A Parallel Implementation of the Network Identification by Multiple Regression (NIR) Algorithm to Reverse-Engineer Regulatory Gene Networks

Francesco Gregoretti^{1*}, Vincenzo Belcastro², Diego di Bernardo², Gennaro Oliva^{1*}

¹ Institute of High Performance Computing and Networking, Naples, Italy, ² Telethon Institute of Genetics and Medicine, Naples, Italy

Abstract

The reverse engineering of gene regulatory networks using gene expression profile data has become crucial to gain novel biological knowledge. Large amounts of data that need to be analyzed are currently being produced due to advances in microarray technologies. Using current reverse engineering algorithms to analyze large data sets can be very computational-intensive. These emerging computational requirements can be met using parallel computing techniques. It has been shown that the Network Identification by multiple Regression (NIR) algorithm performs better than the other ready-to-use reverse engineering software. However it cannot be used with large networks with thousands of nodes - as is the case in biological networks - due to the high time and space complexity. In this work we overcome this limitation by designing and developing a parallel version of the NIR algorithm. The new implementation of the algorithm reaches a very good accuracy even for large gene networks, improving our understanding of the gene regulatory networks that is crucial for a wide range of biomedical applications.

Citation: Gregoretti F, Belcastro V, di Bernardo D, Oliva G (2010) A Parallel Implementation of the Network Identification by Multiple Regression (NIR) Algorithm to Reverse-Engineer Regulatory Gene Networks. PLoS ONE 5(4): e10179. doi:10.1371/journal.pone.0010179

Editor: Carl Kingsford, University of Maryland, United States of America

Received: December 2, 2009; **Accepted:** March 22, 2010; **Published:** April 21, 2010

Copyright: © 2010 Gregoretti et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Funding: The authors have no support or funding to report.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: francesco.gregoretti@na.icar.cnr.it (FG); gennaro.oliva@na.icar.cnr.it (GO)

Introduction

Microarray analysis methods produce large sets of gene expression data that can be exploited for novel insights into the fundamentals of molecular biology research. Inferring gene regulating networks from microarray gene expression data has become one of the major topics in system biology. Inferring or ‘reverse-engineering’ gene networks can be defined as the process of identifying regulatory gene interactions from experimental data through computational analysis.

Since the advent of microarray, various methods have been developed to infer the underlying gene regulatory network. The pioneer methods were simple clustering algorithms [1] where the similarity between genes were measured by a distance or “pseudo-distance” metric such as the clustering coefficient.

Since the structure to infer is a gene network, graphical models have been proposed and developed. This is the case of BANJO [2], that assumes that the gene network can be modeled as a Bayesian network. Bayesian statistics can be applied under some constrain [3].

Information theoretic approaches have been first proposed in [4] but the very first application was ARACNe [5]. ARACNe computes a pairwise pseudo-distance between each pair of genes to check their dependence. Theoretically ARACNe can be run to infer networks of any dimension.

All the mentioned algorithms have a ready-to-use software, and their performances have been tested and compared on both in-silico and in-vivo gene networks [6].

Here we will focus on an ODE-based algorithm, NIR [7], that relates the expression of each gene with the expression of other

genes in the cell. It has been shown that NIR is able to correctly identify these relations called “influence interactions”. The ensemble of these interactions is referred to as gene network. Each of the recovered interactions within the gene network implies a regulatory interaction between components (Proteins, mRNAs, metabolites, etc.) of the cell. Gene networks can be used to identify the functional modules i.e. the subset of genes that regulate each other with multiple indirect interactions; to predict the response of the network to external perturbations and identify the genes directly hit; to identify physical interactions when integrated with additional information coming from sequence data and other experimental data.

Gene network inference algorithms based on ODEs relate gene transcript concentration changes to each other and to an external perturbation. The external perturbation is generally an experimental treatment that can alter the transcription rate of the genes in the cell. An example of perturbation is the treatment with a drug or a genetic perturbation that results in the overexpression or the downregulation of particular gene.

NIR algorithm achieves better results than the other network inference algorithms being able to reach peaks of 95% of correct predicted interactions. This has been shown in [8] where there have been selected and compared the algorithms capable of solving the network inference problem with an available ready-to-use software. In that work it wasn’t possible to run NIR on data set with more than 100 genes because it would have taken too much time. Microarray technology allows the measurements of many thousands of transcripts at once and the mammalian gene regulatory network is of the order of 10^4 . The complexity in

handling networks of this size [6], motivated us to design, develop and run a parallel computing algorithm.

In section Methods we describe the NIR algorithm and its parallel implementation; in section Results we describe experimental results and in section Discussion we give our conclusions.

Methods

The NIR algorithm

It is possible to describe the gene network as a system of differential equations in which each equation describes the variation in time of the concentration of a particular transcript (gene), x_i , as a non linear function, f_i , of the concentrations of the other transcripts:

$$\dot{x}_i(t) = f_i(\underline{x}(t), u_i(t), \theta_i), \quad i = 1, \dots, N, \quad (1)$$

where $\underline{x}(t) = [x_1(t), \dots, x_n(t)]$ is a vector whose components are the concentrations of the transcripts measured at time t , $u_i(t)$ is the external perturbation applied at gene i at time t , θ_i is a set of parameters describing interactions amongst genes, $\dot{x}_i(t) = \frac{dx_i}{dt}(t)$ is the rate of change in transcription of transcript x_i and N is the number of genes.

To reverse-engineer a network using ODEs means to choose a functional form for f_i and then to estimate the unknown parameters θ_i for each i by using the gene expression data. With the ODE-based approach the resulting gene network will be a directed graph (i.e., if a_{ij} is the interaction between genes i and j , it specifies the direction of the interaction, that is, gene j regulates gene i and not vice versa, $a_{ij} \neq a_{ji}$).

We expand f_i in a Taylor series around x_0 , the point in which measurements were made. If we assume that perturbations around this point are sufficiently small, it is possible to truncate the Taylor series after the first order term, and obtain a linear expression:

$$\dot{x}_i(t) = \sum_{j=1}^N a_{ij} x_j(t) + b_i u_i(t), \quad i = 1, \dots, N, \quad (2)$$

where a_{ij} represents the influence of gene j on gene i , b_i represents the effect of the external perturbation on x_i (in subsequent analysis, we set b_i equal to one for the sake of simplicity) and $\underline{u}(t)$ is the vector of the external perturbations at time t (a_{ij} and b_i are the θ_i in equation (1)). $\dot{x}_i(t)$ is the rate of change of concentration of gene i at time t , i.e., the first derivative of the mRNA concentration of gene i at time t . In proximity of the steady-state the concentrations of the N transcripts don't change in time ($\dot{x}_i(t) = 0 \forall i$) so that the (2) can be rewritten as:

$$\underline{a}_i^T \underline{x} = -b_i u_i, \quad i = 1, \dots, N. \quad (3)$$

Let us suppose that we have conducted M experiments such that we know the genes directly perturbed ($u_i(k)$, $k = 1, \dots, M$) as well as the expression profiles following the experiments (transcript concentration levels from microarray data $x_j(k)$, $k = 1, \dots, M$). We can then solve the equation (3) for the unknown parameters a_{ij} , and thus obtain the incoming edges per gene.

NIR applies the multiple linear regression method to estimate the unknown model parameters. It relies on the assumptions that the data x are realizations of a normally distributed random variable with known variances and the perturbations, u , are random variables, also normally distributed with known variances. Generally, the response y may be related to k regressors and the

model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k, \quad (4)$$

is termed a multiple linear regression with k regressors.

Having M experiments (response observation points) at our disposal, the model for each gene of the network becomes:

$$y_i = \beta_0 + \sum_{j=1}^k \beta_j x_{ij}, \quad i = 1 \dots M. \quad (5)$$

The response (y) is given by the experimental perturbation values $u_i \in \mathbb{R}^{1 \times M}$, the regression variable values (X) are given by the concentrations of the gene transcripts and the regression variable parameters are given by the components of the vector $\underline{a}_i$, so that, in matrix form, the model becomes:

$$-\underline{u}_i^T = \underline{a}_i^T X, \quad i = 1 \dots N, \quad (6)$$

with $X \in \mathbb{R}^{N \times M}$. The regression analysis aims to best-fit the data by estimating the parameters of the model. NIR estimates the parameters of the regression variables for each gene, using the least squares method. These are the values for which the first derivative of the residual sum square function is zero:

$$\hat{\underline{a}}_i = -\underline{u}_i X^T (X X^T)^{-1}, \quad (7)$$

under the assumption that the regressors are linearly independent.

Biological networks are sparse [9], thus NIR adopts the sparsity assumption that imposes an upper bound on the number of incoming edges per gene (i.e. maximum number of regulators per gene), $restk < N$, which can be chosen by the user.

For each gene the $restk$ parameters that result in the smallest mean square deviation identify the $restk$ incoming edges for that gene. The weight of the identified edges is given by the value of the estimated parameters. The choice of $restk$ affects either the sensitivity to measurement errors or the execution time. A low value of $restk$ induces an increase in the solution sensitivity to measurement errors. A high value prohibitively increases the computational time needed to identify the regulatory network due to the high number of the regressor combinations to be included in the model. This number is equal to the number of combinations without repetitions of N objects taken $restk$ at a time:

$$D_{N,restk} = \frac{N!}{restk!(N-restk)!}. \quad (8)$$

This is polynomial of degree $restk$ in the number of genes. The exhaustive approach which evaluates the regression for each combination is not feasible for gene networks larger than 100 genes (with 100 genes and $restk = 10$ the number of combinations is of the order of 10^{13}), thus NIR uses the following heuristic approach.

For each gene i :

- At the first step NIR computes (7) N times by considering the regression variables one at time; the $topd$ variables for which the sum of the squared deviations is minimized are selected as possible incoming edges for the gene.
- At the second step NIR computes (7) by considering the remaining $N - 1$ variables jointly with each of the first $topd$ selected ones, that is $\frac{topd(2N - topd - 1)}{2}$ (Gauss formula)

pairs of variables; the *topd* pairs of variables for which the sum of the squared deviations is minimized are selected as possible pairs of ingoing edges for the gene.

- At step $k+1$ NIR computes (7) by considering the remaining $N-k$ variables jointly with each of the *topd* sets of k variables selected at the previous step, that is $\frac{\text{topd}(2N-2k-\text{topd}+3)}{2}$ sets of k variables are considered; the *topd* sets of $k+1$ variables for which the sum of the squared deviations is minimized are selected as possible sets of $k+1$ ingoing edges for the gene.
- The process ends when the number of regression variables selected reaches *restk*; the set of *restk* variables for which the sum of the squared deviations is minimized identifies the set of parameters a_i corresponding to the input regulations affecting expression profile of gene i .

The final output is an adjacency matrix, where each element is the edge a_{ij} , that encodes the directed graph. The number of times (7) is calculated for each gene is $O(\text{restk} \cdot \text{topd} \cdot N)$, so the overall number of times (7) is calculated is $O(\text{restk} \cdot \text{topd} \cdot N^2)$. The computational complexity of (7) at step $k+1$, for the submatrix of X whose rows correspond to a set of k variables ($0 \leq k \leq \text{restk}-1$), is $O(k^2 N)$. The overall computational complexity is therefore $O(\text{topd} \cdot \text{restk} \cdot N^3)$.

The NIR parallel version and its implementation

The NIR algorithm can be easily parallelized to handle large problems in a computationally efficient manner by distributing the overall computational burden among different processors to reduce the total execution time. In order to address the high computational cost issue of the NIR algorithm we have applied some specific implementation optimizations along with parallel programming techniques.

The computational core of the NIR algorithm is the equation (7) where X is a submatrix of the gene expression matrix composed only of k rows (with $k=1,2,\dots,\text{restk}$). From the matrix-matrix product definition applied to the submatrix $X(\underline{v}, \underline{v})$, with $\underline{v} = [i_1, \dots, i_k]$ vector of k indices, it follows that:

$$X(\underline{v}, \underline{v})(X(\underline{v}, \underline{v}))^T = (XX^T)(\underline{v}, \underline{v}). \quad (9)$$

Therefore for each step of the algorithm, we don't compute any matrix-matrix product operation $X(\underline{v}, 1:N)(X(\underline{v}, 1:N))^T$. On the contrary the product XX^T is computed once and for all at the beginning of the program. At each step k , our implementation just selects the symmetric submatrix of XX^T whose row and column indices correspond to the k possible ingoing edges for the gene. Let S be this submatrix of dimension k stored in packed format.

In each experiment only one gene is perturbed. This implies that for each gene i the perturbation vector $\underline{u}_i$ is equal to $(0, \dots, 0, 1, 0, \dots, 0)$ and then the product $\underline{u}_i X^T$ reduces to the i -th row of X^T . Denote this row by r .

S is positive definite so we can apply the Cholesky factorization to the matrix S in order to compute $\hat{\underline{a}}_i$ as solution of the system of equations $S \hat{\underline{a}}_i = -r$, thus avoiding the matrix inversion. We rely on the LAPACK [10] routine DPPSV to solve this system of equations with a computational complexity of $O(k^3)$.

By avoiding the matrix product in (7), the parallel algorithm complexity is decreased by one order of magnitude: at the generic step the computational complexity is $O(k^3)$, the overall computational complexity is therefore $O(\text{topd} \cdot \text{restk}^4 \cdot N^2)$.

The parallelization is implemented by assigning different genes to different computing processes: each process takes care of N/p genes where p is the number of processes available. The computing steps described in the previous section can be performed independently for each gene so each process can compute the results for its genes independently without communication. The parallel algorithm has been implemented in C using the MPI standard.

Results

We carried out two kind of tests: to measure the result accuracy of the algorithm and to measure the efficiency in terms of speed-up. In order to measure the result accuracy we ran the program, by using the 'in silico' data generated by [8], on 20 different networks counting 1000 genes, with 10 as average in degree per gene. 'In silico' data are gene expression data generated by a computer model of the gene regulation that enable one to check the performance of algorithms against a perfectly known truth. For each network we generated 1000 experiments perturbing a different single gene at time (local steady-state data).

Table 1. Results of the application of network inference algorithms on the simulated dataset.

Data Sets	ARACNe		BANJO		NIR		Clustering		Random
	PPV	Se	PPV	Se	PPV	Se	PPV	Se	PPV
Local (steady-state)									
10×10	0.53 ^u	0.61 ^u	0.41 ^u	0.50 ^u	0.63 ^u	0.96 ^u	0.39 ^u	0.38 ^u	0.36 ^u
			0.25 ^d	0.18 ^d	0.57 ^d	0.93 ^d			0.20 ^d
			0.15 ^s	0.05 ^s	0.57 ^s	0.93 ^s			0.10 ^s
100×100	0.56 ^u	0.28 ^u	0.71 ^u	0.00 ^u	0.97 ^u	0.87 ^u	0.29 ^u	0.18 ^u	0.19 ^u
			0.42 ^d	0.00 ^d	0.96 ^d	0.86 ^d			0.10 ^d
			0.60 ^s	0.00 ^s	0.96 ^s	0.86 ^s			0.05 ^s
1000×1000	0.66 ^u	0.65 ^u	-	-	0.91^u	0.82^u	0.20^u	0.10^u	0.02^u
					0.84^d	0.84^d			

PPV: Positive Predicted Value (or accuracy) defined as $(TP)/(TP+FP)$, where TP is true positive and FP is false positive; Se: Sensitivity defined as $TP/(TP+FN)$ with FN false negative. ^d: directed graph; ^u: undirected graph. In bold are the results obtained by using our parallel implementation of the NIR algorithm which could not be obtained in [8]. NIR performs significantly better than other software even for the 1000 gene networks.

doi:10.1371/journal.pone.0010179.t001

The program has been executed on 100 processors of an HP XC6000 Cluster with Itanium 2 biprocessors nodes and a Quadrics ELAN 4 network. On average it took 984 seconds to generate the results for each gene network. The program recovered most of the true interactions as shown in Table 1. We compared the results from the simulations obtained by our implementation with the ones obtained by the network inference software reviewed in [8], setting the parameters $restk=11$, $topd=50$. Some software infers the network just as an undirected graph (i.e. the direction of the interaction is not specified, $a_{ij}=a_{ji}$), while NIR generates directed and signed graphs. In order to be able to compare the different types of software, we computed PPV and Se by first transforming the real (signed directed graphs) and inferred networks yielded by our implementation into undirected graphs (labeled u in Table 1). As shown in Table 1, NIR performs better than the other ready-to-use software even for 1000 gene networks.

Moreover we ran the program on a 2500 gene network, even though we didn't have any other software predictions to compare the results with. We set the same values for the parameters $restk$ and $topd$ as before. It took around 12450 seconds to generate the results and we obtained the following values for PPV and Sensitivity: 0.26^d , 0.27^u and 0.10^d , 0.11^u respectively. In order to measure the parallel efficiency we run the program for a 1000 gene network on different number of processors. The execution times and speedups are shown in Table 2. To evaluate the parallel speed-up, we developed a serial version of the algorithm with the implementation optimizations discussed in the previous paragraph.

Discussion

We have designed, developed and tested a parallel version of the NIR algorithm whose purpose is to infer gene regulating networks from microarray gene expression data. Our parallel algorithm reduces the time complexity of the original NIR algorithm by one order of magnitude by avoiding the useless repetition of matrix

Table 2. Total execution times in seconds.

number of procs	time (secs)	speedup
1	98896	-
10	9725	10.1
20	4798	20.6
40	2406	41.1
60	1643	60.2
80	1259	78.6
100	969	102.0

doi:10.1371/journal.pone.0010179.t002

multiplication. The algorithm uses data parallelism, distributing the gene expression data over the available processors. The tests were performed on large networks ($N=2500$) that couldn't be efficiently analyzed by the original serial version. The results confirm the improvements in accuracy, in terms of positive predicted values, and sensitivity, in terms of false negatives, with the respect to the other inferring methods. In addition, the parallel algorithm scales well as the number of processors increases and has a linear speedup.

Acknowledgments

The authors would like to thank Sara Geneletti of Imperial College London for valuable comments on the manuscript.

Author Contributions

Conceived and designed the experiments: FG VB DDB GO. Performed the experiments: FG GO. Analyzed the data: FG VB GO. Contributed reagents/materials/analysis tools: FG VB DDB GO. Wrote the paper: FG VB DDB GO.

References

1. Eisen MB, Spellman PT, Brown PO, Botstein D (1998) Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the United States of America* 95: 14863–14868.
2. Yu J, Smith VA, Wang PP, Hartemink AJ, Jarvis ED (2004) Advances to bayesian network inference for generating causal networks from observational biological data. *Bioinformatics* 20: 3594–3603.
3. Heckerman D (1995) A tutorial on learning with bayesian networks. Technical report, Microsoft Research, Redmond, Washington. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.1431>.
4. Butte AJ, Kohane IS (2000) Mutual information relevance networks: functional genomic clustering using pairwise entropy measurements. *Pac Symp Biocomput*. pp 418–429.
5. Margolin AA, Nemenman I, Basso K, Wiggins C, Stolovitzky G, et al. (2006) ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics* 7 Suppl 1: S7.
6. Bansal M, Belcastro V, Ambesi-Impiombato A, di Bernardo D (2007) How to infer gene networks from expression profiles. *Mol Syst Biol* 3: 78.
7. Gardner TS, di Bernardo D, Lorenz D, Collins JJ (2003) Inferring genetic networks and identifying compound mode of action via expression profiling. *Science* 301: 102–105.
8. Bansal M, Belcastro V, Ambesi-Impiombato A, di Bernardo D (2007) How to infer gene networks from expression profiles. *Mol Syst Biol* 3.
9. Newman MEJ (2003) The structure and function of complex networks. *SIAM Review* 45: 167–256.
10. Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J, et al. (1992) LAPACK's user's guide. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.