Reverse Engineering of Temporal Boolean Networks from Noisy Data Using Evolutionary Algorithms

Carlos Cotta and José M. Troya

Dept. Lenguajes y Ciencias de la Computación, University of Málaga, ETSI Informática, Campus de Teatinos, 29071 - Málaga, SPAIN

Abstract

We consider the problem of inferring a genetic network from noisy data. This is done under the Temporal Boolean Network Model. Due to the hardness of the problem, we propose an heuristic approach based on the combined utilization of evolutionary algorithms (EAs) and other existing algorithms. The main features of this approach are the heuristic seeding of the initial population, the utilization of a specialized recombination operator, and the use of a majority-voting procedure in order to build a consensus solution. Experimental results provide support for the potential usefulness of this approach.

Key words: Biocomputation, Genetic Network Inference, Temporal Boolean Networks, Evolutionary Algorithms, Noisy Data

1 Introduction

Nearly all the information needed to set up and control the different processes that take place in a living cell is contained in its genes. According to the so-called Central Dogma of Biology, the information stored in DNA molecules is transcribed into RNA molecules that will in turn direct the production of proteins. When considered at cell scale, this is a complex dynamical process, since the activity of particular genes (i.e., whether they are expressed or not at a certain time) is regulated by the activity of other genes and their byproducts. It has been only recently that we have had the means for obtaining experimental data regarding these interactions: by using the DNA microarray technology

Email address: ccottap@lcc.uma.es (Carlos Cotta).

[17] we can monitor the activity of a whole genome in a single experiment. Huge amounts of data are becoming available thanks to the utilization of this technique. The challenge is now unraveling the complex interactions behind these data.

Genetic Networks are tools for modeling gene interactions that can be used to explain a certain observed behavior. In a genetic network, the functionality of a gene is described by the global effect it has on the cell, as a result of its interaction with other genes. Such interactions must be inferred from experimental data in order to construct an adequate genetic network for modeling the biological process under scrutiny. The way this inference is done depends on the particular genetic-network model used. For example, one can consider Bayesian Networks [20], Boolean Networks [1–4,22–24,36], Petri Nets [26], Qualitative Networks [4], and Weight Matrices [39] among others (see also [38]). Each of these approaches can be useful under different circumstances (available computational resources, properties of the experimental data, properties of the model sought, etc.). In this work we will focus on Temporal Boolean Networks (TBNs) [34], a generalization of the Boolean Network model that takes into account the time-series nature of the data, and tries to incorporate into the model the possible existence of delayed regulatory interactions among genes. The basic notions about this model of genetic networks will be presented in Section 2.

Several exact algorithms have been proposed for the inference of TBNs from experimental data. A brief outline of these algorithms will be provided in Section 2 as well. It turns out that the inference problem is very hard, and some of these algorithms can become impractical up from a certain problem size. This is specifically true if the data contains noise due to, e.g., measurement errors during acquisition. For this reason, the use of heuristic approaches is in order. In this sense, we will study the utilization of evolutionary algorithms (EAs) [9] for this purpose.

EAs are optimization techniques inspired on the principles of natural evolution, namely adaptation and survival of the fittest. They are based on the iterative generation of tentative solutions for the problem under consideration, and constitute a very appealing option for dealing with otherwise-hard-to-solve problems. In this work we consider the use of EAs for TBN inference from noisy data. The details of the application of EAs to this problem will be provided in Section 3.

It is important to notice that we consider the utilization of EAs not substituting but complementing other existing algorithms. In effect, EAs can successfully use information provided by the latter in order to build improved solutions. Section 4 will provide empirical evidence of this fact. This work will close with some discussion and prospects for future research in Section 5.

2 Background

In this section we will introduce the essentials of the TBN model considered in this work. Firstly, some basic definitions and notation will be presented in Subsection 2.1. Subsequently, we will provide some highlights on existing algorithms for the inference of TBNs in Subsection 2.2.

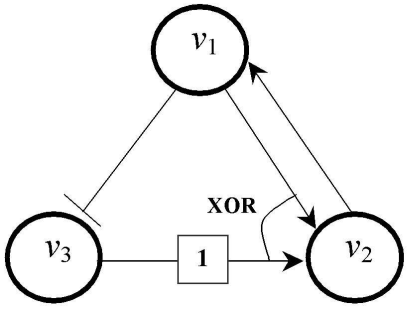
2.1 Temporal Boolean Networks

As mentioned in the previous section, TBNs are a generalization of Boolean Networks. It would be then appropriate to start defining the latter.

A Boolean Network is a tuple $BN(G, F)$, where $G(V, E)$ is a directed graph, and $F = \{f_v \mid v \in V\}$ is a set of Boolean functions, such that f_v is attached to vertex v. Let $K_G(v)$ be the in-degree of vertex v in graph G. Then, the signature of the Boolean function attached to v is $f_v : \mathbb{B}^{K_G(v)} \longrightarrow \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$. Each vertex v represents a different gene, and can be labeled with a binary value –0(OFF) or 1(ON)– that indicates its expression level. The existence of an edge (v', v) in G indicates that the value of gene v' exerts some influence on the value of gene v, i.e., v' is a regulatory factor of v. The precise way in which this regulation works, and how it is combined with other regulatory factors that bind to v is captured by means of the corresponding Boolean function f_v.

Having defined these concepts, the dynamics of the genetic network is modeled as follows: first of all, time is divided in discrete time steps. In each step, the values of all genes are synchronously updated using the attached Boolean functions. More precisely, the value of gene v at time $t + 1$ is computed by having function f_v being fed with the values at time t of all genes $v', v'', \cdots, v^{(K_G(v))}$ for which incoming edges to v exist. The output of f_v for this particular input is the expression level of v at time $t + 1$.

Let $\psi : V \longrightarrow \mathbb{B}$ be a labeling of vertices in V, and let Ψ be the set of all such labelings. Now, we define an example as a pair $(I, O) \in \Psi^2$. A Boolean network BN is said to be *consistent* with this example if it holds for each $v \in V$ that $f_v\left(I(v'), \cdots, I(v^{(K_G(v))})\right) = O(v)$. In the context of time-series data, we will have a sequence $\Lambda = \langle \lambda_1, \cdots, \lambda_m \rangle \in \Psi^m$, and we will be interested in checking the consistency of the network with examples $(\lambda_i, \lambda_{i+1})$, $1 \leq i < m$. Plainly, this represents the capability of the network for reproducing the observed data. In case the network were not consistent with the whole time series, it would make sense to measure the degree of agreement with it. This is done using the *accuracy* measure. First, let the error $\epsilon^{\Lambda}_{BN}(v)$ for gene v be defined as the fraction of states of v that were incorrectly predicted by the network across



INPUT			OUTPUT		
v_1	v_2	$v_3^{(-1)}$	v_1	v_2	v_3
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	1	1	0
1	1	1	1	0	0

Fig. 1. A Temporal Boolean Network, and the equivalent truth table defining its behavior. Notice that the edge (v_3, v_2) has a 1-step delay.

the time series Λ. Now the accuracy of the network for a time series Λ is

$$accuracy_{BN}(\Lambda) = 1 - \frac{1}{|V|} \sum_{v \in V} \epsilon^{\Lambda}_{BN}(v). \tag{1}$$

It can be easily seen that the accuracy of a network is 1.0 if, and only if, it is fully consistent with the time series Λ. In case more than one time series were available, i.e., $\Lambda_1, \cdots, \Lambda_q$, a combined accuracy measure can be computed by averaging the accuracy values for all time series Λ_i, $1 \leq i \leq q$.

Some of the assumptions behind the Boolean network model are clearly unrealistic [37], e.g., transcription processes are known to work asynchronously and at different rates, in contrast to the synchronous functioning of this model. Despite this, Boolean networks exhibit very interesting properties, such as simplicity and explanatory power. While idealized, they can constitute a valid starting point for gaining insights into the dynamics of genetic networks. Furthermore, the model can be extended to cope with its limitations to some extent. Temporal Boolean networks are one of such extensions.

Unlike the case of plain Boolean networks, in TBNs the state of a gene at a certain time step is not only relevant for the next time step; on the contrary, its regulatory influence can span across several time steps. The rationale for allowing this is twofold [34]: firstly, there may exist a latency period between the expression of a gene and the observation of its effect (e.g., because this effect depends on the prior binding with a certain inducer). Secondly, a gene g may exert an indirect regulatory influence on another gene g' through a third gene g'', i.e., g' regulates g'' and g'' regulates g; if g'' were not observable, this would appear as a delayed influence of g on g'.

In order to formally define a TBN we only need to specify a labeling of edges in the graph. Thus, a TBN is a tuple $TBN(\bar{G}, F)$, where F has the same interpretation as above, and G is a graph $G(V, E, \varphi)$ with φ being defined as a function $\varphi : E \longrightarrow \mathbb{N}$. Were a certain edge (v', v) be labeled with l, this would

mean that the state of v' at time t is relevant for computing the new state of v at time $t + (l + 1)$. Then, it can be seen that a plain Boolean network is a particular case of TBN in which all edges are labeled with 0. We will denote by T the maximum size of the time window in which the state of a gene can have regulatory effects, i.e., $T = 1 + \max_{e \in E} \varphi(e)$ (hence $T = 1$ for a plain Boolean network). An example of TBN is shown in Fig. 1.

2.2 Inference of TBNs

A number of algorithms have been proposed for the inference of TBNs from data. Actually most of them correspond to the simpler Boolean network model, but they can be readily extended to deal with TBNs. We will briefly overview some of them.

After the seminal work of Somogyi *et al.* [35], one of the first algorithms that were proposed is REVEAL (REVerse Engineering ALgorithm) [24]. This algorithm combines Information-Theory tools and exhaustive search in order to find a network consistent with the data (full consistency is not possible in general in the case of noisy data though; we will return to this point below). More precisely, the algorithm tries to identify an adequate set of inputs for each gene g by considering all possible k-tuples of genes for increasing values of k (1 up to n, the total number of genes). A k-tuple Γ is considered a valid input if the *mutual information* [31] between gene g and genes in Γ is equal to the entropy of g, i.e., Γ is enough to explain all state variations for g. If the underlying model behind the data is known to have in-degree bounded by K, then the complexity of this algorithm can be shown to be $O(mn\binom{n}{K}) = O(mn^{K+1})$, where m is the size of the data set.

Another algorithm was proposed by Akutsu *et al.* [2]. This algorithm was termed BOOL-1 in a later work [4], and consists of examining all possible K-tuples of inputs, testing all Boolean functions of each K-tuple until a consistent set of inputs is found. As it is the case for REVEAL, this algorithm has $O(mn^{K+1})$ worst-case complexity. Quite interestingly, the authors also showed that the size of the data set suffices to be $O(2^{2K}(2K + \alpha) \log n))$ in order to uniquely identify the underlying Boolean network with probability at least $1 - n^{-\alpha}$, as long as the expression patterns are given uniformly randomly. This is not the case in general: expression data exhibits some correlation due to its time-series nature, and the existence of attractors in the network dynamics [42]. This fact notwithstanding, this result still suggests that relatively small sets of data can be enough for identifying the underlying Boolean network. The same authors have developed another version of this algorithm –BOOL-2– to deal with noisy data [4]. The functioning of BOOL-2 is very similar to that of BOOL-1, but with an important difference: rather than discarding a certain Boolean function as soon as it reveals itself as inconsistent with the data, a certain number of mismatches θ is allowed ($\theta = \frac{m}{2^{2K+1}}$ in their theoretical

analysis). The computational complexity of BOOL-2 is the same as BOOL-1, i.e., $O(mn^{K+1})$.

Although both REVEAL, BOOL-1 and BOOL-2 run in polynomial time for any fixed K, the inference problem cannot be considered *fixed parameter tractable* (FPT) [18]. The notion of FPT emanates from the Parameterized Complexity paradigm [19], and tries to capture the fact that the problem hardness is concentrated on a certain parameter that when extracted from the problem input and kept fixed allows for an efficient resolution of the problem, i.e., in $O(f(k)n^c)$, where k is the parameter, f is an arbitrary function of k, and c is a constant independent of n and k[1]. Problems not in FPT have algorithms whose complexity is $O(n^{g(k)})$ –much harder in general– as it is the case of the algorithms mentioned above, and other approaches proposed for this purpose such as the predictor-chooser algorithm [22], or the Best-Fit Extension Problem [32].

This problem has been recognized in the literature, and more efficient algorithms have been sought. In this sense, Akutsu *et al.* designed an improved version of BOOL-1 that runs in $O(mn^K)$ [1], and a Monte-Carlo type randomized algorithm running in $O(m^{\omega-2}n^K + mn^{K+\omega-3})$, where $\omega < 2.376$ is the exponent of matrix multiplication [3]. Still, these improvements do not lead to FPT algorithms. As a matter of fact, it is now known that such FPT algorithms are theoretically impossible. This has been shown by Cotta and Moscato [13] by establishing the completeness for $W[2]$ (a parameterized complexity class substantially harder than FPT) of the k-Feature Set Problem, i.e., the problem of determining a subset of k out of n Boolean variables such that a certain target variable can be expressed as a function of the former.

A hard complexity barrier is thus being faced here. The fact that when dealing with TBNs an additional T^K factor must be considered in the complexity expressions above (to account for the T possible labels every edge in a K-tuple can have) only strengthens this barrier. Whenever the size of the problem instance allows it, any of the algorithms mentioned above would be a good choice for they guarantee finding the best solution. However, heuristic approaches are clearly required in general for larger problem instances.

One of these possible heuristic approaches is the ID3 algorithm [27], a well known algorithm in Machine Learning. This algorithm is based on the incremental construction of the input set for each variable using a greedy search. This search is guided by the information gain criterion, and thus exhibits some similitude with the REVEAL algorithm mentioned above. More precisely, a decision tree modeling the evolution of the target variable is generated, trying to select at each step the variable that provides the largest entropy reduction. This approach has been used in the context of TBN inference by Silvescu and Honavar [34] with encouraging results. In this work we propose an approach

based on the synergistic utilization of evolutionary algorithms and existing heuristics such as ID3. This approach will be presented in next section.

3 An EA-based Approach for Inferring TBNs

Evolutionary algorithms are heuristic search techniques based on the iterative generation of tentative solutions for a target problem [9]. Starting from a pool –*population*– of initial solutions –*individuals*– (generated at random, or by some other procedure), an EA repeatedly performs a process consisting of *selection* (promising solutions are picked form the pool), *reproduction* (new solutions are created by modifying selected solutions), and *replacement* (the pool is updated by replacing some existing solutions by the newly created ones). A *fitness* function measuring the goodness of solutions is used to drive the whole process, specially during the selection stage.

The generic template sketched above must be adequately instantiated in order to be deployed on a specific problem. This instantiation must be carefully done, trying to use and incorporate available problem-dependent knowledge in the algorithm. Despite some voices objecting to such a specialization of EAs in the past, it is nowadays accepted that it constitutes a recipe for success, as it has been shown both in theory [15,41] and in practice [11,16]. In this section we will show how to adapt the EA for the inference problem we are considering.

3.1 Representation and Evaluation

Choosing a representation of solution for the problem being considered is a crucial decision in any EA application. Binary representations were very common in the past, to the point of being considered predetermined, specially by part of the genetic-algorithm [21] community. This attitude has been judiciously criticized by many researchers (e.g., [7,16]), and now binary representations are not considered globally superior to other representations nor even preferable by default any longer. On the contrary, nowadays it is sought to use "natural" representations for the problem at hand, i.e., representations in which the relevant properties of solutions for the problem considered are explicitly shown [28].

In the problem of inferring TBNs from data, the relevant properties of solutions are the edges among genes, and their labels. The chosen representation is then based on these units. More precisely, we have considered solutions as a list of triplets (v', v, l), $v, v' \in V$, $l \in \{0, \cdots, T-1\}$, each one corresponding to an edge present in the TBN. It turns out that such lists can be efficiently stored and manipulated in terms of an $(n \times n)$-matrix M of natural numbers in the range $[0, T]$; having $M_{ij} > 0$ implies that an edge exists from v_i to

v_j, and its label is $M_{ij} - 1$ (conversely, the edge list can be viewed as a sparse encoding of this matrix; notice that in case we were considering plain Boolean networks, the natural representation of solutions would be binary). This matrix is not allowed to have more than K non-zero entries per column, i.e., K is the maximum in-degree of any node. Repairing by pruning the input set of a node is performed whenever its size is greater than allowed, e.g., after applying mutation.

EA individuals thus specify the wiring of the TBN. When submitted to evaluation, the Boolean functions attached to each vertex must be firstly learned. This is done by scanning the data set Λ and assigning the most common output to each input combination found in Λ. This is similar to the maximum likelihood estimation of parameters done in, e.g., Bayesian networks, and can be done in linear time in the number of patterns in Λ. Once these functions have been determined, the TBN is effectively evaluated using accuracy –recall Equation (1)– as the fitness function to be maximized.

3.2 Reproductive Operators

The construction of new individuals during reproduction is done in an EA via the application of two major mechanisms: recombination and mutation. The former consists of creating a new solution by picking information from a pair of selected solutions (commonly termed parents), as well as possibly using some exogenous information. The main goal of recombination is combining valuable parts of solutions that have been independently discovered, and hence can be seen as a somewhat exploitative procedure. As to mutation, it consists of cloning a single solution, subsequently performing some modifications on it. The purpose of mutation is introducing exploring capabilities in the EA by injecting new information in the solution pool. Precisely, the final performance of the algorithm will crucially depend on the adequate balance between exploration and exploitation during reproduction. This balance has been achieved in this work by using a heuristic recombination operator for boosting the exploitative capabilities of the EA, and a blind mutation operator for providing unbiased exploration.

Unlike classical recombination operators, the recombination operator used by the EA does not blindly mix edges from the parents, but tries to select the most relevant edges. Clearly, it is then necessary to firstly define a measure of relevance. Pairwise mutual information (MI) between the edge endpoints could serve as a first approximation. This measure has the advantage of being pre-computable, but it is inadequate for several reasons. First of all, it is a peer-to-peer measure that would induce a total order in the set of edges, i.e., some edges would be always preferable to some other edges. The recombination operator would thus turn into a naïve greedy selection that would make the EA quickly converge to the upper portion of the ordered list of edges. For

the same reason, it is a context-unaware measure, incapable of grasping the possible combined effect that a group of genes may have on another one.

To cope with these drawbacks, a alternative measure is proposed. Let $\Gamma = \{(v', l_1) \cdots, (v^{(s)}, l_s)\}$, $s = K_G(v)$, be the input set of v in one of the parents being recombined, and let $(v^{(i)}, l_i)$ be a regulatory factor that does not belong to the input set of v in the other parent. Let $\tilde{\Gamma} = \Gamma \setminus (v^{(i)}, l_i)$, and let $\langle\tilde{\Gamma}\rangle$ be a certain combination of states for regulatory factors in $\tilde{\Gamma}$. Then, the quality Q of this edge is:

$$Q\left((v^{(i)}, v, l_i)\right) = \sum P\left(\langle\tilde{\Gamma}\rangle\right) H\left(f_v(\Gamma)|\langle\tilde{\Gamma}\rangle\right) \tag{2}$$

where the sum ranges across all states $\langle\tilde{\Gamma}\rangle$ of regulatory factors in $\tilde{\Gamma}$, f_v is the Boolean function attached to v in the corresponding parent (available after having been computed during evaluation), and $H(f_v(\Gamma)|\langle\tilde{\Gamma}\rangle)$ is the entropy in the output of f_v when the states of $\tilde{\Gamma}$ are kept fixed to $\langle\tilde{\Gamma}\rangle$ and the state of $(v^{(i)}, l_i)$ varies. In the Boolean scenario considered, $H(f_v(\Gamma)|\langle\tilde{\Gamma}\rangle)$ collapses into one out of two values: 0.0 if $f_v(\Gamma)$ is constant for a certain state of $\tilde{\Gamma}$ no matter the state of $(v^{(i)}, l_i)$, and 1.0 if $f_v(\Gamma)$ changes when the state of $(v^{(i)}, l_i)$ varies. In the former situation, we have that $\langle\tilde{\Gamma}\rangle$ determines the value of $f_v(\Gamma)$, and the state of $(v^{(i)}, l_i)$ is irrelevant, while in the latter situation the state of $(v^{(i)}, l_i)$ is crucial for determining $f_v(\Gamma)$. By summing these values across all states of regulatory factors in $\tilde{\Gamma}$ an indication is obtained on how relevant $(v^{(i)}, l_i)$ is in the presence of $\tilde{\Gamma}$. The factor $P(\langle\tilde{\Gamma}\rangle)$ just normalizes this sum, taking into account the relative importance in the data set of each state $\langle\tilde{\Gamma}\rangle$ of $\tilde{\Gamma}$. This term is the fraction of expression patterns in the data set in which $\tilde{\Gamma}$ takes the precise state $\langle\tilde{\Gamma}\rangle$. Notice that if $(v^{(i)}, l_i)$ is relevant for determining v in a MI sense (e.g., if v were a function of $(v^{(i)}, l_i)$ alone), then it will also appear as relevant using Q. However, the reverse is not true since combined effects cannot be captured using MI as mentioned above.

Having defined this quality measure, the functioning of the recombination operator can be described as follows:

(1) Find common edges and copy them to the offspring.
(2) Let L be a list of non-common edges; compute quality values and sort L in decreasing quality values.
(3) Determine the number of edges the offspring will have (randomly using a binomial distribution centered in the parents' mean number of edges). Pick edges from L in order until the offspring is completed.

Regarding step 1, this property of respecting features shared by both parents has revealed itself as very positive in closely related domains such as the induction of Bayesian networks [14]. By doing so, a coarse skeleton of the TBN is being constructed as the population converges, being steps 2 and 3 responsible for the fine tuning of solutions. According to this template,

irrelevant edges (those with quality values near 0) will be filtered out. On the contrary, strong relationships (indicated with quality values near 1) will be preserved.

Unlike recombination, mutation is performed using a much more simple mechanism: a random edge (v', v, l) is selected and removed from the solution if it is present, or added to it otherwise (any (v', v, l') that existed would then be removed). The objective here is obtaining an unbiased source of fresh information that keep diversity in the population.

3.3 Initialization and Post-Processing

In order to have the EA started, it is necessary to create the initial population of solutions. This is typically addressed by randomly generating the desired number of solutions. When the alphabet used for representing solutions has low cardinality, this random initialization provides a more or less uniform sample of the solution space. The EA can subsequently start exploring the wide area covered by the initial population, in search of the most promising regions.

In this application, the alphabet used to represent solutions has T symbols (the possible values of each entry M_{ij}). The value of T will be small when compared to the population size (several tens or even hundreds of individuals usually) in most situations. Thus, the risk of not having any sample of a particular regulatory factor in the initial population is rather low, and it is not necessary to resort to systematic initialization procedures [30].

This random initialization can be complemented with the inclusion of heuristic solutions in the initial population. The EA can thus benefit from the existence of other algorithms, using the solutions they provide both for refinement and as information source. This is termed *seeding*, and it is known to be very beneficial in terms of convergence speed, and quality of the solutions achieved [29]. The potential drawback of this technique is having the injected solutions taking over the whole population in a few iterations, provoking the stagnation of the algorithm. This problem can be remedied in several ways, e.g., by using a non-fitness-proportionate selection mechanism such as ranking [40], or tournament [10]. These selection schemes focus on qualitative fitness comparisons instead of on quantitative comparisons (as for example roulette-wheel selection does), and hence are less prone to allow these super-individuals spread without control. This has been the approach we have considered: injecting the solution provided by the ID3 algorithm in the initial population, and using binary tournament as selection scheme.

Once the constituents of the algorithm have been fully specified, it can be deployed on particular data sets. EAs are stochastic techniques, and hence mul-

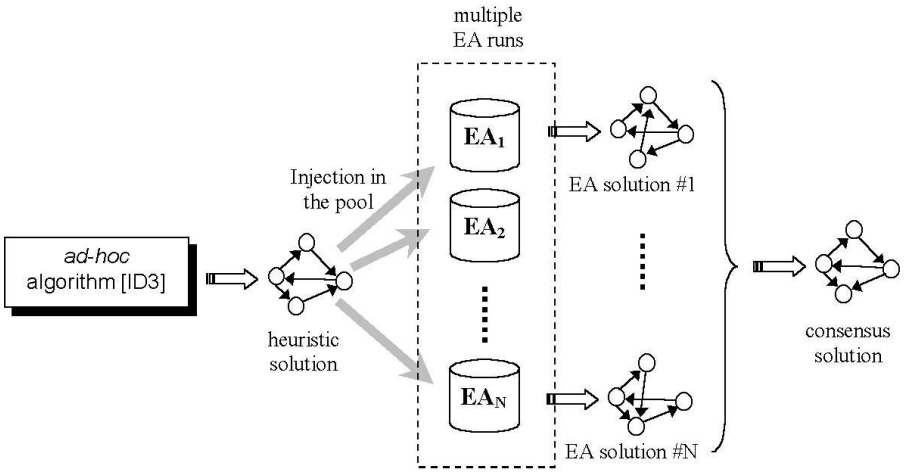


Fig. 2. Overview of the proposed approach.

tiple runs of the algorithm will provide different results. Quite surprisingly, a simple strategy such as performing N_R independent runs of the algorithm and keeping the best solution can be shown to provide in some cases superlinear speed-ups [33] with respect to individual runs. Since the fitness function is in this case defined as the accuracy of the network being evaluated, this speed-up would refer to an improved ability for finding high-accuracy networks. Nevertheless, it must be noted that the goal is to extract the underlying genetic network rather than accurately predicting gene expression levels. This distinction is important in the case of noisy data: although a correlation between these two objectives exists, they are not equivalent because the algorithm can obtain solutions of improved accuracy by fitting them to the noise. In fact, this is not a specific problem of EAs, but a problem of learning algorithms in general.

A possible solution to this problem can be found in the stochastic nature of the algorithm. By running it several times and keeping the best solution of each run, a set of potential networks is obtained. The ability of the EA for capturing in each of these networks a precise regulatory relationship depends on both the strength of such a relationship, and the level of noise. Unless the latter were too high, the EA will be capable of extracting a coarse skeleton of the network with the strongest links. Of course, the EA cannot recover very weak relationships located at the level of noise or under it, but has growing chances of finding them as their strength increases. Furthermore, when fitting to noise happens, it is likely that different runs of the EA will produce different solutions with different spurious relationships (due to the isolated evolution of different populations in those several runs). Based on this, the proposed strategy consists of performing N_R isolated runs, and collecting the best solution in each of them. Subsequently, a consensus network is produced by majority voting, i.e., for each pair of genes (v', v) the majority label l_{maj} across the N_R

networks is determined; if there are more networks voting for that label than networks voting for having no edge between v' and v then (v', v, l_{maj}) is added to the consensus network.

This latter post-processing step completes the description of the approach. An illustration of the whole process is given in Fig. 2.

4 Experimentation

In this section we will report experimental results of the proposed approach. Firstly, the parameters of the algorithm, and the characteristics of the benchmark will be described in Subsection 4.1. Subsequently, empirical data will be presented in Subsection 4.2.

4.1 Experimental Setup

The experiments have been conducted in order to test the ability of the proposed approach in recovering specific TBNs using a sample of their output. To do so, we have randomly generated networks of different sizes, in-degrees, and temporal delays. More precisely we have considered networks of $n = 16$ and $n = 32$ genes, in-degrees of $K = 5$ and $K = 7$ edges, and temporal delays from 0 up to 3 time steps (i.e., $1 \leq T \leq 4$). For each parameter combination we have generated 5 different networks. These have been created by firstly deciding at random a set of K inputs per gene, and subsequently assigning delays to the corresponding edges. Similarly to [34], the probability that a gene have a regulatory influence over a window of τ time steps is made proportional to ζ^τ. We have chosen ζ such that $\sum_{\tau=1}^{T} \zeta^\tau = K/n$. Once all networks have been generated, their output is sampled by randomly setting their states for T time steps, and then iterating their behavior for 100 time steps. This is repeated 5 times, so a total of 5 time series of 100 patterns each is obtained for each network. As a final step, white noise has been added to these time series. This has been done by randomly flipping entries of each of the time series with a certain probability p. Three different levels of noise have been considered: $p = 1\%$, $p = 2.5\%$, and $p = 5\%$. Each of the resulting groups of 5 noisy time-series constitute the input data for the EA.

The evolutionary algorithm used is a (50,1)-EA, i.e., an EA with a population size of 50 individuals, generating a single descendant in each step, and inserting this new individual in the population by substituting the worst one. Recombination is done with probability $p_R = 0.9$, and mutation of genes with probability $p_M = 1/n^2$. Binary tournament is used for selecting individuals for reproduction as mentioned in Subsection 3.3. Finally, the EA is run for a total number of evaluations $maxevals = 10,000$ for $n = 16$, and $maxevals = 20,000$ for $n = 32$. No fine tuning of these parameters has been attempted.

Using these parameters, the EA is run 20 times on each of these 5 time-series data sets. Besides considering the accuracy of the solutions obtained, their similarity to the target network is also measured. This can be done using the *sensitivity* and *specificity* metrics [22]. These are defined as follows: let M^{EA} be the matrix representing a solution provided by the EA; analogously, let M^{TBN} be the matrix corresponding to the target TBN. Then,

$$sensitivity_i(M^{EA}, M^{TBN}) = \frac{\sum_{j=1}^{n} \delta\left(M_{ji}^{EA}, M_{ji}^{TBN}\right)}{\sum_{M_{ji}^{TBN}>0} 1} \tag{3}$$

$$sensitivity(M^{EA}, M^{TBN}) = \frac{1}{n}\sum_{i=1}^{n} sensitivity_i(M^{EA}, M^{TBN}) \tag{4}$$

where δ is the Kronecker-delta function. This metric measures the degree to which real dependencies are captured in the EA solution (many of them for high –near 1.0– sensitivity, and few of them for low –near 0.0– sensitivity). Similarly, specificity is defined as sensitivity, but using $\sum_{M_{ji}^{EA}>0} 1$ as the denominator in Eq. (3). By doing so, specificity measures the degree to which the EA solution postulates spurious dependencies (many of them for low specificity, and few of them for high specificity).

Notice that both metrics –sensitivity and specificity– will yield the same result when the original network and the solution provided by the EA indicate the same number of dependencies for each gene.

4.2 Results

First of all, the accuracy results of the EA are shown in Fig. 3. Notice that in all cases the EA manages to improve the accuracy of the ID3 solution. This improvement is generally larger for low values of T. In this case, the EA is able to find the target network most of the times. This can be corroborated by taking a look at Fig. 4, where sensitivity values are shown, and Table 1, where the number of successful runs (i.e., runs in which the target network is found) is shown. As it can be seen, the performance line is located very close to 1.0 in this case (specificity values are equivalent in all cases to those for sensitivity, due to the fact that the networks generated by the EA have exactly K inputs per gene, as target networks do). Obviously, the search space is smaller for lower values of T, and this motivates a higher performance of the EA. For higher T, say $T \leq 2$, the EA is capable of improving over the results of ID3, but would require more iterations in order to increase accuracy.

The performance curve of the EA drops at a slightly higher rate in the case $K = 7$. However, this is also the case for ID3, and reflects the higher difficulty of this set of instances. Actually, this in-degree value can be considered as rather high since genes are believed to be influenced on average by no more

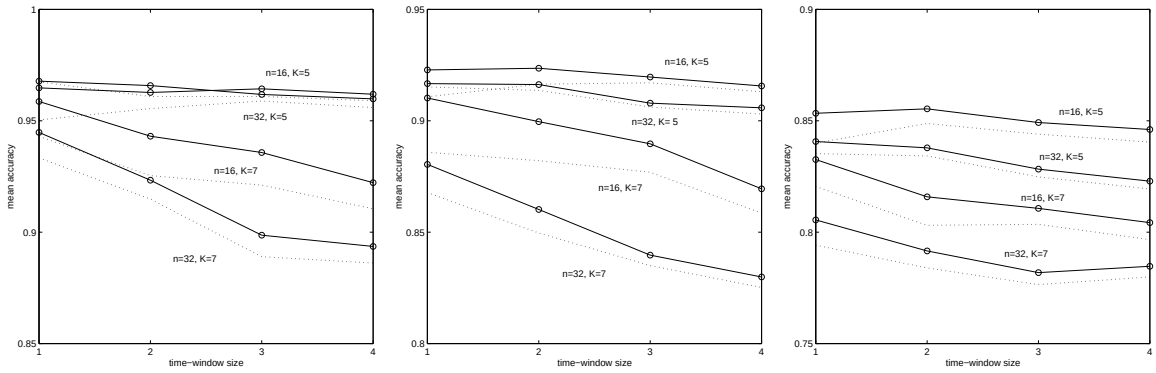


Fig. 3. Accuracy of the TBNs generated by the GA (solid line) for different levels of noise (1%, 2.5%, and 5% from left to right). The results of ID3 are included as a reference (dotted line).

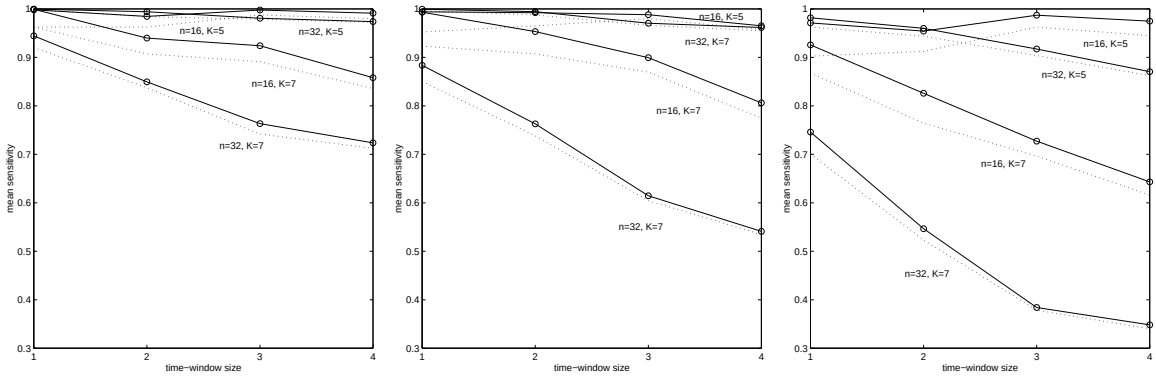


Fig. 4. Sensitivity of the TBNs generated by the GA (solid line) for different levels of noise (1%, 2.5%, and 5% from left to right). The results of ID3 are included as a reference (dotted line). Specificity values are equivalent in this case.

Table 1
Number of times (out of 100) the EA finds the target TBN for different network sizes, in-degree bounds, time-window sizes, and levels of noise.

		$K=5$				$K=7$			
size	noise	$T=1$	$T=2$	$T=3$	$T=4$	$T=1$	$T=2$	$T=3$	$T=4$
	1%	92	57	80	60	92	11	0	0
$n=16$	2.5%	78	62	58	43	72	20	0	0
	5%	45	44	42	20	33	0	0	0
	1%	96	56	25	48	0	0	0	0
$n=32$	2.5%	89	60	20	14	0	0	0	0
	5%	20	18	20	0	0	0	0	0

Table 2
Number of times (out of 5) the TBNs generated by the ID3 algorithm agree with target TBNs for different network sizes, in-degree bounds, time-window sizes, and levels of noise.

		$K=5$				$K=7$			
size	noise	$T=1$	$T=2$	$T=3$	$T=4$	$T=1$	$T=2$	$T=3$	$T=4$
	1%	1	1	2	1	0	0	0	0
$n=16$	2.5%	0	1	1	1	0	0	0	0
	5%	0	0	1	0	0	0	0	0
	1%	4	2	1	2	0	0	0	0
$n=32$	2.5%	2	2	1	0	0	0	0	0
	5%	0	0	1	0	0	0	0	0

Table 3
Number of times (out of 5) the EA consensus TBN agrees with the target TBN for different network sizes, in-degree bounds, time-window sizes, and levels of noise.

		$K=5$				$K=7$			
size	noise	$T=1$	$T=2$	$T=3$	$T=4$	$T=1$	$T=2$	$T=3$	$T=4$
	1%	5	3	5	4	5	1	0	0
$n=16$	2.5%	4	4	4	3	5	1	0	0
	5%	3	2	2	0	2	0	0	0
	1%	5	3	1	2	0	0	0	0
$n=32$	2.5%	5	3	1	1	0	0	0	0
	5%	1	2	1	0	0	0	0	0

than eight to ten other genes [8]. As it was the case for increasing T, longer evolution times may be required for larger K.

Finally, consider the consensus networks generated by the EA. Tables 2 and 3 show the number of times the target network could be successfully recovered when using ID3 and the evolutionary approach respectively. Notice the clear improvement in the number of hits in the latter. Furthermore, by using this consensus construction it is possible to recover the original network in situations in which individual runs of the EA could not do it consistently.

5 Discussion and Future Work

An evolutionary approach for the inference of genetic networks from noisy data has been presented in this work. This approach can exploit pre-existing

heuristics by incorporating the solutions they provide to the initial population. This seeding boosts convergence towards probably optimal solutions. It also utilizes a majority-voting mechanism for constructing a consensus solution. This appears as a useful option for dealing with noisy data, since spurious relationships introduced in the EA solutions can be filtered this way. Of course, this could not be the case if such unrealistic relationships were consistently present in the solutions; then again, this would indicate the presence of some degeneracy in the data, e.g., due to high noise.

The empirical results obtained from its evaluation are encouraging. It has been shown that the results of an ad-hoc algorithm (ID3) can be improved. This has been achieved despite the fact that the optimization task the EA faced was really challenging. Recall that a limit on the complexity (the in-degree) of the networks generated by the EA was imposed, and the target network was located precisely on that limit. This means that the EA had to find this boundary, and then explore it, seeking the best solution by moving on it or below it. Certainly, the search capabilities of the EA would be stronger if it were allowed to temporarily travel above the boundary. In this sense, notice that the accuracy measure used as guiding function is monotonic in the in-degree: if the EA is allowed a higher level of complexity in the networks, it will not descend to lower levels since accuracy can only improve when new edges are added to a network. It would be thus precise to introduce a penalization term in the fitness function to amount for the complexity of solutions, and make the EA look for simpler solutions if accuracy is not strongly degraded. We are currently working in this, and the preliminary results we are obtaining indicate that it can produce improved solutions.

We would also like to emphasize the generalizability of the approach. We have considered the Temporal Boolean Network model of genetic networks, and ID3 as the heuristic for seeding the initial population. The latter can be easily changed for any other heuristic; hence, if an improved algorithm is devised, it can be readily used to seed the initial population. As to the underlying model, it is also possible to change it without changing the philosophy of the approach. It would be then necessary to change some of the EA details, such as the representation and the operators. In this sense, there exist some works dealing with the application of EAs for finding genetic networks under other models, e.g., [5,6,25]. These works could pave the way for the application of this approach to other models of genetic networks.

Notes

[1]Vertex Cover is a paradigmatic example of an NP-hard problem that is in

FPT: it can be solved in linear time in the number of vertices when the size of the vertex cover sought is fixed [12].

References

[1] T. Akutsu, S. Miyano, and S. Kuhara. Fast identification of boolean networks. Technical Report 99-AL-66, Information Processing Society of Japan, 1999.

[2] T. Akutsu, S. Miyano, and S. Kuhara. Identification of genetic networks from a small number of gene expression patterns under the boolean network model. *Pacific Symposium on Biocomputing*, 4:17–28, 1999.

[3] T. Akutsu, S. Miyano, and S. Kuhara. Algorithms for identifying boolean networks and related biological networks based on matrix multiplication and fingerprint function. In *RECOMB 2000*, pages 8–14, Tokyo, Japan, 2000. ACM Press.

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

[5] S. Ando and H. Iba. Inference of gene regulatory model by genetic algorithms. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pages 712–719, Seoul, Korea, 2001. IEEE Press.

[6] S. Ando and H. Iba. Inference of linear gene regulatory model –reverse engineering by genetic algorithms–. In *Proceedings of Atlantic Symposium on Computational Biology, and Genome Information Systems & Technology*, Durham NC, 2001. Association for Intelligent Machinery.

[7] J. Antonisse. A new interpretation of schema notation that overturns the binary encoding. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 86–91, San Mateo, CA, 1989. Morgan Kaufmann.

[8] A. Arnone and B. Davidson. The hardwiring of development: Organization and function of genomic regulatory systems. *Development*, 124:1851–1864, 1997.

[9] T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, New York NY, 1997.

[10] T. Bickle and L. Thiele. A mathematical analysis of tournament selection. In L.J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, San Francisco, CA, 1995. Morgan Kaufmann.

[11] L.D. Chambers, editor. *The Practical Handbook of Genetic Algorithms: Applications*. CRC Press, Boca Ratón FL, 2000. Second edition.

[12] J. Chen, I.A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science*, pages 313–324. Springer-Verlag, Berlin Heidelberg, 1999.

[13] C. Cotta and P. Moscato. The k-Feature Set problem is $W[2]$-complete. *Journal of Computer and System Sciences*, 67(4):686–690, 2003.

[14] C. Cotta and J. Muruzábal. Towards a more efficient evolutionary induction of bayesian networks. In J.J. Merelo et al., editors, *Parallel Problem Solving from Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 730–739. Springer-Verlag, Berlin Heidelberg, 2002.

[15] J. Culberson. On the futility of blind search: An algorithmic view of "no free lunch". *Evolutionary Computation*, 6(2):109–128, 1998.

[16] L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York NY, 1991.

[17] J.L. DeRisi, V.R. Lyer, and P.O Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–686, 1997.

[18] R. Downey and M. Fellows. Fixed parameter tractability and completeness I: Basic theory. *SIAM Journal of Computing*, 24:873–921, 1995.

[19] R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

[20] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. *Journal of Computational Biology*, 7:601–620, 2000.

[21] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Harbor, 1975.

[22] T.E. Ideker, V. Thorsson, and R.M. Karp. Discovery of regulatory interactions through perturbation: Inference and experimental design. *Pacific Symposium on Biocomputing*, 5:305–316, 2000.

[23] S.A. Kaufmann. *The Origins of Order, Self-Organization, and Selection in Evolution*. Oxford University Press, 1993.

[24] S. Liang, S. Fuhrman, and R. Somogyi. REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. *Pacific Symposium on Biocomputing*, 3:18–29, 1997.

[25] Y. Maki, D. Tominaga, M. Okamoto, S. Watanabe, and Y. Eguchi. Development of a system for the inference of large scale genetic networks. *Pacific Symposium on Biocomputing*, 6:446–458, 2001.

[26] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid Petri net representation of gene regulatory network. *Pacific Symposium on Biocomputing*, 5:338–349, 2000.

[27] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[28] N.J. Radcliffe. Non-linear genetic representations. In R. Männer and B. Manderick, editors, *Parallel problem Solving from Nature II*, pages 259–268, Amsterdam, 1992. Elsevier Science Publishers.

[29] C. Ramsey and J.J. Grefensttete. Case-based initialization of genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo CA, 1993. Morgan Kauffman.

[30] C.R. Reeves. Using genetic algorithms with small populations. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 92–99, San Mateo CA, 1993. Morgan Kaufmann.

[31] C.E. Shannon. The mathematical theory of communication. *Bell System Technical Journal*, 27:379–423/623–656, 1948.

[32] I. Shmulevich, O. Yli-Harja, and J. Astola. Inference of genetic regulatory networks under the best-fit extension paradigm. In W. Zhang and I. Shmulevich, editors, *Computational and Statistical Approaches to Genomics*. Kluwer Academic Publishers, Boston MA, 2002.

[33] R. Shonkwiler. Parallel genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 199–205, San Mateo CA, 1993. Morgan Kauffman.

[34] A. Silvescu and V. Honavar. Temporal boolean network models of genetic networks and their inference from gene expression time series. *Complex Systems*, 13(1):54–70, 2001.

[35] R. Somogyi, S. Fuhrman, M. Askenazi, and A. Wuensche. The gene expression matrix: towards the extraction of genetic network architectures. In *Proceedings of the Second World Congress of Nonlinear Analysts (WCNA96)*, volume 30(3), pages 1815–1824. Elsevier Science Publishers, 1997.

[36] R. Somogyi and C. Sniegoski. Modeling the complexity of genetic networks: Understanding multigenetic and pleiotropic regulation. *Complexity*, 1(6):45–63, 1996.

[37] Z. Szallasi and S. Liang. Modeling the normal and neoplastic cell cycle with "realistic boolean genetic networks": Their application for understanding carcinogenesis and assesing therapeutic strategies. *Pacific Symposium on Biocomputing*, 3:66–76, 1998.

[38] E.P. van Someren, L.F.A. Wessels, and M.J.T. Reinders. Genetic network models: A comparative study. In *Proceedings of SPIE, Micro-arrays: Optical Technologies and Informatics (BIOS01)*, volume 4266, pages 236–247, San José, CA, 2001.

[39] D.C. Weaver, C.T. Workman, and G.D. Stormo. Modeling regulatory networks with weight matrices. *Pacific Symposium on Biocomputing*, 4:112–123, 1999.

[40] D. Whitley. Using reproductive evaluation to improve genetic search and heuristic discovery. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 116–121, Hillsdale NJ, 1987. Lawrence Erlbaum Associates.

[41] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[42] A. Wuensche. Genomic regulation modeled as a network with basins of attraction. *Pacific Symposium on Biocomputing*, 3:66–76, 1998.