Learning Dynamic Systems from Time-series Data
An Application to Gene Regulatory Networks

Ivo J. P. M. Timoteo and Sean B. Holden
University of Cambridge, Computer Laboratory,
William Gates Building, 15 JJ Thomson Avenue, Cambridge CB3 0FD, U.K.

Keywords: Graphical Models, Local Search, Bioinformatics Application.

Abstract: We propose a local search approach for learning dynamic systems from time-series data, using networks of differential equations as the underlying model. We evaluate the performance of our approach for two scenarios: first, by comparing with an l_1-regularization approach under the assumption of a uniformly weighted network for identifying systems of masses and springs; and then on the task of learning gene regulatory networks, where we compare it with the best performers in the DREAM4 challenge using the original dataset for that challenge. Our method consistently improves on the performance of the other methods considered in both scenarios.

1 INTRODUCTION

We address the following general inference problem. We have a partially observable system consisting of many components, some of which may interact with one-another. Our aim is to infer which components interact, and possibly the nature of the interaction. Such systems will typically be partially observable in the sense that, while measurements over time of one or more properties of the system are available, these time series do not directly expose the underlying interactions.

In studying problems of this kind we have in mind the specific application of inferring *gene regulatory networks (GRNs)*. The action of a gene within an organism is often regulated and coordinated by its interaction with other genes, and considerable effort is invested in trying to discover exactly how genes interact. It is hoped that a better understanding of the network of interactions might lead to improved understanding of the basic processes occurring in a cell, and thus of how specific traits or diseases develop. In turn this might lead to new techniques to treat disease or control cell development. Inferring such relationships from observed data is, however, difficult because the network is large and contains redundant paths, and because data is generally noisy due to the nature of the experimental techniques used.

Biological processes are *dynamic* and we are interested in interactions between molecules occurring *over time* and *in highly complex networks of interdependent nodes*. This suggests that in order to understand a GRN it will be necessary to consider *observations at multiple time instances*, allowing us to observe temporal patterns and thus to better understand causality and predict future behaviour. As a result researchers have increasingly collected time-series gene expression data—according to (Barrett et al., 2011), the number of time-series datasets has been growing exponentially.

Given the dynamic and temporal nature of the typically available data, it is natural to model gene expression networks using stochastic differential equations as suggested by (Ackers et al., 1982), (Gardner and Collins, 2000) and (Marbach et al., 2010); see also (Ly and Lipson, 2012) and (Voortman et al., 2010) for further, related methods. Further, it seems reasonable to assume that a similar model could be used to *learn the structure of the network itself*, although we might expect this to be a computationally expensive process given the use of such expressive models—for example, the one used by GeneNetWeaver (Schaffter et al., 2011) and explained in (Marbach et al., 2010).

In this paper we propose a simple approach for the general problem of learning the structure of systems of differential equations. We demonstrate its effectiveness first by comparing it to an existing l_1-regularization method using data for a system of masses and springs. We then apply it to the prob-

J. P. M. Timoteo I. and B. Holden S..
Learning Dynamic Systems from Time-series Data - An Application to Gene Regulatory Networks.
DOI: 10.5220/0005282303240332
In *Proceedings of the International Conference on Pattern Recognition Applications and Methods* (ICPRAM-2015), pages 324-332
ISBN: 978-989-758-077-2

lem of inferring GRNs. We address the DREAM4 Challenge data (Dream4Challenge, 2009) and compare our method against the methods that performed best in the challenge. These methods did not take time series data into account, and we find that our method can improve on their performance by using them to obtain an initial solution which it then develops further using time series data.

2 INTERACTING DYNAMIC SYSTEMS

2.1 General Model

We have a system of p time-dependent variables $\mathbf{x}^T(t) = \begin{bmatrix} x_1(t) & \cdots & x_p(t) \end{bmatrix}$, and describe the evolution of each variable using differential equations relating it to other variables in the system. The evolution of the ith variable $x_i(t)$ is described by the expression

$$dx_i(t) = F_i(\mathbf{x}(t))dt. \tag{1}$$

The functions F_i are selected from a family $\mathcal{F}$ of functions defined as follows. Begin with a set of *m basis functions* $f = \{f_1(\cdot), f_2(\cdot), ..., f_m(\cdot)\}$; these will be chosen according to the domain of interest. Class $\mathcal{F}$ consists of all functions of the form

$$F(\mathbf{x}) = \sum_{j=1}^{p}\sum_{k=1}^{m} a_{jk} f_k(x_j(t)). \tag{2}$$

where the parameters a_{jk} define a specific F. Given the set f of basis functions, define $\mathbf{f}^T(x) = \begin{bmatrix} f_1(x) & \cdots & f_m(x) \end{bmatrix}$ and

$$\mathbf{f}^T(\mathbf{x}) = \begin{bmatrix} \mathbf{f}^T(x_1) & \cdots & \mathbf{f}^T(x_p) \end{bmatrix}.$$

Then the full system of p variables can be described by

$$d\mathbf{x}(t) = \mathbf{A}\mathbf{f}(\mathbf{x}(t))dt \tag{3}$$

where $\mathbf{A}$ is a $p \times pm$ matrix containing p sets of parameters a_{jk}.

Our goal is as follows: given the set f of basis functions and an observed trajectory

$$X_{0:T} \equiv \{\mathbf{x}(0), \mathbf{x}(\eta), ..., \mathbf{x}(T)\} \tag{4}$$

we wish to recover the structure of the system by identifying $\mathbf{A}$. In (4), η is the interval between observed values, and is assumed to be constant.

Example: Masses and Springs
Consider a dynamic system of masses connected by springs. From the observation $X_{0:T}$ of the positions of the masses over a time interval $[0,T]$ we aim to infer which masses are connected by springs. In this case we know precisely which basis functions to use as they follow directly from Newtonian mechanics. Also, the matrix $\mathbf{A}$ now contains elements with values in $\{0,1\}$ representing the presence or absence of a spring.

Example: Gene Regulatory Networks
As noted in the introduction it is natural to model gene expression networks using stochastic differential equations, and this is our central aim in what follows. Further, just as in the example of masses and springs, we aim to infer useful information regarding the structure of the network by inferring the matrix $\mathbf{A}$.

2.2 Learning Algorithm

Since we observe a trajectory $X_{0:T}$ with interval η between consecutive values we should consider a discrete model. The natural discretization of (3) is to rewrite it as

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \eta\mathbf{A}\mathbf{f}(\mathbf{x}(t)) \tag{5}$$

where $\mathbf{x}(t)$ and $\mathbf{x}(t+1)$ are consecutive observations. Assuming that our model is perturbed by Gaussian noise we employ the least squares error function

$$\text{Error}(\mathbf{A}) = \sum_{i=1}^{p}\sum_{t=0}^{n-1} [x_i(t+1) - x_i(t) - \eta\mathbf{A}_i\mathbf{f}(\mathbf{x}(t))]^2. \tag{6}$$

Where $\mathbf{A}_i$ is the ith row of $\mathbf{A}$. This error function has been shown to be convex when $\mathbf{A}$ is real-valued (Banerjee et al., 2008).

Our proposed algorithm assumes that $\mathbf{A}$ has elements $A_{i,j} \in W$ where W is a finite set. This assumption is not too restrictive as in many interesting problems the existence of an edge and its nature are all we are looking for. For example, in addressing gene interactions we are mainly interested in identifying whether a gene inhibits, enhances or does not influence other genes. In this case $W = \{-1,0,1\}$ would be an appropriate choice provided we have an adequate set f of basis functions. Also, determining the structure of the system in this way can form a first step making the edge weights easier to estimate afterwards.

Even though (6) is convex for a real-valued $\mathbf{A}$, our choice to restrain $A_{i,j}$ to W entails the loss of that guarantee. Nevertheless, we show that restraining the search space according to the domain of interest, despite the loss of convexity, can lead to improved performance (Experiment 1).

We use a local search strategy to avoid searching all $|W|^{p^2 m}$ possible structures. We start with an arbitrary structure—perhaps chosen according to the specific domain to reduce the search time—and generate new candidates by selecting an edge and altering the value among the possibilities in W. If the change results in a decrease in the error, we keep the new structure and restart. When we cannot generate new candidates that decrease the error we stop the search. This strategy can no longer guarantee that we achieve a global minimum so we consider additional strategies that may help us avoid local minima. For convenience, we will refer to this approach as *flip-edge* throughout the paper due to the way new structures are generated.

It is of interest to note that the error function used is decomposable, in the sense that is familiar from probabilistic graphical models (Koller and Friedman, 2009). This leads to a simple method for quickly evaluating the error. The error function is a sum of p summands, one for each variable and each itself having limited dependencies. The total error is a sum of individual *node errors*

$$\text{Error}(\mathbf{A}) = \sum_{i=1}^{p} \text{nodeError}_i(\mathbf{A}) \tag{7}$$

where

$$\text{nodeError}_i(\mathbf{A}) = \sum_{t=0}^{n-1} \left[x_i(t+1) - x_i(t) - \eta \mathbf{A}_i \mathbf{f}(\mathbf{x}(t))\right]^2 . \tag{8}$$

This is useful as our algorithm, from an initial structure, generates neighbouring structures based on one edge flip, and we want to minimise the error function. Let $e_{(j,k),i}$ be the edge from variable j to variable i associated with f_k. We are seeking the value $w \in W$ for $e_{(j,k),i}$ that minimises the error function. Since $e_{(j,k),i}$ is only associated with row A_i we know that minimising the error of $\mathbf{A}$ is equivalent to minimising $\text{nodeError}_i(\mathbf{A})$. Let $\delta(e)$ be a function that returns the w minimising $\text{Error}(\mathbf{A})$ when flipping edge e. This function can easily be cached as it will only change when some other element of the relevant $\mathbf{A}_i$ is changed.

3 EXPERIMENTS

3.1 Masses and Springs

We first compare the flip-edge algorithm with a l_1-regularised least squares (l_1-rls) algorithm in a masses and springs scenario as presented in (Bento et al., 2010) and introduced above. We will assume rest distances, masses and elastic coefficients are equal to 1 and focus on inferring which masses are connected. The general equations that model the dynamics of such a system are

$$\begin{aligned} d\mathbf{q}(t) &= \mathbf{v}(t)dt \\ d\mathbf{v}(t) &= -\nabla U(\mathbf{q}(t))dt + \sigma d\mathbf{W}(t) \\ U(q_i(t)) &= \frac{1}{2}\sum_j \mathbf{A}_{i,j}(||q_i - q_j|| - D_{i,j})^2 \end{aligned} \tag{9}$$

where $\mathbf{q}$ is the position, $\mathbf{v}$ is the velocity, $D_{i,j}$ is the resting distance, $d\mathbf{W}(t)$ denotes the Wiener process and σ^2 is the noise variance. The simulation is performed using the Euler-Maruyama method for a time step of 0.01. We sample using $\eta = 0.1$ and use $\sigma = 0.2$ unless explicitly stated.

The same dynamic system was previously used as an example by (Bento et al., 2011) though with a different target structure. (Bento et al., 2010) propose the l_1-rls algorithm for the recovery of the structure of a network of stochastic differential equations[1]. The idea is to recover each row of the support matrix $\mathbf{A}$ independently by solving a convex optimisation problem. For each row i, $\mathbf{A}_i$ is estimated by minimising

$$L(\mathbf{A}_i; X_{0:T}) + \lambda||\mathbf{A}_i||_1 \tag{10}$$

where

$$L(\mathbf{A}_i; X_{0:T}) = \frac{1}{2\eta^2 n}\sum_{t=0}^{T} \left[x_i(t+1) - x_i(t) - \eta \mathbf{A}_i \mathbf{f}(\mathbf{x}(t))\right]^2 . \tag{11}$$

We used the CVX package (Grant and Boyd, 2008; Grant and Boyd, 2012) to describe and solve the convex program.

Our experiments used the following protocol: a network is generated, the behaviour of this network over an interval of time is simulated, and the resulting data is supplied to the algorithms. Each network is generated as an $N \times M$ grid of masses, separated along each axis by a unit of distance and connected to all eight neighbours where possible. To generate different structures, masses and springs are removed with probability 0.1. We guarantee that all the springs in the final structure are connected to two masses. In the experiments a 5×5 initial grid is used except when explicitly studying the performance under different network sizes.

Our goal was to compare our method against an optimally tuned instance of l_1-rls. We therefore started by running preliminary experiments to study the response of the l_1-rls algorithm to different values

[1]Though their focus is on the theoretical bounds of the recovery rather than on performance.

of its parameters λ and τ (a threshold above which we consider the existence of an edge). The preliminary study led to the following approach: we run the algorithm for $\lambda \in \{0, 0.05, 0.10, \ldots, 0.90, 0.95\}$ and $\tau \in \{0, 0.1, 0.2, \ldots, 1.8, 1.9\}$ and use the best pair (λ, τ), chosen by direct comparison against the results obtained with the known correct structure, to extract the structure for comparison with our algorithm. Furthermore, we guaranteed that the chosen pair was a local minimum by 8-neighbour comparison in the $\lambda \times \tau$ space considered.

In a realistic setting, we would not be able to tune l_1-rls using the correct structure. Nevertheless, we wanted to compare our suggested flip-edge algorithm with the best possible l_1-rls results.

Figure 1 shows the results from these experiments. These results suggest that *flip-edge*, while undoubtedly a straightforward algorithm, performs very well in uniform weight networks. In fact, in our experiments it surpasses the l_1-regularization approach common in the literature in its ability to recover the structure of the network. Specifically, it proved more resistant to noise, small trajectories of observed data, and size of the network (for the same length of trajectory). Furthermore, the fact that it does not require tuning of the parameters, for which there seems to be no obvious method in the l_1-rls algorithm, makes it a more straightforward method to use.

3.2 Learning Gene Regulatory Networks from Time-series Knockout Data

We assessed our algorithm in the context of identifying GRNs by employing data from the *DREAM4 In Silico Network Challenge* (Dream4Challenge, 2009) (see also (Marbach et al., 2010; Marbach et al., 2009; Prill et al., 2010)). This allowed us to assess our algorithm against many other methods—at least 25 in the challenge itself—evaluated under the same conditions.

Time-series data are especially useful if gathered after applying perturbations to the regulatory equilibrium of the cell, as the relationships are then more apparent than in a stable cell where the expression of each gene remains reasonably constant. A common experiment is to knock out a gene, that is, to completely block its expression, and observe the system as it re-adapts. With this in mind, the DREAM4 challenge involves inferring the structure of five gene regulatory networks, each with 100 genes. Two were extracted from E. coli and three from Yeast. We are provided with data on the steady-state expression levels of the wild-type, and after the knockout and knockdown of every gene. (*Knockdowns* reduce the expression of some gene to half of that in the wild-type.) We are also provided with 10 time-series of 21 time points where an unknown perturbation is applied at the start and released half way through the time-series.

We used the three best-performing methods from the DREAM4 Challenge as comparisons for our method. Pinna et al. (Pinna et al., 2010) presented the best method by applying a pruning procedure on top of *z-score* to remove potentially redundant edges. This algorithm has a parameter—denoted t in the original paper and renamed to θ in the following—used to decide when an edge should be included. The *z-score* method of Prill et al. (Prill et al., 2010) was placed second and the *TRANSWESD* method of Klamt et al. (Klamt et al., 2010) achieved the third best performance with a variant of transitive reduction for weighted signed digraphs.

It is of interest to note that the three top scorers in the DREAM4 Challenge did not make any use of the time-series data, as it seems clear that it might provide interesting insights into the correct structure. Figure 2 shows an example of three genes (P: dotted line, D: dashed line and S: solid line) where there are inhibitory relationships from P to S, P to D and S to D. However, from the observation of the initial and final states only, it would be impossible to infer anything more than the inhibitory relationship from P to S.

We applied the flip-edge algorithm to this data. Our method relies on time-series data to improve the inferences made by the existing static methods. Unlike in the example using masses and springs we do not in this case know a correct set of basis functions. Bonneau et al. (Bonneau et al., 2006) suggest the use of the logistic sigmoid (while ultimately using a linear approximation) and this agrees with our initial intuition. The model thus becomes, for some general gene i,

$$dx_i(t) = [-x_i(t) + F(\mathbf{A}_i, \beta, \mathbf{x}(t))]\,dt + \sigma d\mathbf{W}(t) \quad (12)$$

where

$$F(\mathbf{A}_i, \beta, \mathbf{x}(t)) = \frac{\max_{eq}(x_i)}{1 + \exp\left(-\sum_j A_{i,j}\beta_j x_j(t)\right)}. \quad (13)$$

Here, $d\mathbf{W}(t)$ is a Wiener process, β_j denotes the weight the expression of the jth gene would have on the activation of the other genes, and and $\max_{eq}(x_i)$ denotes the maximum observed value of x_i in an equilibrium state. We added the scaling factor $\max_{eq}(x_i)$ since the observed maximum expression values may vary considerably for each variable. However, this results in a need to estimate $\boldsymbol{\beta}$. With the above model,

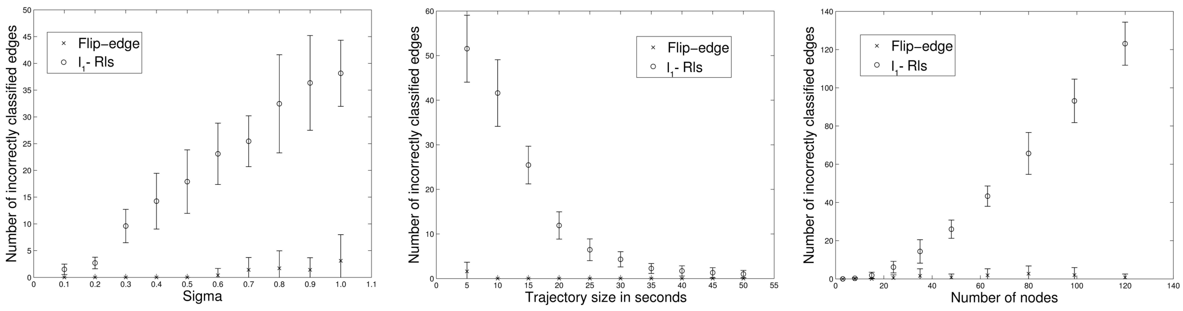

Figure 1: Identification of systems of masses and springs. We plot the number of incorrectly classified edges against, respectively: noise variance, size of the observed time-series, and size of the structure being learnt. Each data point was obtained by running the algorithm 10 times; the bars correspond to one standard deviation.

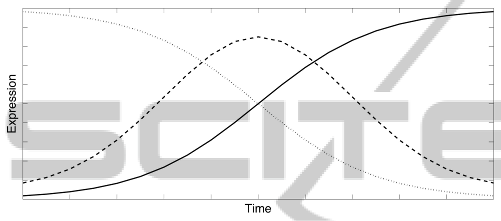

Figure 2: Example of the evolution of the expression of three genes where time-series data, rather than just the observation of the initial and final state, is crucial to infer all three relationships.

the error function becomes

$$\text{Error}(\mathbf{A},\boldsymbol{\beta}) = \sum_{i=1}^{p}\sum_{t}\left(\frac{x_i(t+1)-x_i(t)}{\eta} + x_i(t) - \frac{\max_{eq}(x_i)}{1+\exp\left(-\sum_j A_{i,j}\beta_j x_j(t)\right)}\right)^2 \tag{14}$$

We used $W = \{-1,0,1\}$ as we are interested in whether a gene is inhibiting, enhancing or just unrelated to the expression of some other gene. We ran all experiments with $\eta = 0.001$ after observing the gene expression changes, from step to step, in the time series data.

The flip-edge algorithm was modified as follows to incorporate the estimation of $\boldsymbol{\beta}$:

1. Generate an initial network structure.
2. Use the current structure to estimate $\boldsymbol{\beta}$.
3. Run *flip-edge* using the estimated $\boldsymbol{\beta}$.

In step 1 of the algorithm we generated prior structures using the two most successful methods from the DREAM4 Challenge, and using different values for θ. After this step we have the first candidate **A**.

In step 2 we estimate $\boldsymbol{\beta}$ by performing ordinary least squares multivariate regression. Note that in this algorithm β_i is constant—it does not itself vary with time. The motivation for this assumption was that β would be modelling transcription rates rather than the effectiveness of the resulting protein in influencing the transcription of the other genes. For the estimation of $\boldsymbol{\beta}$ we only consider the gene expressions at stable states such as the wild-type—that is, the expression values of the non-perturbed network—and the stable states after the knockout of a single gene. In a general stable state $\mathbf{s}$ we have $ds_i(t)/dt = 0$, along with $s_i = F(\mathbf{A}_i, \boldsymbol{\beta}, \mathbf{s})$ and

$$\left(\mathbf{A}_i \circ \mathbf{s}^T\right)\boldsymbol{\beta} = -\log\left(\frac{\max_{eq}(s_i)}{s_i} - 1\right) = g(s_i) \tag{15}$$

where $\circ$ denotes the Hadamard product. That is, the expression of the gene $\mathbf{s}_i$ is regulated by its parents $(\mathbf{A}_i \circ \mathbf{s}^T)$ weighted by $\boldsymbol{\beta}$. Let $\mathbf{S}$ be the square matrix of stacked $\mathbf{s}^T$ and define $\mathbf{g}^T(\mathbf{s}) = \begin{bmatrix} g(s_1) & \cdots & g(s_p) \end{bmatrix}$ such that

$$\mathbf{g}(\mathbf{s}) = (\mathbf{A} \circ \mathbf{S})\boldsymbol{\beta}. \tag{16}$$

Finally let $\mathbf{y}$ be a vector of stacked $\mathbf{g}(\mathbf{s})$ and $\mathbf{X}$ a matrix of stacked $\mathbf{A} \circ \mathbf{S}$ so that we can estimate $\boldsymbol{\beta}$ using all the instances of stable states available. We have

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} \tag{17}$$

and we want to estimate $\boldsymbol{\beta}$ by least squares

$$\underset{\boldsymbol{\beta}}{\text{argmin}} \sum_{i=1}^{k} (\mathbf{y}_i - \mathbf{X}_i\boldsymbol{\beta})^T(\mathbf{y}_i - \mathbf{X}_i\boldsymbol{\beta}) \tag{18}$$

where k denotes the number of stable state data instances, $\mathbf{y}_i$ corresponds to the ith $\mathbf{g}(\mathbf{s})$ and $\mathbf{X}_i$ corresponds to the ith $(\mathbf{A} \circ \mathbf{S})$, which gives us

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{19}$$

In step 3 we use $\mathbf{A}$ from step 1 and $\boldsymbol{\beta}$ from step 2, and run *flip-edge* gradient descent using (14) which is also decomposable. When applying *flip-edge* we decided to use only the second half of the time-series data provided; that is, from the point when the perturbation is lifted, as we do not know the exact nature of the perturbation.

Table 1: Comparison of the results of other methods with our method run for three rounds of estimation and *flip-edge* with different values of θ.

	overall	auroc	aupr
Pinna et al	71.589	40.110	103.068
z-score	71.297	39.737	102.857
TRANSWESD	64.715	37.324	92.106
$\theta = 2.5$	78.232	48.684	107.779
$\theta = 3$	79.116	49.278	108.953
$\theta = 3.5$	82.962	52.707	113.217
$\theta = 4$	80.955	50.584	111.326
$\theta = 4.5$	77.892	49.051	106.732

The DREAM4 Challenge evaluated performance using the *p*-values for the area under the ROC curve (AUROC) and area under the precision-recall curve (AUPR) measures for each network—see (Schaffter et al., 2011) and (Dream4Challenge, 2009) for a complete description. The *p*-values provide an indication of the probability that random predictions would be of equal or better quality than those produced by the method of interest. Let p_{AUROC} and p_{AUPR} be the geometric means of the corresponding *p*-values for the five networks of the dataset. The scores are given by

$$\begin{aligned} \text{auroc score} &= -\log_{10}(p_{\text{AUROC}}) \\ \text{aupr score} &= -\log_{10}(p_{\text{AUPR}}) \\ \text{overall score} &= -0.5\log_{10}(p_{\text{AUROC}} \times p_{\text{AUPR}}). \end{aligned} \quad (20)$$

Figure 3 (a) shows the performance of our method alongside that of the top three methods presented at DREAM4. (As pointed out in (Schaffter et al., 2011), the best performer in the DREAM3 Challenge (Yip et al., 2010) did not repeat its previous success. The method presented in (Yip et al., 2010) uses an approach similar to ours in trying to learn first from knockout and then from time-series data using differential equation models.)

Figure 3 (b) shows how the performance of our algorithm is affected by using different methods to choose the initial structure. Clearly the use of *z-score* or the method of Pinna et al. has little effect on the score achieved by our algorithm. This suggests that our algorithm is improving due to the time series data, independently of the initial structure and most likely improving on different features than those targeted by Pinna et al.'s pruning step. The average difference in the score over the values of θ considered is 0.052 in favour of Pinna et al., which is considerably smaller than the original difference before the application of our method (see Table 1).

Unfortunately, comparison using only the data of the DREAM4 Challenge does not allow us to have statistical confidence in the performance of our method, since we address only one prediction task. However, the GeneNetWeaver distribution includes the kinetic models of the networks used in the DREAM4 Challenge. This allows us to generate new data from the same networks. We ran our method as described above with $\theta = 3.5$. We used 10 new datasets from each individual network and we compared our score with the score obtained by our implementation of Pinna et al.'s method, in this case using $\theta \in \{2, 2.5, 3, 3.5\}$ and finding that $\theta = 3.0$ maximised its average score over the set of 50 prediction tasks (ten datasets drawn from five networks). In Figure 4, we present the percentage improvement of our method over Pinna et al. with $\theta = 3.0$ for the ten datasets for each of the five DREAM4 Challenge networks. In Figure 4 we see a box plot where the whole boxes and whiskers are above zero. Furthermore, the p-values for the t-test on the improvement for the different networks are 0.00064, 1.43057e-09, 0.00637, 1.88231e-05, and 6.6216e-10 meaning that the improvement of our method over the best scorer at the DREAM4 Challenge is strongly significant. In fact, it boasts quite surprising improvement in some networks suggesting it is considerably more efficient at discovering particular graphical features.

4 FURTHER REFINEMENTS

Since the local search approach cannot guarantee to end at the global minimum of the error function it is interesting to consider some strategies to try to avoid finishing the search in local minima. Many different possibilities are presented in the literature, such as random restarts, tabu searches and annealing methods, among others. Given the characteristics of our problem we were led to exploring data perturbation methods.

The idea is that it might be possible to escape local minima if we slightly change the weight given to each data point. Intuitively, we want to change the weights of the data in the error function enough to escape local minima while keeping the general shape of the error function unaltered. A common approach is to consider a probability distribution over the weights and make it converge to the uniform distribution so that we get closer and closer to the original dataset over time.

Looking back at (14) we can see that the error function can be decomposed as a sum of terms each corresponding to a pair i, t.

$$\text{Error}(\mathbf{A}, \boldsymbol{\beta}) = \sum_{i=1}^{p} \sum_{t} \text{error}(i, t, \mathbf{A}, \boldsymbol{\beta}) \quad (21)$$

to which we can add a matrix of weights

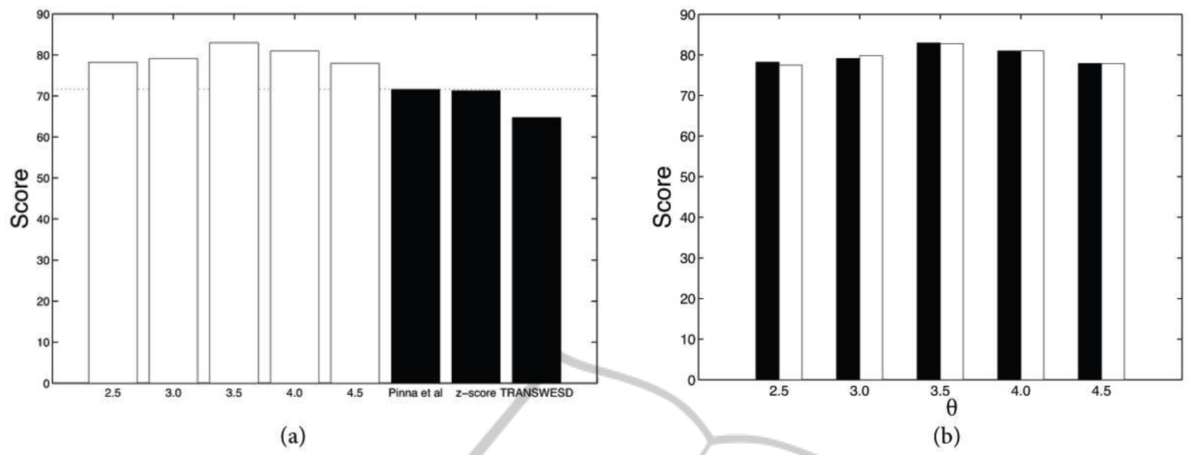

Figure 3: (a) Comparison of the scores of our method (white), for different values of θ, and the top three scorers (black) in the DREAM4 Challenge. The horizontal dotted line represents the score of the best method at the DREAM4 Challenge.
(b) Comparison of the scores achieved by our method from different starting structures and using different values for θ. Black bars: using the method of Pinna et al. as initial structure; white bars: using *z-score* as initial structure.

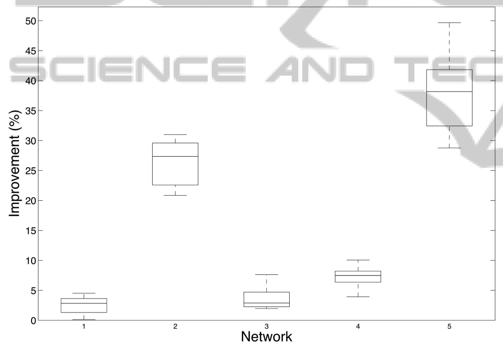

Figure 4: Percentage improvement of *flip-edge* with $\theta = 3.5$ and three rounds, over our implementation of the method of Pinna et al. with $\theta = 3.0$ (the best score from $\theta \in \{2, 2.5, 3, 3.5\}$) for ten new datasets generated from each original DREAM4 network. The p-values for the t-test on the improvement for the different networks are 0.00064; 1.43057e-09; 0.00637; 1.88231e-05; 6.6216e-10.

$$\mathrm{Error}(\mathbf{A}, \boldsymbol{\beta}, \mathbf{w}) = \sum_{i=1}^{p} \sum_{t} w_{i,t}\,\mathrm{error}(i, t, \mathbf{A}, \boldsymbol{\beta}) \qquad (22)$$

4.1 Random Perturbation

For the random perturbation we sample each $w_{i,t}$ from a Gamma distribution (as it is non-negative)

$$w_{i,t} \sim \mathrm{Gamma}(1/\tau, \tau) \qquad (23)$$

where τ is a “temperature” variable that decreases over time. The above distribution has mean equal to one and variance proportional to τ therefore forcing $w_{i,t}$ to approach one over time.

4.2 Adversarial Perturbation

With the aim of escaping a possible local minimum, it is possible to employ a more informed strategy when choosing the weights used to perturb the data. The general idea, as presented by (Elidan et al., 2002), is as follows. We aim to modify the error function such that the current minimum becomes less desirable. In this way the optimization is able potentially to escape from the minimum, should it be local. This means that instead of randomly generating a new weight vector, we will iteratively update it. In general one can consider:

$$w_{i,t}^{n+1} = w_{i,t}^{n} + \zeta \frac{\partial \mathrm{Error}}{\partial w_{i,t}} \qquad (24)$$

so that we increase the relevance of the weights that contribute more towards an increased value of the error function. (Elidan et al., 2002) further develops this idea to ensure convergence towards the uniform distribution and avoid negative weight values. Following their results we have the following weight update equation

$$w_{i,t}^{n+1} = \alpha^{n+1} \left(w_{i,t}^{0}\right)^{\frac{\kappa}{\kappa+\lambda}} \left(w_{i,t}^{n}\right)^{\frac{\lambda}{\kappa+\lambda}} e^{-\frac{\zeta}{\kappa+\lambda}\left(\frac{\partial \mathrm{Error}}{\partial w_{i,t}}\big|_{w_{i,t}^{n}}\right)} \qquad (25)$$

where α^{n+1} is a normalisation constant and both κ and λ are inversely proportional to the temperature τ.

In both the random and adversarial approaches, we start with uniform weights and allow our algorithm to converge to a (possibly local) minimum. We then select a new set of weights using the appropriate update equation and temperature, and optimise again starting from the previously found minimum. This process is repeated several times, updating the temperature at each iteration, and the best minimum found is returned.

We further extended our method by checking whether the best solution was in a plateau, and if so continuing the search in case we were able to escape the plateau.

4.3 Results

All the experiments were run for ten temperature updates where the temperature was reduced by a factor of a ($\tau \leftarrow a\tau$). For the random perturbation, where $w_{i,t} \sim \text{Gamma}(1/\tau, \tau)$, we used an initial $\tau = 1$, $a = 0.9$ for five runs. For adversarial perturbation we used initial $\tau = 0.001$, $a = 0.9$, $\zeta = 1$ and $\kappa, \lambda = 1/\tau$ unless otherwise stated. Note that adversarial perturbation is a deterministic procedure so only one run per parameter configuration was performed.

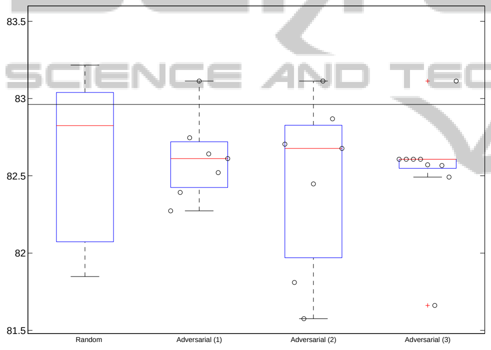

Figure 5: Score of different runs after including data perturbation. The horizontal line marks the best score achieved without data perturbation. The first box refers to random perturbation and the following three refer to adversarial perturbation: (1) for different $\zeta = [0.7, 0.8, \ldots, 1.3]$; (2) for different initial $\tau = [10, 1, \ldots, 1e-05]$; (3) for different temperature fall rates $a = [0.1, 0.2, \ldots, 0.9]$. Also plotted are the actual values for each value of ζ, τ and a.

Figure 5 presents the results obtained using the refinements proposed above. We can see that they do not consistently improve on the results obtained using non-perturbed data though some configurations of parameters and some runs using random perturbation indeed improved on the score. Checking for the existence of plateaus produced no improvement on the score. On the rare occasions it lead to an escape the search ended on a minimum equivalent to one found for another temperature.

In terms of the value of the error functions, instead of the DREAM4 score, data perturbation was consistently reaching lower values. Also of note is the fact that the correct network, with the data given, has a higher value on the error function than the structures found by those methods, meaning that our method would discard it in favour of the networks it eventually found. This could be due to the noise in the data or to the unavoidable misrepresentation of the phenomenon by our chosen basis function. Nevertheless, all the alternatives studied here are clear winners when compared to the best methods that tackled the DREAM4 challenge, strongly suggesting that the use of time series data and these simple, yet effective, methods is beneficial.

5 FINAL REMARKS

It is important to note that some of the data provided in the DREAM4 Challenge datasets—namely, information on the steady-state expression after the knock out of each gene in the network—is very unrealistic. However that data was only used in the creation of an appropriate prior structure, and all subsequent improvement was made using time-series perturbation data from ten experiments. This data is considerably more realistic in terms of what might be produced in a laboratory environment, especially since we were not informed of the exact characteristics of the perturbation.

It might be argued that with the release of the information concerning GeneNetWeaver (Schaffter et al., 2011)—the *in silico* simulator used to generate the datasets—and the datasets themselves with the gold standard, we would have an unfair advantage. Note, however, that our method is completely independent from the design of GeneNetWeaver and all the parameters apart from θ—used to create the initial structure before the time series dataset is used—are estimated using only the initial dataset.

In terms of application to a real-world scenario, it is fair to assume that there would be prior knowledge concerning the organism being studied, and that the experiments themselves would have a specific goal. In such a scenario we feel that the prior knowledge could help in the construction of an appropriate prior structure (a fairly sparse but high-confidence prior seems to be favourable to *flip-edge*). *Flip-edge* would then be run on time-series data from appropriately designed experiments. We also feel that knowing the characteristics of the perturbation could help in making the method more efficient.

ACKNOWLEDGEMENTS

Ivo Timoteo is supported by an FCT Individual Doctoral Fellowship, number SFRH/BD/88466/2012.

REFERENCES

Ackers, G. K., Johnson, A. D., and Shea, M. A. (1982). Quantitative model for gene regulation by lambda phage repressor. *Proceedings of the National Academy of Sciences*, 79(4):1129–1133.

Banerjee, O., Ghaoui, L., and D'Aspremont, A. (2008). Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *Journal of Machine Learing Research*, 9:485–516.

Barrett, T., Troup, D. B., Wilhite, S. E., Ledoux, P., Evangelista, C., Kim, I. F., Tomashevsky, M., Marshall, K. A., Phillippy, K. H., Sherman, P. M., Muertter, R. N., Holko, M., Ayanbule, O., Yefanov, A., and Soboleva, A. (2011). NCBI GEO: archive for functional genomics data set - 10 years on. *Nucleic Acids Research*, 39:D1005–D1010.

Bento, J., Ibrahimi, M., and Montanari, A. (2010). Learning networks of stochastic differential equations. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *NIPS*, pages 172–180. Curran Associates, Inc.

Bento, J., Ibrahimi, M., and Montanari, A. (2011). Information theoretic limits on learning stochastic differential equations. In Kuleshov, A., Blinovsky, V., and Ephremides, A., editors, *ISIT*, pages 855–859. IEEE.

Bonneau, R., Reiss, D. J., Shannon, P., Facciotti, M., Hood, L., Baliga, N. S., and Thorsson, V. (2006). The inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets *de novo*. *Genome Biology*, 7(5).

Dream4Challenge (2009). DREAM4, Challenge 2 – In Silico Network Challenge. http://wiki.c2b2.columbia.edu/dream/index.php?title=D4c2.

Elidan, G., Ninio, M., Friedman, N., and Schuurmans, D. (2002). Data perturbation for escaping local maxima in learning. In *In AAAI*, pages 132–139.

Gardner, T. S. and Collins, J. J. (2000). Neutralizing noise in gene networks. *Nature*, 405(6786).

Grant, M. and Boyd, S. (2008). Graph implementations for nonsmooth convex programs. In Blondel, V., Boyd, S., and Kimura, H., editors, *Recent Advances in Learning and Control*, volume 371 of *Lecture Notes in Control and Information Sciences*, pages 95–110. Springer London.

Grant, M. and Boyd, S. (2012). CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx.

Klamt, S., Flassig, R., and Sundmacher, K. (2010). TRANSWESD: inferring cellular networks with transitive reduction. *Bioinformatics*, 26(17):2160–2168.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT Press.

Ly, D. L. and Lipson, H. (2012). Learning symbolic representations of hybrid dynamical systems. *Journal of Machine Learning Research*, 13:3585–3618.

Marbach, D., Prill, R. J., Schaffter, T., Mattiussi, C., Floreano, D., and Stolovitzky, G. (2010). Revealing strengths and weaknesses of methods for gene network inference. *Proceedings of the National Academy of Sciences*.

Marbach, D., Schaffter, T., Mattiussi, C., and Floreano, D. (2009). Generating realistic in silico gene networks for performance assessment of reverse engineering methods. *Journal of Computational Biology*, 16(2):229–239.

Pinna, A., Soranzo, N., and de la Fuente, A. (2010). From knockouts to networks: establishing direct cause-effect relationships through graph analysis. *PLoS ONE 5(10): e12912. doi:10.1371/journal.pone.0012912*.

Prill, R. J., Marbach, D., Saez-Rodriguez, J., Sorger, P. K., Alexopoulos, L. G., Xue, X., Clarke, N. D., Altan-Bonnet, G., and Stolovitzky, G. (2010). Towards a rigorous assessment of systems biology models: The DREAM3 challenges. *PLoS ONE*, 5(2):e9202.

Schaffter, T., Marbach, D., and Floreano, D. (2011). GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods. *Bioinformatics*, 27(16):2263–2270.

Voortman, M., Dash, D., and Druzdzel, M. (2010). Learning why things change: The difference-based causality learner. In *Proceedings of the Twenty-Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*.

Yip, K., Alexander, R., Yan, K., and Gerstein, M. (2010). Improved reconstruction of *In Silico* gene regulatory networks by integrating knockout and perturbation data. *PLoS ONE 5(1): e8121. doi:10.1371/journal.pone.0008121*, 5(1):e8121.