory Network

A method, widely used to identify interactions among genes from gene expression data is the reverse-engineering method. Typically, the data consist of measurements at steady state after multiple perturbations (like gene overexpression, knockdown or drug treatment) or at multiple instants after one or more perturbations (i.e. time series data), as the ones used to test the system presented here. Successful applications of these approaches have been demonstrated in bacteria, yeast and, recently, in mammalian sys-

tems [Della Gatta et al., 2008, Di Bernardo et al., 2005, Faith et al., 2007, Gardner et al., 2003]. Many reverse-engineering approaches have been proposed to date, and their assessment and evaluation is of critical importance [Stolovitzky et al., 2007]. For this reason, in [Cantone et al., 2009], a synthetic network of five genes in the yeast *Saccharomyces cerevisiae*, regulating each other, called IRMA, was defined. This network was defined with the explicit goal of enabling in vivo reverse-engineering and modeling assessment.

Principally for this reason, the IRMA network is used to test the gene regulatory network generator and dynamic simulator system proposed in this thesis and presented at the 20th Italian Workshop on Neural Network [Farinaccio et al., 2011] and in the 5th European Graduated Student Workshop on Evolutionary Computation [Farinaccio, 2010] .

The system takes as input a dataset of temporal series of gene expression levels and tries to reconstruct the underlying gene regulatory network by taking advantage of some characteristics of GP. These characteristics are the ability of GP to solve complex regression problems, where little or no information is known about the underlying data, and its ability to perform an automatic feature selection during the learning phase, as already pointed out, for instance, in [Langdon and Barrett, 2004, Vanneschi et al., 2010, Farinaccio et al., 2010, Farinaccio et al., 2009]. The peculiarity of such a system, compared to many existing reverse-engineering systems (like for instance the ones used in [Cantone et al., 2009]), is that it is able to generate both the topology of the target network and the activation functions of the different genes at the same time. This allows not only to know the structure of the network, but also to simulate its dynamics. Given its ability to automatically generate networks the system is called Gene Regulatory Nework Generator (GRNGen).

7.3 The Proposed System: GRNGen

The proposed system (GRNGen) works on datasets containing temporal series of gene expressions [Quackenbush, 2001] and it can be summarized in three steps, the last of which is composed of five phases:

I. The construction of a regression problem for each gene represented in

the dataset;

II. The generation of the regression models, interpreted as the activation functions of the various genes;

III. The building and simulation of the network using the information contained in those models;

III. 1 The generation, for each gene, of its own activation function, expressed in form of tree structure;

III. 2 The identification, for each gene, of the genes involved in its activation function, and between them, the construction of the subgraph relating to its dependencies;

III. 3 The construction of the complete graph that represents the whole network;

III. 4 The collection of the activation rule of each gene;

III. 5 The dynamic simulation of the constructed gene regulatory network.

The system is general and can potentially be implemented using any regression technique at step (II). Nevertheless, it could be particularly suitable to use GP, given that GP has the ability to perform an automatic feature selection at learning time. The intuition of GRNGen is simple, even though its formal definition may be complicated. For this reason, it is preferable to present it using a description of its application to an example of gene expression time series.

7.4 Example of GRNGen Application

Assume we have the following matrix of temporal series of gene expression data, denoted as X (Time Series Matrix):

	$TIME1$	$TIME2$	$TIME3$	$TIME4$	$TIME5$	$TIME6$
$GENE1$	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
$GENE2$	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}
$GENE3$	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}
$GENE4$	x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}
$GENE5$	x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}

where each line *GENEi* of this matrix represents the activation values of gene i in the different considered instants and each element x_{ij} represents the activation value of $i-th$ gene at instant j.
According to the matrix, in this example we have five genes (lines) and six instants (columns).

Step I: Building Regression Problems from Matrix X. We assume that the activation value of each gene at a given instant $t > 1$ is given by the activation values of all the *other* genes at time $t-1$, as drafted in Fig. 7.2.

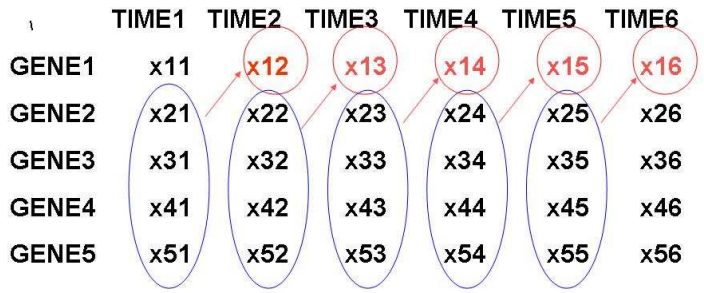


Figure 7.2: Example of Gene Expression TimeSeries

Thus, to obtain the Activation Function of GENE1 in time series matrix X, we have to find (or approximate) the function ACT_1 such that:

$$
\begin{aligned}
&ACT_1(x_{21}, x_{31}, x_{41}, x_{51}) = x_{12},\\
&ACT_1(x_{22}, x_{32}, x_{42}, x_{52}) = x_{13},\\
&ACT_1(x_{23}, x_{33}, x_{43}, x_{53}) = x_{14},\\
&ACT_1(x_{24}, x_{34}, x_{44}, x_{54}) = x_{15},\\
&ACT_1(x_{25}, x_{35}, x_{45}, x_{55}) = x_{16}.
\end{aligned}
$$

So for GENE1, we build the dataset showed in Fig. 7.3
The Activation Functions ACT_2, ACT_3 and ACT_4 of the other genes are obtained in the same way.

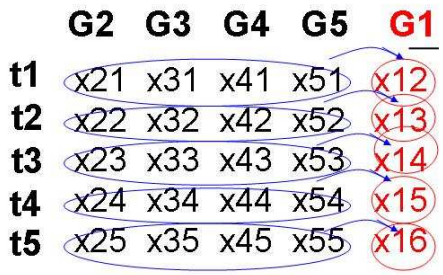


Figure 7.3: Example of Regression Model for Gene 1

Step II: Regression for Each Gene We perform a regression on the dataset built for each gene by means of GP. The result will be the approximation of the Activation Function of each gene.

For example, possible expressions for functions ACT_1, ACT_2, ACT_3, ACT_4 and ACT_5 could be the following (denoting $GENE_i$ with G_i, for the sake of readability):

$$ACT_1(G_2,G_3,G_4,G_5) = (G_4 + (G_4 + G_5) + ((G_2 - G_4) - G_3) - G_2)$$

$$ACT_2(G_1,G_3,G_4,G_5) = G_4$$

$$ACT_3(G_1,G_2,G_4,G_5) = G_2$$

$$ACT_4(G_1,G_2,G_3,G_5) = ((G_2 + G_5) - ((G_5 - G_5) + G_5))$$

$$ACT_5(G_1,G_2,G_3,G_5) = (G_1 + (G_3 - G_4))$$

Of course, given its stochastic nature, in practice the regression by means of GP has to be executed several independent times for each gene. For example, the regression function chosen for each gene may be the one with the best fitness over all these runs.

Suppose we are in the case stated above, so we assume that the activation values of the various genes have the following dependencies:

G_1 depends on the activation values of G_2, G_3, G_4 and G_5;
G_2 depends on G_4;
G_3 depends on G_2;
G_4 depends on G_2, G_5;
G_5 depends on G_1, G_3, G_4;

Note that the use of GP in performing the regression has the two advantages of being readily generalized to arbitrary non-linear dependencies and of automatically performing a selection of the most relevant features (in our case, of the genes whose expression is most relevant in determining the activation of the target gene).

Step III: Network Reconstruction and Simulation

Phase1

In order to reconstruct/approximate the topology of the network that induces time series X, the activation functions of each gene found by means of GP regression, denoted with ACT_i from now on, have to be collected. Due to the nature of the tree-based GP used, they have a tree form, and so they are expressed as a tree structure.

Phase2

It is necessary, at this point, to extract, for each gene, the genes involved in its activation function. In this way it is possible to build the subgraph relating to each gene, basing it on its dependencies, as shown in Fig. 7.4.

Phase3

Now all the subgraphs can be connected to build the whole gene regulatory network, as shown Fig. 7.5.
In this network each gene is represented by a node and links represent the mutual relationships between genes. In this reconstructed network, a directed link from $gene_i$ to $gene_j$ exists if and only if $gene_j$ appears in the expression of function ACT_i found by GP.

Phase4

Contrarily to what happens in many existing gene regulatory network

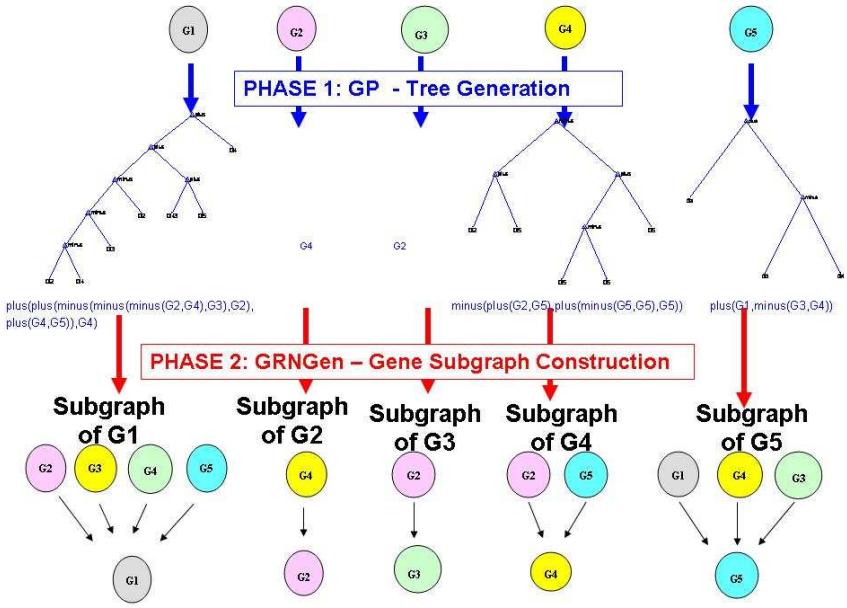


Figure 7.4: Scheme of Phase1 and Phase2: GRNGen collects the activation function of each gene found by means of GP linear regression and extracts, for each gene, the ones involved in its activation function. Then it builds the subgraph related to each gene, basing it on its dependencies.

generator systems, GRNGen is able not only to reconstruct the topology of the network, but it also allows to know the activation rules of each nodes. With this aim, in phase 4 it collects in a table the activation rules of each gene, see Fig. 7.6. Note that the difference between this phase and the 2th phase has a logic nature rather than being an implementation constraint: in the 2nd phase the mutual interactions between genes are identified, instead in the 4th phase the activation function are detected and extracted.

Phase5

Knowing the activation rules of each node, GRNGen is now able to simulate the dynamics of the gene regulatory network. A scheme is shown in Fig. 7.7 and it is based on a realistic reconstruction and dynamic simulation of a gene regulatory network, described in depth in the following section. For this reason the name of the genes are now the real ones, i.e.:
$G_1 = CBF1$,
$G_2 = GAL4$,

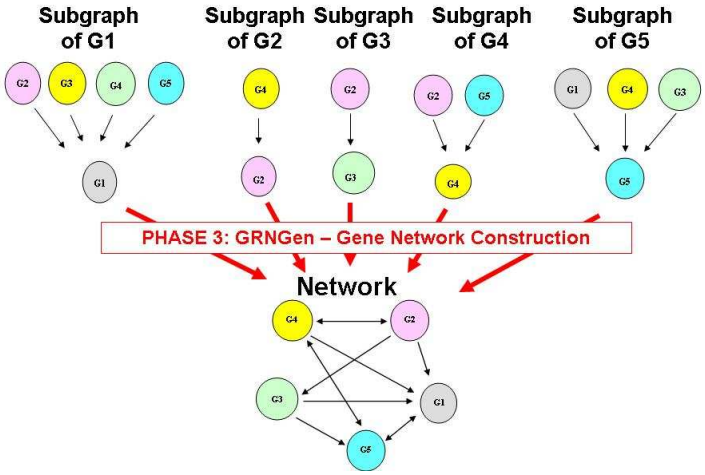


Figure 7.5: Scheme of Phase3: GRNGen connects all the subgraphs to build the whole gene regulatory network

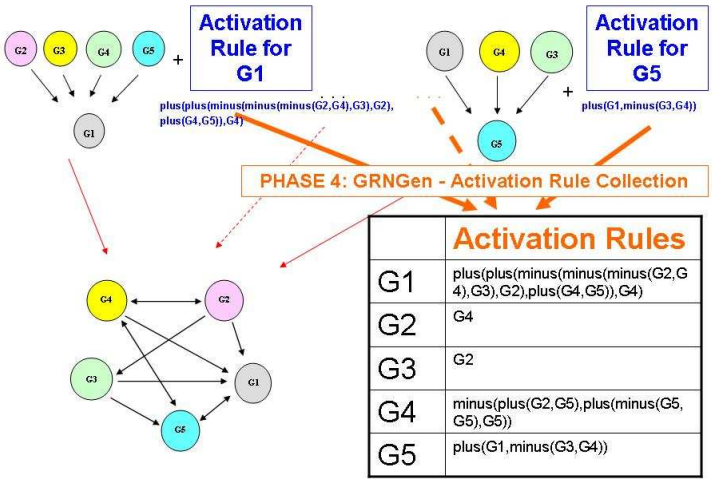


Figure 7.6: Scheme of Phase4: GRNGen collects in a table the activation rules of each gene

$G_3 = SWI5$,
$G_4 = GAL80$,
$G_5 = ASH1$.

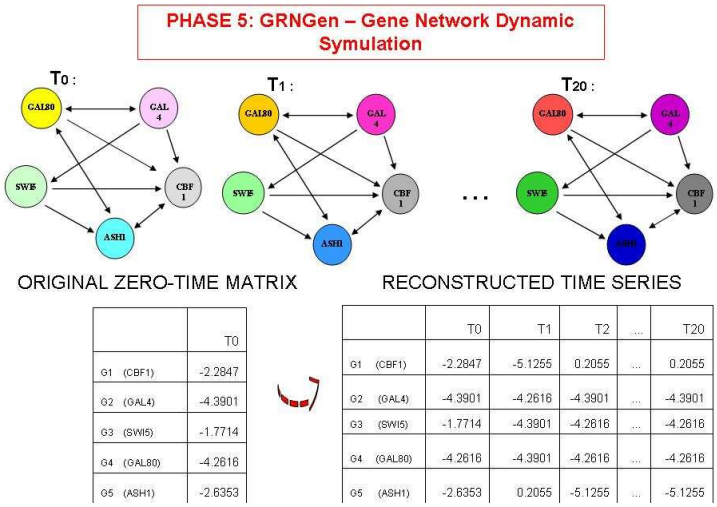


Figure 7.7: Scheme of Phase5: GRNGen simulates the dynamic of the gene regulatory network initializing the network with the first instant of the gene expression time series

The initialization is obtained by assigning the values at the first instant of matrix X to all genes. The subsequent states of the network are obtained by applying to each gene its activation function.

7.5 A Test Case: The Reverse Engineering of IRMA Network

In order to validate the system a reverse engineering of a gene regulatory network has been performed as a test case. The IRMA synthetic network, introduced in [Cantone et al., 2009] has been used (Fig. 7.8).

This network was built in the yeast *Saccharomyces cerevisiae* for in vivo benchmarking of reverse-engineering and modeling approaches. It is composed by five genes regulating each other through a variety of regulatory interactions. A matematical model of the regulatory interactions among genes in IRMA is shown in Figure 7.11(a). IRMA is organized in such a way that each gene controls transcription of at least another gene in the network and it can be *switched on* or *off* by culturing cells in galactose or in glucose, respectively.

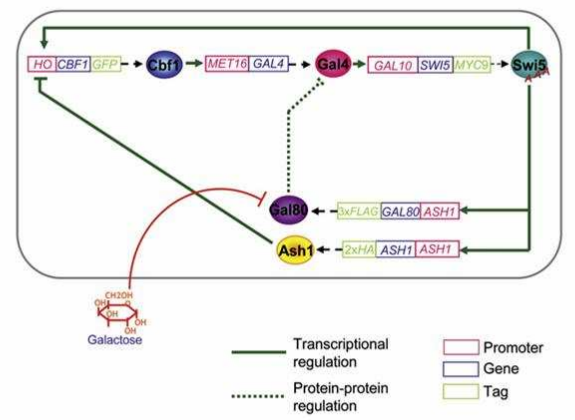


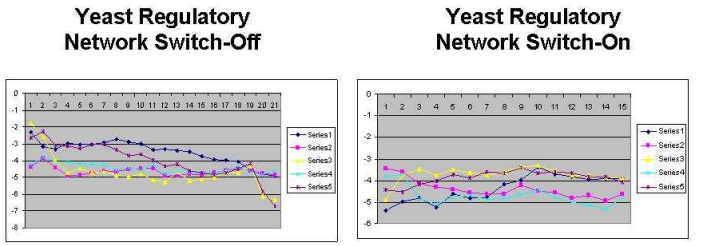
Figure 7.8: IRMA Synthetic Network

The genes involved in this network are SWI5, ASH1, CBF1, GAL4 and GAL80. A description of these genes is beyond the objectives of this thesis. The interested reader is referred to [Cantone et al., 2009] or to *http://www.yeastgenome.org* for a detailed description.
In this work, both the *switch on* and *switch off* time series presented in the supplementary material of [Cantone et al., 2009] have been used in order to test GRNGen (drafted in Fig. 7.9)

7.6 Experimental Setting

The different regression problems, aimed at finding the activation functions of the different genes, have been solved using GP with the following parameter configuration:

- Ramped Half-and-Half initialization;
- Tournament Selection of size 10;
- Standard Subtree Crossover with rate equal to 0.9;



	T0	T1	T2	...	T20
G1	-2.2847	-3.1725	-3.3104	...	-4.9337
G2	-4.3901	-3.8776	-4.4063	...	-4.8536
G3	-1.7714	-2.577	-3.9846	...	-6.32
G4	-4.2616	-3.7942	-4.0456	...	-4.5756
G5	-2.6353	-2.2701	-3.0748	...	-6.7254

	T0	T1	T2	...	T14
G1	-5.4037	-4.9762	-4.8159	...	-3.8728
G2	-3.4296	-3.5936	-4.1541	...	-4.6052
G3	-4.8929	-3.7423	-3.4546	...	-3.8873
G4	-3.8122	-3.7297	-4.8536	...	-4.7795
G5	-4.4228	-4.5282	-4.1605	...	-4.0745

Figure 7.9: IRMA SwitchOn and SwitchOff TimeSeries

- Standard Subtree Mutation with rate equal to 0.1;
- Maximum Tree Depth for Initialization equal to 6;
- Maximum Tree Depth for Variation equal to 10;

The function set $\mathcal{F}$ contained the two binary operators $+$ and $-$.
The terminal set $\mathcal{T}$ contained five floating point variables.
The fitness function is the root mean squared error. Only in the case of the *switch on* dataset, the numerical constant 0.5 has been added to the terminal set. Several preliminary tests induced this choice, showing better results having this constant in the terminal set in the case of *switch on* dataset.

For the *switch on* data a population composed by 50 individuals has been used and GP was run for 100 generations, while for the *switch off* data a population composed by 100 individuals has been used and GP was run for 50 generations.

Given the regression trees obtained by means of GP (performed with a modified version of the GPLab tool [Silva, 2007]), GRNGen builds, for each gene, the graph of its dependencies with the other genes and finally it aggregates all these (sub-)graphs, building the complete graph that represents the network. GRNGen is developed in Matlab v7.0.

7.7 Experimental Results

Although the procedure by which the GRNGen builds the network is mostly deterministic, the result is non deterministic due to the stochastic nature of the regression step executed by means of GP. For this reason a statistical approach has been used to evaluate the results, as sketched in Fig. 7.10

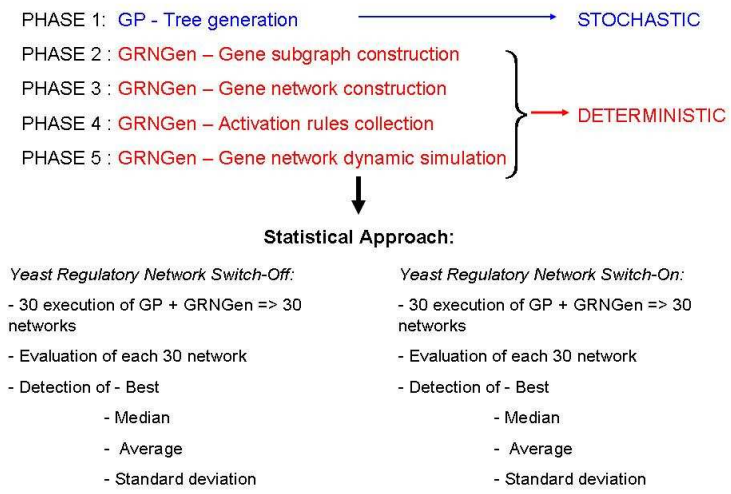


Figure 7.10: A sketch of the phases of GRNGen and the statistical approach used to evaluate its results

As outlined, GRNGen has been executed 30 independent times for the *switch on* data and 30 independent times for the *switch off* data. This has produced 30 different reconstructions of the target network for the *switch on* data and 30 different reconstructions for the *switch off* data.
In order to evaluate the ability of these reconstructed networks in approximating the topology of the target one, the same performance measures as in [Cantone et al., 2009] have been used, i.e.:

- Positive Predictive Value (PPV) = TP / (TP + FP)
- Sensitivity (Se) = TP / (TP + FN)

where TP (True Positives) is the number of links that are both in the reconstructed network and in the true one, FP (False Positives) is the number of links that are in the reconstructed network, but not in the true one and FN

(False Negatives) is the number of links that are not in the reconstructed network, but are contained in the true one. For both PPV and Se, the best, median, average and standard deviations obtained over these 30 runs are reported.

The results obtained by GRNGen are compared with the ones returned by three algorithms that have been studied in [Cantone et al., 2009]: BANJO (Bayesian network) [Yu et al., 2004], NIR and TSNI (ordinary differential equations) [Della Gatta et al., 2008, Gardner et al., 2003]. For the sake of completeness, it is necessary to point out that also a fourth algorithm was studied in [Cantone et al., 2009], i.e. ARACNE (information theoretic). Nevertheless, the authors of [Cantone et al., 2009] state that this algorithm cannot be used for time series. For this reason, the results of the ARACNE algorithm are not reported here.

The work presented in this thesis, as stated above, is focused on identifying the interactions between genes, building the topology of the network and simulating its dynamic. So, for the moment, the network constructed in this manner neither distinguish between activatory and inhibitory links, nor between protein-protein interactions and other kinds of interactions. As suggested in section 7.8 the particular operators used by the activation functions found by GP could help to make this distinction in future.

The IRMA network used as a test case, instead, presents some of these elements. This has, of course, an impact on the PPV and Se values. For this reason, the PPV and Se of the networks reported in [Cantone et al., 2009] for BANJO, NIR and TSNI have been recalculated in such a way that inhibitory and activatory links and protein-protein interactions and other kinds of interactions have not been distinguished among them. For this reason, the results reported here are different from the ones reported in [Cantone et al., 2009].

The comparison of the results returned by GRNGen and BANJO, NIR and TSNI on the *switch off* data is reported in Table 7.1, while the comparison on the *switch on* data is reported in Table 7.2 (results of methods NIR and TSNI have been reported in the same column, given that these two methods have allowed to reconstruct the same networks, as shown in [Cantone et al., 2009]).

Looking at the results obtained on the *switch off* data (Table 7.1), we can see that the best, median and average values found by GRNGen outperform BANJO, NIR and TSNI both for PPV and Se. Looking at the results ob-

	BANJO	**NIR & TSNI**	**GRNGen (best)**	**GRNGen (median)**	**GRNGen (avg.)**	**GRNGen (std.dev.)**
PPV	0.60	0.60	0.80	0.66	0.66	0.097
Se	0.42	0.42	0.75	0.62	0.58	0.081

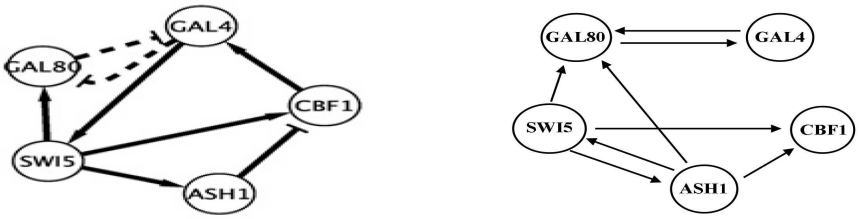
Table 7.1: PPV and Se values returned by the considered methods on the *switch off* data. Results of BANJO, NIR and TSNI calculated on the networks found by these methods as reported in [Cantone et al., 2009].

	BANJO	**NIR & TSNI**	**GRNGen (best)**	**GRNGen (median)**	**GRNGen (avg.)**	**GRNGen (std.dev.)**
PPV	0.33	0.75	0.80	0.71	0.68	0.10
Se	0.25	0.42	0.75	0.56	0.57	0.084

Table 7.2: PPV and Se values returned by the considered methods on the *switch on* data. Results of BANJO, NIR and TSNI calculated on the networks found by these methods as reported in [Cantone et al., 2009].

tained on the *switch on* data (Table 7.2), we can see that the best PPV and Se values found by GRNGen outperform BANJO, NIR and TSNI.
Median and average PPV values are outperformed by NIR and TSNI, while they outperform BANJO.
Finally, median and average Se values found by GRNGen outperform all methods studied in Ref. [Cantone et al., 2009].

The target network and one of the networks found by GRNGen (in particular, the one with the best Se on the *switch off* data found over the performed 30 runs) are reported in Figure 7.11.



(a) (b)

Figure 7.11: (a): Model of the regulatory interactions among genes in IRMA. Dashed lines represent protein-protein interactions. Directed edges with an arrow end represent activation, whereas a dash end represents inhibition. Figure taken from [Cantone et al., 2009]. (b): Network with the best Se on the *switch off* data found by GRNGen.

As pointed out above, one interesting characteristic of GRNGen is that, once the topology of the target network has been reconstructed, it is also able to simulate its dynamics.

The activation values of the different genes are initialized with the time zero values of the original time series and evolved applying the activation functions found by GP.
This simulation has been performed for all the 30 networks obtained by GRNGen on the *switch off* data and for all the networks obtained by GRNGen on the *switch on* data. For each of these networks, at each time step, the mean squared error has been computed between the activation values obtained by this simulation and the ones that are contained in the original time series, calculated over all genes. Finally the average of these averages has been performed over the 30 runs.

What has been obtained is one average error at each time step for the *switch off* data and one for the *switch on* data. These results are reported in Figure 7.12.

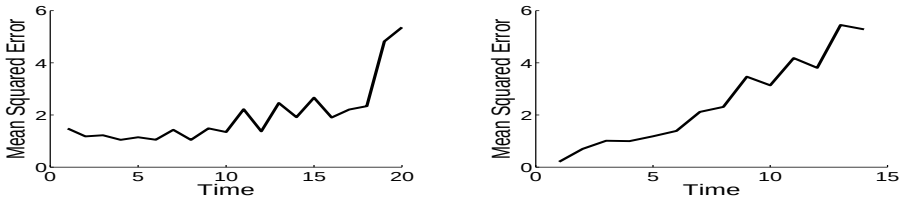
For the *switch off* data (Figure 7.12(a)), we can see that the average error remains limited (smaller than 2) and more or less constant in the first 10 time steps, while it slightly grows in the subsequent 10 steps, however without reaching the value of 6.

For the *switch on* data (Figure 7.12(b)), the error plot is constantly increasing, but also in this case the average error does not reach the value of 6 in the first 15 instants.

A comparison between these errors and the ones obtained by simulating the networks found by BANJO, NIR and TSNI is impossible, given that BANJO, NIR and TSNI are only able to reconstruct the topology of the target network, without generating the activation functions of the different genes. Thus, the dynamics of the networks reconstructed by BANJO, NIR and TSNI cannot be simulated. On the other hand, GRNGen is able to reconstruct the topology and the activation functions "in one shot", thus allowing to simulate the network dynamics.

7.8 Discussion

The application of GRNGen to large genetic networks will require some preprocessing of the data since the regression problem is underdetermined in



(a) (b)

Figure 7.12: Average mean squared error of the gene activation values at the different time steps obtained by simulating the networks reconstructed by GRNGen for the *switch off* (a) and *switch on* (b) data.

the likely scenario in which the number of genes is much larger than the number of available time-points. This problem could be solved either by limiting the genes involved in the activation function to known transcription factors or by preprocessing the expression data to identify clusters of coexpressed genes and then running GRNGen on the clusters rather than on the genes, based on the reasonable assumption that coexpressed genes should have similar activation functions.

In order to approach this aim, a new cluster kernel method for gene expression time series is studied in this thesis and is presented in the next chapter. It is useful for the execution of a preliminary clustering step [Sheng et al., 2005] before running the method presented here, in which genes are grouped into a reasonable number of clusters. Genes belonging to the same clusters should present similar behaviours and thus could have similar activation functions.

Many other issues could be considered in the future to improve and refine the system presented here. First of all, the system could be extended to distinguish activatory from inhibitory connections. It is reasonable to say that this can be done easily, given that the activation functions found by GP are constructed using the operators + and −: for instance, a majority of + symbols in the expressions could represent activation, while a majority of − symbols could represent inhibition. Furthermore the full power of GP as a regression method can be better exploited by considering non-linear activation functions that could describe threshold and saturation effects in gene regulation.

Chapter 8

An Application of Kernel Methods to Gene Cluster Temporal Meta-Analysis

8.1 Cluster Meta-Analysis: Problem Context

Real application of medical decision support systems and gene regulatory networks, as stated in the previous chapters, needs data pre-processing in order to perform a dimensional reduction of data manageable by actual systems, as the ones presented in this thesis.
Cluster analysis is one of the most widespread techniques used in the pre-processing phase of gene expression data investigation. Following a very brief introduction to this concept is given, that, far from being a complete description, has the unique scope to introduce the reader to the concepts expressed later.

Cluster analysis is a set of methodologies for automatic classification of samples into a number of groups using a similarity measure, so that the samples in one group are similar and samples belonging to different groups are not similar. The input to a cluster analysis algorithm is a set of samples and a measure of similarity (or dissimilarity) between two samples. The output of cluster analysis is a number of groups (clusters) that form a partition, or a structure of partition, of the data set. One additional result of cluster analysis is a generalized description of every cluster, and this is especially important for a deeper analysis of the dataset's characteristic. Samples for clustering are represented as a vector of measurements, or more formally, as a point in a multidimensional space.

Clustering is a very difficult problem because data can be organized in clusters with different shapes and size in a n-dimentional data space. To make the problem more difficult, the numbers of clusters in the data often

depends on the resolution (fine vs. coarse) at which we view the data.
Of course, the process of clustering points in the Euclidean 2D space is quite straightforward because clusters can be recognized by sight. For a set of points in a higher-dimensional Euclidean space, instead, we cannot recognize clusters visually, especially for gene expression time course microarray experiments, that generally comprise tens of thousands of transcripts for a given cellular extract. Accordingly, we need an objective criterion for clustering, as the one presented in this thesis.

To describe this criterion, a more formalized approach in describing the basic concepts and the clustering process is introduced later.

The input to a cluster analysis algorithm can be described as an ordered pair (X,s), or (X,d), where X is a set of samples, and s and d are measures for similarity or dissimilarity (distance) between samples, respectively. Output of the clustering system is a partition $\Lambda = \{G_1, G_2, \ldots, G_N\}$ where G_k, $k = 1,\ldots,N$ is a crisp subset of X such that:

$G_1 \cup G_2 \cup \cdots \cup G_N = X$, and
$G_i \cap G_j = \emptyset, i \neq j$

The members $G_1, G_2, \ldots, G_N$ of Λ are called *clusters*.
It is important to mention that there is no clustering technique that is universally applicable in uncovering the variety of structures present in multidimensional datasets [Kantardzic, 2003]. In fact, the cluster kernel method based on Gene Ontology presented later, is conceived specifically for the application domain of time course microarray gene expression.

As stated in the first chapter, microarrays are one of the most successful and widespread technologies in the field of gene expression studies, enabling the parallel measurement of thousands of transcripts for a given cellular extract [Eisen et al., 1998, Lockhart and Winzeler, 2000].

The Gene Ontology (GO) project [Gene Ontology Consortium, 2006], provides an ontology of defined terms representing gene product properties. The ontology covers three domains: cellular component, the parts of a cell or its extracellular environment; molecular function, the elemental activities of a gene product at the molecular level, such as binding or catalysis; and biological process, operations or sets of molecular events with a defined beginning and end, pertinent to the functioning of integrated living units: cells, tissues, organs, and organisms.

So GO is used in this work to fulfill the requirements for the application domain of microarray of gene expression data, in order to perform a more accurate clustering of it.

The state-of-the-art of the clustering approach is to perform clustering, in order to group together genes with similar expression profiles across experiments, and then summarize the clusters by identifying the most prominent functional groups [1] [Eisen et al., 1998].

If the same experiment is sampled using different technological platforms, or if different experiments generate highly correlated data, it is informative to track similarities among different clusters. The most elementary solution to evaluate cluster similarity is to count the number of overlapping genes; however, this strategy is suboptimal when the clusters are generated from partially or non-overlapping gene sets, as can happen when different technological platforms are used. A more general and high-level approach is to compute similarity according to the functional profiles of the clusters; this approach is usually called *cluster meta-analysis*.

A specific case of cluster meta-analysis is encountered when analyzing time-course gene expression experiments. In this case the temporal dimension can be exploited to gather information about the dynamics of the biological system under observation. Most analysis setups for time-course experiments look at the overall time-span of the measurements, [Bar-Joseph, 2004, Ernst and Bar-Joseph, 2006], and may generate clusters spanning the full length of the time-course; consequently, genes are associated into a cluster only if their expression is coordinated for the entire duration of the experiment.

In this thesis a different approach is followed where gene expression experiments are analyzed by splitting the time course into shorter *time-windows*, as proposed in [Kleinberg et al., 2006, Ramakrishnan et al., 2005] and implemented in the GOALIE system [Antoniotti, 2007].

The key observation is that temporally localized (i.e., within a limited sequence of time-steps) relationships among genes are worth detecting.

If the time-course is split into shorter time-windows, and clusters are generated separately within each window, it is possible to concentrate on temporally localized gene relationships. In this framework [2] a cluster meta-

[1] Functionally-correlated genes are expected also to be expressed according to coordinated patterns.

[2] Of course, such a framework opens up several interesting problems, which will be

analysis method is required to detect further relationships among clusters belonging to consecutive time-windows. The result of this meta-analysis will be the *reconnection* of the clusters from different windows, in order to better visualize the temporal evolution of the system.

Clustering meta-analysis is a valuable resource for understanding complex microarray data-sets. In particular, segmenting time-course experiments enables one to identify local patterns; clustering meta-analysis enables one to track the differences and similarities among such patterns. A typical question answered by the proposed framework is the following: are two functional groups of genes constantly co-expressed along a time-course, or in related yet different experiments?

Finally, the overall cluster meta-analysis model and framework proposed have been evaluated in the context of a split time-course analysis, and tested comparing two stand-alone experiments.

8.1.1 Cluster Meta-Analysis: Problem Description

The most popular resource for functional profiling is Gene Ontology (GO) [Gene Ontology Consortium, 2006], a hierarchical controlled vocabulary for the annotation of gene function. GO is organized as a directed acyclic graph (DAG), in which the vertices are terms and arcs are IS-A or PART-OF relations. A team of curators regularly reviews the most reliable literature on gene function, in order to annotate genes using GO terms; the results are then stored in the Gene Ontology Database. The functional evaluation of a cluster is performed by testing every GO term for over-representation (also referred as *enrichment*) in the clustered genes, hence determining a significance score [Gene Ontology Consortium, 2006].

Available cluster meta-analysis methods do not take into account the GO structure [Doherty et al., 2006], as it is common to visualize the enrichment result as a flat list of terms. Nevertheless, a GO enrichment can be conveniently regarded as a labeled graph, where the significance score (usually a p-value) is associated to every term in the DAG. We argue that exploiting the GO structure can significantly improve the cluster meta-analysis.

To this end, in [Zoppis et al., 2007] the use of a *kernel function* [Ben-Hur and Noble, 2005, Schölkopf et al., 2004] is proposed in order to compute the similarity among the GO enrichment graphs of different

briefly discussed later on.

clusters. The graph similarity is computed by identifying common paths in the graphs, and then computing a more elementary path similarity score, exploiting the enrichment scores associated to the vertices. In order to evaluate the advantages of using a structured approach, cluster similarities are computed using an unstructured approach (based on the *Jaccard coefficient index* [Jaccard, 1901] for set comparison), and the comparative performance of the two competing methods is determined.

In this thesis, an improved evaluation framework for the numerical results is proposed, based it on an information-theoretic approach and it is published in [Antoniotti et al., 2010].

The proposed kernel function can be extended to other annotation ontologies, and applied to other gene-expression data-sets, hence constituting a generally-valid solution to the cluster meta-analysis problem.

8.2 Dataset Description and Pre-processing

8.2.1 Dataset Description

The Kernel Function proposed for clustering meta-analysis was tested on the following data-sets:

- Spellman's [Spellman et al., 1998] time-course microarray study of the yeast cell cycle;
- yeast sporulation [Chu et al., 1998] and diauxic shift [DeRisi et al., 1997], two short time-course microarray experiments.

These datasets can be regarded as an optimal testing yard for the kernel function, as they are well known in the bioinformatics community, and constitute a very well characterized biological system.

Spellman's Cell cycle Data-set

The original Spellman's cell cycle data-set is composed by three subsets (Cdc15, Alpha factor and Elutriation), obtained using different experimental techniques [3], and having different time-course length.

[3]The study of the cell cycle is carried out on a population of cells; therefore, it is mandatory to synchronize their cycles, in order to obtain coherent expression profiles. Cdc15, Alpha Factor and Elutriation are the three different techniques used by Spellman et al. for synchronization.

After extensive exploratory data analysis, it was decided to work on a subset of the Alpha (factor) data-set, comprising ten time-points [4]; the analyzed time-course covers a full round of cell cycle, during which the cell undergoes different stages of the cell cycle transcriptional program [5].

The clustering meta-analysis is required to evaluate whether functional groups maintain the same inter-association patterns over time, or what changes occur.

Yeast Sporulation and Diauxic Shift Datasets

Yeast sporulation and diauxic shift are two short time-course microarray experiments, composed by seven time-points each. They are too short for segmentation; nevertheless, the two cluster-sets referring to the full time-course can be productively meta-analyzed using the same framework introduced for split time-course experiments.

The two data-sets sample different yeast stress response programs, involving extensive structural and/or metabolic remodeling. Sporulation consists in the generation of reproductive spores after nutrient starvation, whereas diauxic shift consists in the switch from anaerobic to aerobic metabolism after glucose exhaustion.

The data-sets were used in their original form, without excluding any sample.

[4]The Cdc15 subset was not analyzed as some of its time-points presented systematic biases. Part of the Elutriation time-course was affected by heavy noise; as a matter of fact, relying on size sorting, this method does not grant accurate separation of different cell cycle stages. Initial time-points of the Alpha subset were discarded as dominated by the response to the Alpha factor, a yeast pheromone; final time-points were discarded because of de-synchronization, a phenomenon due to stochastic events, and to the typically asymmetric division of budding yeast. These features were studied using Principal Component Analysis, to identify global gene expression patterns, and looking at the profiles of cell cycle marker genes, such as Cyclins.

[5]The time-course starts with the cell in G2 phase, a growth phase preparatory for cell division; then it undergoes M phase, when cell division occurs; the next phase, G1, is again a growth phase; the S phase is devoted to DNA replication. The execution of different activities (e.g. DNA condensation required for chromosome segregation in the M phase, versus DNA replication) requires different programs of transcriptional regulation; hence the rationale for studying the patterns of association and dissociation of functional groups.

8.2.2 Clustering and Data Pre-processing

The cell cycle data-set was split into 6 overlapping windows, each composed by 5 subsequent time-points. Clustering was performed on each of the cell cycle windows, on the sporulation and on the diauxic datasets, using the CLICK algorithm [Sharan and Shamir, 2000].

The algorithm utilizes graph-theoretic and statistical techniques to identify tight groups of highly similar elements (kernels), which are then expanded into the full clusters. CLICK was picked as it was proved superior to K-means, both in terms of cluster homogeneity and separation. The euclidean distance was used as dissimilarity measure, after fixed-norm scaling of the gene profiles.

The CLICK homogeneity target value was set in a range between 0.1 and 0.4, in order to obtain a comparable number of clusters for every window (in a range between 3 and 4).

The number of genes in a cluster ranges between 100 and 1500.

Clustering failed in cell cycle window 3, as only one cluster was generated with all possible values of target homogeneity; in addition, a large part of the genes (more than one third) was discarded as outliers.

Therefore, the cell cycle window 3 is neglected during the evaluation of the Jaccard's index and the kernel comparative performances [6].

Functional enrichment profiles were generated for every cluster. Functional Enrichment p-values were calculated according to Fisher's exact test.

These p-values reflect the significance, compared to the random expectation, of the set size obtained by intersecting the genes grouped into a certain cluster with the genes annotated under a certain GO term.
In the bioinformatics community, this procedure is usually termed *functional enrichment* or *over-representation analysis*.

It is common to pre-filter Gene-Ontology before computing enrichment profiles. A common choice is to exclude terms that are too small, for several reasons:

- multiple test correction burden;
- extremely narrow biological scope;

[6] Window 3 is centered on the M phase of the cell cycle, and is thus possible that the observed phenomenon is due to significant transcriptional reprogramming of gene expression after cell division.

- higher instability of statistical test outcome;

In this work, GO terms annotating less than 4 genes were excluded from the analysis. The p-value of GO terms with less than 5 genes in a cluster was arbitrarily set to 1 (not significant), in order to reduce their leverage in the similarity computation.

8.3 Graph and Kernels Description

First of all, the entities manipulated later are described in this section, using standard mathematical notation.

A *graph* $G = (V,E)$ consists of a finite set of nodes (or vertices) V and a set of edges $E \subseteq V \times V$.

A *path* of length $k-1$ in G is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ where $(v_{i-1}, v_i) \in E$ for all $1 < i \leq k$.

Definition 8.1 ((Labeled) GO graph)**.** *A* GO Graph *is a graph* $G(V,E)$ *with a double label for its vertices defined by* $\mathsf{label}_{GO} : V \to GO$ *and* $\mathsf{label}_p : V \to [0,1]$ *where* GO *is the set of Gene Ontologies terms and* $\mathsf{label}_p(v)$ *is the p-value of statistical significance associated to the term* $\mathsf{label}_{GO}(v)$*.*

With a slight abuse of terminology, a random variable is referred to as a function $X : \Omega \to \mathcal{X}$ where Ω is the sample space and $\mathcal{X}$ is a discrete set. When X has distribution $P_X(x)$ then the (Shannon) entropy of X is given by

$$H(X) = \sum_{x \in \mathcal{X}} p_x \log(1/p_x).$$

In order to measure the average amount of information conveyed when an outcome of a conditional random variable is observed, the conditional entropy is also used in the following. It is defined by

$$H(X|Y) = \sum_{y \in \mathcal{Y}} p_y \sum_{x \in \mathcal{X}} p_{x|y} \log(1/p_{x|y})$$

for some Y taking values in $\mathcal{Y}$.

8.3.1 Kernel Functions Review

Kernel methods are a family of algorithms that, thanks to their theoretical potential, have been applied successfully to a variety of problems. The main

idea is to map any input set $\mathcal{X}$ into a new *feature* space (generally a Hilbert space) $\mathcal{F}$ in order to find there some suitable hypothesis: in this way complex relations in $\mathcal{X}$ can be simplified and more easily discovered. The "feature map" Φ used for this task is defined by a *kernel function* k which allows to compute the inner product in $\mathcal{F}$ using only objects of the input space, hence without "computing" Φ. This is sometimes referred as the *kernel trick*.

Definition 8.2 (Kernel function)**.** *A kernel is a function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *capable of representing through* $\Phi : \mathcal{X} \to \mathcal{F}$ *the inner product of* $\mathcal{F}$ *i.e.*

$$k(x,y) = \langle \Phi(x), \Phi(y) \rangle. \tag{8.1}$$

To ensure that such equivalence exists a kernel must satisfy Mercer's Theorem [Courant and Hilbert, 1953]. Hence, under certain conditions, by fixing k one is able to ensure the existence of a mapping Φ and a Hilbert space $\mathcal{F}$ for which (8.1) holds. Therefore these functions can be interpreted as *similarity measures* of data objects in the new feature space [Schölkopf and Smola, 2002].

Graph Kernels

A common strategy adopted to formulate similarity measures for structured objects is to assemble kernel functions operating on the object parts; in the case of graphs, various measures based on the count of *matching labeled paths* have been proposed [Gärtner et al., 2003, Kashima et al., 2003, Kondor and Lafferty, 2002].

Given a pair of nodes with matching labels, the (kernel) similarity between two random paths is the product of more elementary similarity values, which are defined on the node and edges encountered along the path.

The kernel value of two graphs is then the sum over the kernel values of all pair of paths within these two graphs:

$$k_{\text{graph}}(G_1, G_2) = \sum_{path_1 \in G_1} \sum_{path_2 \in G_2} k_{\text{path}}(path_1, path_2), \tag{8.2}$$

where the membership notation $w \in G$ is abused in order to denote a "path" w in the graph G.

An elegant approach to construct such a similarity measure uses the direct product graph [Gärtner et al., 2003]:

Definition 8.3 (Direct product of two labeled graphs)**.** *Given two labeled graphs $G_1 = (V,E)$, $G_2 = (W,F)$ the direct product is denoted by $G_1 \times G_2$. The vertex set $V_\times$ and edge set $E_\times$ of this direct product are respectively defined as:*

$$\begin{aligned} V_\times(G_1 \times G_2) &= \{(v_1,w_1) \in V \times W : \mathsf{label}(v_1) = \mathsf{label}(w_1)\}, \\ E_\times(G_1 \times G_2) &= \{((v_1,w_1),(v_2,w_2)) \in V_\times^2(G_1 \times G_2) : \\ &\qquad (v_1,v_2) \in E \wedge (w_1,w_2) \in F \wedge \mathsf{label}((v_1,v_2)) = \mathsf{label}((w_1,w_2))\}, \end{aligned} \tag{8.3}$$

where $\mathsf{label}(\cdot)$ *is an uninterpreted "labeling" function for vertices and edges* [7]*.*

With this definition the *random path kernel* is computed as follows

Definition 8.4 (Random Path Kernel)**.**

$$k_\times(G_1,G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda_n A_\times^n \right]_{i,j} \tag{8.4}$$

where $A_\times$ is the adjacency matrix of the product graph [8].

The sum in (8.4) converges for a suitable choice of $\lambda_0, \lambda_1, \lambda_2 \ldots$ [Gärtner et al., 2003]. Here, the random walk kernel is computed only for walks up to a predetermined length (i.e. by truncating the summation for walks ≥ 7).

Function (8.4) is redefined in [Borgwardt et al., 2005] to measure similarities between paths (up to a fixed lengths) that are not identically labeled since the application has to cope with this case (i.e. the comparison between GO graphs having almost different labeled paths).

Note that the work is performed actually with label couples (GO term label, p-value label); the GO term label needs to be matching, whereas the p-value label does not need to be matching. This distinction will be

[7] The $\mathsf{label}(\cdot)$ function will be eventually re-interpreted for the needs of this work on the basis of Definition 8.1.

[8] The applicability of this elegant formulation is limited by the high polynomial time complexity, a problem particularly significant with large data sets. Nonetheless, some speed up methods are available [Vishwanathan et al., 2007]. Moreover, the *tottering problem* [Borgwardt and Kriegel, 2005] may lead to high similarity score even when only small identical substructures are present. This phenomenon is due to the existence of cycles, which can cause the iterated visit of the same node sequence. In our case, tottering does not occur, as the graphs we treat do not contain cycles.

implemented in the kernel function definition.

Definition 8.5. *Given two graphs* $G_1 = (V,E)$ *and* $G_2 = (W,F)$ *and two paths* $path_1 = (v_1, v_2, \ldots, v_n) \in G_1$ *and* $path_2 = (w_1, w_2, \ldots, w_n) \in G_2$, *with* $v_i \in V$, $w_i \in W$, *the path kernel is defined as*

$$k_{\text{path}}(path_1, path_2) = \prod_{i=1}^{n-1} k_{\text{step}}((v_i, v_{i+1}), (w_i, w_{i+1})) \tag{8.5}$$

for each i *and* j.

The random kernel is still the sum over all kernels on pairs of path and can be computed with the adjacency matrix:

$$[A_\times]_{((v_i,w_i),(v_j,w_j))} = \begin{cases} k_{\text{step}}((v_i,v_j),(w_i,w_j)) & \text{if } ((v_i,v_j),(w_i,w_j)) \in E_\times \\ 0 & \text{otherwise} \end{cases} \tag{8.6}$$

with $E_\times = E_\times(G_1 \times G_2)$ and $(v_i, v_j) \in E$ and $(w_i, w_j) \in F$.

8.3.2 Kernel for GO graph

Based on the ideas stated in [Zoppis et al., 2007] a similarity score capable of taking into consideration the graph structure nature of GO terms was applied.

In [Merico et al., 2007] an evaluation criterion to compare the performance of this kernel approach versus an unstructured one (based on Jaccard's index) is proposed. In order to improve on such result, the information-theoretic model presented above is developed; more specifically, the amount of residual uncertainty after applying the two different methods (Kernel and Jaccard's index) is considered.

More formally, the goal is to infer a relation $\mathcal{L}_{1,m} \subseteq S_1 \times S_2 \times \ldots \times S_m$ among sets S_i of all possible clusters $\mathcal{C}_{i,u}$ (with $1 \leq u \leq |S_i|$) of gene expressions. A tuple in $\mathcal{L}_{1,m}$ represents a sequence of functionally homogeneous clusters.
For each cluster $\mathcal{C}_{i,u}$ a labeled GO graph $G_{\mathcal{C}_{i,u}}$ is computed by assigning the GO term code $\mathsf{label}_{\mathsf{GO}}(v)$ and the enrichment p-value $\mathsf{label}_p(v)$ to each graph vertex v; in other words, the input set $\mathcal{X}$ can also be represented by the space of all labeled GO graphs. The enrichment p-value is an index of statistical significance, quantifying the relevance of a GO term within a specific cluster

(as in, for example, [Beißbarth and Speed, 2004]).
GO graphs are then filtered, excluding terms with non-significant p-values, in order to avoid noise in distance computation. By neglecting these vertices, we work on potentially disconnected GO sub-graphs.

Following is described in details how the kernel function is structured.

The start is as in [Borgwardt et al., 2005] by using (8.6) as adjacency matrix. In order to compare different paths, the *steps* are compared in the following way.

Definition 8.6 (Step kernel for GO graphs)**.** *For $i = 1,\ldots,n-1$, the* step kernel k_{step} *is defined as*

$$\begin{aligned} & k_{\text{step}}((v_i, v_{i+1}), (w_i, w_{i+1})) \\ = \;& k_{\text{pv}}(v_i, w_i) \times k_{\text{term}}(v_i, w_i) \times k_{\text{pv}}(v_{i+1}, w_{i+1}) \times k_{\text{term}}(v_{i+1}, w_{i+1}) \end{aligned} \tag{8.7}$$

where $k_{\text{pv}}(v_i, w_i) = \max(0, c - |\text{label}_p(v_i) - \text{label}_p(w_i)|)$ [9] is the Brownian bridge kernel [Schölkopf and Smola, 2002] and k_{term} is a Dirac function, i.e.,

$$k_{\text{term}}(v_i, w_i) = \begin{cases} 1 & \text{if label}_{\text{GO}}(v_i) = \text{label}_{\text{GO}}(w_i) \\ 0 & \text{otherwise.} \end{cases} \tag{8.8}$$

Armed with these definitions, the procedure for inferring $\mathcal{L}_{1,m}$ is now presented. It is done in the section 8.3.3, by defining a distance function based on the kernel $k_{\times}$.

8.3.3 Connection Selection Algorithm

The set of tuples in $\mathcal{L}_{1,m}$ is iteratively inferred by using the following definition.

Definition 8.7. *Given $\mathcal{L}_{1,m} \subseteq S_1 \times S_2 \times \ldots \times S_m$ and $\mathcal{L}_{m,m+1} \subseteq S_m \times S_{m+1}$, the aggregation $\mathcal{L}_{1,m} \circ \mathcal{L}_{m,m+1}$ is given by:*

$$\mathcal{L}_{1,m} \circ \mathcal{L}_{m,m+1} = \{(\mathcal{C}_{1,u_1}, \mathcal{C}_{2,u_2}, \ldots, \mathcal{C}_{m,u_k}, \mathcal{C}_{m+1,u_s})\}$$

where $1 \leq u_k \leq |S_i|$, $1 \leq u_s \leq |S_{i+1}|$ and

$$\exists\, \mathcal{C}_{m,u_k} t.c. (\mathcal{C}_{1,u_1}, \mathcal{C}_{2,u_2}, \ldots, \mathcal{C}_{m,u_k}) \in \mathcal{L}_{1,m} \wedge (\mathcal{C}_{m,u_k}, \mathcal{C}_{m+1,u_s}) \in \mathcal{L}_{m,m+1}$$

[9] c is fixed in order to accounting only for distance lower then 0.1 (i.e $c = 0.1$), This setting reflects a non-similarity (i.e. $K_{\text{pv}} = 0$) when p-values are more distant.

Therefore, given the kernel $k_\times$ (8.4) (with the adjacency matrix 8.6) one can always induce a (non Euclidean) distance $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $d^2(x,y) = k(x,x) + k(y,y) - 2k(x,y)$. Now, let the set of pairs in $\mathcal{L}_{i,i+i} \subseteq S_i \times S_{i+1}$ defined by:

$$\mathcal{L}_{i,i+1} = \{(\mathcal{C}_{i,u_k}, \mathcal{C}_{i+1,u_s}) | 1 \leq u_k \leq |S_i|, 1 \leq u_s \leq |S_{i+1}|\}$$

where each pair is either

$$(\mathcal{C}_{i,u_k}, \mathcal{C}_{i+1,u_s}) = \underset{\substack{\mathcal{C}_{i,u_k} \in S_i \\ \mathcal{C}_{i+1,u_s} \in S_{i+1}}}{\text{argmin}} \{d(\mathcal{C}_{i,u_k}, \mathcal{C}_{i+1,u_s})\}$$

or a pair such that

$$\theta_L \leq d(\mathcal{C}_{i,u_k}, \mathcal{C}_{i+1,u_s}) \leq \theta_R$$

Where $\theta_L = m - \mathsf{IQR} * \gamma$, $\theta_R = m + \mathsf{IQR} * \gamma$, m being the median and IQR being the *Inter-Quartile Range*, both referred to the distribution over the distance d.

This is done in order to possibly take into account pairs that are still significant while avoiding to limit the attention to those that minimize $d(.,.)$. In other words, by fixing $\mathcal{L}_{i,i+1}$, the first step was the identification of the most similar destination cluster $\mathcal{C}_{i+1,k}$, for every cluster $\mathcal{C}_{i,u} \in S_i$;
then the results are furthermore refined, by adding other pairs with a high similarity score, and removing those pairs with limited similarity.

Given this background, it is a simple matter to implement an iterative algorithm to compute the relevant relations $\mathcal{L}_{1,m}$ (cfr. Algorithm 1).

Algorithm 1 Compute the relevant relations $\mathcal{L}_{1,m}$

1: $Z_{1,2} := \mathcal{L}_{1,2}$ // Initialization step.
2: **for** $i = 2$ to m **do**
3: $\quad T := \mathcal{L}_{i,i+1}$
4: $\quad Z_{1,i} := Z_{1,i} \circ T$
5: **end for**

8.3.4 Clustering Quality Score

The focus of this section is the comparative evaluation of connections among clusters of consecutive windows determined using different methods. Un-

der an information-theoretical perspective, establishing connections among clusters of consecutive time-windows modifies the uncertainty of the system. Connections that lower the uncertainty are better. This idea can be suitably exploited to evaluate the comparative performance of the kernel function versus the Jaccard's index. The first part of this section provides a qualitative description of the score; the second part is devoted to a more rigorous definition.

In order to evaluate the uncertainty, the distribution of its gene counts over the clusters of two consecutive windows is considered, for every GO term; if the distribution is non-uniform, i.e. with low uncertainty, that implies the term is mostly represented in a limited number of clusters, and not all over them.

In order to determine the uncertainty reduction induced by establishing a connection between two clusters, the connected clusters are merged, and the corresponding uncertainty after the merge are evaluated. Indeed, if the connected clusters are functionally homogeneous, the gene count is expected to significantly increase in the merged cluster, for all the GO terms represented in the original clusters; as a consequence, for these GO terms the gene count distribution becomes less and less uniform, thus inducing a reduction of the entropy value.

Since the entropy associated to the disconnected setting is the same for all methods, comparing the entropy difference associated to a connection, and comparing the absolute entropy associated to the setting with that connection, lead to equivalent results. Therefore, the quality score is constructed considering only the *absolute* entropies associated to each single connection.

As a further refinement, the quality score for each term separately are not computed; instead, it is averaged over the union of size-one neighborhood of GO terms. To induce the neighborhoods, all GO terms having a significant enrichment score in the clusters are used. The rationale of this choice is to restrict the uncertainty estimation to GO terms important for the functional profiles: these are the very genes driving the establishment of cluster connections in both the methods. We now proceed with a more formal treatment: let's consider a pair of consecutive windows (S_i, S_{i+1}). The clusters in the window pair belong to the set $P_{i,i+1} = S_i \cup S_{i+1}$. Let's also consider the set of GO terms $\mathsf{GO}_{i,u}$, which is the set of *nearby significant* terms in a GO graph – the actual definition of $\mathsf{GO}_{i,u}$ will be given shortly. It is then

possible to introduce the conditional distribution $f_{C|G}(c|g)$, representing the count number of 'time-localized' genes associated to the specific GO term g, over the different clusters C; C and G are two random variables taking values respectively on $P_{i,i+1}$ and $\mathsf{GO}_{i,u}$. Therefore it is also possible to consider the associated entropy function:

$$H(f(C|G=g)) = \sum_{c \in P_{i,i+1}} f(c|g) \log(1/f(c|g)). \tag{8.9}$$

When the clusters are disconnected, $H(f(C|G=g))$ can be thought as an index of the background uncertainty, which clearly assumes the same value whatever method is used to compute the cluster connections.

Time-localized genes are specifically introduced in order to treat the same gene in different time-windows as a different entity. As a matter of fact, when cluster connections $(\mathcal{C}_{i,u}, \mathcal{C}_{i+1,k}) \in \mathcal{L}_{i,i+1} \subseteq S_i \times S_{i+1}$ are introduced, it is possible, for each connection, to merge the corresponding clusters. This operation modifies $P_{i,i+1}$ from

$$P_{i,i+1} = S_i \cup S_{i+1}$$

to

$$P_{i,i+1} = (S_i \backslash \mathcal{C}_{i,u}) \cup (S_{i+1} \backslash \mathcal{C}_{i+1,k}) \cup \{\tilde{\mathcal{C}}_{i,i+1}\},$$

where $\mathcal{C}_{i,u}$ and $\mathcal{C}_{i+1,k}$ are the connected clusters, and $\tilde{\mathcal{C}}_{i,i+1} = \{\mathcal{C}_{i,u} \cup \mathcal{C}_{i+1,k}\}$. Thanks to the use of time-localized genes, the size of $\tilde{\mathcal{C}}_{i,i+1}$ is equal to the sum of the sizes of $\mathcal{C}_{i,u}$ and $\mathcal{C}_{i+1,k}$, a property which enables a more meaningful treatment of the conditional entropy [10]. After the merging operation, it is possible to recompute $H(f(C|G=g))$ using the new value of $P_{i,i+1}$. We interpret a low value of $H(f(C|G=g))$ as a successful reduction of the system uncertainty after the establishment of the cluster connection.

The most represented GO terms are expected to be more relevant for the entropy evaluation.Therefore, firstly are looked for the "most significant" GO terms (using the label_p values) and then are looked at their neighboring ones in the GO graph. More formally, the final score used for the comparative

[10] If the time-localized genes are not introduced, the reduction of entropy is greater when different genes supporting the same term are merged, compared to when the same genes supporting the same term are merged; this different behavior is not meaningful: on the contrary, the latter situation is expected to be associated to an even more consistent reduction of the uncertainty.

performance evaluation is computed only on the GO terms subset $\mathsf{GO}_{i,u}$, which is defined as:

$$\mathrm{GO}_{i,u} = \bigcup_{v \in V(G_{C_{i,u}}):\mathsf{label}_p(v) \leq \delta} O_v.$$

where $V(G_{C_{i,u}})$ and $E(G_{C_{i,u}})$ are vertices and edges of the GO graph associated to the cluster $C_{i,u}$, $O_v = \{\mathsf{label}_{\mathsf{GO}}(v_i) \mid v_i \in N[v]\}$, $N[v]$ is the closed neighborhood of a vertex v, i.e., $N[v] = \{v\} \cup N(v)$ and $N(v) = \{v_i \mid (v, v_i) \in E(G_{C_{i,u}})\}$. Therefore, the final formulation of the entropy becomes:

$$\mathbf{I} = \frac{1}{m-1}\left[\sum_{i=1}^{m-1} \frac{1}{|\mathcal{L}_{i,i+1}|}\left[\sum_{(C_{i,u}, C_{i+1,k}) \in \mathcal{L}_{i,i+1}} H(C|G)\right]\right] \tag{8.10}$$

where $1 \leq u \leq |S_i|$ and $1 \leq k \leq |S_{i+1}|$ and

$$H(C|G) = \sum_{g \in \mathsf{GO}_{i,u}} f_G(g) H(f(C|G = g)),$$

i.e. the conditional entropy of clusters given gene distributions.

This result is compared with the score obtained from a method which simply treats cluster enrichments as a flat lists of GO terms. In this case, firstly those terms whose p-values were below a detection threshold are removed from each cluster. Then, the Jaccard's index distance is applied:

$$J(C_{i,u}, C_{i+1,v}) = 1 - \frac{|C_{i,u} \cap C_{i+1,v}|}{|C_{i,u} \cup C_{i+1,v}|}.$$

8.3.5 Kernels and Algorithms Application Results

The procedures described in the previous sections are implemented with four computational modules developed in Matlab 7: functional enrichment, kernel, Jaccard, contingency. The connections among the clusters of the yeast cell cycle and yeast sporulation / diauxic shift time course data are determined using both the kernel function and the Jaccard's index (p-value threshold set to 0.05). For every pair of consecutive windows (or cluster pairs in case of sporulation/diauxic data), the quality score defined in Section 8.3.4 is computed; in the cell cycle data, pairs formed by window 3 (2, 3 and 3, 4) were excluded from the computation, as previously discussed in Section 8.2.2; in addition, the presence of one single cluster in window 3 in yeast, would induce a zero entropy. The results are displayed in Table 8.1. Clearly, the performance of the kernel function is systematically

Yeast cell cycle: quality score	win 1, 2	win 4, 5	win 5, 6
Jaccard	1.038	0.873	1.138
Kernel	0.951	0.856	1.078

Sporulation/Diauxic: quality score	Cluster Pairs
Jaccard	0.0670
Kernel	0.0670

Table 8.1: The comparison of the quality scores obtained for the connections induced by the kernel function and the Jaccard's index. Every column represents a window pair (or a pair of cluster sets in case of Sporulation/Diauxic data).

better than the performance of the Jaccard's index for yeast cell cycle while the performance for the sporulation/diauxic pair are equivalent.

In addition, the biological significance of the results induced by the kernel function is checked. For the window pair (1, 2), the meta-analysis is performed manually: for every cluster, the gene ontology terms is selected passing a relatively stringent threshold (p-value threshold set to 0.01); then the enriched terms are grouped into aggregated and biologically homogeneous groups, considering the graph structure and additional biological knowledge [11]; the connections were eventually determined considering the matching of the aggregated groups, and sorted into high-confidence and low-confidence. The strategy adopted for the manual method does not strictly reproduce neither the Jaccard's index, nor the kernel function, avoiding circularity issues. The results are displayed in figure 8.1. All the high-confidence connections manually determined overlap totally with the kernel function results (three out of three connections), whereas the overlapping with the Jaccard coefficient results is poor (none of the manually determined, high-confidence connections overlaps with the Jaccard-determined ones).

[11] There are well known relations between GO terms which are not rendered by the graph; most of these relations occur among terms belonging to the three GO partitions, Molecular Function, Biological Process and Cellular Component

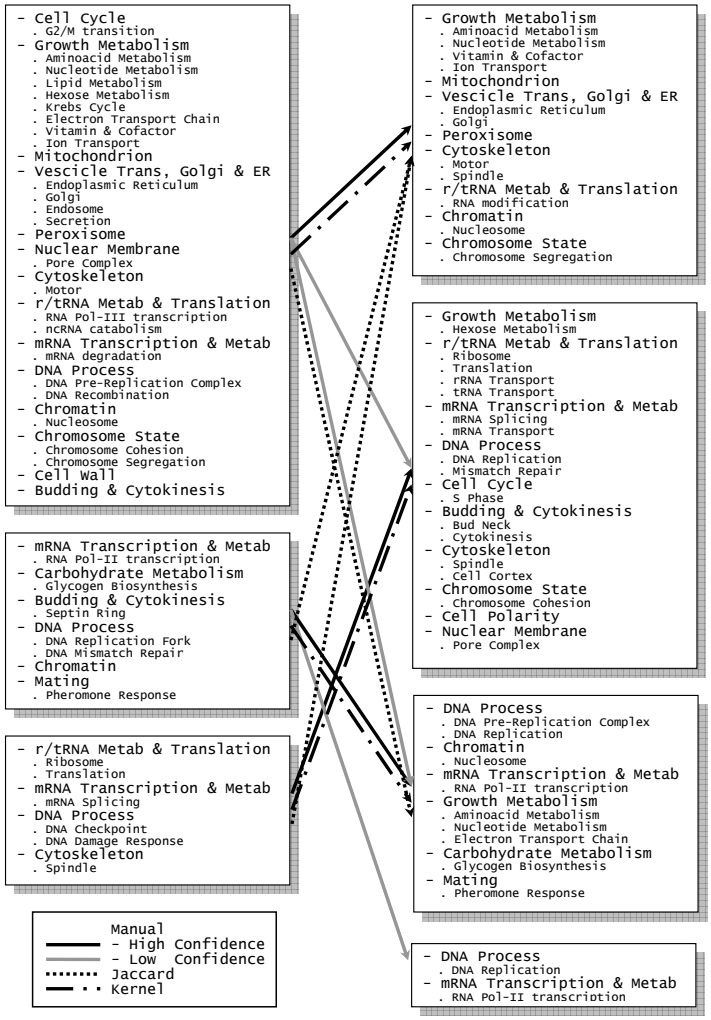


Figure 8.1: The results of the cluster meta-analysis for windows 1 and 2 of the yeast cell cycle data. The boxes represent the different clusters, and the manually determined GO term groups are used to summarize their respective GO enrichment profiles. The connections were determined using the manual method (solid line), the kernel function (dash-dotted line), and the Jaccard's index (dotted line); in particular, manually determined connections were sorted into high-confidence (black) and low confidence (gray).

PART IV

FINAL CONSIDERATIONS

Chapter 9

Conclusions and Assessments

9.1 Medical Decision Support System for Survival Prediction in Breast Cancer using a binary dataset

The goal of the investigation on the 70-gene signature was to refine the set of criteria that could lead to an individualized diagnosis of variations of Breast Cancer. To reach this goal the study was based on the well known "70-genes signature" and led to the development of a Medical Decision Support System based on GP able to predict the outcome of the therapy predicting the survival of breast cancer patients. It is also able to perform an automatic feature selection, giving the possibility of generating biological insights and hypotheses that are intrinsic to the method.

Subsequently, several machine learning schemes, such as Support Vector Machines, Multilayered Perceptron, Random Forest, Radial Basis Function Network and Voted Perceptron, have been applied to the 70-genes signature dataset, in order to perform a comparison between their respective performances. Some simplifying assumptions were made, the data were pre-processed accordingly and several evaluation experiments were run.

The results showed that while all the machine learning algorithms that have been used do have predictive power in classifying breast cancer patients into risk classes, GP-MDSS clearly outperforms all other methods. Of course there is no way to do such a comparison in a completely unbiased way, as one could always argue that the numerous parameters used by the different methods have been set to favour one of them. To minimize bias as much as possible, default implementation and parameters setting of all methods was used. The fact that all methods other than GP-MDSS had very similar performances suggests that GP-MDSS is indeed the most promising method.

The improvement in performance shown by GP-MDSS compared to the original scoring method presented in [van't Veer et al., 2002,

van de Vijver et al., 2002] was rather small and not statistically significant. As expected, the scoring method was superior to all machine learning algorithms in minimizing false negatives.

9.2 Medical Decision Support System for Survival Prediction in Breast Cancer using floating point dataset

The study of the survival prediction of breast cancer patients based on the binarized 70-gene signature was preliminary, mainly because the validation dataset was preprocessed and all its features binarized in order to use logical operators for the GP functional nodes. If, an the one hand, this choice allowed a simple interpretation of the solutions from the biological viewpoint, on the other hand the binarization of data was limiting, since it caused a sizable loss of information.
To overcome this limitations, the use of the original floating point valued expression data was investigated, abandoning the binarization of the 70-genes dataset.

In the first phase of the investigation, it has been showed that all the machine learning algorithms used do have predictive power in classifying breast cancer patients into risk classes and GP-MDSS is outperformed by Support Vector Machines and Voted Perceptron in the minimization of the number of incorrectly classified instances and by Voted Perceptron in the minimization of false negatives. To minimize bias, in this first phase default implementation and parameter setting was used for all methods.

In the second phase, GP-MDSS was enriched by changing its fitness function so that false negatives (positives) were penalized more than false positives (negatives), hoping to tune the classifiers towards better sensitivity (specificity).
Some experiments were performed with different values of these weights and showed that, when larger weight is given to false negatives, GP-MDSS obtains results comparable to all other machine learning methods in the minimization of the number of incorrectly classified instances while being able to outperform the other machine learning methods in a statistically significant way in the minimization of the false negatives.

It is also pointed out that:

- GP-MDSS used on the original floating point valued expression data outperforms GP-MDSS used on binarized data (results shown in Chapter 5) both in the minimization of the incorrectly classified instances and in the minimization of the false negatives in a statistically significant way.
- The improvement in performance shown by GP-MDSS in minimizing false negatives compared to the original scoring method presented in [van't Veer et al., 2002, van de Vijver et al., 2002] was rather small and not statistically significant.

9.3 Generator of Gene Regulatory Network and Dynamic Simulator System

A new Gene Regulatory Network Generator (GRNGen) driven by data has been developed in this thesis.

Taking as input a dataset of temporal series of gene expression levels, the proposed system is composed of three macro-steps:

In the first step a regression dataset is built for each gene. The targets of this dataset are the activation values of that gene at all instants $t > 1$, while the features are the activation values of all other genes at the previous instants.

In the second step, the regression models are computed using the GP and interpreted as the activation function for that gene.

In the third step the network is reconstructed and simulated interpreting the previously generated regression models as the activation functions for the various genes.

The proposed system has been tested on the IRMA gene regulatory network defined in [Cantone et al., 2009].

This network contains five genes in the yeast *Saccharomyces cerevisiae*, regulating each other and has been defined with the explicit objective of being a simplified biological model to benchmark reverse-engineering approaches.

An experimental comparison has been performed between the ability of GRNGen to reconstruct the topology of the IRMA network and the three different well-established reverse engineering methods used in [Cantone et al., 2009]: BANJO, NIR and TSNI.

In this experimental study, the performance measures used are the same as in [Cantone et al., 2009]: Positive Predictive Value (PPV) and Sensitivity (Se).

The results found by GRNGen generally outperform BANJO, NIR and TSNI both for PPV and Se.

Even though this system is not yet able to distinguish between activatory and inhibitory connections and between protein-protein interactions and other kinds of interactions, it is able to simulate the dynamics of the network, a feature that BANJO, NIR and TSNI do not have.

9.4 An application of Kernel Methods to Gene Cluster Temporal Meta-Analysis

The application of GRNGen to large genetic networks will require some preprocessing of the data since the regression problem is underdetermined in the likely scenario in which the number of genes is much larger than the number of available time-points. This problem could be solved either by limiting the genes involved in the activation function to known transcription factors or by preprocessing the expression data to identify clusters of coexpressed genes and then running GRNGen on the clusters rather than on the genes, based on the reasonable assumption that coexpressed genes should have similar activation functions. To reach this aim a new cluster kernel method has been identified and developed, and a meta-analysis procedure for time-course gene expression experiments based on the time-windowing segmentation has been introduced.
The contributions are the following:

- An application of kernel methods to the initial problem of enriching clusters of genes has been developed, in order to take into account the hierarchical structure of the Gene Ontology, and the related problem of tracking relations of similarity among clusters, exploiting the hierarchical structure of their functional profiles (clustering meta-analysis).
- A new scheme for the ranking of various relationships between clusters across time-steps has been defined.
- A performance measure for the results of clustering meta-analysis has been identified and compared with the simpler and widely used Jaccard

index.

Chapter 10

Future Works

10.1 Medical Decision Support System for Survival Prediction in Breast Cancer

In Chapters 5 and 6, the Medical Decision Support System(GP-MDSS) for survival prediction in breast cancer has been described. It has been shown that this system outperforms other machine learning methods as a tool to extract predictions from an established breast cancer gene signature, the well known "70-genes signature". Since the method is able to generate biological insights and hypotheses, it deserves deeper investigation along the following three lines:

- The implementation of GP-MDSS was purposely not optimized, and substantial improvements in performance can be expected aimed at tuning the various GP parameters.
- Maybe more importantly, GP-MDSS can potentially offer biological insights and generate hypotheses for experimental work (see also [Yu et al., 2007]). Indeed, an important result of this analysis is that the trees produced by GP-MDSS tend to contain a limited number of features, and therefore are easily interpretable in biological terms.

Future work along these lines should therefore focus on both improving the performance of GP-MDSS and interpreting the results from the biological point of view.

Finally, it will be a future task to test the GP-MDSS approach on other features/gene sets that account for other kinds of cancer or other diseases, always with the objective of providing clinicians with more precise and individualized diagnosis criteria.

10.2 Generator of Gene Regulatory Network and Dynamic Simulator System

The Gene Regulatory Network Generator and Simulator System(GRNGen) presented in this thesis is rather preliminary and many issues could be considered in the future to validate and establish it. First of all, the system could be extended to distinguish activatory from inhibitory connections. This can be done easily, given that the activation functions found by GP are constructed using the operators + and −: for instance, a majority of + symbols in the expressions could represent activation, while a majority of − symbols could represent inhibition.

Moreover the full power of GP as a regression method can be better exploited by considering non-linear activation functions that could describe threshold and saturation effects in gene regulation.

The realization of this extension of the system is a hard and ambitious task and many issues still have to be understood in order to perform this extension effectively. This is one of the main objectives of the current research.

10.3 An application of Kernel Methods to Gene Cluster Temporal Meta-Analysis

The application of Kernel Methods to Gene Cluster Temporal Meta-Analysis presented in this thesis could be extended in several directions in the future. First of all there is the issue of how to semi-automatically evaluate the dependency of the presented methods on the clustering algorithm chosen (all our experiments have been conducted using the CLICK system) This opens up a vast search space where each potential clustering method yields a result as a function of some input parameters; while searching such space is very likely to be unwieldy, the problem points in the direction of studying various formulations as an optimization problem of such dependency.

Second, the method which leads to the most “informative” segmentation of a time course experiment could be determined. This problem is also formalizable as an optimization problem and a first discussion appears in [Kleinberg et al., 2007, Tadepalli et al., 2008].

Third, data from time-course experiments are not very easy

to obtain due to their cost. This has led researchers to try to reconstruct time-course "snapshots" starting from various "static" experiments [Gupta and Bar-Joseph, 2008, Magwene et al., 2003]; in this last case, this kernel method could be proposed as a variation of the optimality measure that is used to re-order the experiment snapshots.

Finally, while the work presented uses the notion of time intrinsic in the windowing segmentation, such constraint can be easily lifted and the method applied also in cases where the objects to be compared for similarity are not ordered in time.

Bibliography

[Abraham, 2002] Abraham, A. (2002). Intelligent systems: Architectures and perspectives, recent advances in intelligent paradigms and applications. In Abraham, A., Jain, L., and Kacprzyk, J., editors, *Studies in Fuzziness and Soft Computing*, Heidelberg. Springer.

[Abraham, 2005] Abraham, A. (2005). Nature and scope of AI techniques. In Sydenham, P. and Thorn, R., editors, *Handbook for Measurement Systems Design*, Chichester. John Wiley and Sons Ltd.

[Albert et al., 2010] Albert, B., Bray, D., Hopkin, K., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Water, P. (2010). *Essential Cell Biology*. Gardland Science, New York and London.

[Alon et al., 1999] Alon, U., Barkai, N., Notterman, D., Gish, K., Ybarra, S., Mack, D., and Levine, A. J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumour and normal colon tissues probed by oligonucleotide arrays. In *Proc. Nat. Acad. Sci.*, pages 6745–6750. USA 96.

[Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning (Adaptive Computation And Machine Learning)*. The MIT Press, Cambridge, Massachusetts, London, England.

[Altman et al., 2001] Altman, R., Valencia, A., Miyano, S., and Ranganathan, S. (2001). Challenges for intelligent systems in biology. *IEEE Intelligent Systems*, 6(16).

[Amaratunga and Cabrera, 2004] Amaratunga, D. and Cabrera, J. (2004). *Exploration and Analysis of DNA Microarray and Protein Array Data*. Wiley-Interscience, New Jersey.

[Antoniotti, 2007] Antoniotti, M. (2004-2007). GOALIE site. `http://bioinformatics.nyu.edu/Projects/GOALIE/`.

[Antoniotti et al., 2010] Antoniotti, M., Carreras, M., Farinaccio, A., Mauri, G., Merico, D., and Zoppis, I. (2010). An application of kernel methods to gene cluster temporal meta analysis. *Computers and Operations Research*, 37(8):1361–1368.

[Archetti et al., 2006] Archetti, F., Lanzeni, S., Messina, E., and Vanneschi, L. (2006). Genetic programming for human oral bioavailability of drugs. In M. Cattolico *et al.*, editor, *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation*, pages 255 – 262, Seattle, Washington, USA.

[Archetti et al., 2007a] Archetti, F., Messina, E., Lanzeni, S., and Vanneschi, L. (2007a). Genetic programming and other machine learning approaches to predict median oral lethal dose (LD50) and plasma protein binding levels (%PPB) of drugs. In E. Marchiori *et al.*, editor, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Proceedings of the Fifth European Conference, EvoBIO 2007*, Lecture Notes in Computer Science, LNCS 4447, pages 11–23. Springer, Berlin, Heidelberg, New York.

[Archetti et al., 2007b] Archetti, F., Messina, E., Lanzeni, S., and Vanneschi, L. (2007b). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, 8(4):17–26.

[Baldi and Brunak, 1998] Baldi, P. and Brunak, S. (1998). *Bioinformatics, The Machine Learning Approach*. The MIT Press.

[Banzhaf et al., 1998] Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic Programming - An Introduction*. Morgan Kaufmann Publishers, San Francisco, Heidelberg.

[Bar-Joseph, 2004] Bar-Joseph, Z. (2004). Analyzing time series gene expression data. *Bioinformatics*, 20(16):2493–2503.

[Beißbarth and Speed, 2004] Beißbarth, T. and Speed, T. P. (2004). GOstat: find statistically overrepresented Gene Ontologies within a group of genes. *Bioinformatics*, 20(9):1464–1465.

[Ben-Hur and Noble, 2005] Ben-Hur, A. and Noble, W. S. (2005). Kernel methods for predicting protein-protein interactions. *Bioinformatics*, 21(Supplement 1):38–46.

[Blickle and Thiele, 1995] Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. Technical Report TIK-Report 11, TIK Institut fur Technische Informatik und Kommunikationsnetze, Computer Engineering and Network Laboratory, ETH, Swiss Federal Institute of Technology, Gloriastrasse 35, 8092 Zurich, Switzerland.

[Bojarczuk et al., 2001] Bojarczuk, C., Lopes, H., and Freitas, A. (2001). Data mining with constrained-syntax genetic programming: applications to medical data sets. *Proceedings Intelligent Data Analysis in Medicine and Pharmacology*, 1.

[Borgwardt et al., 2005] Borgwardt, K. M., Cheng, S. O., Schönauer, Vishwanathan, S., Smola, A., and Kriegel, H. (2005). Protein Function Prediction via Graph Kernel. *Bioinformatics*, 21(Supplement 1):47–56.

[Borgwardt and Kriegel, 2005] Borgwardt, K. M. and Kriegel, H. (2005). Shortest-path Kernels on Graphs. In *ICDM*, pages 74–81.

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

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Belmont, California, Wadsworth International Group.

[Cantone et al., 2009] Cantone, I., Marucci, L., Iorio, F., Ricci, M., Belcastro, V., Bansal, M., Santini, S., Di Bernardo, M., Di Bernardo, D., and Cosma, M. (2009). A yeast synthetic network for in vivo assessment of reverse-engineering and modeling approaches. *Cell*, 137(1):172–81.

[Chen et al., 1999] Chen, T., He, H., and Church, G. (1999). Modeling gene expression with differential equations. In *Pacific Symposium on Biocomputing*, pages 29–40.

[Chu and Wang, 2005] Chu, F. and Wang, L. (2005). Applications of support vector machines to cancer classification with microarray data. *Int J Neural Syst*, 15(6):475–484.

[Chu et al., 1998] Chu, S., DeRisi, J. L., Eisen, M., Mulholland, J., Botstein, D., Brown, P. O., and Herskowitz, I. (1998). The Transcriptional Program of Sporulation in Budding Yeast. *Science*, 282:699–705.

[Cios et al., 2005] Cios, K., Mamitsuka, H., Nagashima, T., and Tadeusiewicz, R. (2005). Computational intelligence in solving bioinformatics problems. *Artificial Intelligence in Medicine*, 1(35).

[Courant and Hilbert, 1953] Courant, R. and Hilbert, D. (1953). *Methods of Mathematical Physics*, volume 1.

[Deb and Reddy, 2003] Deb, K. and Reddy, A. R. (2003). Reliable classification of two-class cancer data using evolutionary algorithms. *Biosystems*, 72(1-2):111–129.

[Della Gatta et al., 2008] Della Gatta, G., Bansal, M., Ambesi-Impiombato, A., Antonini, D., Missero, C., and Di Bernardo, D. (2008). Direct targets of the TRP63 transcription factor revealed by a combination of gene expression profiling and reverse engineering. *Genome Res.*, 18:939–948.

[DeRisi et al., 1997] DeRisi, J. L., Iyer, V. R., and Brown, P. O. (1997). Exploring the Metabolic and Genetic Control of Gene Expression on a Genomic Scale. *Science*, 278:680–686.

[Deutsch, 2003] Deutsch, J. M. (2003). Evolutionary algorithms for finding optimal gene sets in microarray prediction. *Bioinformatics*, 19(1):45–52.

[Di Bernardo et al., 2005] Di Bernardo, D., Thompson, M. J., Gardner, T. S., Chobot, S. E., Eastwood, E. L., Wojtovich, A. P., Elliot, S. J., Schaus, S. E., and Collins, J. J. (2005). Chemogenomic profiling on a genome-wide scale using reverse-engineered gene networks. *Nat. Biotechnol.*, 23:377–383.

[Di Ventura et al., 2006] Di Ventura, B., Lemerle, C., Michalodimitrakis, K., and Serrano, L. (2006). From in vivo to in silico biology and back. *Nature*, 443:527–533.

[Doherty et al., 2006] Doherty, J. M., Carmichael, L. K., and Mills, J. C. (2006). GOurmet: a tool for Quantitative Comparison and Visualization of Gene Expression Profiles Based on Gene Ontology (GO) Distributions. *BMC Bioinformatics*, 7(151).

[Eisen et al., 1998] Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Science*, 95(25):14863–14868.

[Emmert-Streib and Dehmer, 2008] Emmert-Streib, F. and Dehmer, M. (2008). *Analysis of Microarray Data - A Network-Based Approach*. Wiley-Vch Verlag GmbH and Co. KGaA, Weinheim.

[Ernst and Bar-Joseph, 2006] Ernst, J. and Bar-Joseph, Z. (2006). STEM: a tool for the analysis of short time series expression data. *BMC Bioinformatics*, 7(191).

[Ezziane, 2006] Ezziane, Z. (2006). Applications of artificial intelligence in bioinformatics: A review. *Expert Syst. Appl.*, 1(30):2–10.

[Faith et al., 2007] Faith, J. J., Hayete, B., Thaden, J. T., Mogno, I., Wierzbowski, J., Cottarel, G., Kasif, S., Collins, J. J., and Gardner, T. S. (2007). Large-scale mapping and validation of escherichia coli transcriptional regulation from a compendium of expression profiles. *PLoS Biol.*, 5(1):e8.

[Farabee,] Farabee, J. On-line biology book. `http://mac122.icu.ac.jp/biobk/biobooktoc.html`.

[Farinaccio, 2010] Farinaccio, A. (2010). Modelling a complex system: from microarray time series to data driven gene regulatory network. In *Proceedings of EvoPhD in Evo* 2010 Conference*, Istanbul. Springer.

[Farinaccio et al., 2010] Farinaccio, A., Vanneschi, L., Giacobini, M., Mauri, G., and Provero, P. (2010). On the use of genetic programming for the prediction of survival in cancer. In *Proceedings of the GECCO, Genetic and Evolutionary Computation, 2010 Conference*, ACM Press, pages 163–170.

[Farinaccio et al., 2009] Farinaccio, A., Vanneschi, L., Muppirisetty, S., Giacobini, M., Antoniotti, M., Mauri, G., and Provero, P. (2009). Genetic programming for survival prediction in breast cancer. In Alberghina, L. and Milanesi, L., editors, *Proceedings of the SysBioHealth 2009 Symposium*, pages 69–71.

[Farinaccio et al., 2011] Farinaccio, A., Vanneschi, L., Provero, P., Mauri, G., and Giacobini, M. (2011). A study on gene regulatory network reconstruction and simulation. In Apolloni, B., S.Bassis, Esposito, A., and Morabito, C., editors, *Proceedings of the 20th Italian Workshop on Neural Nets WIRN 2010 Conference*, volume 226, pages 235–242.

[Freund and Schapire, 1998] Freund, Y. and Schapire, R. E. (1998). Large margin classification using the perceptron algorithm. In *Machine Learning*, pages 277–296.

[Friedberg, 1958] Friedberg, R. (1958). A learning machine, part I. *IBM, J. Research and Development*, 2:2–13.

[Friedberg et al., 1959] Friedberg, R., Dunham, B., and North, J. (1959). A learning machine, part II. *IBM, J. Research and Development*, 3:282–285.

[Friedman et al., 2000] Friedman, N., Linial, M., Nachmann, I., and Peer, D. (2000). Using bayesian networks to analyze expression data. *J. Computational Biology*, 7:601–620.

[Gardner et al., 2003] Gardner, T. S., Di Bernardo, D., Lorenz, D., and Collins, J. J. (2003). Inferring genetic networks and identifying compound mode of action via expression profiling. *Science*, 301:102–105.

[Gärtner et al., 2003] Gärtner, P., Flach, P., and Wrobel, S. (2003). On Graph Kernels: Hardness Results and Efficient Alternatives. In *COLT/Kernel*, volume 2777 of *Lecture Notes in Artificial Intelligence*, pages 129–143. Springer-Verlag.

[Gene Ontology Consortium, 2006] Gene Ontology Consortium (2006). The Gene Ontology (GO) project in 2006. *Nucleic Acid Research (Database issue)*, 34:D322–D326.

[Gentleman et al., 2005] Gentleman, R., Carey, V. J., Huber, W., Irizarry, R., and Dudoit, S. (2005). *Bioinformatics and Computational Biology*

Solutions Using R and Bioconductor. Springer Science-Business Media, LLC, New York.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

[Goodsell, 2009] Goodsell, D. (2009). *The machinery of life*. Springer Verlag.

[Grefenstette and Baker, 1989] Grefenstette, J. and Baker, J. (1989). How genetic algorithms work: A critical look at implicit parallelism. In *Proc. 3rd International Conference on Genetic Algorithms*, pages 20–27. San Mateo, CA. Morgan Kaufmann, San Francisco CA.

[Guigo et al., 1992] Guigo, R., Knudsen, S., Drake, N., and Smith, T. (1992). Prediction of gene structure. *Journal of Molecular Biology*, 226:141–157.

[Gupta and Bar-Joseph, 2008] Gupta, A. and Bar-Joseph, Z. (2008). Extracting dynamics from static cancer expression data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2):172–182.

[Gusfield, 2004] Gusfield, D. (2004). Introduction to the ieee/acm transactions on computational biology and bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):2–3.

[Guyon et al., 2002] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422.

[Harris et al., 2002] Harris, S., Sawhill, B., Wuensche, A., and Kauffman, S. (2002). A model of transcriptional regulatory network based on biases in the observed regulation rules. *Complexity*, V:23–40.

[Hassanien et al., 2008] Hassanien, A., Milanova, M. G., Smolinski, T. G., and Abraham, A. (2008). Computational intelligence in solving bioinformatics problems: Reviews, perspectives, and challenges. In Smolinski, T., M.G.Milanova, and Hassanien, A., editors, *Computational Intelligence in Biomedicine and Bioinformatics*, Heidelberg. Springer.

[Hasty and McMillen, 2002] Hasty, J. and McMillen, D. (2002). Engineered gene circuits. *Nature*, 420:224–230.

[Hayete et al., 2007] Hayete, J., McMillen, D., and Collins, J. J. (2007). Size matters: network inference tackles the genome scale. *Mol. Syst. Biol.*, 3:77.

[Haykin, 1999] Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, London, UK.

[Helmbold and Warmuth, 1995] Helmbold, D. P. and Warmuth, M. K. (1995). On weak learning. *J. Comput. Syst. Sci.*, 50(3):551–573.

[Hernandez et al., 2007] Hernandez, J. C. H., Duval, B., and Hao, J. (2007). A genetic embedded approach for gene selection and classification of microarray data. *Lecture Notes in Computer Science*, 4447:90–101.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan.

[Hong and Cho, 2006] Hong, J. and Cho, S. (2006). The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming. *Artif. Intell. Med*, 36:43–58.

[Hsu et al., 2003] Hsu, A., Tang, S., and Halgamuge, S. (2003). An unsupervised hierarchical dynamic self-organizing approach to cancer class discovery and marker gene identification in microarray data. *Bioinformatics*, 19(16):2131–40.

[Hunga et al., 2006] Hunga, C., Huanga, Y., and Changb, M. (2006). Alignment using genetic programming with causal trees for identification of protein functions. *Nonlinear Analysis*, (65):1070–1093.

[Jaccard, 1901] Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, (37):547–579.

[Kantardzic, 2003] Kantardzic, M. (2003). *Data Mining: concepts, models, methods and algorithms*. Piscataway(N.J.): IEEE Press, J.Wiley, New York.

[Kashima et al., 2003] Kashima, H., Tsuda, K., and Inokuchi, A. (2003). Marginalized Kernels between Labelled Graphs. In *Proceedings of International Conference on Machine Learning(ICML)*, pages 321–328.

[Kaufmann, 1971] Kaufmann, S. (1971). Gene regulation networks: A theory of their global structure and behaviour. *Curr. Top. Dev. Biol.*, (6):145–182.

[Kaufmann, 1993] Kaufmann, S. (1993). *Origins of Order: Self-organization and Selection in Evolution*. Oxford University Press, New York.

[Kaufmann et al., 2003] Kaufmann, S., Peterson, C., Samuelsson, B., and Troein, C. (2003). Random boolean network models and the yeast transcriptional network. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, (25):14796–14799.

[Kelemen et al., 2008] Kelemen, A., Abraham, A., and Chen, Y. (2008). Computational intelligence in bioinformatics. In *Studies in Computational Intelligence*, Heidelberg. Springer Publishing Company.

[Kleinberg et al., 2006] Kleinberg, S., Antoniotti, M., Tadepalli, S., Ramakrishnan, N., and Mishra, B. (2006). Remembrance of Experiments Past: A Redescription Based Tool for Discovery in Complex Systems. In *Proceedings of the International Conference on Complex Systems*, pages 1–8, Boston, MA, U.S.A.

[Kleinberg et al., 2007] Kleinberg, S., Casey, K., and Mishra, B. (2007). Systems Biology via Redescription and Ontologies: Untangling the Malaria Parasite Life Cycle. In *Proceedings of the LSMS-07*.

[Knudsen, 2002] Knudsen, S. (2002). *Analysis of DNA Microarray Data*. Wiley-Liss, New York.

[Kondor and Lafferty, 2002] Kondor, R. S. and Lafferty, J. (2002). Diffusion Kernels on Graphs and Other Discrete Structures. In *Proceedings of ICML*.

[Koza et al., 2003] Koza, J., Keane, M., Streeter, M., Mydlowec, W., and Lanza, J. Y. G. (2003). *Genetic Programming IV, Routine Human-Competitive Machine Intelligence*. Klugen Academic Publishers, Norwell, Massachusetts.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming*. The MIT Press, Cambridge, Massachusetts.

[Langdon and Barrett, 2004] Langdon, W. and Barrett, S. (2004). Genetic Programming in data mining for drug discovery. *Evolutionary Computing in Data Mining*, pages 211–235.

[Langdon and Buxton, 2004] Langdon, W. B. and Buxton, B. F. (2004). Genetic programming for mining DNA chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257.

[Langdon and Poli, 2002] Langdon, W. B. and Poli, R. (2002). *Foundations of Genetic Programming*. Springer, Berlin.

[Larranaga et al., 2006] Larranaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., Lozano, J., Armananzas, R., Santafe, G., Perez, A., and Robles, V. (2006). Machine learning in bioinformatics. *Briefings in Bioinformatics*, 7(1):86–112.

[Liu et al., 2005] Liu, J., Cutler, G., Li, W., Pan, Z., Peng, S., Hoey, T., Chen, L., and Ling, X.-B. (2005). Multiclass cancer classification and biomarker discovery using ga-based algorithms. *Bioinformatics*, 21:2691–2697.

[Lockhart and Winzeler, 2000] Lockhart, D. J. and Winzeler, E. A. (2000). Genomics, gene expression and dna arrays. *Nature*, 405(6788):827–36.

[Lodish et al., 2008] Lodish, H., Berk, A., Kaiser, C., Keiger, M., Scott, M., Bretscher, A., Ploegh, H., and Matsudaira, P. (2008). *Molecular Cell Biology*. W. H. Freeman and Company, New York.

[Lu and Han, 2003] Lu, Y. and Han, J. (2003). Cancer classification using gene expression data. *Inf. Syst.*, 28(4):243–268.

[Luscombe et al., 2001] Luscombe, N., Greenbaum, D., and Gerstein, M. (2001). What is bioinformatics? a proposed definition and overview of the field. *Yearbook of Medical Informatics*.

[Magwene et al., 2003] Magwene, P. M., Lizardi, P., and Kim, J. (2003). Reconstructing the temporal ordering of biological samples using microarray data. *Bioinformatics*, 19(7):842–850.

[Merico et al., 2007] Merico, D., Zoppis, I., Antoniotti, M., and Mauri, G. (2007). Evaluating Graph Kernel Methods for Relation Discovery in GO-

Annotated clusters. In *KES-2007/WIRN-2007, Part IV*, volume 4694, pages 892–900, Lecture Notes in Artificial Intelligence. Springer-Verlag.

[Michie et al., 1994] Michie, D., Spiegelhalter, D., and Taylor, C. (1994). *Machine learning, neural and statistical classification.* Prentice Hall.

[Mitchell, 1996] Mitchell, T. (1996). *Machine Learning.* McGraw Hill, New York.

[Mitra et al., 2007] Mitra, S., Banka, H., and Paik, J. (2007). Evolutionary fuzzy biclustering of gene expression data. In Yao, J., Lingras, P., Wu, W., Szczuka, M., N.J.Cercone, and Slezak, D., editors, *RSKT 2007. LNCS (LNAI)*, Heidelberg. Springer.

[Mitra and Hayashi, 2006] Mitra, S. and Hayashi, Y. (2006). Bioinformatics with soft computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, (36):616–635.

[Moore et al., 2001] Moore, J., Parker, J., and Hahn, L. (2001). Symbolic discriminant analysis for mining gene expression patterns. *Lecture Notes in Artificial Intelligence*, 2167:372–381.

[Muller-Esterl et al., 2004] Muller-Esterl, W., Brandt, U., Anderka, O., Kieb, S., Ridinger, K., and Plenikowski, M. (2004). *Biochemie.* Elsevier-GmbH.

[Nelson and Cox, 2000] Nelson, D. L. and Cox, M. (2000). *Lehninger Principles of Biochemistry.* Worth Publisher, Inc.

[Nevins and Potti, 2007] Nevins, J. R. and Potti, A. (2007). Mining gene expression profiles: expression signatures as cancer phenotypes. *Nat Rev Genet*, 8(8):601–609.

[Park and Sandberg, 1991] Park, J. and Sandberg, J. W. (1991). Universal approximation using radial basis functions network. *Neural Computation*, 3:246–257.

[Paul and Iba, 2005] Paul, T. K. and Iba, H. (2005). Gene selection for classification of cancers using probabilistic model building genetic algorithm. *Biosystems*, 82(3):208–225.

[Platt, 1998] Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods – Support Vector Learning*.

[Poggio and Girosi, 1990] Poggio, T. and Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497.

[Poli and Langdon, 1997] Poli, R. and Langdon, W. B. (1997). Genetic programming with one-point crossover and point mutation. Technical Report CSRP-97-13, University of Birmingham, B15 2TT, UK.

[Poli and Langdon, 1998] Poli, R. and Langdon, W. B. (1998). Genetic programming with one-point crossover. In Chawdury, P. K., Roy, R., and Pant, R. K., editors, *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pages 23–27. Springer-Verlag, London.

[Polymeropoulos et al., 1996] Polymeropoulos, M. H., Higgins, J. J., Golbe, L. I., Johnson, W. G., Ide, S. E., Iorio, G. D., Sanges, G., Stenroos, E. S., Pho, L. T., Schaffer, A. A., Lazzarini, A. M., Nussbaum, R. L., and Duvoisin, R. C. (1996). Mapping of a gene for parkinson's disease to chromosome 4q21-q23. *Science*, 274.

[Quackenbush, 2001] Quackenbush, J. (2001). Computational analysis of microarray data. *Nat Rev Genet*, 2(6):418–427.

[Ramakrishnan et al., 2005] Ramakrishnan, N., Antoniotti, M., and Mishra, B. (2005). Reconstructing Formal Temporal Models of Cellular Events using the GO Process Ontology. In *Bio-Ontologies SIG Meeting, ISMB*, Detroit MI, U.S.A.

[Rosskopf et al., 2007] Rosskopf, M., Schmidt, H., Feldkamp, U., and Banzhaf, W. (2007). Genetic programming based dna microarray analysis for classification of tumour tissues. Technical Report Technical Report 2007-03, Memorial University of Newfoundland.

[Sakamoto and Iba, 2000] Sakamoto, E. and Iba, H. (2000). Identifying gene regulatory network as differential equation by genetic programming. In *Poster Session of Genome Informatics Workshop*, volume 3.

[Schölkopf and Smola, 2002] Schölkopf, B. and Smola, A. J. (2002]). *Learning with Kernels*. MIT Press.

[Schölkopf et al., 2004] Schölkopf, B., Tsuda, K., and Vert, J. P. (2004). *Kernel Methods in Computational Biology*. MIT Press.

[Semeria et al., 2004] Semeria, A., Villani, M., Serra, R., and Kauffman, S. (2004). Perturbation in genetic regulatory network: Simulation and experiments. In *Proceedings of ACRI 2004 Conference*, number 3305 in Springer, LNCS, pages 533–542.

[Serra et al., 2008] Serra, R., Villani, M., Graudenzi, A., Colacci, A., and Kauffman, S. (2008). The simulation of gene knock-out in scale-free random boolean models of genetic networks. *AIMS' Journals*, 3(2):333–343.

[Sharan and Shamir, 2000] Sharan, R. and Shamir, R. (2000). CLICK: A Clustering Algorithm with Applications to Gene Expression Analysis. In *Proceedings of ISMB 2000*, pages 307–316. AAAI Press, Menlo Park, CA, U.S.A.

[Sheng et al., 2005] Sheng, Q., Moreau, Y., Smet, F. D., Marchal, K., and Moor, B. D. (2005). Advances in cluster analysis of microarray data. *Data Analysis and Visualization in Genomics and Proteomics*, pages 153–173.

[Silva, 2007] Silva, S. (2007). GPLAB – a genetic programming toolbox for MATLAB, version 3.0. `http://gplab.sourceforge.net`.

[Skovgaard et al., 2001] Skovgaard, M., Jensen, L. J., Brunak, S., Ussery, D., and Krogh, A. (2001). On the total number of genes and their length distribution in complete microbial genomes. *Trends Genet.*, 17:425–428.

[Speed, 2003] Speed, P. T. (2003). *Statistical Analysis of Gene Expression Microarray Data*. Interdisciplinary Statistics Ser. CRC Press, first edition.

[Spellman et al., 1998] Spellman, P. T., Sherlock, G., Zhang, M. Q., Iyer, V. R., Anders, K., Eisen, M. B., Brown, P. O., Botstein, D., and Futcher, B. (1998). Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast Saccharomyces Cerevisiae by Microarray Hybridization. *Molecular Biology of the Cell*, 9:3273–3297.

[Sprinzak and Elowitz, 2005] Sprinzak, D. and Elowitz, M. B. (2005). Reconstruction of genetic circuits. *Nature*, 438:443–448.

[Stolovitzky et al., 2007] Stolovitzky, G., Monroe, D., and Califano, A. (2007). Dialogue on reverse-engineering assessment and methods: the dream of high-throughput pathway inference. *Ann. N Y Acad. Sci*, 1115:1–22.

[Szallasi et al., 2006] Szallasi, Z., Stelling, J., and Periwal, V. (2006). System modeling in cellular biology: From concepts to nuts and bolts. *The MIT Press*.

[Tadepalli et al., 2008] Tadepalli, S., Ramakrishnan, N., Watson, L. T., Mishra, B., and Help, R. F. (2008). Simultaneuosly Segmenting Multiple Gene Expression Time Courses by Analyzing Cluster Dynamics. In *Proceedings of the Sixth Asia-Pacific Bioinformatics Conference (APBC 2008)*, pages 297–306, Tokyo, Japan.

[Tettamanzi and Tomassini, 2001] Tettamanzi, A. and Tomassini, M. (2001). *Soft Computing - Integrating Evolutionary, Neural and Fuzzy Systems*. Springer, Berlin.

[Tominaga et al., 2000] Tominaga, D., Koga, N., and Okamoto, M. (2000). Efficient numerical optimization algorithm based on genetic algorithm for inverse problem. In *Genetic and Evolutionary Computation Conference*, pages 251–258.

[van de Vijver et al., 2002] van de Vijver, M. J., He, Y. D., van't Veer, L. J., Dai, H., Hart, A. A. M., Voskuil, D. W., Schreiber, G. J., Peterse, J. L., Roberts, C., Marton, M. J., Parrish, M., Atsma, D., Witteveen, A., Glas, A., Delahaye, L., van der Velde, T., Bartelink, H., Rodenhuis, S., Rutgers, E. T., Friend, S. H., and Bernards, R. (2002). A gene-expression signature as a predictor of survival in breast cancer. *N Engl J Med*, 347(25):1999–2009.

[Vanneschi, 2004] Vanneschi, L. (2004). *Theory and Practice for Efficient Genetic Programming*. Ph.D. thesis, Faculty of Sciences, University of Lausanne, Switzerland.

[Vanneschi et al., 2010] Vanneschi, L., Farinaccio, A., Giacobini, M., Antoniotti, M., Mauri, G., and Provero, P. (2010). Identification of individualized feature combinations for survival prediction in breast cancer: A

comparison of machine learning techniques. In *Proceedings of 8th European Conference, EvoBIO 2010*, volume 6023 of *LNCS*, pages 110–121, Istanbul. Springer.

[van't Veer et al., 2002] van't Veer, L. J., Dai, H., van de Vijver, M. J., He, Y. D., Hart, A. A. M., Mao, M., Peterse, H. L., van der Kooy, K., Marton, M. J., Witteveen, A. T., Schreiber, G. J., Kerkhoven, R. M., Roberts, C., Linsley, P. S., Bernards, R., and Friend, S. H. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415(6871):530–536.

[Vapnik, 1998] Vapnik, V. (1998). *Statistical Learning Theory*. Wiley, New York, NY.

[Vishwanathan et al., 2007] Vishwanathan, S. V. N., Borgwardt, K. M., and Schraudolph, N. N. (2007). Fast Computation on Graph Kernels. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19*, pages 1449–1456. MIT Press.

[Watson et al., 2004] Watson, J., Baker, T., Bell, S., Gann, A., Levine, M., and Losick, R. (2004). *Molecular Biology of the Gene*. Pearson Education, Inc.

[Weka, 2006] Weka (2006). A multi-task machine learning software developed by Waikato University. `www.cs.waikato.ac.nz/ml/weka`.

[Whitley, 1989] Whitley, D. (1989). The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In Schaffer, J. D., editor, *Proc. 3rd Int. Conference on Genetic Algorithms, San Mateo, CA.*, pages 116–121. San Mateo, CA. Morgan Kaufmann, San Francisco CA.

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

[Yu et al., 2007] Yu, J., Yu, J., Almal, A. A., Dhanasekaran, S. M., Ghosh, D., Worzel, W. P., and Chinnaiyan, A. M. (2007). Feature selection and molecular classification of cancer using genetic programming. *Neoplasia*, 9(4):292–303.

[Zhang and Rajapakse, 2008] Zhang, Y. and Rajapakse, C. (2008). *Machine Learning in Bioinformatics*. Wiley.

[Zoppis et al., 2007] Zoppis, I., Merico, D., Antoniotti, M., Mishra, B., and Mauri, G. (2007). Discovering Relations among GO-annotated Clusters by Graph Kernel Methods. In *Proceedings of the 2007 International Symposium on Bioinformatics Research and Applications*, volume 4463 of *lncs*, pages 158–169.