Evolving a Fuzzy Rulebase to Model Gene Expression

Ricardo Linden[1] and Amit Bhaya[2]

[1] Faculdade Salesiana Maria Auxiliadora, CEPEL
`rlinden@pobox.com`
[2] COPPE-UFRJ
`amit@nacad.ufrj.br`

Summary. This chapter describes the use of genetic programming to evolve a fuzzy rule base to model gene expression. We describe the problem of genetic regulation in details and offer some reasons as to why many computational methods have difficulties in modeling it. We describe how a fuzzy rule base can be applied to this problem as well as how genetic programming can be used to evolve a fuzzy rule base to extract explanatory rules from microarray data obtained in the real experiments, which give us data sets that have thousands of features, but only a limited number of measurements in time. The algorithm allows for the insertion of prior knowledge, making it possible to find sets of rules that include the relationships between genes that are already known.

10.1 Biological Introduction

Biology has expanded its knowledge of the gene regulation process and a full description of known features of the regulation process would require a huge compendium. In this section we will describe the main features of gene regulation in order for the reader to understand the specific class of problems studied in this chapter. We will also describe the main characteristics of the data sets used in this problem.

10.1.1 Gene Regulation

All information necessary for the creation of the proteins that are necessary for the cell is coded in DNA. This information is extracted in a multi-step path that is called the central dogma of molecular biology that could be summarized as DNA $\longrightarrow$ RNA $\longrightarrow$ Protein, meaning that cell response and differentiation steps can be controlled in many different steps. This ability to respond to different needs and the basic process of differentiation are fundamental to an organism. No one would confuse a neuron with a liver or a heart cell, either based on their shape or on their function. Nevertheless, this difference is not due to major differences in their DNA sequence, but rather to differences in the expression level of each gene. In every cell, at every time point, only a fraction of the total DNA is expressed, that is, transcribed into mRNA, which is subsequently translated into protein. The amount of proteins and other elements produced is regulated so that the necessary amounts are produced without depleting excessively the energetic reserves of the cell.

At every given time, a gene may be expressed at different levels, including zero (not expressed). Its expression level has a strong correlation to the amount of mRNA in the cell that is coded by the gene. It is important to understand that there are other control mechanisms in the cell, such as protein degradation speed, RNA transport and localization control *etc*, but for most cells the initiation of RNA transcription is the most important point of control [1]. Nevertheless, one should understand that considering only transcriptional control in the reverse engineering problem is a limitation of any method based on microarrays (see Section 10.1.2 for further details).

A cell can change the expression level of its genes in response to external signals. At any given time, the cell needs to inhibit or activate groups of genes in response to changing organism and environment requirements. In order to perform this regulation, cells have an elaborate mechanism that include several control areas in every gene, the catalysis of gene expression by several other substances and other mechanisms.

A gene can be regulated by several molecules and the ones it codes for can regulate one or more other genes, creating a pleiotropic regulation network, which occurs by the binding of molecules to gene control areas. Since many of these molecules are proteins synthesized based on the mRNA of other genes, we can say that one gene regulates another, even though DNA regions do not interact directly.

10.1.2 Microarray Data Sets

Nowadays, the data available on gene expression is growing exponentially. The creation of new tools has caused a genomics revolution, flooding scientists with huge amounts of data. DNA microarray technology is one of the technologies that has caused this huge impact in the biological sciences. It describes the state of each cell by measuring the mRNA expression levels, which, according to the central dogma, is a good approximation of protein levels. It is not precise however, because there are other mechanisms of control besides DNA transcription control, but measurement techniques for other substances such as proteins are not as accurate and widespread as microarrays.

A DNA array, or microarray, is defined as an orderly arrangement of tens to hundreds of thousands of unique radioactive or fluorescent DNA molecules (probes) of known sequence attached to a fixed surface. The DNA samples are then presented to the microarray and bind to their complementary sequences (hybridize), thus allowing for the determination of their sequence or their relative abundance (expression level) [26].

Microarrays allow one to study expression levels in parallel thus providing static information about gene expression (i.e., in which tissue(s) the gene is expressed) and dynamic information (i.e., how the expression pattern of one gene relates to those of others). The high degree of digital data extraction and processing of these techniques supports a variety of samples or experimental conditions [6]. Microarray technology, as a high throughput approach of differential gene expression studies, efficiently generates massive amounts of gene regulation data, facilitating rapid identification of gene candidates to follow up with functional characterization.

There are, of course, some pitfalls when dealing with this technology that a cautious experimenter must deal with. We will not describe the problems in detail, for this is not the main goal of this chapter, but, in brief, the main problems with this technology are:

- Differences among samples
- Measurement errors from CCD or radioactivity detectors
- Normalization methods
- Background noise elimination
- Stochastic binding effects

A cautious experimenter will try to ensure that these problems are dealt with and that experiments are repeated in order to avoid the common errors associated with microarrays.

Once these problems are dealt with, the main problem with a microarray data set becomes its dimensions. Usually, thousands of genes are measured simultaneously for a small number of time steps. Given thousands of expression levels, it is a complex task to identify the differentially expressed genes and understand the myriad relationships that define the genetic regulation network. There are several well known statistical techniques for this [12], but they all require massive amounts of data. There are some techniques, such as those described in [34] and [25] that try to minimize the number of experiments, by reducing the dimensionality of the data, either with Fisher Discriminant Analysis (FDA), singular value decomposition (SVD) or by Wilks Lambda Score. Nevertheless, these techniques, even though they manage to reduce data dimensionality drastically, still require a significant number of measurements (of the order of tens, in both cases).

Unfortunately, given the high cost of experiments, this volume of data is not available. Thus, it is necessary to propose new techniques that can cope with lack of data and still mine the gene expression matrix for useful relationships. There is, of course, a price for dealing with such a small data set. First, we can point out to the loss of certainty, making it very difficult to make accurate statements about relationships between genes. Second, we incur the "blessing of dimensionality". Given the high number of genes measured and the small number of measurement points, we face a situation where there are many degrees of freedom and few constraints, so that there are many possible "good fit" solutions. Thus, it can be argued that any search method will stumble upon a reasonable solution, and the key is how to differentiate between various solutions. Therefore, any proposed method should be able to deal with both these problems.

10.2 Pros and Cons of Gene Regulatory Networks (GRN) Models

In this section we discuss some methods that are successful and popular in other fields of application but have inherent characteristics that make them less advisable in this specific application. This does not mean, of course, that these methods are of inferior quality or that it is impossible to obtain meaningful results with them, but only that their inherent characteristics make it more difficult to achieve good or reliable results when dealing with microarray data sets.

10.2.1 Boolean Networks

Perhaps the simplest model is the Boolean Network. Boolean networks were first introduced by Kauffman in the late 60's as an abstraction of genetic regulatory networks: each gene is modeled as being either "ON" or "OFF", and the state of each gene at the next time step is determined by a Boolean function of its inputs at the current time step [15].

Several papers use Boolean networks as a tool to model gene expression, either regular [21] or probabilistic [27]. This model assumes that each gene can take either 0 (not-expressed) or 1 (expressed) as its state value. This approach can model some aspects of real regulation, as discussed in [29]. Nevertheless, since gene regulation is gradual and varies with gene expression levels, the ability of Boolean Networks to model real genetic regulation is limited.

10.2.2 Neural Networks

The data set available in a typical gene regulatory network problem generally consists of hundreds to thousands of signals, usually measured for no more than twenty time-steps. This paucity of data renders network models inferred from this data statistically insignificant [31] and requires methods that can deal with this specific situation to produce results that have real value to "wet lab" experiment design, even though they may not be statistically significant for predictive purposes.

Artificial neural networks (ANN) are computational models that try to mimic, in an extremely simple way, the way the human brain works. Just like the brain, ANNs consist of a group of simple cells (artificial neurons) connected by synapses that form a massively parallel and distributed processing system. The structure of this system (connections and their weights) is formed during a training phase, when the ANN is presented with data examples, actually acquiring knowledge from its environment [18].

To perform this learning, neural networks rely on massive data sets to perform their training process, so they cannot be trained correctly, using traditional algorithms, in situations where data is scarce. Unfortunately, each experiment has a non negligible cost, so that it is financially not practical for most labs to perform a large number of experiments, which makes it impossible to generate the amount of data a neural network would require. When presented with small data sets, neural networks tend to simply memorize them, instead of actually learning the underlying structure that generated them.

10.2.3 Association Rules

Creighton and Hanash [7] describe a technique that uses association rules to mine for regulatory relationships among genes. The algorithm proposed uses data binning (actual discretization) to overcome measurement errors and inherent noise. The dangers associated with discretization are highlighted in [16] in which it is also proposed to search for association rules, using planes to separate data.

In [7] all expression levels are set to three different states: up, down and neither-up-nor-down, while in [5] all expression levels are transformed into a boolean variable (over-expressed and under-expressed). An idea that might improve these methods would be to adapt them to a fuzzy sets based approach, allowing them to have multiple fuzzy sets that would correspond to, for instance, highly expressed, very highly expressed, etc.

This technique suffers from being data intensive, because it needs several microarray runs to create frequent itemsets and establish a pattern of co-occurrences and, therefore, the cost limitations described above still apply. Using a small number of data points may cause the frequent itemsets to be relatively small, allowing for spurious effects (inherent to a multiply connected graph such as the biological regulation one) to create rules that do not have real importance.

This method, in common with all others that rely on large data sets, will improve as experimental costs begin to fall. As [16] points out, it is becoming increasingly common to see data sets with tens of experiments. In the future, when hundreds or even thousands of experiments become common, the ability of methods based on association rules to model data will improve considerably.

10.2.4 Linear Regression

If gene regulation could be assumed to be linear, then linear regression would be an effective tool to model it. However, there are commonly occurring biological effects that make the linearity assumption fail:

1. Saturation: at a concentration that is specific to each binding site and substrate, increasing the regulator concentration will not change the speed and/or the amount of regulatee action.
2. Catalysis: the presence of certain enzymes will change the speed of the reaction and this effect, together with saturation, will cause the reaction curve to be S-shaped. A piecewise linear method, such as described in [9], may minimize this as well as he effect mentioned in the previous item, by creating close linear approximations for each of the main function regions.
3. Inhibitory effects: The presence of certain substances will inhibit the regulation process, either shutting it down completely, or making it extremely inefficient. This effect can be modeled by combining a Boolean network with a linear one, as done in [4].

Because of all these effects, any technique that is based on a linearity assumption tends to yield poorer results. They can serve as a first approximation, but non-linearity will always insidiously decrease the veracity of their results, an effect that may be decreased if one combines a piecewise linear approach with Boolean networks.

10.2.5 Perturbation Analysis

The idea behind such methods, described in papers such as [30] and [3], is that additional information about a genetic network may be gleaned experimentally by applying a directed perturbation to the network, and observing the steady-state expression levels of every gene in the network in the presence of the perturbation.

The problem with this technique is similar to the one described for neural networks: perturbation experiments may become expensive, especially if they need to be repeated in order to eliminate spurious results. Another problem with this approach lies in the fact that there seems to be some evidence that nature rewards redundancy in the form of alternative pathways performing the same function, so that there are backups in case that one pathway fails. Therefore a knock-out experiment may not lead to the pathway being switched off, even if a seemingly crucial component of a pathway was knocked out, since a backup component is activated and takes over the function of the one knocked out.

Another issue is that in higher metazoa each gene is associated with an average of ten different biological functions [2]. This means that perturbation studies may disrupt several different pathways and cause unforeseen effects that may make it very difficult for the analyst to understand what is really happening in the single pathway he is studying.

10.2.6 Differential Equations

A genetic regulatory network can be understood as a dynamic system whose states are defined by the expression levels as measured by microarray experiments. As such, they could be described by differential or difference equation based models, which at once can be seen as superior to Boolean networks, given their ability to deal with intermediate expression levels.

The main problems with these models are the following:

- Discrete aspects: they cannot easily describe the discrete aspects of gene regulation such as binding of a transcription factor to the DNA, which is essentially an on/off event [4].
- Linear models vs Excessive data requirements: scientists tend to concentrate on linear models because they require less data, but at the cost of placing strong constraints on the nature of regulatory interactions in the cell that may lead to less accuracy in the regulatory interaction description [15]. The use of differential equations allows for the introduction of explicit rate constants and one could go beyond the linear additive summation and include higher order terms, the determination of the parameters of a network incorporating higher-order terms would require much more data than is generally available for these systems [32].
- If the algorithm uses time-series data, it must estimate the rates of change of the transcripts ($\frac{dx}{dt}$) from the series. This can be problematic because calculating the derivative can amplify the measurement errors (noise) in the data [15].

This section has not made an exhaustive list of methods applied to the gene regulation problem. Many other methods have been applied to this problem. The reader may refer to [35], for instance, to learn about other computational techniques to analyze genomic data.

10.3 Fuzzy Logic Applied to the Gene Regulation Problem

Now that we have discussed what methods should not be used, we can proceed to the discussion of the method proposed in this chapter. In this section we will describe the fuzzy logic model used to deal with microarray data.

10.3.1 Beneficial Characteristics of a Fuzzy Model

Fuzzy rules model naturally ill-posed problems characterized by very little information such as the one we are facing using a linguistic approach that is a form of information useful to human experts. Therefore, they can be used to convey imprecise information for experts that are available and can seed the systems with a number of effective rules from the outset and verify the linguistic results obtained [28]. Using this prior knowledge is a very interesting feature that we explored in this work and that becomes very difficult to implement in numeric methods, like differential equations and neural networks.

Whenever discovered knowledge is to be used to assist in decision-making by a human user, it is important that it be comprehensible to the user [13]. Given the discrepancy between data dimensions (high number of genes with small number of points) we cannot arrive at a definitive model, no matter what method is used. Therefore, the main purpose of purposing a fuzzy model is to create hypotheses that can subsequently be tested in the biological workbench ("wet lab"). This implies that simplicity of the rules is an essential feature.

Fuzzy rules also make it very easy to include delays in the model. It is enough to include a parameter t in the rule, where t stands for the delay. For instance, a rule with the format *IF HighlyExpressed(f_1(-1)) AND LowlyExpressed(f_2(-2)) THEN HighlyExpressed(f_3)* means that if in the previous time step the substance f_1 had a high expression level and two time steps before substance f_2 had a low expression level, then the substrate f_3 will have a calculated high expression value. The default value for t will always be 1, meaning that we are referring to the previous expression value available. Obviously, when we introduce this feature we create more degrees of freedom and the model must be able to deal with this issue.

10.3.2 Definition of the Fuzzy Sets

It is assumed that each gene may be associated with several fuzzy sets whose universes of discourse will cover all the numeric space that its expression levels can span.

In our work, the expression space of each feature is divided evenly, so that each fuzzy set has the same support. The process consists of the following steps:

1. Choose the number of fuzzy sets for the gene of interest (n_{fuzzy});
2. The universe of discourse for the variable is defined by reading the interval of expression values from the microarray data and expanding it a little (10%, for instance) in each direction;
3. The universe of discourse is then divided in ($n_{fuzzy} - 1$)parts;
4. Create the fuzzy sets using triangular functions, using the following scheme:
   - The first half part is the support of the first descending function;
   - The last half part is the support of the last ascending function;
   - The $n_{fuzzy}-2$ remaining parts are the supports of triangular membership functions.

We could assign meaning to each fuzzy set. For instance, if we divide the space into three fuzzy sets, they could be understood as representing *Low expression level*, *Medium expression level* and *High expression level*. If we divide the space in five fuzzy sets, we could interpret the fuzzy sets as meaning *Very low expression level*, *Low expression level*, *Medium expression level*, *High expression level* and *Very high expression level*.

There is overlapping among adjacent fuzzy sets, so that more than one rule may be active for each expression value.

10.3.3 Using Fuzzy Rules to Calculate Expression Levels

Having defined the fuzzy sets and the fuzzy rule base, calculating next time step expression levels is a simple matter of applying the current time step expression levels to the rules and defuzzifying the results. There are many defuzzification methods that could be used in this process and the choice of method is analogous to that used in any other fuzzy application.

In this application, we applied the medium of maxima (MoM) defuzzification method, expressed by the formula:

$$y_j = \frac{\sum_{i=1}^{n_{fuzzy}} \mu_i * y_i^{max}}{\sum_{i=1}^{n_{fuzzy}} \mu_i} \tag{10.1}$$

In this formula, n_{fuzzy} stands for the number of fuzzy sets defined for the gene of interest (see Section 10.3.2), μ_i is the calculated membership of the rule to the fuzzy set i and y_i^{max} is the expression value where the membership function of set i is at its maximum.

There may be rules that have zero support. In our experience, this may lead to poorer results, and a minimum degree should be designated for each rule before the defuzzification process. A small value, such as 0.05 will guarantee set participation and allow the algorithm to achieve better results.

10.3.4 Rule Introduction by an Expert

Fuzzy logic offers an appealing method for describing phenomena by a set of rules and data sets that are based on linguistic expressions very similar to the ones we use daily to keep and transfer knowledge. These expressions include "high level of expression", "low level of expression" etc and the rules based on those concepts express knowledge in approximately the same way that a human expert would. An example of a fuzzy rule would be "if gene A is at a low level of expression, then gene B is at a high level of expression", which clearly means that gene A is an inhibitor to gene B.

Experts are already aware of many regulatory pathways and ignoring this prior knowledge is certainly wasteful, while exploiting it leads to a reduction in the size of the search space, which is always desirable.

It is not easy for an expert to incorporate prior knowledge into certain methods, such as differential equation based and neural networks based methods, which can be considered to be "numerically encoded", thus requiring translation of qualitative knowledge into appropriate quantitative knowledge, usually a difficult task. This is not the case with fuzzy systems. Since we have a rule base that can have many rules applied to a single fuzzy variable, experts can include their knowledge and this will be used together with the rules discovered by any algorithm. Using this prior knowledge makes the method smarter and also makes it possible for any tool created to become a hypothesis tester. In the next section we will discuss rule insertion by experts when describing a full genetic programming algorithm used to discover a fuzzy rule base.

10.4 Using Genetic Programming to Evolve a Fuzzy Rule Base

The basic concepts of using a fuzzy rule base were described in the previous section, but we still need an algorithm that can help us find what rules to use. In this section we will describe the evolutionary algorithm we use to create the fuzzy rule base that can best describe the gene regulation implicitly presented in the (small) data set.

10.4.1 Basic Concepts

Evolutionary algorithms (EA) are inspired by Nature. The idea is to mimic the natural evolution of the species using operators that simulate both sexual reproduction and random mutation in order to create a new kind of search technique that is robust and intelligently seeks solutions in a search space that may be too big for conventional techniques [22]. EA spawns several different techniques, including genetic algorithms (GA), genetic programming (GP) and evolutionary programming (EP).

Genetic Programming is a branch of evolutionary algorithms that simulates natural evolution to find complex structures such as programs and rules [19]. As with other EAs, it uses the concept of a population that reproduces and suffers mutation (as in natural, biological processes) and with high probability evolves toward a better solution that fits better to real data available. It is a heuristic that, although dependent on many probabilistic factors, tends to span much of the solution space and is less affected by common data problems such as local minima in an energy function.

All evolutionary algorithms use a population of competing solutions subjected to random variation and selection for a specific purpose [10], which is to evolve the population to one that contains a higher proportion of superior (fitter) individuals. The fitness of each individual in the population (its quality) is a measure of how well that individual achieves the desired goal and is the main connection between the EA and the problem at hand. Therefore, the formulation must contain all aspects of the problem, including constraints.

The variation and selection are usually based on two operators, the crossover operator which combines two different individuals into a new one and the mutation operator, which randomly changes parts of one individual in order to increase diversity. Both are very important, representing two different aspects of the natural search: exploitation (using current solutions information to derive a new and possibly better solution), which is performed by the crossover operator, and exploration (venturing into new areas of the search space), which is performed by the mutation operator.

An evolutionary algorithm could be described by the following pseudo-code:

```
Create Initial Population
While termination criteria not met
    Select parents which will generate offspring from current population
    Apply genetic operators to the selected parents and generate offspring
    Select next population from current individuals and generated offspring
End While
Present best solution(s)
```

In this algorithm, termination is usually based on one or more of the following criteria:

- time based: a number of generations has elapsed;
- quality based: a certain performance has been achieved;
- stagnation based: the set of best individuals has not improved for a certain number of generations.

Parent selection must be done in a way so that best solutions have the bigger probability of reproducing, but not at the expense of preventing the worse solutions from also doing so. The idea is that these worse solutions may have important information coded in their "genes" that would be missed if they did not participate in the creation of new offspring. Therefore, a variation of a roulette approach is usually used in which the parents with highest evaluation ("fittest") correspond to a bigger fraction of the roulette wheel when a random "spin" of the wheel is performed.

Thus, in order to define an EA one must define the coding scheme (how each individual will be represented in the computer), the operators (both mutation and crossover and any other specific one that will be used), the evaluation or fitness function (i.e., a measure of the quality of the current solutions to the problem at hand), the termination criteria used, the parent selection scheme and the next generation choice algorithm. We will discuss all these items applied to the problem of gene regulation reverse engineering in the following sections.

10.4.2 Dealing with Microarray Data Characteristics

The use of GP methods is justified by their ability to generate and test a broad spectrum of solutions, even from incomplete or insufficient data sets. As discussed in Section 10.1.2, microarray data usually consists of hundreds or thousands of genes whose expression levels are measured at only a few time points. This low dimensionality generally implies that statistical methods are liable to generate solutions of low statistical significance, but does not stop a GP method from generating testable hypotheses for the biology lab.

The high number of degrees of freedom of the data set gives rise to a phenomenon that we have called "the blessing of dimensionality", which will cause any search method to stumble upon a reasonable solution. The key is how to differentiate between various spurious solutions and a possibly true regulatory relationship. Since financial constraints make it very hard to conduct the experiments that would generate the amount of data necessary to find a unique optimally fitted network, this problem must be dealt with computationally.

Our approach is based on the assumption that genes that exhibit the same behavior are under the same kind of control. This assumption seems to be biologically sound, however, when we deal with a limited number of experiments, we may be fooled into believing that two genes that display the same behavior are part of the same regulatory pathway, since different strategies could generate the same response over a limited window of observation and under a limited set of conditions. Nevertheless, although this strategy could lead to false positives, it should not, in principle, hide any meaningful relationships.

Therefore, it was decided to use a clustering algorithm that assumes that genes that exhibit the same behavior are under the same kind of control, which means that strongly correlated genes should be treated together in order to find the single underlying regulation network that controls them all. It is important to understand that this clustering approach may not be correct, but searching for techniques that work in the entire cluster helps to eliminate many hypotheses and helps limit the number of control mechanisms found by the algorithm. Since the goal of this algorithm is to find testable hypotheses for the "wet lab", this helps to reduce the set of candidates.

One useful measure that indicates similarity between expression levels and their changes is the Pearson Product-Moment Correlation Coefficient (correlation coefficient for short) denoted ρ, which is a measure of the degree of linear relationship between two variables, usually labeled X and Y. While in regression the emphasis is on predicting one variable from the other, in correlation, on the other hand, the emphasis is on the degree to which a linear model may describe the relationship between two variables. In regression the interest is directional, one variable is predicted and the other is the predictor; in correlation the interest is non-directional, the linearity relationship is the critical aspect [12].

We calculate the correlation values for the gene of interest with every other gene in the microarray and establish an arbitrary cutoff point (in our case, we worked with a value of 0.95). Using only high correlation values is also interesting because it may generate overlapping clusters. The fact that gene g_1 is correlated to a certain degree with gene g_2 and the latter is correlated to the same certain degree with gene g_3 does not imply that gene g_1 is also correlated with gene g_3 to a degree above the cutting point. Therefore, we create one cluster for each gene of interest, containing all the elements that are correlated to it.

Pearson correlation values are in the range $[-1, 1]$, where -1 stands for perfect anti-correlated (when gene g_1 expression level rises, gene g_2 expression level lowers in the same proportion). We could include anti-correlation in our groups, just arranging for a special processing of the trees that turns inhibition into promotion and vice-versa, but we decided against it, because we do not need an additional hypothesis to constrain the veracity of our results.

10.4.3 Chromosome Structure

In our algorithm, a rule is represented as a tree whose linear representation is in reverse polish notation (RPN), where all operators precede their operands. This tree representation has already been used, although somewhat differently, in works such as [24, 8, 36].

In the model proposed, expressions are defined recursively by the following syntax:

$$\begin{cases} <\mathit{Expression}>::= & \text{<Antecedent> <Consequent>;} \\ <\mathit{Antecedent}>::= & \text{<BinOperator> <Expression> <Expression>} \\ <\mathit{Antecedent}>::= & \text{NOT <Expression>;} \\ <\mathit{Antecedent}>::= & \text{<FuzzySet>;} \\ <\mathit{BinOperator}>::= & \text{AND | OR;} \\ <\mathit{FuzzySet}>::= & \text{<set> <variable> <time>;} \\ <\mathit{Consequent}>::= & \text{<set> <variable>.} \end{cases}$$

In this syntax, <set> defines the fuzzy set used, <variable> is one of the many genes whose expression levels were measured by the microarray and <time> is an integer that represents the delay between measurement and control. This delay is important because sometimes, due to kinetic energy in the molecules, probabilistic effects or mere reaction times, the control performed by one gene can occur later in time, especially if the interval between microarray measurements is small enough. This delay can go from 1 (previous time step) to $(n-1)$, where n is the number of measurements. It is important to understand that the number of time steps used to match the data will be limited by the maximum delay allowed (d). In this case, we will only have $n-d$ instances of data, starting at time step $n-d+1$.

The use of past data is important in several genetic processes. For instance, there is evidence that the phenomenon of cell memory is a prerequisite for the creation of organized tissues and for the maintenance of stably differentiated cell types [1]. In other cases, such as bacterial tryptophan control, only the immediate availability of tryptophan is relevant. So, using the delay mechanism is a good addition to the toolbox but will not necessarily be used in all problems we might face.

This notation leads to a tree representation, where non-leaf nodes consist of operators, either binary (*AND*; *OR*) or unary (*NOT*) and leafs consist of the fuzzy set, the variable and the time delay. An example can be seen in Figure 10.1.

We must guarantee that we have at least one rule for each fuzzy set created for our gene of interest. We could consider having only one rule per gene and joining the various rules with an *OR* node, which would have the same effect in the defuzzification process. Nevertheless, this idea leads to higher trees and we choose to have several lower trees instead. This will cause only a minor impact on the application of the crossover operator (see Section 10.4.4 for further information). Therefore, our chromosome will consist of a "forest", with at least n_{fuzzy} trees, where n_{fuzzy} is the number of fuzzy sets created for our gene of interest.

Having more than one tree per fuzzy set can lead to having contradictory rules. Wang and Mendel [33] suggest having only one rule: the one with the maximum degree. We do not need to apply this rule because the *OR* interpretation in our algorithm will already do that, with the added benefit of being adaptive, meaning that in each different situation, a different rule may have the highest degree and will be chosen for application.

10.4.4 Genetic Operators

In order to define completely the proposed GP, we need to define the mutation and crossover operators. As is typical in evolutionary algorithms that use competing operators, the GP proposed in this paper uses a roulette wheel and assigns a time varying probability for the mutation and crossover operators.

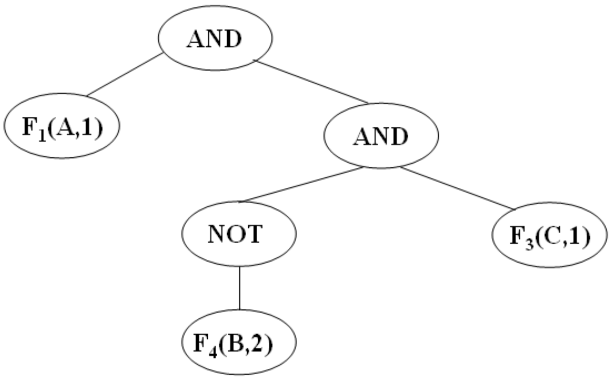


Fig. 10.1. Example of the representation. Inner nodes are the operators, while outer nodes represent a fuzzy set, a variable and a delay. In the figure, $F_1(A, 1)$ means that we are going to use the membership value of expression level of variable A in the previous time step (delay 1) to fuzzy set F_1.

The probability variation is due to the fact that in the beginning of each run we want to emphasize the exploitation aspect of the algorithm, by trying to allow the best characteristics in the population to propagate. This means that the crossover operator should have a higher choice probability. In the final generations, genetic convergence tends to occur and the population tends to be filled with similar individuals, which makes it more profitable for the GP to emphasize the exploration aspect, giving the mutation operator a higher probability.

Some researchers prefer to emphasize the exploration aspect in the early stages of the GA, but given the fact that the population is randomly initialized and applying further randomness on the population tends to yield little gain. On the other hand, giving the mutation operator higher probabilities in the later stages helps the GA fight the convergence effect that usually affects populations.

Therefore, we start with a high preference for the crossover operator (90% chance, for instance) and linearly decrease it with each successive generation. There are other strategies to make this chance (namely, quadratic and step decrease), but the choice of change mode does not seem to have a high impact on the GP performance.

Crossover Operator

The main goal of the crossover operator is to exchange information between two different individuals in a way similar to sexual reproduction, allowing the EA to exploit the best characteristics of the current population and hopefully transmitting it to the newly generated offspring.

The crossover operator used in this work is a version of the operator commonly used in genetic programming [19], having a performance that is quite similar to uniform crossover for genetic algorithms, both in its *modus operandi* and in the number of schemes preserved.

The operator works by randomly choosing sub-trees to interchange between the chosen parents, using the following algorithm:

```
For each parent do
    node $n_c$ = tree root
    initialize selection probability $p_s$
    While $n_c$ not equals to null do
        Make a random draw with probability $p_s$
        If $n_c$ is chosen
            Store node $n_c$ for the current parent
            Exit while
        Else
            raise probability $p_s$
        End if
    End While
End For
Create offspring by exchanging the selected sub-trees
```

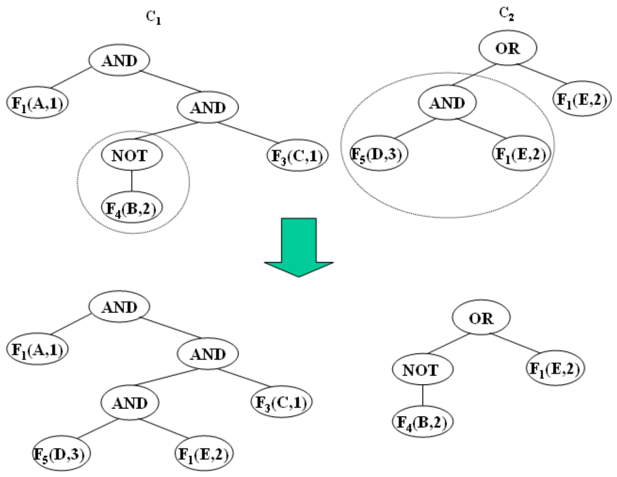


Fig. 10.2. Example of crossover action in two random parents

An example of this operator can be seen in Fig. 10.2. This strategy preserves the sub-expressions rooted at the selected nodes and therefore the crossover is not too disruptive with regard to the current population, which is a common problem with genetic programs.

Since there may be multiple rules per fuzzy set, there must be an additional control mechanism to choose which rules will exchange sub-trees. In this case, the crossover operator guarantees that the rules for a fuzzy set X in the first parent (C_1) only crosses with rules for the same fuzzy set in the second parent (C_2), even if there are more rules for this fuzzy set in one chromosome than in another. This means that if chromosome C_1 has two rules for fuzzy set X and chromosome C_2 has only one, the two rules from C_1 will cross with the single one from C_2. If the situation is reversed and C_1 has only one rule while has C_2 two or more, the number of rules in the resulting chromosome will still be equal to the number of rules in C_1.

If both of them have more than one rule for fuzzy set X, a random choice will be made between the alternative rules, only ensuring that each rule crosses at least once. Obviously, crossover generates two rules per operation, which are done applying the rules described here twice: first considering the order C_1/C_2 and secondly, the order C_1/C_2. Therefore, children with the characteristic structure of both parents will be generated.

We could create a multiple parent version of this crossover operator by allowing multiple cutting points in each parent and creating a pool of selected sub-trees that would be randomly selected to replace the cut branches of each parent. This version would be more disruptive and should be used together with an elitist operator and an increase in the number of generations per run.

Mutation Operator

The goal of the mutation operator is to insert random variation into the population so that we can explore areas of the search space that have not yet been visited. In order to perform this task, we create three different mutation operators:

1. rule mutation: The rule mutation randomly chooses one node in the rule tree and prunes the whole branch. Following this, a new sub-tree is generated using the same generator that created the initial population and the branch is then replaced. The population generator in this specific case is instructed to generate short trees (trees whose height is equal to or less than 3). An example of its operation is described in shown in Figure 10.3.
2. insertion mutation: Insertion mutation randomly chooses a fuzzy set for which to create a rule. The new rule is generated using the same random rule generator that generates the initial population. It is reasonable to give the inserted rules a high choice probability in this isolated creation, for we cannot raise the probability with the number of selections, as we did in the population initialization (see Section 10.4.7).
3. deletion mutation: Deletion mutation randomly chooses one rule to delete in one of the fuzzy sets available. In order to keep the rule base efficient, we cannot allow a fuzzy set to have zero rules, so the chosen rule will be deleted if it is not the only rule for a fuzzy set. In this case we will perform the

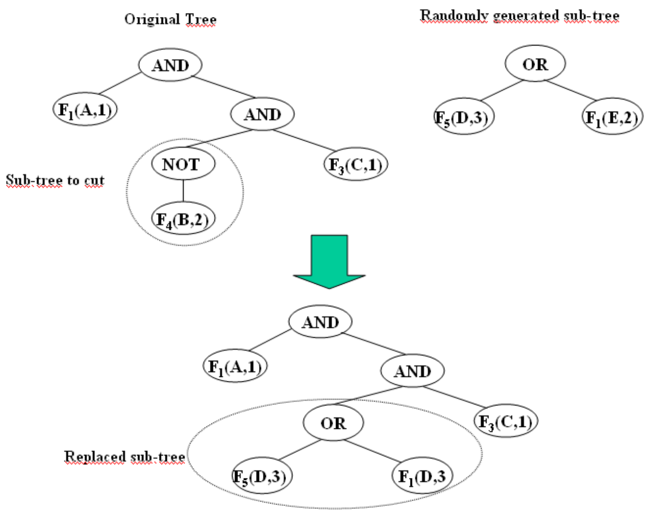


Fig. 10.3. The mutation operator chooses randomly a sub-tree to cut (circled) and replaces it with a newly generated short tree. The choice can be done algorithmically by performing a draw at each node with increasing probability. If the draw fails, we randomly choose to go to the left or right node and increase the probability in such a way that at a leaf, the draw probability will be 1.

random selection again, but one must be careful not to get into an infinite loop if all the fuzzy sets have exactly one rule, which can be avoided by allowing only a maximum number of tries for this operator. One must also be careful when excluding rules to verify whether the chosen tree is the only one that contains a rule inserted by the user.

10.4.5 Evaluation Function

In order to evaluate the performance of the chromosome, we will consider microarray data as a trajectory with N steps. This trajectory represents the "real" behavior of the network to be modeled, given the conditions it was submitted to. There are one or more genes of interest in that network, whose regulatory networks we intend to find.

In order to evaluate a proposed solution, the network it represents receives the first state of each trajectory and the GA calculates the intermediate and final steps for the genes of interest.

In order to calculate the expression levels at time t, one must decide whether to use the real or calculated values at each time step. As stated in the previous paragraph, the expression levels at time $t = 2$ are calculated using the real initial conditions at time $t = 1$ ($real_1$). But for every $t > 2$, how should we calculate the genes expression value ($calc_t$)? Should we use the real value at time $t - 1$ ($real_{t-1}$) or the previously calculated expression value at time $t - 1$ ($calc_{t-1}$)?

There are good arguments for both alternatives. Using the calculated values verifies if the network can model the entire trajectory, but allows errors to accumulate. Using the real values at each time point to calculate the next expression values verifies if the rules can, given a real value, predict the behavior of the network at the next point. Both are valid alternatives, but in our work we opted for the latter, because some experiments showed that some cumulative errors influenced the ability of the network to behave like the real one, specially when we use many fuzzy sets, causing them to have a small domain.

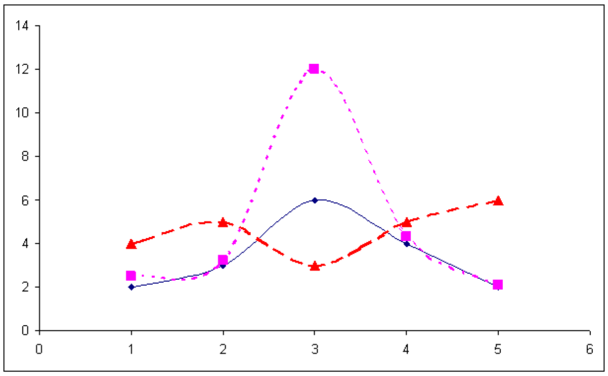
In order to avoid scale errors due to different expression levels at different time steps, the mean absolute percentage error (MAPE) was used, instead of using the absolute difference. This metric is defined by the following formula:

$$MAPE = \frac{1}{N} \sum_{t=1}^{N} \frac{real_t - calc_t}{real_t} \tag{10.2}$$

We will use the inverse of this function because the smaller the error, the bigger should be the evaluation function. We will also add 1 to the denominator in order to avoid infinity with perfect fit solutions. Therefore, the first try for our evaluation function will be:

$$F = \frac{1}{1 + MAPE} \tag{10.3}$$

The first impression is that this function will capture easily the good solutions, but there is a potential pitfall. In many situations a single outlier can make



x	real	dotted	traced
1	2	2,5	4
2	3	3,2	5
3	6	12	3
4	4	4,3	5
5	2	2,1	6

d(real, dotted)=36,39
d(real,traced)=34

Fig. 10.4. Example of the outlier effect. The solid line represents the real data and there are two calculated solutions represented by the dotted and the traced lines. Even though the dotted one models the underlying process better, it has an outlier at time t=3 that makes it MAPE grow and may cause it to be considered worse than the traced line whose behavior shows less resemblance to the real one.

the function obtain a low evaluation, even though it may correctly capture the changes in expression levels with time. An example of this problem is shown in Figure 10.4

In order to minimize this effect and reward those solutions that capture the directional changes in the target genes, we created a directional coefficient, which is given by the a function of the directional changes in the measured values ($real_t$) and the calculated values ($calc_t$), that is given by the following formula:

$$dc = \begin{cases} 0.7, \text{ if } (\uparrow real_t \wedge \downarrow calc_t) \vee (\downarrow real_t \wedge \uparrow calc_t); \\ 0.9, \text{ if } (\uparrow real_t \wedge \leftrightarrow calc_t) \vee (\downarrow real_t \wedge \leftrightarrow calc_t) \\ \qquad \vee (\leftrightarrow real_t \wedge \downarrow calc_t) \vee (\leftrightarrow real_t \wedge \uparrow calc_t); \\ 1.1, \text{ if } (\downarrow real_t \wedge \downarrow calc_t) \vee (\uparrow real_t \wedge \uparrow calc_t). \end{cases} \tag{10.4}$$

In this formula, a $\uparrow$ means that the expression level has grown at time t. Conversely, a $\downarrow$ means that the expression level has diminished and a $\leftrightarrow$ means that the expression level has remained the same. An expression level is considered constant if it varies less than $\pm 1\%$ from the previous time point. This evaluation improves the capture of directional expressional movement in time. For instance, if an expression value increases from time t to time $t+1$ and the calculated value decreases in the same period of time, the error value at this time point may not be very high, even as a percentage. Therefore, on adding this coefficient, every chromosome that has a high evaluation will calculate a solution that has high correlation with the function that maps real expression values changes over time.

The blessing of dimensionality, discussed above, may cause the GP to find an over-specific rule that matches the numbers available without really uncovering the underlying process that generated them. This will be reflected in "tall" trees that should be discouraged by a reduction in their evaluation. This is achieved by creating a coefficient $c \leq 1$ to multiply the evaluation function we discussed so far. The idea is that the higher the tree represented by the chromosome, the smaller is the coefficient c. Therefore, in the case where there are two different chromosomes with the same evaluation, the simpler and shorter one will be preferred. This does not imply that nature necessarily rewards simplicity, but rather that we have a small amount of data and should be cautious about over-fitting it. This coefficient is given by the following formula, dependent on the tree height h:

$$hc = \begin{cases} 1, & h \leq 2; \\ \frac{1}{h-1}, & h > 2. \end{cases} \tag{10.5}$$

The final formula is the multiplication of all three elements obtained in formulas 10.3, 10.4 and 10.5, creating the following evaluation function:

$$Eval(network) = hc * dc * F \tag{10.6}$$

This evaluation function is quite resilient and searches for the underlying process, not only achieving a data fitting, but also trying to avoid data over-fitting, by simplifying the regulatory process found and being aware of the dangers of the blessing of dimensionality.

10.4.6 Population Module and Execution Mechanism

In the GP proposed, we used an elitist population module and defined the following three different termination criteria:

- number of generations: no more than a predefined number of generations per run;
- stagnation: stop the run if the best solution stagnates for the last 20 generations. It was not used in our work, but an alternative solution is to increase the probability of the mutation operator, so that we give more emphasis to the exploration effect in order to find new solutions that break the stagnation. Of course, a fine control must be established so that the probability is decreased when the GP starts to have a better performance;
- quality of the solution found: stop if the data was fit to a maximum error of 1%. This number is arbitrary and appropriate for this specific problem. We know that the numbers obtained from microarrays are inherently imprecise (see Section 10.1.2) and that given the blessing of dimensionality, any perfect fit might consist of over-fitting. In other problems, the 1% rate of error might be too high;

Usually, genetic programs are seeded with large populations that execute for a large number of generations, because genetic programming operators tend to be very destructive, which causes a long execution time. The destructive effect can be minimized by using elitist strategies to preserve the best solutions and by using operators that prioritize the lower tree levels when deciding where to exchange material between genes.

In this work a different strategy was adopted. Ten independent runs with initial random populations were executed and the top 2% of each execution was used to seed the 11th run, whose initial population was completed randomly. This allowed for a faster execution time with smaller stagnation effects in each population, while allowing the best solutions found in different initial populations (seeded randomly) to interact in order to find a better solution. It is even possible to get an even bigger speed up by using a parallel execution of the algorithm, using 10 different machines that will allow their best solutions to migrate at the end of their execution.

10.4.7 Tree Processing

The rule generator used to initialize the population was created to guarantee that some chromosomes would incorporate prior knowledge, expressed as a specific set of rules defined by the user . This is an important advantage of fuzzy rules that was enforced in this work.

This incorporation was done by creating a random draw for each sub-tree generated to decide whether that sub-tree would be one of the rules inserted by the user. Since we wish for their value to be used as entered, placing those rules as descendants of a NOT node should be avoided. The probability assigned to this draw increased linearly with the number of chromosomes, up to the point where the rules were used or, in the final chromosome, the chance amounted to 100%, assuring rule usage.

A second passage was made through all generated chromosomes in order to assure that rules that were considered as "forbidden" by the user were absent from every chromosome. When cropping the tree to remove "forbidden" rules, a new sub-tree was generated with the characteristic discussed above.

After every reproduction/mutation cycle, a full pass through the new population is performed in order to verify that required rules are still present in the population and have not been disrupted by the genetic operators. If they are not present anymore, a new incorporation is performed, as discussed in the paragraph above.

A rule simplifier was created that allows us to substitute some sub-trees for simpler ones that still represent the same logical expression, allowing us to, for example, reduce expressions such as A AND A to their simpler form (in this case, A). The rules shown here have already gone through this simplification process. This simplification decreases the computational time spent by the proposed algorithm in the tree evaluation step, which is the longest one in the whole search process, decreases average tree height and also allows the genetic material exchanged between trees to be meaningful.

10.5 Results

The results obtained with the application of this algorithm to microarray data sets are described in great details in [23]. We will highlight their most important features in this section.

The data set tested was obtained from The Arabidopsis Internet Research project (TAIR) and measures the reaction to cold of *Arabidopsis thaliana*, giving the value of close to 8000 genes at seven data points. Because of this small number of measurements it was not feasible to use the time dependence available in the chromosomes. Therefore, every calculated value depends solely on the expression values available at the previous time point.

A. thaliana, like many plants, increases its freezing tolerance when exposed to low nonfreezing temperatures. This process of cold acclimation is a multigenic and quantitative trait that is associated with complex physiological and biochemical changes [17].

The genes that are hypothesized to be responsible for the cold response are 16062_s_at, 17520_s_at and 16111_f_at. The genes 13018_at and 13785_at are two of the genes regulated by the above named ones. The algorithm was applied to search for regulatory strategies for these last two genes, modeling correctly both trajectories and giving us the following interesting results:

- In the rules discovered for 13018_at, the three known regulators were present and a candidate regulator (17034_s_at) that was considered interesting enough by the biologists who provided the data to warrant further investigation in the near future. Other genes that show high correlation with known regulators were present in the rules, an effect that may be due to the small number of points available in the data set.
- In the second case (13785_at, some prior knowledge was included and the program was asked to include necessarily activation from 17520_s_at and preferentially an inhibition from 16111_s_at. The resulting rules included the required and the desirable relationships. Another interesting feature is the presence of gene 17034_s_at, which was also deemed interesting in the previous set of rules.
- In both cases, the rules present a few genes that don't seem to "belong" in terms of previous knowledge. This kind of spurious control relationship will always be present in any method and is a consequence of the blessing of dimensionality previously mentioned and cannot be avoided with such a small data set.

One can understand better those results by reviewing the best chromosome found for element 13875_at, which is described by the following rules:

(a) IF AND Low_Level_of_Expression(17413_s_at)
NOT High_Level_of_Expression(16111_f_at)
THEN Low_Level_of_Expression(13785_at)
(b) IF NOT Average_Level_of_Expression (15714_at)
THEN Low_Level_of_Expression(13785_at)
(c) IF NOT Average_Level_of_Expression (17834_at)
THEN Low_Level_of_Expression(13785_at)
(d) IF Low_Level_of_Expression (17421_s_at)
THEN Average_Level_of_Expression(13785_at)

(e) IF Average_Level_of_Expression (16062_s_at)
THEN Average_Level_of_Expression(13785_at)
(f) IF OR Low_Level_of_Expression(17050_s_at)
High_Level_of_Expression(16062_s_at)
THEN High_Level_of_Expression(13785_at)
(g) IF Average_Level_of_Expression (17034_s_at)
THEN High_Level_of_Expression(13785_at)
(h) IF NOT OR High_Level_of_Expression(16062_s_at)
Low_Level_of_Expression(15140_s_at)
THEN High_Level_of_Expression(13785_at)

Those rules have the following features worthy of note:

- Rules (a), (e) e (f) show relationships to known regulators;
- Rule (g) presents a new regulator (17034_s_at) that was considered promising by A. thaliana researchers;
- Rules (f) e (h) show relationships with elements 15140_s_at and 17050_s_at that are highly correlated to element 17520_s_at, which is a known regulator and is absent from the rules found. This situation is expected and is due to the small amount of data points.

10.6 Conclusion

The algorithm described was applied to a data set with many degrees of freedom and yielded interesting results. When applied to previously investigated regulation models, the results generated are very similar to known results in biology. This suggests that the algorithm may be a tool to uncover other regulation processes, but must be used with caution. All results found by this algorithm must be tested afterwards in a biological lab and all limitations associated with microarray data sets must be taken into consideration. Besides, if data is not scarce, other methods could be more effective, but, at the present time, obtaining a large number of microarray measurements is not financially feasible.

The approach described in this chapter to model gene expression is a very simplified view of the actual process, appropriate for exploratory data analysis. In reality, gene expression is a complex process regulated at several stages in the synthesis of proteins that also involves molecular movement and binding, which is a probabilistic event.

Apart from the regulation of DNA transcription, the best-studied form of regulation, the expression of a gene may be controlled during RNA processing and transport (in eukaryotes), RNA translation, and the post-translational modification of proteins. The degradation of proteins and intermediate RNA products can also be regulated in the cell, and the modeling of this process can be seen, for example, in [11]. Regulatory molecules can control the concentration and form of the product of each step. These regulators are usually fully-formed proteins, but any of the intermediate products (RNA, polypeptides, or proteins) also may act as regulators of gene expression. Reverse-engineering techniques usually concentrate on protein transcription control, mainly because DNA microarray technology has become an abundant data source. Measuring peptide, protein and metabolite regulators of gene expression is generally more difficult, and such data are not often available [14].

In this situation, the model proposed here serves as a rough sketch of the regulatory process, but it still must be considered as an initial step and must be augmented in the direction of methods that can understand and model cellular context, RNA translation and protein folding; thus understanding the network as a whole.

It is important to understand that most large-scale data sets contain only information from cells exposed to a single condition. The approach proposed here does not attempt to analyze the dynamics of complex biological networks. In order to carry out such an analysis of dynamics, we would have to deal with more interaction data sets under different cellular conditions, and more importantly, integrate with gene expression profile data under various conditions. The reader interested in biological networks can refer to [37].

It is also important to understand that many different genetic networks can generate the same phenotype, specially under data scarcity conditions, a phenomenon called "gene elasticity" and discussed in detail in [20]. The approach proposed here will not find many different possibilities for the same data in the same run, but given the fact that there is a random initialization step in the GP, it is possible to find different networks in different runs.

Besides incorporating the issues described above, a full model must also incorporate different regulatory mechanisms at different time points. For instance, a gene may regulate another only at a certain time point, while remaining quiescent during the rest of the interval evaluated. A possible solution would be to add, to each fuzzy rule in the base, an application condition that would determine when to apply it. Many difficult issues arise from this idea: for example, ensuring that at least one condition applies to a controlled gene at every time step and determining how to submit these rules to a genetic operator. We are currently studying the best way to represent these application conditions in our rules.

References

1. Alberts, B., Johnson, A., Lewis, J., et al.: Molecular biology of the cell, 5th edn. Garland Science (2007)
2. Arnone, M.I., Davidson, E.H.: The hardwiring of development: organization and function of genomic regulatory systems. Development 124, 1851–1864 (1997)
3. Bansal, M., della Gatta, G., di Bernardo, D.: Inference of gene regulatory networks and compound mode of action from time course gene expression profiles. Bioinformatics 22(7), 815–822 (2006)
4. Brazma, A., Rukliza, D., Viksna, J.: Reconstruction of gene regulatory networks under the finite state linear model. Genome Informatics 16(2), 225–236 (2005)
5. Carmona-Saez, P., Chagoyen, M., Rodriguez, R., et al.: Integrated analysis of gene expression by association rules discovery. BMC Bioinformatics 7(54) (2006)
6. Chen, J.J., Chen, C.-H.: Encyclopedia of biopharmaceutical statistics. In: Microarray Gene Expression, pp. 599–613. Informa Healthcare (2003)

7. Creighton, C., Hanash, S.: Mining gene expression databases for association rules. Bioinformatics 19(1), 79–86 (2003)
8. Dasgupta, D., Gomes, J.: Evolving fuzzy classifiers for intrusion detection. In: Proceedings of the 2002 IEEE Workshop on Information Assurance, US Military Academy (2002)
9. De Jong, H., Gouze, J.L., Hernandez, C., et al.: Qualitative simulation of genetic regulation models using piecewise-linear models. Bulletion of Mathematical Biology 66(2), 301–340 (2004)
10. Fogel, G.B., Corne, D.W.: Evolutionary computation in bioinformatics. Morgan Kaufmann, San Francisco (2003)
11. Foteinu, P., Yang, E., Saharidis, G.K., et al.: A mixed-integer optimization framework for the synthesis and analysis of regulatory networks. Journal of Global Optimization (online) (2007)
12. Freedman, D., Pisani, R., Purves, R.: Statistics, 4th edn. W. W. Norton Publisher (2007)
13. Freitas, A.A.: A survey of evolutionary algorithms for data mining and knowledge discovery. In: Ghosh, A., Tsutsui, S. (eds.) Advances in Evolutionary Computation, pp. 819–845. Springer, Heidelberg (2003)
14. Gardner, T.S., Fainth, J.J.: Reverse-engineering transcription control networks. Physics of Life Reviews 2(1), 65–88 (2005)
15. Gardner, T.S., Faith, J.J.: Reverse-engineering transcription control networks. Physics of Life Reviews 2(65), 88 (2005)
16. Georgii, E., Richter, L., Ruckert, U., Kramer, S.: Analyzing microarray data using quantitative association rules. Bioinformatics 21(suppl. 21), ii123–ii129 (2005)
17. Hannah, M.A., Heyer, A.G., Hincha, D.K.: A global survey of gene regulation during cold acclimation in Arabidopis thaliana. PLoS Genet. 1(2), 26–43 (2005)
18. Heaton, J.T.: Introduction to neural networks with java, 1st edn. Heaton Research, Inc. (2005)
19. Koza, J.R., Keane, M.A., Streeter, M.J., et al.: Genetic programming iv: Routine human-competitive machine intelligence (genetic programming), 1st edn. Springer, Heidelberg (2005)
20. Krishnan, A., Giuliani, A., Tomita, M.: Indeterminacy of reverse engineering of gene regulatory networks: The curse of gene elasticity. PLOS One (6), e652 (2007)
21. Laubenbacher, R., Stigler, B.: A computational algebra approach to the reverse engineering of gene regulatory networks. Journal of Theoretical Biology (229), 523–537 (2004)
22. Linden, R.: Algoritmos geneticos (genetic algorithms). Brasport (2006)
23. Linden, R., Bhaya, A.: Evolving fuzzy rules to model gene expression. BioSystems 88(1), 76–91 (2007)
24. Martinek, D.: A tree representation of fuzzy inference rules. In: Proceedings of 40th Spring International Conference MOSIS 2006, Prerov, CZ, p. 6 (2006)
25. Mistra, J., Schmitt, W., et al.: Iterative explorations of microarray gene expression patterns in a reduced dimensional space. Genome Research, 1112–1120 (2002)
26. Nuber, U. (ed.): DNA microarrays, 1st edn. Advanced Methods Series. Taylor and Francis, Inc., Abington (2005)
27. Pal, R., Datta, A., Bittner, M.L., Dougherty, E.: Intervention in context-sensitive probabilistic boolean networks. Bioinformatics 21(7), 1211–1218 (2005)
28. Ross, T.H.: Fuzzy logic with engineering applications. Wiley, Chichester (2004)
29. Schlitt, T., Brazma, A.: Current approaches to gene regulatory network modelling. BMC Bioinformatics 8(suppl. 6), 9 (2007)

30. Tegner, J., Yeung, M.K.S., Hasty, J., Collins, J.J.: Reverse engineering gene networks: Integrating genetic perturbations with dynamical modeling. PNAS 100(10), 5944–5949 (2003)
31. Van Someren, E.P., Wessels, L.F.A., Reinders, M.J.T.: Linear modeling of genetic networks from experimental data. In: Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, pp. 355–366. AAAI Press, Menlo Park (2000)
32. Wahde, M., Hertz, J.: Coarse-grained reverse engineering of genetic regulatory networks. BioSystems 55(1), 129–136 (2000)
33. Wang, L., Mendel, J.M.: Generating fuzzy rules by learning from examples. IEEE Transactions on Systems, Man and Cybernetics 22(6), 1414–1427 (1992)
34. Wang, Y., Tetko, I.V., Hall, M.V., et al.: Gene selection from microarray data for cancer classification–a machine learning approach. Computational Biology and Chemistry 29(1), 37–46 (2005)
35. Wehrli, A.V., Grzegorczyk, M., Husmeier, D.: Comparative evaluation of reverse engineering gene regulatory networks with relevance networks, graphical gaussian models and bayesian networks. Bioinformatics 22(20), 2523–2531 (2006)
36. Yang, Z.R., Thomson, R., et al.: Searching for discrimination rules in protease proteolytic cleavage activity using genetic programming with a min-max scoring function. BioSystems 72, 159–176 (2003)
37. Zhu, X., Gerstein, M., Snyder, M.: Getting connected: analysis and principles of biological networks. Genes and Dev. 21, 1010–1024 (2007)