A Novel Reverse Engineering Approach for Gene Regulatory Networks

Francesco Zito, Vincenzo Cutello, and Mario Pavone(✉)

Department of Mathematics and Computer Science, University of Catania, v.le Andrea Doria 6, 95125 Catania, Italy
francesco.zito@phd.unict.it, cutello@unict.it, mpavone@dmi.unict.it

Abstract. Capturing the rules that govern a particular system can be useful in any field where the causes of its effects are unknown. Indeed, discovering the causes that produced a particular effect is extremely useful in fields such as biology. In this paper, a reverse engineering method based on machine learning is proposed. This method was used to replicate real world behaviour and use this knowledge to generate the relative Gene Regulatory Network. The datasets from the DREAM4 Challenge were used to validate this method.

Keywords: Reverse engineering · Complex network · Machine learning · Gene regulatory network · Multivariate time-series forecasting

1 Introduction

Most real-world systems can be represented by a graph with a nontrivial topology, called a complex network, where the nodes represent real-world entities (or variables) and the arcs represent the interactions between the entities themselves [15]. The goal is to better understand the functioning of the real system by studying the interactions between the entities involved and the properties of the network [12]. To build the complex network associated with a real-world system we need to carefully reverse engineering it. In other words, we have to build the complex network representing the real system by measuring only the outputs of the system over a period of time [22].

Unlike forward problems, where the effects can be accurately predicted by the system model given one or more causes as inputs, inverse problems are the opposite [20]. Here, the mathematical formulation of the problem is not known and therefore it is not possible to assert anything about it [5]. The only known information come from the observations of the system. Explaining the actual causes of a particular phenomenon, however, can be an extremely difficult, if not impossible, because small differences in effects can lead to large differences in causes, or the same effect can result from more than one cause [21]. In reality, estimating the parameters of the model is a difficult and time-consuming process due also to the large dimensional space, although some efficient algorithms were proposed [6,7]. Several approaches can be found in literature to solve such inverse problems. For example, in [9] different mathematical approaches to solve this problem are explained. However, as stated in [4], mathematical approaches may not be sufficient to find a solution in reasonable time with high accuracy. Therefore, in recent years, many studies have been focused on the application of deep learning techniques to inverse problems, e.g., in image processing [14], astronomy [8], physics [16], biology [1], civil engineering [3], and so on.

In this paper, we present a novel approach based on machine learning techniques that has two goals: create an artificial environment capable of replicating the behaviour of a real environment based solely on observations of the variables of interest, and generate a complex network that reveals the relationships among variables in the system.

One of the most interesting areas of computer science where the application of reverse engineering is on the rise, is in the field of genetics [17]. Micro-array technology allows researchers to examine the presence of multiple genes in a DNA sample and their levels of expression [11]. Using our method, we were able to firstly artificially reproduce this behaviour and secondly create a genetic regulatory network showing the iterations between genes.

In Sect. 2 we will discuss modelling and creating an artificial environment that can replicate the real environment. In Sect. 3, we will discuss a methodology that uses the artificial environment to facilitate the creation of a gene regulatory network. In Sect. 4, we present the results obtained.

2 Modeling

To solve the inverse problem the goal we need to find the best model m such that

$$d = G(m) \tag{1}$$

where $G(\cdot)$ is an operator describing the explicit relationship between the observed data d and the model parameters [2]. In this section, we present a novel approach based on machine learning techniques to determine the best model m that can replicate the behaviour of a real environment when only the observations are available.

2.1 Environment

An environment basically consists of several elements (such as entities or variables) that interact with each other according to well-defined rules established during the design process. The two main components of an environment are: the agents and the rules that govern their interactions. Each agent is responsible for predicting the future value of a variable based on the current state of the environment. The interaction rules between the agents, on the other hand, define the mechanism by which the state of the environment evolves. Figure 1 depicts the environment scheme under consideration.

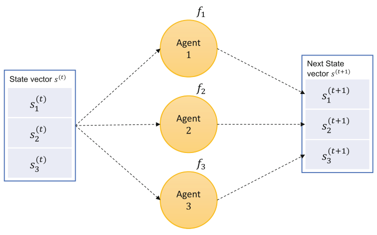


Fig. 1. Environment architecture

Given an environment with k variables, the state of the environment at time t, denoted by $s^{(t)}$, is defined as a k-vector in which the generic element $s_i^{(t)}$ is the value of the i-th variable at time t. The subsequent state of the environment $s^{(t+1)}$ is computed as follows:

$$s^{(t+1)} = f\left(s^{(t)}\right) \tag{2}$$

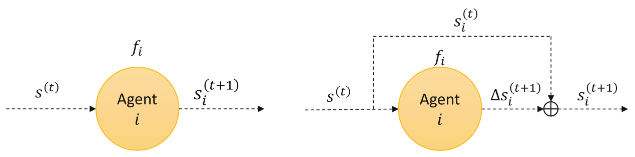
It can be seen from figure 1 that each agent in the environment is responsible for predicting the value of the variable to which it refers. Accordingly, there are as many agents as there are variables in the environment, so the equation 2 can also be written as follows.

$$s^{(t+1)} = f\left(s^{(t)}\right) = \left[f_1\left(s^{(t)}\right), f_2\left(s^{(t)}\right), \ldots, f_k\left(s^{(t)}\right)\right] \tag{3}$$

where f_i denotes the agent function and represents the i-th component of the function f.

2.2 Agent

An agent can be formally described as a function $f_i : S^k \rightarrow S$ that predicts the i-th component of the state of the environment at time $t+1$, given its previous state. We use two different types of prediction scheme: simple agent prediction and delta agent prediction.



(a) Simple Agent prediction (b) Delta Agent prediction

Fig. 2. Agent prediction scheme

Simple Agent Prediction In the first prediction scheme (Fig. 2a), an agent directly estimates the future value of the variable according to Eq. 4.

$$s_i^{(t+1)} = f_i\left(s_1^{(t)}, s_2^{(t)}, \ldots, s_k^{(t)}\right) \tag{4}$$

Agent Delta Prediction On the other hand, Fig. 2b illustrates the second agent prediction scheme. In this case, an agent predicts by how much the current value should be increased or decreased. In other words, it predicts the offset between the value of the i-th component of the state at time $t+1$ and its value at time t. The output of the agent is calculated as follows.

$$s_i^{(t+1)} = s_i^{(t)} + \Delta s_i^{(t+1)} = s_i^{(t)} + f_i\left(s_1^{(t)}, s_2^{(t)}, \ldots, s_k^{(t)}\right) \tag{5}$$

Each agent in the environment can be considered as a black-box function with its own architecture and configuration. The basic model for each agent is shown in Fig. 3.

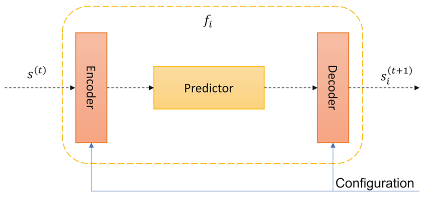


Fig. 3. The basic model of an agent.

The Predictor is the core of the agent and it is responsible for forecasting the value of a variable given a specific value as input. A predictor can be anything from a neural network to a regression model to a decision tree. Different types of predictors were tested to explore different possible configurations of the artificial environment. In Table 1 we report the available configurations for each predictor.

Table 1. Agent configurations for each type of predictor. 1) Predictor specifies the type of predictor used to predict the variable; 2) Machine learning tasks defines whether the predictor is a classifier and returns a class value or whether it is a regressor and predicts a real value; 3) Agent prediction scheme defines how the agent's output is composed; 4) Training parameters are used to train the predictor.

Predictor name	**Machine learning task**	**Agent prediction scheme**	**Training parameters**
Fully Connected Neural Network (FCNN)	Classification (L)/regression	Simple/delta prediction	Epochs, mini batch size, learning rate
Reccurent Neural Network (RNN-LSTM)	Classification (L)/regression	Simple/delta prediction	Epochs, mini batch size, learning rate
Convolutional Neural Network (CNN)	Regression	Simple/delta prediction	Epochs, mini batch size, learning rate
Simple Linear Regression (SLR)	Regression	Simple/delta prediction	

Since an environment may contain different types of agents, it is important to use additional layers to ensure agent interoperability. Therefore two additional blocks have to be added before and after the predictor: encoder and decoder. The encoder's role is to convert the agent's input into a format that the predictor can understand. The decoder's role, on the other hand, is to convert the predictor's output into a format that is compatible with the architecture of the environment ($s_i^{(t+1)}$).

3 Methodology

In a gene regulatory network, each gene can be activated or inhibited depending upon the expression level of another gene, called regulator or regulatory gene. To find the regulatory genes of each gene, we used a simple principle: if varying the expression level of gene G_i causes a significant change in the expression level of gene G_j, then G_i might be a good candidate as a regulator of gene G_j [10]. However, this approach can only be used if the mathematical formulation that determines the relationships between gene expression levels is known. Several methods for constructing a gene inference model have been proposed in the literature. For instance, continuous models such as the ordinary differential equations ODE, which are based on estimates of the inference level over time. Although the ODE approach provides detailed information about the dynamics of gene expression, it requires high quality data to build an accurate model [13]. In this section, we first look at how the model described above can be used to predict the expression level of genes, and then take advantage of the artificial environment to generate GRNs.

3.1 Gene Expression Level Prediction

We create an artificial environment by using: the observations (X) represented by a $k \times n$ matrix, where k is the number of variables and n is the number of observations over time; and the environment's configuration that consists of a collection of k tuples, each associated with an agent and containing the elements listed in Table 1, such as, the predictor, ML Task, prediction scheme, and training parameters. Algorithm 1 shows the procedure for creating an artificial environment and the evaluation of the environment thus created.

Since the model of the i-th agent depends on its associated configuration, each agent must be trained on its own training set consisting of pairs (X, Y_i), where Y_i is a n vector and contains the values of the i-th variable shifted forward by one time unit. The artificial environment E is composed by all the agents created according to the architecture described in Sect. 2.1.

Ideally, an optimal artificial environment without any perturbation and with identical initial conditions should be able to produce the same response as a real environment. Figure 4 depicts the artificial environment's response over a time interval equal to twice the observation time. As we can see, the response of the artificial environment before 1000 time units is similar to the observation in the real environment. On the other hand, after 1000 time units, the artificial

Algorithm 1: Environment-creation algorithm

```
1 $A \leftarrow \emptyset$ ;                                   /* collection of agents */
2 for $i = 1, 2, \ldots, k$ do
3     $Y_i$ = createTargetData ($X$,Config$_i$) ;
4     $A_i$ = createAgent ($X$,$Y_i$,Config$_i$) ;
5     $A = A \cup A_i$
6 end
7 $E$ = makeEnvironment ($A$) ;
8 $\hat{X}$ = simulation($E$,0,$n$) ;    /* the simulation starts at time 0 and ends
  after $n$ time steps, the resulting matrix is a $k \times n$ matrix and
  contains all predicted values for each variable for each time point.
  */
9 $Q_{\%}$ = evaluation($X$, $\hat{X}$) ;                    /* using equation 6 */
```

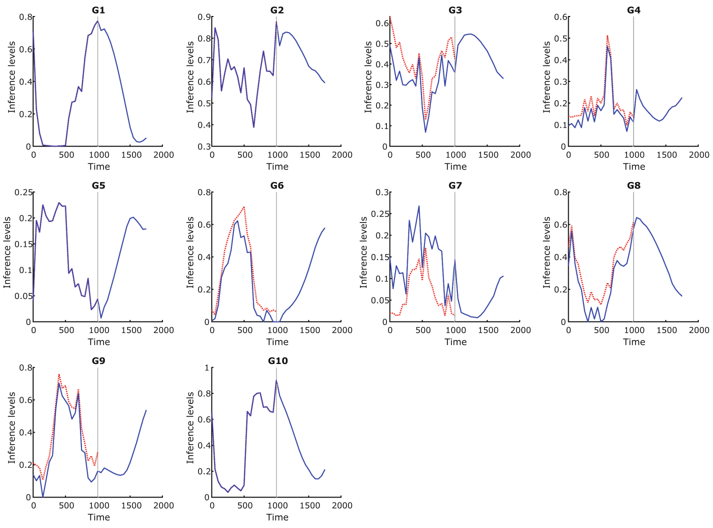


Fig. 4. Prediction of gene expression levels of ten genes over time. The red line represents the actual observations, while the blue line shows the predicted values over time.

environment uses its knowledge to forecast the inference level of the genes for the subsequent time points.

To validate the effectiveness of the prediction, the cosine similarity measure is used. Given two matrices, $X \in \mathbb{R}^{k\times n}$ and $\hat{X} \in \mathbb{R}^{k\times n}$, representing, respectively, the actual and predicted data and the generic element x_{ij}, corresponding to the value of the i-th variable at the j-th time, the similarity index $Q_{\%}$ denotes how close the prediction is to the target values in percentage terms and it is defined as follows:

$$Q_{\%} = \frac{1}{k}\left(\sum_{i=1}^{k}\left|\frac{\sum_{j=1}^{n} x_{ij}\hat{x}_{ij}}{\sqrt{\sum_{j=1}^{n} x_{ij}^2}\sqrt{\sum_{j=1}^{n} \hat{x}_{ij}^2}}\right|\right) * 100, \tag{6}$$

To choose the appropriate configuration of the agents so to maximize the similarity index, we used a local search algorithm [19,24] that continuously improves a possible solution until the best configuration for the environment is found.

3.2 Gene Regulatory Network

Algorithm 2 shows the pseudo-code of the procedure used to determine the regulatory matrix by the artificial environment. Suppose we want to determine whether the gene G_i is a regulatory gene of G_j. Using the artificial environment, the expression value of G_i can be manually set to V_i at time t (lines 5–6), and it will be possible to observe how the gene G_j responds to this change after time $t+1$ (lines 7–8). Clearly, we cannot know whether or not the measurements obtained in this way are correct. For that, we would need a real dataset and would have to validate the measurements in the field using micro-array technology.

We denote by "initial time" $t-1$ the time when the environment does not react to perturbations and evolves naturally, by "transition time" t we denote the time when the perturbation acts on the gene G_i and changes its expression level manually, and finally by "final time" $t+1$ we denote the time after the perturbation when the artificial environment reacts to it and the expression level of the genes is calculated taking this change into account. According to Eqs. 3, 7, 8, and 9 represent the state of the environment calculated at the initial time, at the transition time, and at the final time, respectively.

Algorithm 2: Building a GRN using the artificial environment

```
1  $E_s$ = findStability ($E$) ;  /* Simulates the system until the variables
   no longer greatly fluctuate and are within a certain range. At the
   end of this procedure, the current time point of the environment $E_s$
   will be equal to the initial time. */
2  for $i = 1, 2, \ldots, k$ do
3      $\hat{E}$ = clone ($E_s$) ;
4      $s^{(t-1)}$ = getCurrentState ($\hat{E}$) ;
5      $\hat{E}$ = nextStep ($\hat{E}$,$i$,$V_i$) ;
6      $s^{(t)}$ = getCurrentState ($\hat{E}$) ;
7      $\hat{E}$ = nextStep ($\hat{E}$) ;
8      $s^{(t+1)}$ = getCurrentState ($\hat{E}$) ;
9      for $j = 1, 2, \ldots, k$ do
10         calculate the regulatory value $r_{i,j}$ according to the equation 10
11     end
12 end
```

The initial time defined above is determined by the procedure in line 1 of the Algorithm 2. Basically, we used a linear regression model to determine when all variables in our artificial environment have reached stability. In other words, they have no fluctuations and move in a finite interval. When all variables exhibit this behaviour, the system is considered stable and the current time point corresponds to our initial time.

Although the environment state is calculated as usual using Eqs. 2, 8 used to determine the state of the environment at the transition time is slightly different. Indeed, in this case, the i-th component of the state must be set to a constant V_i.

$$s^{(t-1)} = \left[f_1\left(s^{(t-2)}\right), f_2\left(s^{(t-2)}\right), \ldots, f_i\left(s^{(t-2)}\right), \ldots, f_k\left(s^{(t-2)}\right)\right], \tag{7}$$

$$s^{(t)} = \left[f_1\left(s^{(t-1)}\right), f_2\left(s^{(t-1)}\right), \ldots, V_i, \ldots, f_k\left(s^{(t-1)}\right)\right], \tag{8}$$

$$s^{(t+1)} = \left[f_1\left(s^{(t)}\right), f_2\left(s^{(t)}\right), \ldots, f_i\left(s^{(t)}\right), \ldots, f_k\left(s^{(t)}\right)\right]. \tag{9}$$

By comparing the expression levels at the initial and at the final time, it is possible to determine which genes have been affected by the perturbation (Fig. 5).

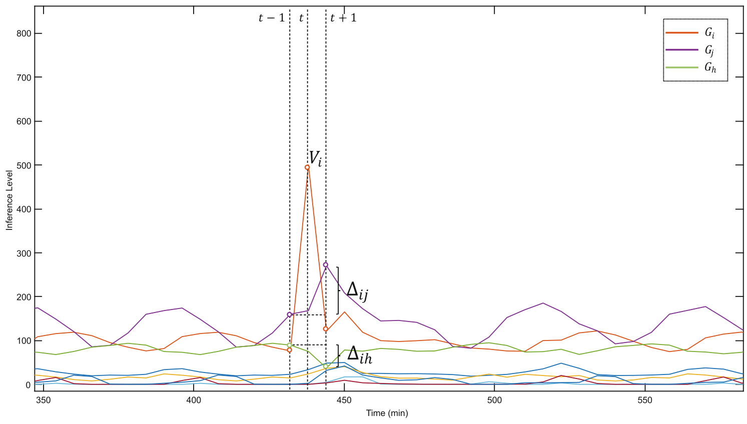


Fig. 5. Example of calculating the regulatory value of two genes G_j and G_h using G_i as their regulatory gene. $\Delta_{i,j}$ and $\Delta_{i,j}$ represent the variation between the expression level of genes at the final time and at the initial time. V_i is the value used to determine which genes are regulated by the i-th gene.

We define the *regulatory value* $r_{i,j}$ as the offset between the expression level of the gene G_j at the final time and at the initial time, taking into account a perturbation of the gene G_i at the transition time (line 10). The regulatory value

	G1	G2	G3
G1	0.14	0	0.01
G2	0.3	0	0
G3	0	0.18	0

Regulatory Matrix

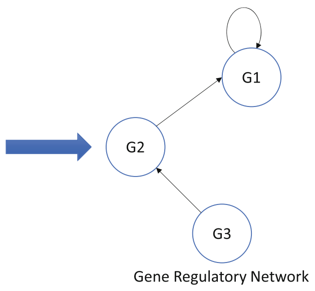


Fig. 6. Method for creating a gene regulatory network starting from the corresponding regulatory matrix with a threshold value of 0.1.

thus obtained is normalized using the maximum value M_j observed for the i-th gene during all the observation time.

$$r_{i,j} = \left| \frac{\Delta_{i,j}}{M_j} \right| = \left| \frac{s_j^{(t+1)} - s_j^{(t-1)}}{M_j} \right|. \tag{10}$$

The comparison between the expression levels of the G_j at the final and at the initial time could provide us with important information about which genes are correlated with each other. Accordingly, G_i is a gene regulator of G_j if and only if the regulatory value $r_{i,j}$ is much larger quantity of a threshold value (> 0). A *regulatory matrix* $R \in \mathbb{R}^{k \times k}$ contains all regulatory values discovered for each gene pair. A gene regulatory network can be extracted using a regulatory matrix. Given a threshold value, it is possible to determine which genes are essentially the regulators of each gene. Figure 6 shows, as an example, how a regulatory matrix with only three genes is transformed into a gene regulatory network using as threshold value of 0.1.

4 Results

In this section, we discuss the results obtained considering ten datasets from DREAM4 Challenge, available in [23]. Each dataset consists of five or ten experiments, each with 21 observations of ten or one hundred genes recorded every fifty minutes. Since each dataset contains multiple experiments, the first half of the experiments were considered as a training set, while the second half served as a validation and testing set.

The obtained results are shown in Table 2. To examine the effectiveness of our methodology, two different groups of metrics were used. The similarity index was used to measure the reliability of the artificial environment. Performance metrics (such as accuracy, precision, sensitivity and specificity) were used to compare the predicted gene regulatory network with the target network. In the field of gene regulation, the typical elements of the confusion matrix also have a biological significance, as reported in [23].

Table 2. The results table shows the metrics used to validate our methodology. $Q_{\%}$ is the similarity index, Thr. stands for threshold, Acc. for accuracy, Prec. for precision, Sens. for sensitivity, and Spec. for specificity.

Dataset name	$Q_{\%}$	**Thr.**	**TP**	**TN**	**FP**	**FN**	**Acc.**	**Prec.**	**Sens.**	**Spec.**
insilico_size10_1	0.97856	0.17	13	72	3	2	0.9444	0.8125	0.8667	0.9600
insilico_size10_2	0.9614	0.25	11	72	2	5	0.9222	0.8462	0.6875	0.9730
insilico_size10_3	0.9794	0.1	12	73	2	3	0.9444	0.8571	0.8000	0.9733
insilico_size10_4	0.9705	0.09	10	75	2	3	0.9444	0.8333	0.7692	0.9740
insilico_size10_5	0.7287	0.17	4	73	5	8	0.8556	0.4444	0.3333	0.9359
insilico_size100_1	0.8990	0.11	146	9621	103	30	0.9866	0.5863	0.8295	0.9894
insilico_size100_2	0.9361	0.08	157	9631	20	92	0.9887	0.8870	0.6305	0.9979
insilico_size100_3	0.9295	0.32	112	9675	30	83	0.9886	0.7887	0.5744	0.9969
insilico_size100_4	0.9239	0.03	118	9599	90	93	0.9815	0.5673	0.5592	0.9907
insilico_size100_5	0.9520	0.07	113	9677	30	80	0.9889	0.7902	0.5855	0.9969

- True Positive (TP) denotes the number of regulatory mechanisms correctly predicted by our approach.
- True Negative (TN) represents the number of arcs that are not present in both the predicted GRN and the target GRN.
- False Positive (FP) denotes the number of regulatory mechanisms predicted by our approach that are incorrect.
- False Negative (FN) denotes the number of regulatory mechanisms not detected by our approach.

As mentioned in the previous section, a threshold needs to be defined to generate a gene regulator from the regulatory matrix. In our experiments, we tested several threshold values ranging from 0 to 1 with a resolution of 0.01. However, due to space limitations, we have only listed the experiments with the highest accuracy and precision for each dataset in the results table.

As it can be observed, the higher the similarity index, the better the accuracy and precision of the generated gene regulatory network. Conversely, the precision decreases when the similarity index is lower, although the accuracy of the predicted gene regulatory network is still quite good. Another aspect that needs to be examined is the number of false negatives. When the number of genes increases, the number of regulatory mechanisms discovered is lower than expected. Conversely, when the number of genes is lower, the number of false negatives is acceptable and the overall sensitivity is higher than when the number of genes is higher. Similarly, false positives are the number of regulatory mechanisms added by our method but not included in the target regulatory network. The resulting gene regulatory network will therefore have more arcs than expected if false positives are high and that will affects the specificity.

5 Conclusions

In this paper, we presented a new method for genetic inference problem. Unlike other methods already present in literature such as Boolean Network [18], our technique allows us not only to perform experiments with our artificial environment, but also to determine the actual interaction between genes with a good accuracy and precision.

In fact, the ability to simulate how the expression levels of a collection of genes change over time is one of the most interesting features of our approach. However, as mentioned above, we are currently unable to demonstrate this part of the work, as all our experiments require to be validated in the laboratory.

Other interesting future work is to compare our method, which is based on an artificial environment, with other existing techniques that use instances with more than a thousand genes.

We are also aware that the threshold defined in Sect. 3.2 was not computed by any method and, therefore, this could be an obstacle. However, we are already working on a solution that consists of using a probabilistic algorithm to define the threshold directly.

Finally, we would like to thank the anonymous reviewers for their careful reading of our manuscript and their insightful comments and suggestions.

References

1. Alvarez, J.M., Brooks, M.D., Swift, J., Coruzzi, G.M.: Time-based systems biology approaches to capture and model dynamic gene regulatory networks. Ann. Rev. Plant Biol. **72**(1) (2021). https://par.nsf.gov/biblio/10231631
2. Aster, R.C., Borchers, B., Thurber, C.H.: Parameter Estimation and Inverse Problems. Elsevier (2018)
3. Avci, O., Abdeljaber, O., Kiranyaz, S., Hussein, M., Gabbouj, M., Inman, D.J.: A review of vibration-based damage detection in civil structures: from traditional methods to machine learning and deep learning applications. Mech. Syst. Sign. Process. **147**, 107077 (2021). https://www.sciencedirect.com/science/article/pii/S0888327020304635
4. Bai, Y., Chen, W., Chen, J., Guo, W.: Deep learning methods for solving linear inverse problems: Research directions and paradigms. Sign. Process. **177**, 107729 (2020). https://www.sciencedirect.com/science/article/pii/S0165168420302723
5. Chikofsky, E., Cross, J.: Reverse engineering and design recovery: a taxonomy. IEEE Softw. **7**(1), 13–17 (1990)
6. Cutello, V., Krasnogor, N., Nicosia, G., Pavone, M.: Immune algorithm versus differential evolution: A comparative case study using high dimensional function optimization. In: 8th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA), Vol. LNCS 4431, pp. 93–101 (2007)
7. Cutello, V., Morelli, G., Nicosia, G., Pavone, M., Scollo, G.: On discrete models and immunological algorithms for protein structure prediction. Nat. Comput. **10**(1), 91–102 (2011). https://doi.org/10.1007/s11047-010-9196-y
8. Flamary, R.: Astronomical image reconstruction with convolutional neural networks. In: 2017 25th European Signal Processing Conference (EUSIPCO), pp. 2468–2472 (2017)

9. Groetsch, C.W., Groetsch, C.: Inverse Problems in the Mathematical Sciences, Vol. 52. Springer (1993)
10. Hecker, M., Lambeck, S., Toepfer, S., van Someren, E., Guthke, R.: Gene regulatory network inference: Data integration in dynamic models-a review. Biosystems **96**(1), 86–103 (2009). https://www.sciencedirect.com/science/article/pii/S0303264708002608
11. Huynh-Thu, V.A., Sanguinetti, G.: Gene regulatory network inference: an introductory survey. In: Gene Regulatory Networks, pp. 1–23. Springer (2019)
12. Kantarci, B., Labatut, V.: Classification of complex networks based on topological properties. In: 2013 International Conference on Cloud and Green Computing, pp. 297–304 (Sep 2013)
13. Karlebach, G., Shamir, R.: Modelling and analysis of gene regulatory networks. Nature Rev. Mol. Cell Biol. **9**(10), 770–780 (2008). https://doi.org/10.1038/nrm2503
14. Lucas, A., Iliadis, M., Molina, R., Katsaggelos, A.K.: Using deep neural networks for inverse problems in imaging: Beyond analytical methods. IEEE Sign. Process. Magaz. **35**(1), 20–36 (2018)
15. Mata, A.S.d.: Complex networks: a mini-review. Brazilian J. Phys. **50**(5), 658–672 (2020). https://doi.org/10.1007/s13538-020-00772-9
16. Pilozzi, L., Farrelly, F.A., Marcucci, G., Conti, C.: Machine learning inverse problem for topological photonics. Commun. Phys. 1(1), 57 (2018). https://doi.org/10.1038/s42005-018-0058-8
17. Rubiolo, M., Milone, D.H., Stegmayer, G.: Extreme learning machines for reverse engineering of gene regulatory networks from expression time series. Bioinformatics **34**(7), 1253–1260 (2017). https://doi.org/10.1093/bioinformatics/btx730
18. Shmulevich, I., Dougherty, E.R., Zhang, W.: Control of stationary behavior in probabilistic Boolean networks by means of structural intervention. J. Biol. Syst. **10**(04), 431–445 (2002). https://doi.org/10.1142/S0218339002000706
19. Talbi, E.G.: Metaheuristics: From Design to Implementation. Wiley Publishing (2009)
20. Vauhkonen, M., Tarvainen, T., Lähivaara, T.: Inverse Problems, pp. 207–227. Springer International Publishing, Cham (2016), https://doi.org/10.1007/978-3-319-27836-0_12
21. Yaman, F., Yakhno, V.G., Potthast, R.: A survey on inverse problems for applied sciences. Math. Prob. Eng., 976837 (2013). https://doi.org/10.1155/2013/976837
22. Yang, Y., Yang, H.: Complex network-based time series analysis. Phys. A Stat. Mech. Appl. **387**(5), 1381–1386 (2008). https://www.sciencedirect.com/science/article/pii/S0378437107011235
23. Zhao, M., He, W., Tang, J., Zou, Q., Guo, F.: A comprehensive overview and critical evaluation of gene regulatory network inference technologies. Brief. Bioinform. **22**(5) (2021). https://doi.org/10.1093/bib/bbab009
24. Zito, F., Cutello, V., Pavone, M.: Optimizing multi-variable time series forecasting using metaheuristics. In: 2022, 14th Metaheuristics International Conference (MIC), vol. LNCS (to appear), pp. 1–15 (2022)