On the Evolutionary Inference of Temporal Boolean Networks

Carlos Cotta

Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga - SPAIN
ccottap@lcc.uma.es

Abstract. The problem of inferring genetic networks under the Temporal Boolean Network model is considered here. This is a very hard problem for which an heuristic approach is proposed. This approach is based on the use of evolutionary algorithms (EAs) to refine the results of a specialized algorithm (ID3). Experimental results provide support for the usefulness of this approach, showing a consistent enhancement of the ID3 solutions.

1 Introduction

Genetic Networks are tools for modeling gene interactions that can be used to explain a certain observed biological 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 [9], Boolean Networks [18], Petri Nets [12], and Weight Matrices [19] among others. In this work we will focus on Temporal Boolean Networks (TBNs) [17], 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. It turns out that the inference problem is very hard, and some of these algorithms can become impractical up from a certain problem size. For this reason, the use of heuristic approaches is in order. In this sense, we will study the utilization of evolutionary algorithms (EAs) [4] for this purpose. 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 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, $1 \leq i \leq q$.

Temporal Boolean networks are a extension of BNs that tries to overcome some of the limitations of the basic model, i.e., the synchronous updating of gene values. 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.

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, the state of v' at time t would be relevant for computing the new state of v at time $t+(l+1)$. It is then easy to see 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).

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.

One of the first algorithms that were proposed is REVEAL (REVerse Engineering ALgorithm) [11]. This algorithm combines Information-Theory tools and exhaustive search in order to find a network consistent with the data. 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* 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.* [1]. This algorithm was termed BOOL-1 in a later work [2], 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.

Since the number of network topologies is $O(n^{K+1})$, these algorithms can be essentially viewed as brute-force approaches. As a matter of fact, it has been shown by Cotta and Moscato [6] that this bound cannot be improved unless a very unlikely condition regarding the structure of parameterized complexity classes held [8].

This hardness has motivated the utilization of heuristic approaches. A popular one is ID3 [13], 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 guided by the information gain criterion. The approach presented in next section is based on the synergistic utilization of evolutionary algorithms and existing heuristics such as ID3.

3 An EA-based Approach for Inferring TBNs

In this section we will show how to deploy EAs on the inference problem we are considering. Firstly, the representation and evaluation function utilized are described in Subsection 3.1. Then, the procedures used for initialization and reproduction are discussed in Subsection 3.2.

3.1 Representation and Evaluation

Choosing a representation in which the relevant properties of solutions for the problem considered are explicitly shown [14] is crucial for the success of the EA. 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). 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 Initialization and Operators

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 (as it is here the case), 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.

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 [16]. In order to avoid injected solutions taking over the whole population in a few iterations, a non-fitness-proportionate selection mechanism (focusing on qualitative fitness comparisons instead of on quantitative comparisons) must be used. This has been the approach we have considered: injecting solutions provided by the ID3 algorithm in the initial population, and using binary tournament [5].

Once the population has been created, the EA conducts its search using two reproductive operators, recombination and mutation. One of the advantages of the representation chosen is the fact that it is *orthogonal* [15], that is, any $(n \times n)$-matrix M of natural numbers in the range $[0, T]$ represents a valid TBN (there are no constraints, e.g., regarding the presence of cycles as it is the case in bayesian networks). This means that classical operators can be used. In this case, we have considered the utilization of uniform crossover (UX) for recombination. This operator ensures a high interchange of gene-values during recombination. As to mutation, it is performed using a 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.

4 Experimentation

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, following the procedure described in [17]. 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.

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 3.2. Finally, the EA is run for a total number of evaluations $maxevals = 10,000$ for $n = 16$, and $maxevals = 20,000$ for $n = 32$.

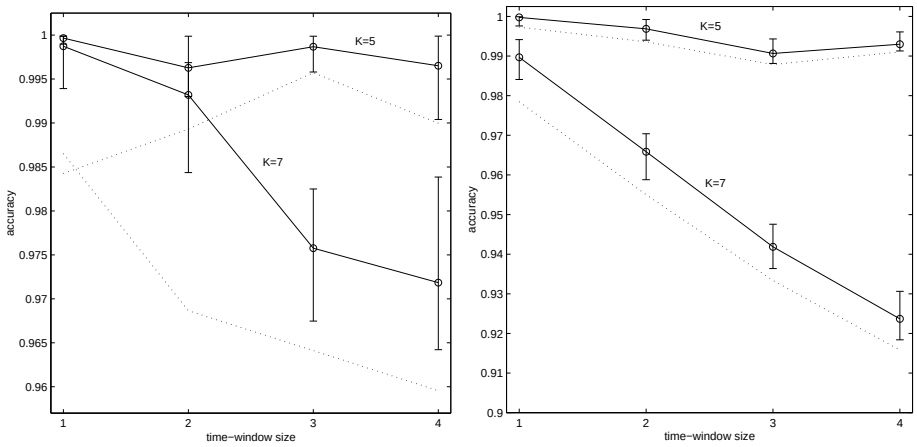

Fig. 1. Accuracy of the TBNs generated by the EA (solid line). The results of ID3 are included as a reference (dotted line). (Left) $n = 16$ (right) $n = 32$

Table 1. Percentage of success of the EA and the ID3 algorithm in finding the target TBN for different network sizes, in-degree bounds, and time-window sizes.

		$K = 5$				$K = 7$			
algorithm	size	$T = 1$	$T = 2$	$T = 3$	$T = 4$	$T = 1$	$T = 2$	$T = 3$	$T = 4$
EA	$n = 16$	95%	69%	82%	71%	81%	43%	0%	1%
	$n = 32$	97%	60%	26%	51%	9%	0%	0%	0%
ID3	$n = 16$	20%	20%	40%	40%	20%	0%	0%	0%
	$n = 32$	60%	40%	20%	40%	0%	0%	0%	0%

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 [10]. These are respectively defined as the fraction of matched edges (edges in both the generated solution and in the original TBN) with respect to the total number of edges in the generated solution or in the original TBN. Notice that both metrics 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.

First of all, the accuracy results of the EA are shown in Fig. 1. 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. 2, 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

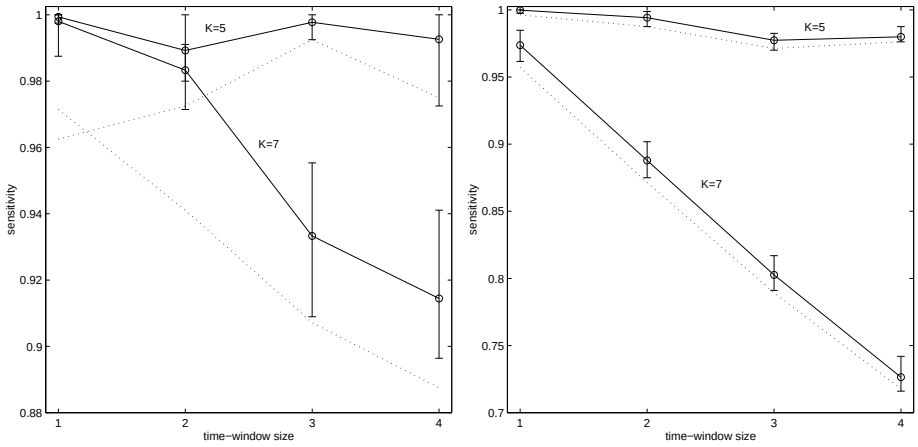

Fig. 2. Sensitivity of the TBNs generated by the EA (solid line). The results of ID3 are included as a reference (dotted line). Specificity values are equivalent in this case. (Left) $n = 16$ (right) $n = 32$

the fact that the networks generated by the EA have exactly K inputs per gene, as target networks do).

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 than eight to ten other genes [3]. Obviously, the search space is also larger for increasing K, so longer evolution times may be required in this case.

5 Conclusions

An evolutionary approach for the inference of genetic networks 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.

The empirical results obtained from its evaluation are encouraging. It has been shown that the results of a specialized algorithm (ID3) can be consistently improved. In this sense, we would like to emphasize the generalizability of the approach: ID3 has been considered as the heuristic for seeding the initial population, but it 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.

Future work will be directed to test the applicability of this approach in more complex scenarios, e.g., assuming the existence of white noise in the data. Work is in progress here. Another interesting line for future developments is the utilization of ideas taken from a related field as it is the inference of bayesian networks [7].

Acknowledgements This work is partially supported by Spanish MCyT and FEDER under contract TIC2002-04498-C05-02.

References

1. 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.
2. T. Akutsu, S. Miyano, and S. Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16(8):727–734, 2000.
3. A. Arnone and B. Davidson. The hardwiring of development: Organization and function of genomic regulatory systems. *Development*, 124:1851–1864, 1997.
4. T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, New York NY, 1997.
5. 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.
6. C. Cotta and P. Moscato. The k-Feature Set problem is $W[2]$-complete. *Journal of Computer and Systems Science*, 2003. In press.
7. C. Cotta and J. Muruzábal. Towards a more efficient evolutionary induction of bayesian networks. In *Parallel Problem Solving from Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 730–739. Springer-Verlag, Berlin Heidelberg, 2002.
8. R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.
9. 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.
10. 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.
11. 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.
12. 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.
13. J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
14. 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.
15. N.J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10:339–384, 1994.
16. 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.
17. 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.
18. R. Somogyi and C. Sniegoski. Modeling the complexity of genetic networks: Understanding multigenetic and pleiotropic regulation. *Complexity*, 1(6):45–63, 1996.
19. D.C. Weaver, C.T. Workman, and G.D. Stormo. Modeling regulatory networks with weight matrices. *Pacific Symposium on Biocomputing*, 4:112–123, 1999.