Chapter 12
Parallel Computing for Gene Networks Reverse Engineering

Jaroslaw Zola

Abstract Gene networks provide a mathematical representation of gene interactions that govern biological processes in every living organism. Given a gene expression data, the goal of network inference is to reconstruct the underlying regulatory network. The problem is challenging owing to the convoluted nature of biological interactions and imperfection of experimental data. In many cases, the resulting computational models are too complex to execute on a sequential computer and require scalable parallel approaches. In this chapter, we describe network inference methods based on information theory and show a parallel algorithm that enables whole-genome networks reconstruction.

12.1 Introduction

Biological processes in every living organism are governed by complex interactions between thousands of genes, gene products, and other molecules. Genes that are encoded in the DNA are transcribed and translated to form multiple copies of gene products including proteins and various types of RNAs. These gene products coordinate to execute cellular processes or to regulate the expression of other genes depending on the signals carried by, e.g., small molecules. Gene regulatory networks are an attempt to develop a system-level model of these complex interactions, using observations of gene expression.

Gene regulatory networks are typically expressed as graphs with vertices representing genes and edges representing regulatory interactions between genes (see Fig. 12.1). The functioning of a gene regulatory network in an organism determines the expression levels of various genes to help carry out a biological process.

J. Zola (✉)
Rutgers Discovery Informatics Institute, Rutgers University, Piscataway, NJ 08854, USA
e-mail: jaroslaw.zola@rutgers.edu

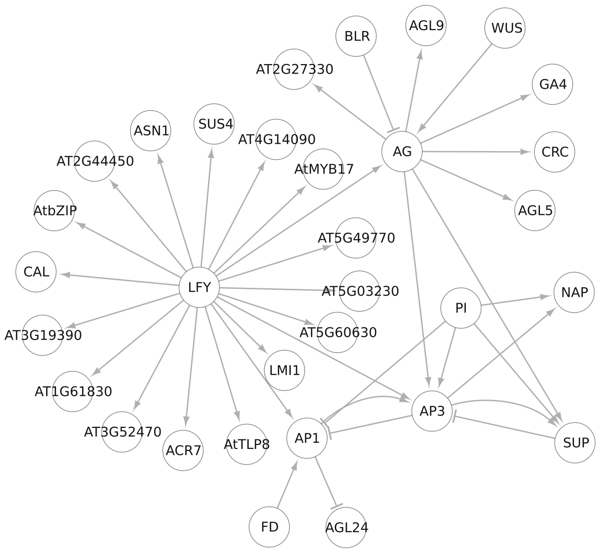

Fig. 12.1 Example gene regulatory network. Here nodes represent genes, T-edges denote regulation in which a source gene inhibits expression of the target gene, and *arrow edges* denote regulation in which a source gene induces expression of the target gene

Network inference, or reverse engineering, is the problem of predicting the underlying network from multiple observations of gene expressions (outputs of the network). To infer a network, one relies on experimental data from high-throughput technologies such as microarrays, or short-read sequencing, which measure a snapshot of all gene expression levels under a particular condition or in a time series. The problem of gene network inference is challenging for several reasons:

- Functioning of any complex organism involves thousands of genes, and usually it is impossible to limit analysis to only a subset of them. In fact, in many cases, the opposite situation takes place – gene networks are used to limit the number of genes that should be target of a biological analysis.
- Despite the rapid progress in high-throughput biotechnology, the number of available expression measurements often falls significantly short of what is required by the underlying computational methods. At the same time, expression data is inherently noisy and significantly influenced by experiment-specific attributes. Consequently in many cases the number of genes in the network significantly outnumbers the number of available expression measurements.
- Finally, our understanding of regulatory mechanisms (e.g., posttranscriptional effects) is still limited, leading to many simplifications in the existing models of regulation.

Due to its importance, gene network inference is an intensely studied problem for which many techniques have been developed. Relevance networks [3, 7], Gaussian graphical models [6, 19], information theory-based methods [2, 9, 25], Bayesian networks [10, 24] and dynamic models [12] are just some examples of the existing approaches. At the same time, two key problems remain with the current methods. The first one is the quality of reconstructed networks. In a recent comprehensive study of 29 network inference methods, Marbach et al. concluded that many do poorly on an absolute basis and 11 do no better than random guessing [15]. The second challenge is computational complexity of the methods and their ability to encompass data from organisms with thousands of genes and large number of expression observations. The computational cost of network inference grows at least as square of the number of the genes and at least linearly with the number of experiments analyzed. Furthermore, statistical methods used to assess significance of the inference, such as bootstrapping, add an extra layer of computational complexity.

To overcome the above limitations and to improve accuracy of network inference while scaling to large expression data, parallel methods for gene networks reverse engineering have been recently proposed [18, 21, 25]. Thanks to the emergence of inexpensive multi and many core processors, and almost ubiquitous adoption of parallel computing, these methods become a solution of choice when large or complex expression data has to be analyzed. More importantly, parallel methods can be used to reconstruct genome-level interactions without sacrificing accuracy, which is where sequential methods fall short.

In this chapter, we focus on application of parallel computing for reverse engineering of gene regulatory networks. First, we explain how the sequential inference process works and then we show how it can be scaled to large distributed memory systems. We base our presentation on information-theoretic methods, a popular class of inference algorithms. Finally, we discuss how to validate inference algorithms *in silico*, and we demonstrate applicability of parallel methods in reconstructing genome-level regulatory networks.

12.2 Network Inference Using Information Theory

In this section, we introduce a more formal statement of the network reconstruction problem, and we present an inference procedure based on information theory. We explain concepts of mutual information and data processing inequality and show how mutual information can be estimated and its significance assessed. We start however with a brief description of a general network inference process.

12.2.1 From Experiment to Network

The goal of network inference is to provide a qualitative, and if possible quantitative, explanation of the observed expression data. The quality of the reconstructed

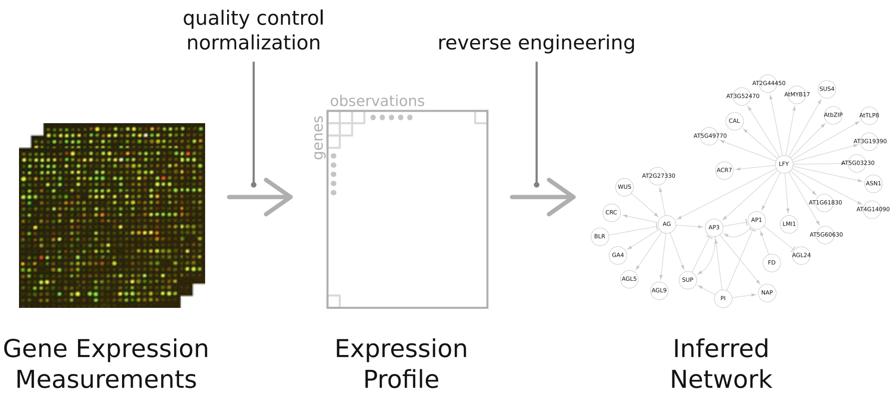

Fig. 12.2 Typical process of reconstructing gene regulatory network

network and its information content are affected not only by the inference method but also by the input data. Gene expression can be measured using several methods, for instance, quantitative PCR, microarrays, or more recently RNA-seq. However, irrespective of which method we select to measure genes expression, three important questions must be answered: first, what should be the set of experimental conditions, how the expression data should be preprocessed to obtain an expression profile suitable for network inference, and finally, which inference method should be used taking into account the two above.

Figure 12.2 illustrates an example inference process. We start with the data acquisition. This step is determined by the underlying scientific hypothesis, which in turn involves careful design of the biological experiment. Note that the data gathered in the experiment can be, and usually is, extended with data deposited in the public repositories, such as Gene Expression Omnibus (GEO) [17]. In general, in this stage, we want to ensure that the collected expression data is sufficient to obtain accurate predictions in the inference process. The following step is to convert the aggregated data into an expression profile. The choice of method depends purely on the experimental platform. For example, processing microarrays will typically require sophisticated signal-calling procedures, followed by filtering and normalization, while RNA-seq will most likely depend on reads mapping and reads counting to obtain a digital expression. This stage is crucial to minimize noise impact, to eliminate low-quality data, and to render different experiments comparable. Only when the expression profile is ready a network can be reconstructed. As we already mentioned, multiple inference methods exist and which method should be used depends on many factors, including size of the expression data and type of queries that the inferred network is meant to answer. In this chapter, however, we consider an information-theoretic approach and its parallel realization.

12.2.2 Problem Formulation

Let us consider the following situation. We performed a set of m experiments, e.g., microarray tests, and obtained expression measurements for n genes. We will represent these genes as a set $G = \{g_1, g_2, \ldots, g_n\}$, where g_i is a gene. Furthermore, we will assume that the expression measurements have been post-processed and normalized accordingly and taking into account properties of the underlying technology. Collectively, we can represent such expression data as a profile matrix $Y_{n \times m}$, where $Y[i, j]$ is an expression of gene g_i in experiment j. The core assumption of virtually all network inference methods is to represent expression of a gene g_i as a random variable $X_i \in \mathcal{X}$, $\mathcal{X} = \{X_1, \ldots, X_n\}$, with marginal probability p_{X_i} derived from some unknown joint probability characterizing the entire system. This random variable is described by a vector of observations $\langle x_{i,1}, \ldots, x_{i,m} \rangle$, where $x_{i,j} = Y[i, j]$. In this form, the network inference problem becomes that of finding a model that best explains the data in Y. The problem can be approached using a variety of methods, including Bayesian networks and Gaussian graphical models. However, one class of methods that have been widely adopted due to their effectiveness uses the concept of mutual information [2, 9, 25]. These methods operate under the assumption that correlation of expression implies co-regulation. Although not always true, the assumption is broadly accepted, especially when analyzing microarray data.

Inference methods based on information theory usually proceed in two phases. First, correlations between pairs of genes are detected. If expression of two genes shows strong correlation, we can assume that they are interacting in the regulatory processes and hence should be connected in the network. Unfortunately, looking solely into pairwise correlations is insufficient to capture more complex regulatory patterns. Consider a scenario where gene g_x regulates gene g_y, which in turn regulates g_z. If we analyze expression of all three genes, it is very likely that g_x and g_z will be significantly correlated, even though they should not be directly connected in the network. To account for such situations, in the second phase, information-theoretic strategies perform additional check to detect and remove indirect interactions.

12.2.2.1 Mutual Information

Let us now focus on how correlation between expression profile of two genes is established. Recall that we represent expression of each gene as a random variable. Although we are given observations of that variable, we do not know its actual distribution. Moreover, the expression observations are delivered from inherently noisy experiments and thus are not perfect. Consequently, to establish whether two expression profiles are correlated, we have to account for potentially complex, e.g., nonlinear, patterns of correlations. This can be achieved using the concept of mutual information [4].

Mutual information is arguably the best measure of correlation between two random variables. It is defined based on the entropy in the following way:

$$\mathcal{I}(X_i; X_j) = \mathcal{H}(X_i) + \mathcal{H}(X_j) - \mathcal{H}(X_i, X_j), \qquad (12.1)$$

where entropy $\mathcal{H}$ is given as

$$\mathcal{H}(X) = -\sum p_X(x) \log(p_X(x)), \qquad (12.2)$$

p_X defines the probability distribution of X, and $\sum$ is replaced by integral if X is continuous. Intuitively, mutual information $\mathcal{I}(X_i; X_j)$ quantifies information that both variables provide about each other. If two variables are correlated, then their joint entropy is smaller than the sum of their individual entropies, and hence greater is their mutual information. Note that from Eqs. (12.1) and (12.2), we can write mutual information as

$$\mathcal{I}(X_i; Y_i) = \sum_{x_i} \sum_{x_j} p_{X_i X_j}(x_i, x_j) \log\left(\frac{p_{X_i X_j}(x_i, x_j)}{p_{X_i}(x_i) p_{X_j}(x_j)}\right), \qquad (12.3)$$

which is equivalent of the Kullback-Leibler divergence between distribution of X_i and X_j when both variables are dependent (i.e., $p_{X_i X_j}$) and when they are independent (i.e., $p_{X_i X_j} = p_{X_i} p_{X_j}$).

Mutual information is a symmetric, nonnegative function and is equal to zero if and only if two random variables are independent. Consequently, to connect two genes in the reconstructed network, we have to check if mutual information between their expression profiles is greater than zero.

12.2.2.2 Data Processing Inequality

Having defined a correlation measure, we are left with the task of identifying indirect interactions between genes. One popular approach to address this problem, which first has been introduced in the ARACNe method [2], is to rely on the data processing inequality principle, or DPI for short. Briefly, DPI states that if three random variables X_i, X_j, X_k form a Markov chain in that order (i.e., conditional probability of X_k depends only on X_j and is independent of X_i), then $\mathcal{I}(X_i; X_k) \leq \mathcal{I}(X_i; X_j)$, which implies also that $\mathcal{I}(X_i; X_k) \leq \mathcal{I}(X_j; X_k)$. In other words, X_k cannot provide more information about X_i than X_j provides about X_i. The DPI reasoning can be used to detect indirect interactions between genes: each time the pair (X_i, X_k) satisfies both inequalities, the corresponding connection between genes g_i and g_k can be removed from the network. Note that the above procedure is based on the assumption that DPI implies independence of X_i and X_k given X_j. This is not always true: in some situations the inequalities may hold even though

X_i and X_k are dependent (consider for example, binary variables, where X_i and X_j are uniform and X_k is a XOR function of X_i and X_j). Nevertheless, DPI has been shown to perform very well in practice.

12.2.2.3 Inference Algorithm

So far we defined two information-theoretic concepts that can be used to infer gene regulatory networks – mutual information and DPI. Algorithm 12.1 shows how the two are combined into a working solution.

We represent the network using adjacency matrix D. Although gene regulatory networks are usually very sparse, initially we have to compute mutual information between all $\binom{n}{2}$ pairs of genes (line 1). Then, we remove edges between genes that are not significantly correlated (lines 2–4) and proceed with the DPI phase (lines 5–9).

Algorithm 12.1 Network inference using information theory

```
Input: Expression profile $Y_{n\times m}$, mutual information threshold $\mathcal{I}^0$
Output: Adjacency matrix $D_{n\times n}$
1: $D[i,j] = $ Estimate $\mathcal{I}(X_i;X_j)$ from $(Y[i,\cdot], Y[j,\cdot])$
2: if $D[i,j] < \mathcal{I}^0$ then
3:     $D[i,j] = 0$
4: end if
5: for all $(i,k)$ do
6:     if $\exists j$ s.t. $D[i,k] \le D[i,j]$ and $D[i,k] \le D[j,k]$ then
7:         $D[i,k] = 0$
8:     end if
9: end for
```

The main component of the algorithm is estimation of mutual information from the expression data. Observe that although in Eq. (12.3) we express mutual information through probability distributions, we do not know the distribution that governs gene expression. Consequently, we have to estimate mutual information from observations provided by the expression profile matrix Y. Fortunately, because mutual information is a widely used concept, there are several estimators available. Here, we will describe the B-spline-based estimator that has been proposed by Daub et al. for analyzing expression data [5].

The estimator works by discretizing observations into b categories, but with the assumption that given observation can be assigned simultaneously to k categories with different weights. The weights are obtained using B-spline functions of order k, defined over b uniformly spaced knot points. Note that knot points define bins (categories) to which each continuous observation can be assigned. Let $\mathcal{B}_k^b$ be a B-spline function of order k defined over b knot points. For a continuous observation, this function returns a vector of size b with k nonnegative weights that indicate to which bins the observation should be assigned. Given two random variables X_i and X_j with m observations, we can discretize them into variables A and B with probabilities:

$$p_A = \frac{1}{m} \sum_{l=1}^{m} \left(\mathcal{B}_k^b(x_{i,l}) \right), \tag{12.4}$$

and

$$p_{AB} = \frac{1}{m} \sum_{l=1}^{m} \left(\mathcal{B}_k^b(x_{i,l}) \times \mathcal{B}_k^b(x_{j,l}) \right), \tag{12.5}$$

where in our case $x_{i,j} = Y[i, j]$. By plugging the resulting probabilities directly into entropy calculations, we can compute marginal and joint entropy for A and B and then approximate $\mathcal{I}(X_i; X_j) \approx \mathcal{I}(A; B)$. A nice property of the B-spline estimator is that it can be very efficiently implemented, and its complexity is of order $O(m)$.

The last element of the inference algorithm is the choice of the threshold value $\mathcal{I}^0$ to decide when correlation is significant. Recall that two random variables are independent only if their mutual information is equal zero. However, because we are estimating mutual information, it would be unrealistic to expect precise results. Therefore, it is a common practice to assume that mutual information lower than the carefully chosen cutoff $\mathcal{I}^0$ implies independence. There are different ways $\mathcal{I}^0$ can be selected, and we describe one particular solution next.

12.2.2.4 Testing Significance of Mutual Information

As we already mentioned, using mutual information requires deciding when its estimate implies independence. This can be regarded as assessing statistical significance of the quantity $\mathcal{I}(X_i; X_j)$ itself. This assessment can be done through permutation testing.

Let $\pi(X_i) = \pi(\langle x_{i,1}, x_{i,2}, \ldots, x_{i,m} \rangle)$ denote a permutation of the vector of m observations of X_i. If there exists dependency between X_i and X_j, it is expected that $\mathcal{I}(X_i; X_j)$ is significantly higher than $\mathcal{I}(\pi(X_i); X_j)$. The permutation testing method involves computing $\mathcal{I}(\pi(X_i); X_j)$ for all $m!$ permutations of $\langle x_{i,1}, x_{i,2}, \ldots, x_{i,m} \rangle$ and accepting the dependency between X_i and X_j to be statistically significant only if $\mathcal{I}(X_i; X_j) > \mathcal{I}(\pi(X_i); X_j)$ for at least a fraction $(1 - \epsilon)$ of the $m!$ permutations tested, for some small constant $\epsilon > 0$. As testing all $m!$ permutations is computationally prohibitive for large m, a large sampling of the permutation space is considered adequate in practice.

Ideally, permutation testing should be conducted for assessing the significance of each pair $\mathcal{I}(X_i; X_j)$ using a large number of random permutations. Clearly, this is computationally prohibitive. However, we proposed a simple solution to overcome this limitation, by using only a few random permutations per pair, while collectively obtaining statistically meaningful results for all pairs [25].

Mutual information has the property of being invariant under homeomorphic transformations:

$$\mathcal{I}(X_i; X_j) = \mathcal{I}(f(X_i); h(X_j)), \tag{12.6}$$

for any homeomorphisms f and h. Consider replacing the vector of observations for X_i, i.e., $\langle x_{i,1}, x_{i,2}, \ldots, x_{i,m} \rangle$ with the vector $\langle \text{rank}(x_{i,1}), \text{rank}(x_{i,2}), \ldots, \text{rank}(x_{i,m}) \rangle$, where $\text{rank}(x_{i,l})$ denotes the rank of $x_{i,l}$ in the set $\{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\}$; i.e., we replace each observation with its rank in the set of all observations. The transformation, which is termed *rank transformation*, while not continuous, is considered a good approximation to homeomorphism [13]. In our case, instead of computing mutual information of pairs of gene expression vectors directly, we equivalently compute the mutual information of their rank transformed counterparts. With this change, each observation vector is now a permutation of $\{1, 2, \ldots, m\}$. Therefore, a permutation $\pi(X_i)$ corresponds to some permutation of the observation vector of any other random variable X_j. More formally, consider applying permutation testing to a specific pair (X_i, X_j) by computing $\mathcal{I}(\pi(X_i); X_j)$ for some randomly chosen permutation π. For any other pair (X_k, X_l), $\exists \pi', \pi''$ such that $\pi(X_l) = \pi'(X_j)$ and $\pi(X_i) = \pi''(\pi'(X_k))$. Since π is a random permutation, so is π'' and $\mathcal{I}(\pi(X_i); X_j)$ is a valid permutation test for assessing the statistical significance of $\mathcal{I}(X_k; X_l)$ as well. Thus, each permutation test is a valid test for all $\binom{n}{2}$ pairs of observations.

Using the above procedure, we can easily find the threshold value $\mathcal{I}^0$. When estimating mutual information for each rank transformed pair (X_i, X_j), we perform additional q permutation tests. Then, $\mathcal{I}^0$ is the rth largest value among all values generated by the permutation test, where $r = (1 - \epsilon) \cdot q \cdot \binom{n}{2}$ and ϵ is a small constant describing the significance level.

12.3 Parallel Method for Networks Inference

In the previous section, we explained how concepts from information theory and statistics can be used for reverse engineering gene regulatory networks. Although the resulting method is considered to be very accurate, it becomes limited for reconstructing genome-level networks with thousands of genes. This is because the whole-genome gene network reconstruction is both compute and memory intensive. Memory consumption arises from the $\Theta(nm)$ size of input data and from the $\Theta(n^2)$ dense initial network generated in the first phase of the reconstruction algorithm. Computational cost is dominated by the $O(n^2)$ computations of mutual information, where the complexity of a mutual information estimator is at least $O(m)$, but it can be $O(m^2)$ for, e.g., Gaussian kernel estimator. As a result, large-scale network construction is out of the scope of sequential methods, and scalable parallel approach becomes necessary.

Fig. 12.3 An example of partitioning of matrix D for eight processors. For each block, the iteration number in which it will be processed is marked

P_0	0	1	2	3	4			
P_1		0	1	2	3	4		
P_2			0	1	2	3	4	
P_3				0	1	2	3	4
P_4	4				0	1	2	3
P_5	3	4				0	1	2
P_6	2	3	4				0	1
P_7	1	2	3	4				0

Let p denote the number of processors in a parallel computer. Recall that we represent the network using the standard adjacency matrix $D_{n \times n}$. To store the matrix in the distributed memory, we use row-wise data distribution where each processor stores up to $\lceil \frac{n}{p} \rceil$ consecutive rows of the expression profile Y and the same number of rows of the corresponding gene network adjacency matrix D. To begin, each processor reads and parses its block of input data and then applies rank transformation. The algorithm proceeds in three phases: in the first phase, mutual information is computed for each of the $\binom{n}{2}$ pairs of genes and q randomly chosen permutations per pair. Note that the total number of permutations used in the test is $Q = q \cdot \binom{n}{2}$, allowing a small constant value of q for large n. In the second phase, the threshold value $\mathcal{I}^0$ is computed. In the final phase, indirect interactions are detected using DPI and removed.

12.3.1 Computing Mutual Information

Without loss of generality, we can assume that n is a multiple of p. To compute mutual information between all pairs of genes, we first partition D into $p \times p$ blocks of submatrices $D_{i,j}$ $(0 \leq i, j < p)$, each of $\frac{n}{p} \times \frac{n}{p}$ size. Then we proceed in $\lceil \frac{p+1}{2} \rceil$ iterations. In each iteration, a processor is assigned a submatrix. Its task is to compute mutual information for each position in the submatrix, along with mutual information of q random permutations for each position. Observe that to do so, it requires the expression profile vectors of all genes representing rows or columns in the submatrix. For blocks on the main diagonal, the same genes represent both rows and columns. For other blocks, the row genes and column genes are distinct. Because the matrix D is symmetric, we need to compute only half of it, i.e., as $D_{i,j} = D^T_{j,i}$ only one of them needs to be computed. We call a set of $\frac{p \cdot (p+1)}{2}$ submatrices containing only one of $D_{i,j}$ or $D_{j,i}$ for each pair (i, j), to be the complete set of unique submatrices. The assignment of submatrices to processors is as follows: in iteration i, processor with rank j is assigned the submatrix $D_{j,(j+i) \bmod p}$ (see Fig. 12.3 for an illustration). It is easy to argue that this scheme computes all unique submatrices.

The assignment of submatrices to processors creates the same workload with the following exceptions: in iteration 0, the submatrices assigned are diagonal, for which we only need the lower (or upper) triangular part. As all processors are dealing with diagonal submatrices in the same iteration, it simply means that this iteration will take roughly half the compute time as others. The other exception may occur during the last iteration. To see this, consider that the processors collectively compute p submatrices in each iteration. The total number of unique submatrices is $\frac{p\cdot(p+1)}{2}$. The following two cases are possible:

1. p is odd. In this case, the number of iterations is $\lceil\frac{p+1}{2}\rceil = \frac{p+1}{2}$. The total number of submatrices computed is $\frac{p\cdot(p+1)}{2}$, which is the same as the total number of unique submatrices. Since the algorithm guarantees that all unique submatrices are computed, each unique submatrix is computed only once.
2. p is even. In this case, the number of iterations is $\lceil\frac{p+1}{2}\rceil = \frac{p}{2}+1$, causing the total number of submatrices computed to be $p\cdot\left(\frac{p}{2}+1\right)$, which is $\frac{p}{2}$ more than the number of unique submatrices. It is easy to show that this occurs because in the last iteration, half the processors are assigned submatrices that are transpose counterparts of the submatrices assigned to the other half (marked with darker color in Fig. 12.3).

When p is even, we can optimize the computational cost by recognizing this exception during the last iteration and having each processor compute only half of the submatrix assigned to it, so that the processor which has the transpose counterpart computes the other half. Note that this will save half an iteration, significant only if p is small. For large p, we can ignore this cost and run the last iteration similar to others.

Let us now compute the parallel runtime of the above method assuming a simple point-to-point communication model with latency τ and bandwidth $\frac{1}{\mu}$ (adequate for most distributed memory parallel systems). Under this model, the first phase takes $O\left(\frac{qn^2m}{p}\right)$ compute time and $O\left(p\tau + \mu nm\right)$ time for communication. Thus, we can scale p linearly with n while maintaining parallel compute time as the dominant factor in runtime.

12.3.2 Computing $\mathcal{I}^0$

Having computed the adjacency matrix D, we have to now remove edges that are not statistically significant, i.e., their mutual information is lower than $\mathcal{I}^0$. Recall that the threshold is computed by finding the element with rank $r = (1-\epsilon)\cdot q\cdot\binom{n}{2}$ among the $q\cdot\binom{n}{2}$ mutual information values computed as part of permutation testing. As each processor stores at most $\frac{qn^2}{p}$ results of the permutation test, we can find the threshold using a parallel selection algorithm. However, ϵ is a very small positive constant close to zero, and hence, the threshold value is close to the largest value in the sorted order of the computed mutual information values. Hence, we can proceed as follows: each processor sorts its $\Theta\left(\frac{qn^2}{p}\right)$ values and selects the r largest values. Then, a parallel reduction operation is applied using the r largest values in each processor as input. The reduction operator performs linear merging of two samples of size r and retains the r largest elements. Once the rth globally largest value is found and broadcast, each processor eliminates edges from its local adjacency matrix that are below the threshold. This phase takes $O\left(\frac{qn^2 \log n}{p} + r \log p\right)$ parallel compute time and $O\left((\tau + \mu r) \log p\right)$ parallel communication time. Assuming $\epsilon < \frac{1}{p}$, we can expect linear scaling.

12.3.3 Removing Indirect Interactions

In the final phase of the algorithm, we have to apply DPI to prune indirect interactions. To decide if a given edge $D[i,j]$ should be removed, we have to compare it with all values $D[i,k]$ and $D[j,k]$. Consequently, complete information about rows i and j is needed. Because matrix D is stored row-wise, we have to stream row j to the processor responsible for row i. Moreover, because matrix D is symmetric, it is sufficient to analyze its upper (or lower) triangular part. We can achieve this in $p - 1$ communication rounds, where in round i only processors with ranks $0, 1, \ldots, p - i$ participate in communication and processing. The parallel runtime of this phase is $O\left(\frac{n^3}{p} + \tau p + \mu n^2\right)$. While this worst case analysis indicates this to be the most compute-intensive phase of the algorithm, it is not so. This is because DPI requires no significant computation and just single memory write, and it needs to be applied only to current existing edges in the network, and the network is expected to be significantly sparse.

12.3.4 Testing Parallel Scalability

The parallel approach described in the previous sections has been implemented in the software package TINGe [1,25]. TINGe is a C++ software based on the Message Passing Interface (MPI) that can be executed on large distributed memory machines. It implements several low-level optimizations to exploit SIMD instructions of modern processors during mutual information computations and uses MPI I/O routines to handle large input and output data. TINGe employs the B-spline-based mutual information estimator.

To demonstrate scalability of the parallel inference method, we executed TINGe on the IBM Blue Gene/L system with $p = 1{,}024$ processors and analyzed four different expression profiles with varying number of genes ($n = 2{,}048$ and $n = 4{,}096$) and varying number of observations ($m = 911$ and $m = 2{,}996$). We tested

Table 12.1 TINGe runtime in seconds for different number of genes n and different number of expression observations m

	$m = 911$		$m = 2{,}996$	
p	$n = 2{,}048$	$n = 4{,}096$	$n = 2{,}048$	$n = 4{,}096$
32	382	1,525	1,489	5,932
64	193	766	752	2,986
128	98	385	378	1,495
256	50	196	193	762
512	27	101	101	386
1,024	17	55	56	203

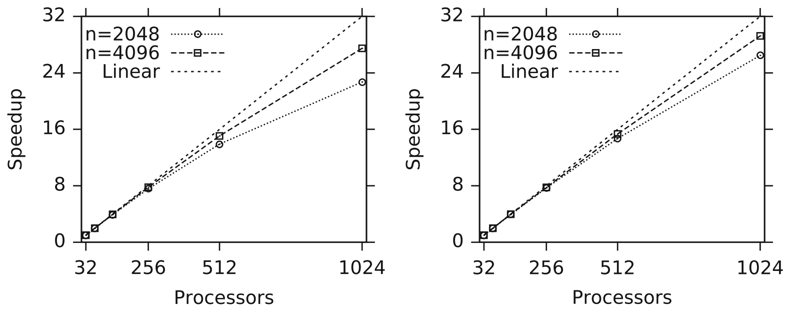

Fig. 12.4 Relative speedup of TINGe as a function of number of processors for the data sets with 911 observations (*left*) and 2,996 observations (*right*)

how runtime changes with the number of processors and what is the relative speedup of the software. Results are summarized in Table 12.1 and Fig. 12.4.

Note that scalability is a crucial characteristic of any parallel software. If a parallel application is scalable, we can decrease its runtime proportionally by executing it on the larger number of processors, and we can solve larger problems. On the other hand, software that scales poorly is of little use as it does not benefit from the parallel hardware.

As we can see, TINGe maintains almost linear scalability up to 1,024 processors, that is, with the increasing number of processors, its runtime decreases linearly. The runtime grows as square function of the number of genes n and linearly with the number of observations m. This is what we expected taking into account the $O(m)$ complexity of the B-spline-based mutual information estimator and the fact that computations are dominated by the first phase, i.e., computing mutual information between all pairs of genes.

12.4 Example Applications

So far we described a parallel information theory-based method for gene network inference. We also demonstrated that the method scales very well on distributed memory parallel systems and hence can be used to process large expression data.

One remaining question that has to be addressed is the accuracy and applicability of the method. In this section, we show how to assess quality of network inference methods. Then, we analyze performance of TINGe, and we explain how it has been used to reconstruct a genome-level regulatory network of the model plant *Arabidopsis thaliana*.

12.4.1 Assessing Quality of Network Inference Methods

Computational methods for regulatory networks reverse engineering are necessarily error-prone, owing to simplifications in the underlying models. This is not surprising taking into account our limited understanding of regulatory processes. When designing a new inference method, we would like to meet three main quality criteria: sensitivity, specificity, and precision. Let TP denote the number of true positives, i.e., the number of correctly predicted gene interactions (network edges); FP be the number of false positives, i.e., incorrectly predicted edges; TN be the number of correctly avoided edges; and, finally, FN be the number of incorrectly avoided interactions. Then, sensitivity relates to the ability of the method to identify positive results Sensitivity $= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$, and it is a fraction of correct interactions predicted. Likewise, specificity relates to the ability of the method to identify negative results Specificity $= \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$, and it is a fraction of missing edges correctly classified. Finally, precision describes predictive power of the method Precision $= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$.

Having defined quality criteria, the question is how can we assess performance of a particular method. Naturally, performing a biological experiment to confirm predictions in the inferred network is the most desired approach. However, in most cases, it is infeasible because of the cost and technical limitations (not all interactions can be easily validated). To address this challenge, several researchers proposed methods to perform quality assessment using a synthetic data [12, 14, 20, 23]. The basic idea of the approach is illustrated in Fig. 12.5.

The process starts with two input components – some known network structure and a model of expression dynamics, which usually involves a set of differential equations describing how regulators affect expression of the target gene [23]. The input network is sampled to obtain a benchmark synthetic network. By combining the synthetic network and the expression model, we can generate a synthetic gene expression profile that next can be used as an input to the tested inference method. Observe that the above process guarantees that we know both expression data and the network from which this data has been derived. Consequently, we can easily assess the quality of the inference method by comparing our prediction with the synthetic network. This approach is very flexible – different models of regulation and different sampling strategies can be used to generate synthetic data, and hence to capture various properties of the real-life biological systems. There are several tools that implement such strategy (see, for example, GNW [20], SynTReN [23],

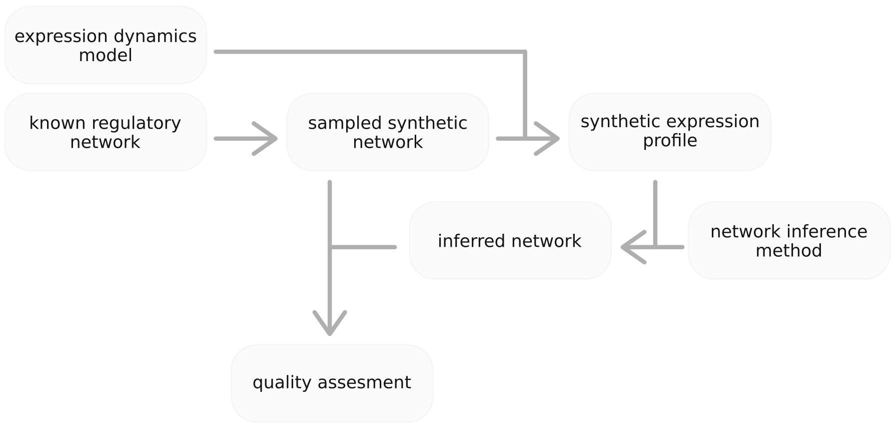

Fig. 12.5 Quality assessment of a network inference method using synthetic data

Table 12.2 Quality of ARACNe and TINGe using a 250-gene synthetic network with two different sets of expression profiles

	$m = 500$		$m = 900$	
	ARACNe	TINGe	ARACNe	TINGe
Time (s)	473	24	1,866	40
Specificity	0.99	0.99	0.99	0.99
Sensitivity	0.42	0.42	0.44	0.44
Precision	0.52	0.52	0.56	0.57
TP	181	180	190	187
TN	30,535	30,538	30,553	30,562
FP	166	163	148	139
FN	243	244	234	237

or COPASI [12]). These tools provide a functionality to create benchmark data sets with desired number of genes and observations and can be readily used to assess quality of inference methods.

We used SynTReN to assess quality of two mutual information-based inference methods: ARACNe and TINGe. Both tools use the same underlying inference algorithm; however, ARACNe uses different methods to estimate mutual information and to establish the threshold $\mathcal{I}^0$. We generated two synthetic regulatory networks, each consisting of $n = 250$ genes, but differing in the number of expression observations ($m = 500$ and $m = 900$, respectively). Table 12.2 shows that both methods preserve very good precision and sensitivity, while TINGe outperforms ARACNe in terms of the runtime. Increasing the number of observations improves performance of both methods, which is expected as we gain more information with more observations.

While synthetic data provides a convenient way to assess quality of inference methods, we should keep in mind that it is not an ultimate quality indicator. This is because the way synthetic data is generated is model dependent, and hence, it is subject to similar limitations as inference methods. Nevertheless, if a given inference method performs well when tested with synthetic data, it is very likely that it will perform well in practice. On the contrary, methods that perform poorly will fail when analyzing real-life data.

12.4.2 Reconstructing Whole-Genome Network of Arabidopsis

Arabidopsis thaliana is the model plant to study plants' biology and hence is of great practical importance. Its genome contains estimated 27,000 genes, and hence, constructing genome-level regulatory network becomes challenging both computationally and in terms of assembling a sufficiently reach expression profile. Consequently, reconstructing *Arabidopsis* network demonstrates applicability and necessity of parallel inference methods.

We used TINGe to reconstruct the gene regulatory network of *Arabidopsis* [1]. We started reconstruction by obtaining a total of 3,546 nonredundant Affymetrix ATH1 microarray observations, grouped into 197 experiments. Here, each experiment contained several gene expression measurements related to the same biological process or condition. The data was aggregated from the main *Arabidopsis* repositories at NASC [16], GEO [17], ArrayExpress [8], and AtGenExpress [22]. It covers different plant development stages and various treatment experiments, and collectively it provides a broad overview of expression profiles in *Arabidopsis*.

To accommodate for the variability in this highly diverse collection, we developed the following pipeline to obtain the final expression profile. We first removed microarrays which did not pass a rigorous quality control (e.g., exhibited problems, with RNA hybridization). For this we depended on several existing quality indicators offered by the Affymetrix platform. The screening process returned 3,137 microarrays that were subject to normalization: we transformed expression measures into $\log_2$ space and changed to $Y[i,j] = S[i,j] - \overline{S}_i$, where $S[i,j]$ represents $\log_2$-transformed expression of gene i in observation j and $\overline{S}_i$ is the average expression of gene i across all the microarray chips in the experiment containing chip j. Finally, the resulting expression was quantile normalized, and to guarantee that the expression profile of every gene covers a wide range of expression levels, expression profiles with interquartile range of expression lower than 0.65 were removed. As a result, we obtained the final expression matrix with $m = 3{,}137$ observations and $n = 15{,}495$ genes.

Using this data, TINGe constructed a whole-genome network in 30 min on the IBM Blue Gene/L with $p = 2{,}048$ processors. I/O operations took 1 min, finding threshold value $\mathcal{I}^0$ required 1 s, and application of DPI ran in 16 s. Analysis of this network enabled several important insights into biological processes in plants, for instance, the carotenoid biosynthesis. More importantly, this experiment demonstrates that thanks to application of parallel computing, mutual information methods can be used to reconstruct genome-level regulatory networks.

12.5 Final Remarks

The problem of gene regulatory networks inference is one of many in the broad area of computational systems biology. In this chapter, we covered information-theoretic approach to the network inference, together with its scalable parallel implementation. We also demonstrated how application of parallel computing can be used to reconstruct some of the largest gene regulatory networks. Recently, several other parallel reverse engineering methods have been proposed [11, 18, 21]. These methods use different criteria to model gene interactions, e.g. based on Bayesian networks, or different approaches to parallelization, e.g., with GPU accelerators. In spite of that parallel processing only recently attracted attention of systems biology researchers. Together with the rapid progress in high-throughput biological technologies, we can expect accumulation of massive and diverse expression data, which will enable more complex and realistic models of regulation. Clearly, these models will require large computational power offered by parallel systems.

References

1. Aluru M, Zola J, Nettleton D et al (2013) Reverse engineering and analysis of large genome-scale gene networks. Nucl Acids Res 41(1):e24
2. Basso K, Margolin A, Stolovitzky G et al (2005) Reverse engineering of regulatory networks in human B cells. Nat Genet 37(4):382–390
3. Butte AJ, Kohane IS (1999) Unsupervised knowledge discovery in medical databases using relevance networks. In: Proceedings of the American medical informatics association symposium, Washington, DC, pp. 711–715
4. Cover TM, Thomas JA (2006) Elements of information theory, 2nd edn. Wiley, Hoboken
5. Daub CO, Steuer R, Selbig J et al (2004) Estimating mutual information using B-spline functions – an improved similarity measure for analysing gene expression data. BMC Bioinform 5:118
6. de la Fuente A, Bing N, Hoeschele I et al (2004) Discovery of meaningful associations in genomic data using partial correlation coefficients. Bioinformatics 20(18):3565–3574
7. D'haeseleer P, Wen X, Fuhrman S et al (1998) Mining the gene expression matrix: inferring gene relationships from large scale gene expression data. In: Information processing in cells and tissues. Plenum Press, New York
8. EMBL-EBI ArrayExpress. http://www.ebi.ac.uk/microarray-as/aer/
9. Faith JJ, Hayete B, Thaden JT et al (2007) Large-scale mapping and validation of Escherichia coli transcriptional regulation from a compendium of expression profiles. PLoS Biol 5(1):e8
10. Friedman N, Linial M, Nachman I et al (2000) Using Bayesian networks to analyze expression data. J Comput Biol 7:601–620
11. Gregoretti F, Belcastro V, di Bernardo D et al (2010) A parallel implementation of the network identification by multiple regression (NIR) algorithm to reverse-engineer regulatory gene networks. PLoS One 5(4):e10179
12. Hoops S, Sahle S, Gauges R, et al (2006) COPASI – a complex pathway simulator. Bioinformatics 22(24):3067–3074
13. Kraskov A, Stogbauer H, Grassberger P (2004) Estimating mutual information. Phys Rev E 69(6 Pt 2):066138

14. Long J, Roth M (2008) Synthetic microarray data generation with RANGE and NEMO. Bioinformatics 24(1):132–134
15. Marbach D, Prill RJ, Schaffter T et al (2010) Revealing strengths and weaknesses of methods for gene network inference. PNAS 107(14):6286–6291
16. NASC European Arabidopsis Stock Centre. http://www.arabidopsis.info/
17. NCBI Gene Expression Omnibus. http://www.ncbi.nlm.nih.gov/geo/
18. Nikolova O, Zola J, Aluru S (2013) Parallel globally optimal structure learning of Bayesian networks. J Parallel Distrib Comput 73(8):1039–1048. ISSN 0743-7315, http://dx.doi.org/10.1016/j.jpdc.2013.04.001
19. Schafer J, Strimmer K (2005) An empirical Bayes approach to inferring large-scale gene association networks. Bioinformatics 21(6):754–764
20. Schaffter T, Marbach D, Floreano D (2011) GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods. Bioinformatics 27(16):2263–2270
21. Shi H, Schmidt B, Liu W et al (2011) Parallel mutual information estimation for inferring gene regulatory networks on GPUs. BMC Res Notes 4:189
22. TAIR. http://www.arabidopsis.org/
23. van den Bulcke T, Van Leemput K, Naudts B et al (2006) SynTReN: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. BMC Bioinform 7:43
24. Yu H, Smith A, Wang P et al (2002) Using Bayesian network inference algorithms to recover molecular genetic regulatory networks. In: Proceedings of the international conference on systems biology, Edmonton
25. Zola J, Aluru M, Sarje A et al (2010) Parallel information-theory-based construction of genome-wide gene regulatory networks. IEEE Trans Parall Distrib Syst 21(12):1721–1733