

# On the use of Population Based Incremental Learning to do Reverse Engineering on Gene Regulatory Networks

Leon Palafox

School of Electrical Engineering  
The University of Tokyo  
Tokyo, Japan  
Email: leon@iba.t.u-tokyo.ac.jp

Hitoshi Iba

Graduate School of Information, Science and Technology  
The University of Tokyo  
Tokyo, Japan  
Email: iba@iba.t.u-tokyo.ac.jp

**Abstract**—Gene Regulatory Networks (GRNs) describe the interactions between different genes. One of the most important tasks in biology is to find the right regulations in a GRN given observed data. The problem, is that the data is often noisy and scarce, and we have to use models robust to noise and scalable to hundreds of genes.

Recently, Recursive Neural Networks (RNNs) have been presented as a viable model for GRNs, which is robust to noise and can be scaled to larger networks. In this paper, to optimize the parameters of the RNN, we implement a classic Population Based Incremental Learning (PBIL), which in certain scenarios has outperformed classic GA and other evolutionary techniques like Particle Swarm Optimization (PSO). We test this implementation on a small and a large artificial networks. We further study the optimal tuning parameters and discuss the advantages of the method.

## I. INTRODUCTION

Through the ages, human beings have suffered from various diseases, while we can prevent some of them with a good diet and exercise, others get triggered by specific gene interactions that happen inside our cells. For example, Cancer and Diabetes are the result of genes being abnormally activated or suppressed in our bodies [1], [2]. These diseases are sometimes fatal, and we lack the means to prevent them. Learning the dynamics among genes in a cell is important, because it allows us to know the genes' interactions, giving important information on the dynamics of illnesses and cell processes. We call these dynamics Gene Regulatory Networks (GRNs)

Microarray data [3] measures the gene's dynamics and represents them as gene's expressions levels over a time series (Fig. 1); we can use this data, along with modeling techniques, to infer the genes' regulations and analyze different genetic conditions. The microarray data, however, is noisy [4], and often, different methods will have problems finding the real regulations among genes. Not only that, but different gene's interactions can also generate similar microarray data.

When we do network inference, we have two problems: the network's model selection and finding the real gene's interactions instead of other interaction that responds to the same data.

For the modeling problem, some groups have used graphical models, which represent the network as a connected graph, with the nodes as the genes and the edges representing the regulations among them, i.e., Bayesian Networks [5], [6] and Boolean Networks [7]. System dynamics models [8] portray the interactions as a set of ordinary differential equations (ODEs), where we wish to find the parameters that best fits the time series. Since the ODEs often are a nonlinear set of equations, to find their parameters we have to use evolutionary optimization techniques. Different groups have used different techniques such as Differential Evolution [9], Genetic Algorithms [10], genetic programming [11], etc.

These models, however good, take a long time to find the regulations, or interactions, in small to medium sized gene chains (5-20), and since the problem at hand tries to find the regulations for hundreds of genes, it becomes infeasible to use these techniques for large networks.

Vohradsk, in his paper [12], showed that the Recursive Neural Networks (RNNs) were a feasible biological model for GRNs, and Xu et al [13] proved that using evolutionary techniques like Particle Swarm Optimization (PSO) we could find the right parameters for the Recursive Neural Net problem applied to GRNs. Yet, RNNs have been used scarcely to find the regulations in gene networks, in spite of the advantages it offers against other models like the system dynamics models. For example, RNNs are faster than the traditional Runge-Kutta's method to solve ODEs in the these methods and have only  $N^2$  solution parameters (N is the number of genes), while the ODEs systems usually have  $2 * N^2$  parameters to find, which for large networks increases the calculations twofold.

In classic PSO it is hard to find a global minimum in a non-convex problem, to solve this, we propose using a different optimization technique from Estimation Distribution Algorithms (EDA), called Population Based Incremental Learning (PBIL) [14]. Using these techniques, we can build a probabilistic model for the candidate populations in an evolutionary setting, so we can keep their best characteristics and evolve them over time. This way, we look to improve the inference's performance obtained using PSO in previous works.

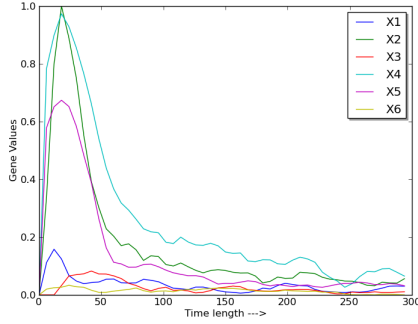


Fig. 1: Example of the microarray data for a 6 gene process, the X axis represents the each gene’s expression.

We also study how different variables of the PBIL, like the standard deviation of the inferred distribution, affects the behavior of the inference. We analyze how increasing the population size decrease the error and increases the reliability of the algorithm.

This paper is organized in the following way, first we present some related work and different approaches to solve the gene network inference problem with different models. Then we show how to represent a GRN using a Recursive Neural Network. We present a description of EDA and PBIL and describe the implementation used to solve the modeling problem. Finally, we present a set of experiments using an artificial small network, in these experiments, we will modify parameters, like the variance of the PBIL’s distribution and the population size, to tune the algorithm, to finally test our tuned algorithm in an artificial large network.

## II. RELATED WORK AND GRN MODELS

Recently, several groups have done inference of gene regulations on small and large networks, in the competition “Dialogue for Reverse Engineering Assessments and Methods” (DREAM) [15], different groups have the task to obtain the best model of a target network. In this competition, the networks’ size is about 100 genes and the judges use a golden standard to score the participant’s results.

We mentioned that there are different approaches to model these networks, some of them, like Bayesian and Boolean Networks model the genes in an intuitive way [5], [6], the links in the graphical model represent links in the genes, and each node represents a gene in the network. Classic Bayesian Networks, however, do not represent dynamic systems, but a static atemporal system, which creates the need of using Dynamic Bayesian Networks. Bayesian structures rely as well on much training data, which sometimes is missing for the inference.

Other approaches use systems of differential equations, like the S-System, to model the dynamics. It has been widely used by Noman [16], Kikuchi [10], Iba [17] and others [18], [19]. These groups have found most of the true regulations in small and middle sized real and synthetic networks, yet, these

models take a long time to find the correct interactions, thus limiting its scalability [16]. Furthermore, these methods often have an  $N^2$  set of parameters for the positive interactions (or activation) and another  $N^2$  set of parameters for the negative interactions (suppression).

The paper [12] by Vohradsky shows that Recursive Neural Networks are a good model for biological systems, he further presented that, compared with other systems like boolean networks, is a better representation for a GRN in a discussion that is out of the scope of this paper. Since then, we have works like [20] or [13], where they have used different training techniques, in the realm of evolutionary computation, to obtain the inference parameters. They have, however, yet to try an implementation on large networks, or using more complex evolutionary optimization techniques.

Combining evolutionary techniques and recursive neural networks for the use of gene network inference is still a very fertile ground, and different optimization algorithms can improve the implementations done so far. In their model, Xu et al [13], proved that using a classic model of PSO was a good solution to infer the structure of small networks, however, their approach was a classic implementation of PSO. The PSO literature [21] [22] shows that a classic PSO scheme is vulnerable to over-fitting the data to some local minimum. Because of this, we extend their approach using a technique robust to over-fitting the data.

Noman and Iba [23] showed that dividing the experimental data in time series helped to prevent local minimums. By dividing the data in different training sets, they increase the richness of the dynamic model, increases the training data, and narrow the solution sets for different gene networks.

We propose a novel implementation, by mixing the division of the training set into different data sets and a better optimization technique, such as PBIL. This approach looks to improve the results obtained using PSO and increase the inference scalability to larger gene networks.

## III. USING RNN TO MODEL GRNs

Unlike tradition Neural Networks (NN), the recurrent neural network (RNN) model is a closed graphical model, where the output layer of the network acts as a feedback for the input layer (Fig. 2).

This architecture is good to model systems that exhibit dynamic behavior by adding a delay variable directly correlated to the output of each neuron. And furthermore, it allows us to connect the output of different elements among them, like the dynamics showed in a true GRN. Like we mentioned before, Vohtadsky [12] used an RNN to model the gene’s regulations in a cell, he further mentioned that this approached presented a promising future to model real networks.

The mathematical model of an RNN is very similar to that of a standard NN, but we have to add the variable of the feedback loop in the system. Thus, the equations that represent a RNN

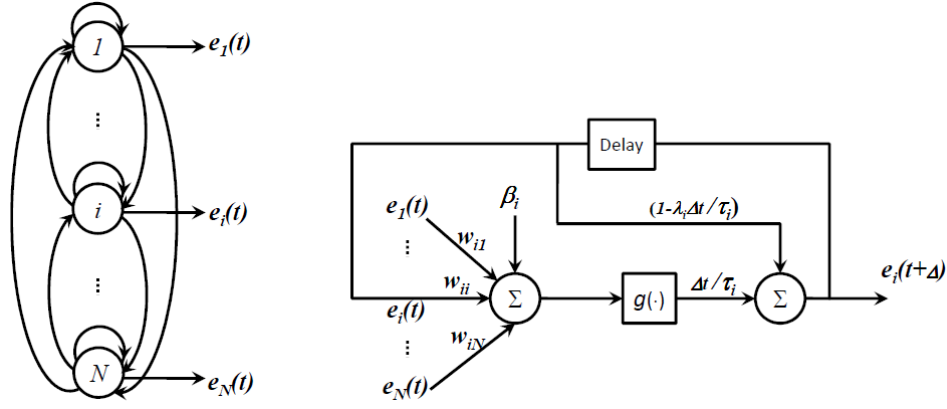


Fig. 2: RNN's structure, with the delay parameters

are:

$$\tau_i \frac{de_i}{dt} = f \left( \sum_{j=1}^N \omega_{ij} e_j + \sum_{k=1}^K v_{ik} u_k + \beta_i \right) - \lambda_i e_i \quad (1)$$

where,  $f(\cdot)$  is a nonlinear function that will act as a classification function, for Neural Networks, we use the sigmoid function  $f(z) = 1/(1 + e^{-z})$ . The values  $w_{ij}$  are the connecting weights of the network, which from a biological point of view represent the connections between gene  $i$  and  $j$ . The variable  $e_j$  represents the expression level for the gene, which is the data from the microarray experiments. The variables  $u$  and  $v$  are defined as external variables, which are often ignored for the true model. And finally, the variable  $\lambda$  is the decay rate parameter and the variable  $\beta$  is the bias parameter of the network.

We use a discretized version of the model to work in the discrete space:

$$e_i(t + \Delta t) = \frac{\Delta t}{\tau_i} \times f \left( \sum_{j=1}^N w_{ij} e_j(t) + \beta_i \right) + \left( 1 - \frac{\Delta t}{\tau_i} \right) e_i(t) \quad (2)$$

In this representation, as we mentioned, we have ignored the external factors and have set the decay rate parameter  $\lambda$  to 1. For a more detailed derivation of eq. 2 from eq. 1, please consult reference [13].

We define forward evaluation as testing the candidate weights  $w_{ij}$  in eq. 2 and obtaining a new time series  $e_i(t)$ . And the optimization problem looks for the optimum values of the weights that will be topologically close to the experimental data after a forward evaluation of the candidate weights  $w_{ij}$ .

#### A. Use of time series segmentation on GRNs

When we do inference on GRNs using RNNs, we often find different sets of parameters  $w_{ij}$  that can fit the experimental data without necessarily being the correct set of parameters. In a mathematical fashion, we call this the problem of the local minimum. Some reasons for this are noisy microarray data, or many gene's chains having many possible solutions.

Because of this, we segment the experimental data into a fixed number of time series  $TS$ . Each time series is defined:

$$x(t) = \sum_{i=1}^{TS} x(ts_i) \quad (3)$$

where  $x(ts_i)$  are disjoint subsets from the experimental data.

Then, we train the system using this segmented data as training sets instead of a single training set from the microarray data. Noman and Iba [23] showed that this approach obtains a better representation and closer values to the true interactions of the GRN, albeit with a different model. This is because multiple training sets force the RNN to recreate the same network dynamics for different initial points of the data rather than one large set of time samples.

#### B. Training of RNN models

The classical approach to train neural networks is forward-backpropagation, where after doing the forward evaluation of the neural network, we evaluate the derivatives and error of the units with respect of the training set, and we modify the weights using a weight correction function, like gradient descent [24].

Xu et al [13] used a very similar system, but to update the weights, they used a classic PSO approach. They initialized a set of random weights, and using PSO they evolved the candidate solutions of the  $w_{ij}$  over time to obtain an approximation to the real values. Traditionally, genetic algorithms are an adequate solution to solve the minimization problem of the weights in a classic Neural Network, some researchers showed that evolutionary techniques work as good as traditional gradient descent and other classic search algorithms. [25], [26]

In this paper, we use a different approach, using PBIL to find the weights of the RNN. PBIL, unlike PSO, has better convergence rates for certain problems, and unlike PSO, where an update has to be done for every candidate solution, in PBIL there is only a general update for an inferred probability distribution. This single updating allows the algorithm to find the solution faster, as well as allowing us to scale up the size of the network.

#### IV. EDA AND PBIL

In Estimation of Distribution Algorithms (EDA), a subcategory of Evolutionary Computation techniques, we fit the best candidates of a problem to a probability distribution and create a new population sampling from this distribution. This way, it extracts the features of the best candidates and transfer them to the next generation, rather than only choosing the best candidates and losing important features from other solutions. The evolutionary nature of EDA consists on iteratively model-sample the trial population to obtain an optimal solution

There are different approaches to EDA, one of them, PBIL, samples each candidate solution from  $D$  distributions, where  $D$  is the dimension of each candidate. We assume that the variables in each candidate are independent from each other; while for some problems this might look as a naive implementation, there has been good results solving complex problems [27], [28].

In classic PBIL, we model  $D$  probability distributions from a  $D$  dimensional space, so each dimension has a probability associated to it (Fig. 3). In turn, we will sample each of these  $D$  distribution to obtain a new set of candidates.

We describe classic PBIL in the following series of steps:

- 1) Generate  $N$  candidate solutions  $X_i \in R^D$  from an uniform distribution  $X_i \sim U(X_{inf}, X_{sup})$ , where  $X_{inf}, X_{sup}$  are the boundaries of the search space and  $D$  is the dimensionality of the data
- 2) Evaluate a fitness function  $f$  over the  $N$  candidates.
- 3) Apply elitism, by choosing the best  $K$  candidates.
- 4) Using the best  $K < N$  candidates, infer  $D$  probability distributions for each dimension of elite candidates.
- 5) Generate  $N$  new candidates from the inferred distributions.
- 6) Go to step 2.

In this algorithm, both the elite candidates  $K$  and the probability distribution are the most important variables. If the search space is multimodal and we choose a unimodal probability distribution, e.g. a Gaussian, we will get a bad fit and a bad solution. We have to select the variable  $K$  so it generates a good probability distribution, using a large  $K$  incurs excessive computational cost, while a small  $K$  creates an non-representative probability distribution.

##### A. Algorithm

For this work, we implemented a simple PBIL, where we modeled each candidate parameter—which are the weights of the RNN—as a normal distribution  $N(\text{mean}, \text{variance})$ . We kept the variance static and calculated the mean with the maximum likelihood equation for the Gaussian:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i \quad (4)$$

We will show, that the variance of this normal distribution is important, since it will be controlling the range of the probability distribution, if the variance is large, we would be accepting many possible solutions, avoiding local minimum;

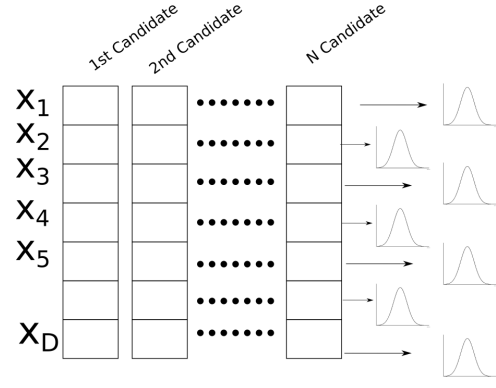


Fig. 3: PBIL's Basic scheme

but we would be prone to delay the convergence, and have difficulty doing local search. On the other side, if the variance is small, the richness of the candidates will be dampened, and thus, our inference algorithm will have difficulty to get out of local minimum.

We show in the experiments how changing the size of the variance affects the final results of the inference in a small network.

To score our candidate solutions, we used the following fitness function:

$$F(Cand_i) = \frac{1}{TN} \sum_{ts=0}^{TS} \sum_{t=0}^T \sum_{i=0}^N (e_i(t) - Cand_i(t))^2 \quad (5)$$

where  $TS$  are the time series we divided the experimental data in,  $T$  is the time samples we used for each time series and  $N$  is the number of genes. The values  $e_i(t)$  and  $Cand_i(t)$  are the values for the experimental data and the RNN evaluation of the candidate  $w_{ij}$ . Next, we describe the overall algorithm including the time series division and the experimental data:

- 1) Take the experimental data and divide it in  $TS$  time series.
- 2) Generate  $N$  candidate solutions  $X_i \in R^D$  from an uniform distribution  $X_i \sim U(X_{inf}, X_{sup})$ , where  $X_{inf}, X_{sup}$  are the boundaries of the search space and  $D$  is the dimensionality of the data. Since we have  $GeneNumber^2$  connections, the dimensionality is equal to  $GeneNumber^2$ .
- 3) Using the candidates, do the forward evaluation of the RNN given in eq. 2 and obtain a set of time series  $TS$  per each candidate.
- 4) Using the cost function (eq. 5), rank the candidates and choose the best  $K$  ones.
- 5) Using the  $K$  candidates, generate  $D$  Gaussian distributions with mean equal to (eq. 4) and variance equal to a fixed value.
- 6) Generate  $N$  new candidates from the inferred Gaussian distributions.
- 7) Go to step 3.

TABLE I: 4 Gene Network's parameters

$w_{i,j}$				$\beta_i$	$\tau_i$
20	-20	0	0	0	10
15	-10	0	0	-5	5
0	-8	12	0	0	5
0	0	8	-12	0	5

## V. RESULTS

### A. Experiments

To tune the variables population size, time series and variance, we observed the effect of changing the variables among different experiments. We changed the variance between 1 and 10 and reported the results in the next section, for each value of the variance we changed time series from 5 to 15, we expect that, as we increase the time series, the standard deviation of the results will decrease, albeit higher computational time, thus, we can do an analysis on the trade-off between accuracy and the growth of the calculations.

We also changed the population size from 100 to 500. Since the accuracy in a distribution's estimation is proportional to the available data, we expect that as we increase the initial candidates in the population, the algorithm's precision will increase. This increase, however, should also present a higher cost in the computations. Thus we have another trade-off that relates our accuracy to the computational overhead we have.

Finally, we use the best values for variance, population size and time series and test them in an artificial 64 gene network. We analyze it using statistical validations such as Precision, Recall and the F-Score, which are standard tests in classification algorithms.

### B. PBIL's parameters Analysis

1) *Analysis for a fixed variance=1*: We fixed the variance of the PBIL's Gaussian distributions to one. With this value, we did 50 simulations and 4000 iterations for 5,10 and 15 time series and 100, 200, 300, 400 candidates for each of them. We had the objective of obtaining the best possible parameters for this architecture. An analysis that previous works had overlooked using only one fixed set of parameters.

For these experiments, we used an artificial 4-Gene network, defined by Table I, this network was used by [13] and [29] to test this algorithm. We also present a box chart (Fig. 4) of the original network, where the X axis are the possible connection states, which are 16, and the Y axis is the value of that connection.

For this small network, we obtained the correct regulations at least 80% of the times for the 50 tests. To visualize this, we present Fig. 5 where we show the box charts that represents the average 16 inferred values of  $w_{i,j}$ . Here we compare the box plots when we use 5 time series and 15 time series. In Fig. 5a we see clearly how for certain values of the weights, the standard deviation is high, which means that for some runs of the algorithm, we obtained false negatives. We see, however, in Fig. 5b that an increase in the time series  $TS$  translates as a decrease in the standard deviations, and that for most of

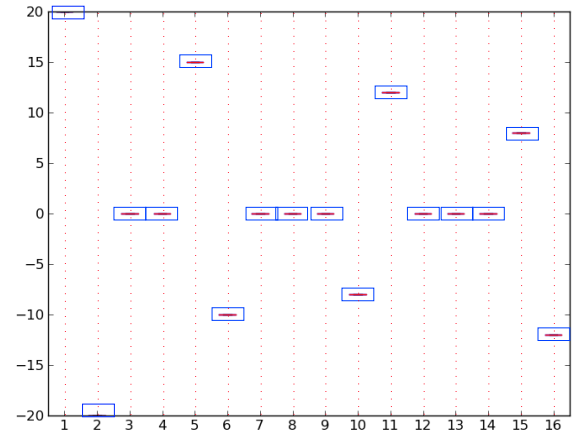


Fig. 4: Box chart of the original network, with 16 possible connections (X-axis) and connection values (Y-axis)

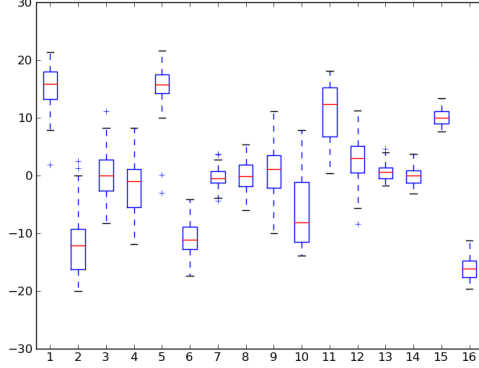
TABLE II: Standard Deviation for a fixed variance of 1 in the PBIL, TS stands for Time series and Pop for population

TS\Pop	100	200	300	400	500
5	4.13	3.64	3.76	3.54	3.52
10	3.06	2.89	2.76	2.74	2.62
15	3.17	2.46	2.45	2.55	2.19

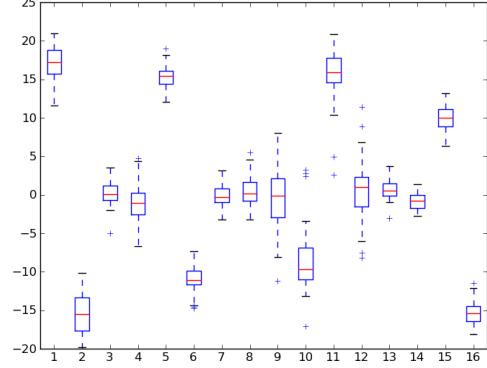
the simulations we found the true values. If we compare both this results with the original plot (Fig. 4) the values of the inference are close to the true values.

We present, in Fig. 6 and Table II, how, for different population sizes, the average standard deviation of the inferred values decreases over time. As we increase the time series and the population size, we obtain smaller standard deviations. Increasing the size of the population and the time series, however, affects badly the inference time. We found that for 300 candidates and 8 time series we obtained good results and the calculation time took around 5 minutes for this small network.

2) *Analysis for fixed variance of 10*: In the next set of experiments, we used the same parameters, and switched the variance to ten, we also did 50 tests with 4000 iterations each. As with the previous case, we present the results obtained for different time series and populations. First, we show the box plot of this evaluation in Fig. 7, we do a qualitative analysis by comparing it with Fig. 5 and fig 4, and the plot shows that increasing the variance has a negative effect on the sparsity of the inferred connections. While for the 16 connections, when using variance of one, we had at least 80% of the times the correct solutions, the Fig. 7b shows how the 10th interaction was a false negative most of the times. The nominal values of the interactions increased, which are irrelevant for this model, since we care about positive and negative interactions, yet, using a variance of one, we found values in the same order as those in Table I. We can conclude from these two tables, that using a large variance has a negative effect in the overall



(a) Inference with 5 Time Series



(b) Inference with 15 Time Series

Fig. 5: Box diagrams for the 16 resulting values of  $w_{ij}$  for 50 experiments and variance of 1, X axis is the interaction and Y is the inferred value.

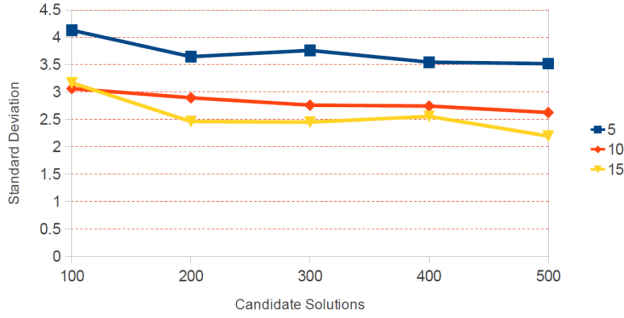


Fig. 6: Value of the result's standard deviation for a small network using different candidate solutions and time series with  $\text{Var}=1$ .

TABLE III: Standard Deviation for a fixed variance of 10 in the PBIL, TS stands for Time series and Pop for population

TS \ Pop	100	200	300	400	500
5	52.97	48.27	49.52	45.30	43.06
10	44.36	38.86	38.35	38.08	37.33
15	38.88	33.07	29.87	33.37	31.76

inference, and that for this specific problem is better to use small variances.

In Fig. 8 we show again how increasing the population's size has a positive effect by reducing the standard deviation of the results. We see it also present in the values in Table III, where we see how the standard deviation decreases inversely proportional with the population size. The decrease was in the same order of that showed in Fig. 6, which means that regardless of the variance in the PBIL, increasing the size of the population always has a positive effect.

This results seems intuitive to the fact that PBIL is based on doing a probabilistic inference over a set of candidates. The larger this set is, the better the inference is, which means the algorithm has more capability to model the best possible solution. And a large variance expands the search space so much that the problem miss the global minimum.

### C. Analysis of large networks

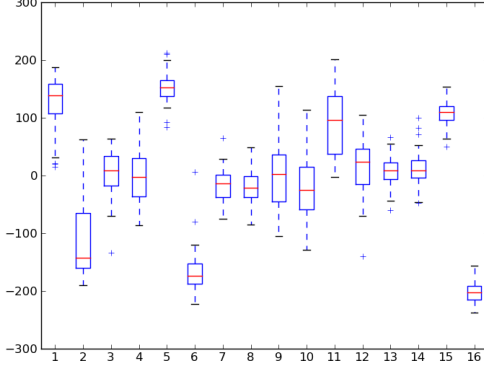
In this part, we present the analysis of a 64 artificial gene network. This network is an extension of the one we used in the previous section, and is the copy of the used matrix

16 times along both axes (Fig. 9) We choose this expansion because it replicates the dynamics of the 4 gene networks in a higher dimension, and as such, we know which are the correct interactions and can evaluate the algorithm counting the true positives and true negatives. Furthermore, true large networks often have these kind of self replicating structures, where several genes behave similarly.

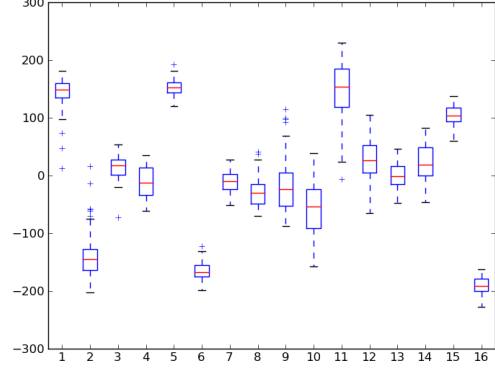
To test our algorithm in this large network, we used the optimum parameters obtained before of 300 candidates, with a fixed variance of 1 and 8 time series. We saw that increasing the number of time series increases the reliability of our results, however it also increases the calculation time, which in a large network like this is an important factor to be considered.

The algorithm took about 30 minutes to solve the 64 gene network, which speaks highly against other approaches using S-System, where it took the same time to obtain the parameters of a 6 gene network. To evaluate the correctness of the inference, we measure the true positives and true negatives and calculated the average Specificity, Precision and Recall





(a) Inference with 5 Time Series



(b) Inference with 15 Time Series

Fig. 7: Box diagrams for the 16 resulting values of  $w_{ij}$  for 50 experiments and variance of 10, X axis is the interaction and Y is the inferred value.

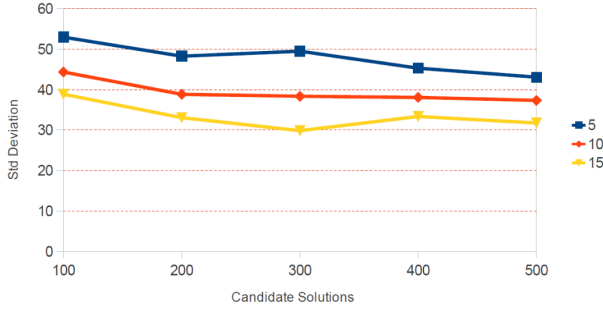


Fig. 8: Value of the result's standard deviation for a small network using different candidate solutions and time series with Var=10.

for 50 runs. The equations for these parameters are:

$$\begin{aligned}
 Recall &= \frac{TP}{TP + FN} \\
 Precision &= \frac{TP}{TP + FP} \\
 S_p &= \frac{TN}{TN + FP} \\
 F - Score &= 2 \frac{Precision * Recall}{Precision + Recall}
 \end{aligned}$$

where TP, FP, TN and FN stand for True and False positive and negative.

In Table IV, we present the values we got for the 64 gene network.

TABLE IV: Results for the 64 gene network inferred

Metric	Value	# of Positives	2879
Specificity	0.280	True Positive	1459
F-score	0.602	True Negative	628
Precision	0.507	False Positive	1420
Recall	0.740	False Negatives	589

As Table IV shows, the inference method has a good recall, it being larger than 70% fares well against other results reported before [9] for smaller networks by other algorithms.

The algorithm, however, reaches a low specificity, this is because even with a high number of time series, we still need to increase either the population or the time series to obtain good results due to the heavy complexity of a 64 gene network. While 8 time series and 300 candidates were a good solution for 16 parameters, for  $64^2$ , we obtained little richness with these values. We see that it is good to have a population size at least 10 times larger than the number of connections.

## VI. CONCLUSION

We have presented an alternative solution to the modeling problem of Gene Regulation Networks. We modeled the chains using a Recurrent Neural Network, and obtained its optimal parameters using a Population Based Incremental Learning, an evolutionary algorithm from the Estimation of Distribution Algorithms family.

We tested the algorithm for a small 4 gene network, and obtained good results for 50 simulations that tested different population sizes and variances.

We then used the parameters to test the same algorithm in a larger network. We obtained good results that show that further work is needed when tuning the parameters. The results, however, showed that the approach is adequate for large networks, and it is capable of finding results faster than ODEs approaches.

The algorithm, however still can be further improved, among those improvements, we could use a different probability distribution to model the candidate solutions. Since a Gaussian distribution is unimodal, it is easier to use multi-modal distributions, like a mixture of Gaussians. We also have yet to test the algorithm using a real small network and a real large network.

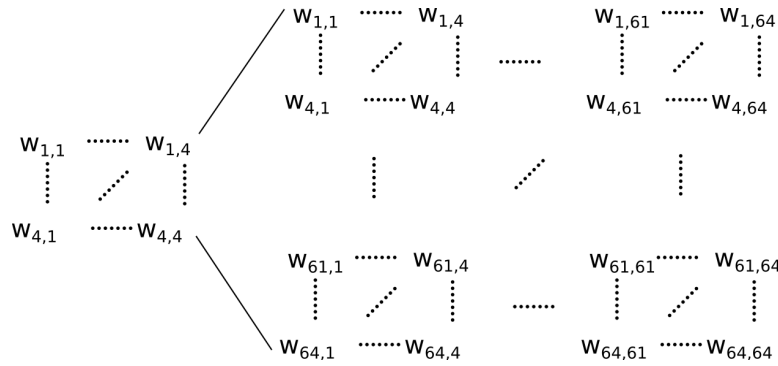


Fig. 9: Expansion form a 4-Gene Matrix to a 64 Gene Matrix

## REFERENCES

- [1] P. A. Futreal, L. Coin, M. Marshall, T. Down, T. Hubbard, R. Wooster, N. Rahman, and M. R. Stratton, "A census of human cancer genes." *Nature reviews. Cancer*, vol. 4, no. 3, pp. 177–83, Mar. 2004.
- [2] F. Busfield, D. L. Duffy, J. B. Kesting, S. M. Walker, P. K. Lovelock, D. Good, H. Tate, D. Watego, M. Marczak, N. Hayman, and J. T. E. Shaw, "A genomewide search for type 2 diabetes-susceptibility genes in indigenous Australians." *American journal of human genetics*, vol. 70, no. 2, pp. 349–57, Feb. 2002.
- [3] J. Quackenbush, "Microarray data normalization and transformation," *Nature Genetics*, vol. 32, no. suppl, pp. 496–501, 2002.
- [4] I. B. Jeffery, D. G. Higgins, and A. C. Culhane, "Comparison and evaluation of methods for generating differentially expressed gene lists from microarray data." *BMC bioinformatics*, vol. 7, p. 359, Jan. 2006.
- [5] B.-E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. D'Alche-Buc, "Gene networks inference using dynamic Bayesian networks," *Bioinformatics*, vol. 19, no. Suppl 2, pp. ii138–ii148, Oct. 2003.
- [6] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using Bayesian networks to analyze expression data." *Journal of computational biology a journal of computational molecular cell biology*, vol. 7, no. 3-4, pp. 601–620, 2000.
- [7] M. I. Davidich and S. Bornholdt, "Boolean network model predicts cell cycle sequence of fission yeast," *PLoS One*, vol. 3, no. 2, p. e1672, 2008.
- [8] M. A. Savageau, *Biochemical systems analysis: a study of function and design in molecular biology*. Addison-Wesley Pub. Co., Advanced Book Program, 1976.
- [9] N. Noman and H. Iba, "On the Reconstruction of Gene Regulatory Networks from Noisy Expression Profiles," *2006 IEEE International Conference on Evolutionary Computation*, no. 1, pp. 2543–2550, 2006.
- [10] S. Kikuchi, D. Tominaga, M. Arita, K. Takahashi, and M. Tomita, "Dynamic modeling of genetic networks using genetic algorithm and S-system," *Bioinformatics*, vol. 19, no. 5, p. 643, 2003.
- [11] D.-Y. Cho, K.-H. Cho, and B.-T. Zhang, "Identification of biochemical networks by S-tree based genetic programming," *Bioinformatics*, vol. 22, no. 13, pp. 1631–1640, 2006.
- [12] J. Vohradský, "Neural network model of gene expression." *The FASEB journal : official publication of the Federation of American Societies for Experimental Biology*, vol. 15, no. 3, pp. 846–54, Mar. 2001.
- [13] R. Xu, I. I. Donald Wunsch, and R. Frank, "Inference of genetic regulatory networks with recurrent neural network models using particle swarm optimization," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 681–692, 2007.
- [14] S. Baluja, "Population-based incremental learning," *Carnegie Mellon Tech Reports*, 1994.
- [15] G. Stolovitzky, D. Monroe, and A. Califano, "Dialogue on reverse-engineering assessment and methods: the DREAM of high-throughput pathway inference." *Annals of the New York Academy of Sciences*, vol. 1115, no. 914, pp. 1–22, Dec. 2007.
- [16] N. Noman and H. Iba, "Inference of gene regulatory networks using s-system and differential evolution," in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, Washington, DC. Citeseer, 2005, p. 439.
- [17] —, "Reverse engineering genetic networks using evolutionary computation." *Genome informatics. International Conference on Genome Informatics*, vol. 16, no. 2, pp. 205–14, Jan. 2005.
- [18] P. K. Liu and F. S. Wang, "Inference of biochemical network models in S-system using multiobjective optimization approach," *Bioinformatics*, vol. 24, no. 8, p. 1085, 2008.
- [19] O. R. Gonzalez, C. Küper, K. Jung, P. C. Naval, and E. Mendoza, "Parameter estimation using Simulated Annealing for S-system models of biochemical networks." *Bioinformatics (Oxford, England)*, vol. 23, no. 4, pp. 480–6, Feb. 2007.
- [20] N. Kasabov, "A method for modelling genetic regulatory networks by using evolving connectionist systems and microarray gene expression data," *ICONIP&#39;02. Proceedings of the 9th*, vol. 2, pp. 596–601, 2002.
- [21] R. Eberhart, "Recent advances in particle swarm," *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, pp. 90–97, 2004.
- [22] X. F. Xie, W. J. Zhang, and Z. L. Yang, "Dissipative particle swarm optimization," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 2. IEEE, 2002, pp. 1456–1461.
- [23] N. Noman and H. Iba, "Inferring gene regulatory networks using differential evolution with local search heuristics." *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, vol. 4, no. 4, pp. 634–47, 2007.
- [24] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- [25] R. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [26] C.-F. Juang, "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design." *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 34, no. 2, pp. 997–1006, Apr. 2004.
- [27] S. Shakya, J. McCall, and D. Brown, "Using a Markov network model in a univariate EDA: an empirical cost-benefit analysis," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, pp. 727–734.
- [28] D. Simon, "Biogeography-based optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 6, pp. 702–713, 2008.
- [29] M. Wahde and J. Hertz, "Coarse-grained reverse engineering of genetic regulatory networks." *Bio Systems*, vol. 55, no. 1-3, pp. 129–36, Feb. 2000.