JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 17, Number 6, 2010

Pp. 853–868
DOI: 10.1089/cmb.2008.0023

Discretization of Time Series Data

ELENA S. DIMITROVA,[1] M. PAOLA VERA LICONA,[2,3,4]
JOHN McGEE,[5] and REINHARD LAUBENBACHER[6]

ABSTRACT

An increasing number of algorithms for biochemical network inference from experimental data require discrete data as input. For example, dynamic Bayesian network methods and methods that use the framework of finite dynamical systems, such as Boolean networks, all take discrete input. Experimental data, however, are typically continuous and represented by computer floating point numbers. The translation from continuous to discrete data is crucial in preserving the variable dependencies and thus has a significant impact on the performance of the network inference algorithms. We compare the performance of two such algorithms that use discrete data using several different discretization algorithms. One of the inference methods uses a dynamic Bayesian network framework, the other—a time-and state-discrete dynamical system framework. The discretization algorithms are quantile, interval discretization, and a new algorithm introduced in this article, SSD. SSD is especially designed for short time series data and is capable of determining the optimal number of discretization states. The experiments show that both inference methods perform better with SSD than with the other methods. In addition, SSD is demonstrated to preserve the dynamic features of the time series, as well as to be robust to noise in the experimental data. A *C*++ implementation of SSD is available from the authors at http://polymath.vbi.vt.edu/discretization.

Key words: gene networks, genetic algorithms, linear algebra, reverse engineering, time discrete dynamical systems.

1. INTRODUCTION

THE DEVELOPMENT OF ALGORITHMS to infer biochemical networks from system-level data, such as DNA microarray or 2D-gel data, is a central problem of computational systems biology. Several such algorithms have been proposed in recent years, using tools from a variety of fields. Many of these methods are statistical in nature, resulting in a ''dependency graph'' for the network. Depending on the method used, the edges represent either a statistical correlation of two variables (de la Fuente, 2004) or a causal relationship

[1]Department of Mathematical Sciences, Clemson University, Clemson, South Carolina.
[2]Institut Curie, 26 rue d'Ulm, Paris, F-75248 France.
[3]INSERM, 11900, Paris, F-75248 France.
[4]Hines Paris Tech, Fontainebleau, F-77300 France.
[5]Mathematics and Statistics Department, Radford University, Radford, Virginia.
[6]Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, Virginia.

(Friedman, 2000). One commonly used modeling framework is that of dynamic Bayesian networks (Pournara, 2004; Yu, 2004; Beal, 2005; Nariai, 2005; Zou, 2005; Dojer, 2006). Toward the other end of the spectrum one finds methods that use a dynamical systems modeling framework, such as systems of differential equations (Gardner, 2003; Andrec, 2005; Kim, 2007), Boolean networks (Liang, 1998; Akutsu, 2000; Mehra, 2004; Martin, 2007), or multi-state discrete models (Laubenbacher, 2004; Jarrah, 2005). The different methods have different data requirements.

We focus here on methods that require discretized data as input, that is, experimental data that have been discretized into a finite number of discrete states. This is a requirement for almost all Bayesian network methods, as well as discrete multi-state model-based methods, such as Boolean networks. Relatively little systematic work has been done on discretization methods for this purpose (Di Camillo, 2005; Ernst, 2006), and such work assumes larger data sets than those available in reality and/or knowledge of the data distribution also rarely available. Di Camillo et al. (2005), for instance, require knowledge of the distribution of the error $(x(t) - x_b)$, where $x(t)$ is the measurement and x_b is its basal value, and suggest using experimental replicates for finding it. Such a requirement makes their method inapplicable when no such experiments are possible due to their prohibitive cost or the particular experimental design. There is also a large selection of discretization methods which cluster data points but many of them are not directly applicable in the network inference context or are not suitable. One important limitation is typically that the number of available data points is very small, e.g., DNA microarray measurements. Also, methods that provide as output a dynamic model rely on appropriate methods for the discretization of time series data.

We argue in this paper that appropriate data discretization is crucial for the optimal performance of network inference algorithms and deserves to be studied more extensively. To this aim we present the results of a comparison of different discretization methods. We compare the performance of two different network inference methods on the same data set using several different discretization algorithms. The two inference methods are the dynamic Bayesian network algorithm (Bernard, 2005) using the software BANJO (Hartemink, 2006) and the polynomial dynamical system method developed in Jarrah (2005). We apply both methods with the discretization algorithms available in BANJO, namely interval, as well as quantile discretization into 2, 3, 4, and 5 states. Similarly, we evaluate the performance of both inference methods using a new discretization algorithm described in this paper. It has been developed specifically for short time course data, with the goal of obtaining dynamic network models. In order to assess the effect of the different discretization algorithms on network inference accurately we use data from a synthetic gene regulatory network, whose structure is known. We emphasize that our purpose is to compare the performance of discretization methods and not inference methods.

2. DATA DISCRETIZATION

Discretization of real data into a typically small number of finite values is often required by machine learning algorithms (Dougherty, 1995), data mining (Han, 2000), discrete dynamic Bayesian network applications (Berlo, 2003), and any modeling algorithm using discrete-state models. Binary discretizations are the simplest way of discretizing data, used, for instance, for the construction of Boolean network models for gene regulatory networks (Kauffman, 1969; Albert, 2003). The expression data are discretized into only two qualitative states as either present or absent. An obvious drawback of binary discretization is that labeling the real-valued data according to a present/absent scheme generally causes the loss of a large amount of information. Discrete models and modeling techniques allowing multiple states have been developed and studied previously (Laubenbacher, 2004; Thieffry, 1998). In order to place the further discussion in a general context we give a definition of discretization (Hartemink, 2001).

Definition 2.1. *A discretization of a real-valued vector* $\mathbf{v} = (v_1, \ldots, v_N)$ *is an integer-valued vector* $\mathbf{d} = (d_1, \ldots d_N)$ *with the following properties:*

1. *Each element of* $\mathbf{d}$ *is in the set* $0, 1, \ldots, D-1$ *for some (usually small) positive integer D, called the degree of the discretization.*
2. *For all* $1 \leq i, j \leq N$, *we have* $d_i \leq d_j$ *if and only if* $v_i \leq v_j$.

Without loss of generality, assume that $\mathbf{v}$ is sorted, i.e., for all $i < j$, $v_i \leq v_j$. Spanning discretizations of degree D are a special case that we focus on in this paper. They are defined in Hartemink (2001) as

discretizations that satisfy the additional property that the smallest element of **d** is equal to 0 and that the largest element of **d** is equal to $D-1$. Given $\mathbf{v}=(v_1, \ldots, v_N)$, there is a large variety of schemes to obtain a discretization that is consistent with the above definition. Here we present two simple ways that are often used as a starting point in more complicated methods.

Interval discretizations divide the interval $[v_1, v_N]$ into k equal sized bins, where k is user-defined. Another simple method is *quantile discretization* which places N/k (possibly duplicated) values in each bin (Dougherty, 1995). Any method based on those two approaches would suffer from problems that make it inapplicable to the type of biological data we consider. Interval discretizations are very sensitive to outliers and may produce a strongly skewed range (Catlett, 1991). In addition, some discretization levels may not be represented at all which may cause difficulties with their interpretation as part of the state space of a discrete model. In fact, for the method that we propose, we assume that for each integer a with $0 \leq a \leq D-1$, there is an entry d_i of **d** such that $a=d_i$.

On the other hand, quantile discretizations depend only on the ordering of the observed values of **v** and not on the relative spacing values. Since distance between the data points is often the only information that comes with short time series, losing it is very undesirable. A shortcoming, common for both interval and quantile, as well as for most other discretization methods, is that they require the number of discrete states, k, to be user-provided. We discuss later why this is impractical.

A number of entropy-based discretization methods deserve attention. An example of those is Hartemink's *Information-preserving Discretization* (IPD). It relies on minimizing the loss of pairwise mutual information between each two real-valued vectors (variables). The mutual information between two random variables X and Y with joint distribution $p(X, Y)$ and marginal distributions $p(x)$ and $p(y)$ is defined as

$$I(X;Y)=\sum_x \sum_y p(x,y) log \frac{p(x,y)}{p(x)p(y)}.$$

Note that if X and Y are independent, by definition of independence $p(x, y)=p(x)p(y)$, so $I(X; Y)=0$. When modeling regulatory networks and having as variables, for instance, mRNA, protein, and metabolite concentrations, the joint distribution function is rarely known and it is often hard to determine whether two variables are independent or not. In fact, finding dependencies is a primary objective of regulatory network inference. Therefore, computing mutual information and basing discretization on its pairwise minimization is inapplicable.

As pointed out earlier, a major challenge of discretizing biological data is the small number of data points. For example, about 80% of microarray time series experiments are short: 3-8 time points (Ernst, 2006). For the case of such small samples of data, many statistical methods for discretization, such as Peer (2001), are not applicable due to the insufficient amount of data. For example, the sample size may be insufficient to estimate distributions.

Another common discretization technique is based on clustering (Jain, 1988). One of the most often used clustering algorithms is k-means developed in MacQueen (1967). It is a non-hierarchical clustering procedure whose goal is to minimize dissimilarity in the elements within each cluster while maximizing this value between elements in different clusters. Many applications of k-means clustering, such as the MultiExperiment Viewer (Saeed, 2003), start by taking a random partition of the elements into k clusters and computing their centroids. As a consequence, a different clustering may be obtained every time the algorithm is run. Another inconvenience is that the number k of clusters to be formed has to be specified in advance. Although there are methods for choosing "the best k", such as the one described in Crescenzi (2001), they rely on some knowledge of the data properties that may not be available.

Another method is single-link clustering (SLC) with the Euclidean distance function. SLC is a divisive (top-down) hierarchical clustering that defines the distance between two clusters as the minimal distance of any two objects belonging to different clusters (Jain, 1988). In the context of discretization, these objects will be the real-valued entries of the vector to be discretized, and the distance function that measures the distance between two vector entries v and w will be the one-dimensional Euclidean distance $|v-w|$. Top-down clustering algorithms start from the entire data set and iteratively split it until either the degree of similarity reaches a certain threshold or every group consists of one object only. For the purpose of data analysis, it is impractical to let the clustering algorithm produce too many clusters containing only one real value. The iteration at which the algorithm is terminated is crucial since it determines the degree of the discretization. SLC with the Euclidean distance function satisfies one of our major requirements: very little

starting information is needed—only distances between points. In addition, being a hierarchical clustering procedure, it lends itself to adjustment in case that clusters need to be split or merged. It may result, however, in a discretization where most of the points are clustered into a single partition if they happen to be relatively close to one another. Another problem with SLC is that its direct implementation takes D, the desired number of discrete states, as an input. However, we would like to choose D as small as possible, without losing information about the system dynamics and the correlation between the variables, so that an essentially arbitrary choice is unsatisfactory.

This brief discussion of commonly used discretization methods and the issues that arise when applying them to data from short time course measurements is intended to demonstrate that data discretization for the purpose of inferring properties of biochemical networks has to be approached with care.

3. NEW DISCRETIZATION METHOD: SSD (SHORT SERIES DISCRETIZATION)

We introduce a new method for discretization of time series of experimental data into a finite number of states. While of interest for other purposes, this method is designed specifically for the discretization of short multivariate time series, such as those used for the construction of discrete models of biochemical networks built from time series of experimental data. An important characteristic of such time series that we kept in mind is the relatively small number of data points—typically no more than ten. We employ a graph-theoretic clustering method to perform the discretization and an information-theoretic technique to minimize loss of information content. One of the most useful features of our method is the determination of an optimal number of discrete states that is most appropriate for the data. Our $C++$ program takes as input one or more vectors of real data and discretizes their entries into a number of states that is most suitable for the data. Our main objective was to construct a method that is capable of preserving information about the network dynamics inherent in the time series as well as performing well in the presence of noise in the experimental data. While in the current paper we use the method only for reconstructing the network interactions, we have evidence that it can be used for the reverse-engineering of the network dynamics as the following example shows.

The current paper acknowledges the problem of having to deal with very small data sizes and presents a method that is specifically designed to work with a small number of data points and does not make ungrounded assumptions about their statistical properties. Some existing discretization techniques assume that the number of discrete classes to be obtained is given (Friedman, 2000). While this number is extremely important, it is not clear how to properly select it in many cases. Also, in these cases it is rarely known what the appropriate discretization thresholds for each gene might be. These two issues were addressed by modifying the SLC algorithm: our method begins by discretizing a vector in the same way as SLC but instead of providing D as part of the input, the algorithm contains termination criteria which determine the appropriate number D. After that each discrete state is checked for information content and if it is determined that this content can be considerably increased by further discretization, then the state is separated into two states in a way that may not be consistent with SLC. We point out that although the discretization method we propose uses clustering as a tool to distribute the data points among the different states, it is not the same as data clustering and pursues different objectives. The details of the SSD algorithm follow next.

3.1. Discretization of one vector

Even if more than one vector is to be discretized, SSD begins by discretizing each vector independently and for some applications this may be sufficient. The example of such a vector to keep in mind is a time series of expression values for a single gene. If the vector contains m distinct entries, a complete weighted graph on m vertices is constructed, where a vertex represents an entry and an edge weight is the Euclidean distance between its endpoints. The discretization process starts by deleting the edge(s) of highest weight until the graph gets disconnected. If there is more than one edge labeled with the current highest weight, then all of the edges with this weight are deleted. The order in which the edges are removed leads to components, in which the distance between any two vertices is smaller than the distance between any two components, a requirement of SLC. We define the distance between two components G and H to be $\min \{|g-h| \; | g \in G, h \in H\}$. The output of the algorithm is a discretization of the vector, in which each

cluster corresponds to a discrete state and the vector entries that belong to one component are discretized into the same state.

3.2. Example

Suppose that vector $\mathbf{v} = (1, 2, 7, 9, 10, 11)$ is to be discretized. The corresponding SLC dendrogram that would be obtained by SLC algorithms such as the Johnson's algorithm (Johnson, 1967) is given in Figure 1. We start with constructing the complete weighted graph based on $\mathbf{v}$ which corresponds to iteration 0 of the dendrogram (Fig. 2). Eight edges with weights 10, 9, 9, 8, 8, 7, 6, and 5, respectively, have to be deleted to disconnect the graph into two components: one containing vertices 1 and 2 and another having vertices 7, 9, 10, and 11; this is the first iteration. Having disconnected the graph, the next task is to determine if the obtained degree of discretization is sufficient; if not, the components need to be further disconnected in a similar manner to obtain a finer discretization. See the following summary for details.

3.3. Algorithm summary

An implementation of the algorithm is available from the authors at polymath.vbi.vt.edu/discretization.

Input: set $S_r = \{v_i | i = 1, \ldots, m\}$ where each $v_i = (v_{i1}, \ldots, v_{iN})$ is a real-valued vector of length N to be discretized.

Output: set $S_d = \{d_i | i = 1, \ldots, m\}$ where each $d_i = (d_{i1}, \ldots, d_{iN})$ is the discretization of v_i for all $i = 1, \ldots, m$.

1. For each $i = 1, \ldots, m$, construct a complete weighted graph G_i where each vertex represents a distinct v_{ij} and the weight of each edge is the Euclidean distance between the incident vertices.
2. Remove the edge(s) of highest weight.
3. If G_i is disconnected into components $C_{i1}^{G_i}, \ldots, C_{iM_i}^{G_i}$, go to 4. Else, go to 2.
4. For each $C_{ik}^{G_i}, k = 1, \ldots, M_i$, apply "disconnect further" criteria 1–3. If any of the three criteria holds, set $G_i = C_{i1}^{G_i}$ and go to 2. Else, go to 5.
5. Apply DF criteria (see below). If this criterion is satisfied, go to 6. Else, go to 7.
6. Sort the vertex values of $C_{i1}^{G_i}$ and split them into two sets: if $|V(C_{i1}^{G_i})|$ is even, split the first $|V(C_{i1}^{G_i})|/2$ sorted vertex values of $G_i = C_{i1}^{G_i}$ into one set and the rest—into another. If $|V(C_{i1}^{G_i})|$ is odd, split the first $|V(C_{i1}^{G_i})|/2 + 1$ sorted vertex values of $|V(C_{i1}^{G_i})|$ into one set and the rest—into another.
7. Sort the components $C_{i1}^{G_i}, k = 1, \ldots, M_i$, by the smallest vertex value in each $C_{i1}^{G_i}$ and enumerate them $0, \ldots, D_i - 1$, where D_i is the number of components into which G_i got disconnected. For each $j = 1, \ldots, N, d_{ij}$ is equal to the label of the component in which v_{ij} is a vertex.

Disconnect Further (DF) Criteria. A component is further disconnected if and only if both (1) and (2) hold:

1. The minimum vertex degree of the component is less than the number of its vertices minus 1. The contrary implies that the component is a complete graph by itself, i.e., the distance between any two of its vertices is smaller than the distance between the component and any other component.

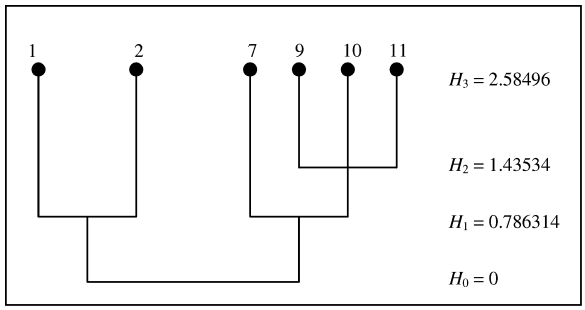


FIG. 1. Dentrogram representing the SLC algorithm applied to the data of Example 3.2. The column on the right gives the corresponding Shannon's entropy increasing at each consecutive level.

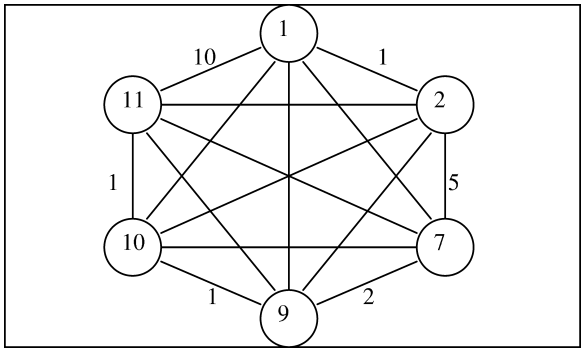


FIG. 2. The complete weighted graph constructed from vector entries 1, 2, 7, 9, 10, and 11. Only the edge weights of the outer edges are given.

2. One of the following three conditions is satisfied:
(a) The average edge weight of the component is greater than half the average edge weight of the complete graph.
(b) The largest distance between two vertices is greater than or equal to half this distance in the complete graph. For the complete graph, the distance is the graph's highest weight.
(c) Finally, if the above two conditions (a) and (b) fail, a third one is applied: disconnect the component if it leads to a substantial increase in the information content carried by the discretized vector (see Section 3.4.)

3.4. Information measure criterion

Discretizing the entries of a real-valued vector into a finite number of states inevitably reduces the information carried by the discrete vector in the sense defined in Shannon (1948). Shannon developed a measure of how much information is produced by a discrete source. The measure is known as entropy or Shannon entropy. Suppose there is a set of n possible events whose probabilities of occurrence are known to be $p_1, p_2, \ldots, p_n$. Shannon proposed a measure of how much choice is involved in the selection of the event or how certain one can be of the outcome, which is given by

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

The base 2 of the logarithm is chosen so that the resulting units may be called bits. In our context the Shannon entropy of a vector discretized into n states is given by

$$H = \sum_{i=0}^{n-1} \frac{w_i}{n} \log_2 \frac{n}{w_i},$$

where w_i is the number of entries discretized into state i. An increase in the number of states implies an increase in entropy, with an upper bound of $\log_2 n$. However, we want the number of states to be small, so it is important to notice that H increases by a different amount depending on which state is split and the size of the resulting new states. For example, if a state containing the most entries is split into two new states of equal size, H will increase more than if a state of fewer entries is split or if we split the larger state into two states of different sizes. Splitting a state into two states of equal size maximizes the entropy increase. For that reason, if a component fails the other conditions, we consider splitting it further only if doing so would provide a very significant increase of the entropy, i.e., if the component corresponds to a "large" collection of entries. In our implementation a component gets disconnected further only if it contains at least half of the vector entries.

3.5. Algorithm complexity

Given M variables, with N time points each, we compute $N(N-1)/2$ distances to construct the distance matrix so the complexity of this step is $\mathcal{O}(N^2)$. The distance matrix is used to create the edge and vertex sets

of the complete distance graph, containing $N(N-1)/2$ edges. This can also be accomplished in $\mathcal{O}(N^2)$ time. These edges are then sorted into decreasing order, so that the largest edges are removed first. A standard sorting algorithm, such as merge sort, has complexity $\mathcal{O}(N \log N)$ (Knuth, 1998). As each edge is removed, the check for graph disconnection involves testing for the existence of a path between the two vertices of the edge. This test for graph disconnection can be accomplished with a breadth-first search, which has order $\mathcal{O}(E+V)$ (Pemmaraju, 2003), with E the number of edges and V the number of vertices in the component. In our case this translates to complexity $\mathcal{O}(N^2)$. Edge removal is typically performed for a large percentage of the $N(N-1)/2$ edges, so this step has overall complexity $\mathcal{O}(N^4)$. The edge removal step dominates the complexity so that the overall complexity is $\mathcal{O}(MN^4)$ to discretize all M variables. While this is the theoretical worst-case performance, because of the heuristics we have added the typical performance is significantly better.

3.6. Requirements on the number of states

While for some applications any number of discretization states is acceptable, there are some cases when there are limitations on this number. For example, if the purpose of discretizing the data is to build a model of polynomials over a finite field, as in Laubenbacher (2004), then the number of states must be a power of a prime since every finite field has cardinality p^n, where p is prime and n is a positive integer. SSD deals with this problem in the following way. Suppose that a vector has been discretized into m states in the way described above. The next step is to find the smallest integer $k=p^n$ such that $m \leq k$. This value for k gives the number of states that needs to be obtained. Since SSD yielded m clusters, the remaining $k-m$ can be constructed by sorting the entries in each cluster and splitting the one that contains the two most distant entries with respect to Euclidean distance. The splitting should take place between these entries. This is repeated until k clusters are obtained. This approach has a potential problem. For instance, if a vector got discretized into 14 states and the total number of distinct entries of the vector is 15, then $k=16$ cannot be reached. In this case the two closest states could be merged together to obtain 13 states. In general it may not be desirable to reduce the number of states because this results in loss of information. We would rather increase the number of states unless it is impossible as in the above example.

3.7. Discretization of several vectors

Some applications may require that all vectors in a data set be discretized into the same number of states. For example, the approach adopted by Laubenbacher (2004) imposes such a requirement on the discretization. The way we deal with this is by first discretizing all vectors separately. Suppose that for N vectors, SSD discretized each into $m_1, m_2, \ldots, m_N$ states, respectively. Let $m = \max\{m_i | i = 1, \ldots, N\}$. Now find the least possible $k=p^n$ such that $m \leq k$. Finally, discretize all variables into k states in the same way that was described for the discretization of a single vector into the required number of states.

3.8. Preservation of correlation

While any discretization inevitably results in information loss, a good multivariate discretization should preserve some important features of the data such as correlation between the variables. To demonstrate that SSD has this property, we use the temporal map of fluctuations in mRNA expression of a set of genes related to rat central nervous system development presented in Wen (1998). This paper focuses on the cervical spinal cord and the genes included in their study are from families that are believed to be important for spinal cord development. They used an RT-PCR protocol to measure the expression of 112 genes in central nervous system development. The data consist of nine expression measurements for each gene: cervical spinal cord tissue was dissected from animals in embryonic days 11, 13, 15, 18, and 21 and in postnatal days 0, 7, 14, and 90.

We calculated the Spearman rank correlation coefficient between each gene's analog time series and discretized time series according to Walpole (1998). Then we identified each coefficient value as representing a "significant" or "not significant" correlation based on a critical value of 0.683, which corresponds to a significance level of 0.025 for a time series of nine points. This means that the probability that the correlation coefficient for a pair of uncorrelated time series of length 9 will be greater than or equal to 0.683 is 0.025. The result is that 71 out of 112 genes were found to be significantly correlated to their discrete version, which is 63.39% of the total number of genes. By changing the level of significance to

0.05, this percentage increases to 76.79%. We also calculated the Spearman rank correlation coefficient for each pair of genes before and after discretizing the data, considering only the genes that discretized into exactly three states (56 of them). For the original data given in Wen (1998), out of the 1540 gene pairs, 234 pairs were identified as significantly correlated at significance level 0.025. Based on the discretized data, 132 pairs were correctly identified as significantly correlated and 1181 pairs were correctly recognized as not significantly correlated. That is, 1313, or more than 85%, of the correlations were correctly classified after discretization. These results suggest that SSD can be successfully applied to cases when preserving the correlation among the variables in a system is important.

We performed more correlation analysis to compare SSD to quantile and interval discretization (see Section 4.5).

3.9. SSD algorithm validation

We now demonstrate how SSD can be used for reconstruction of a gene network. Due to limited knowledge of the dynamic features of real biochemical networks, the validation of our method is best done with a simulated network. In order to extract information about the network topology and dynamics from the discretized data, we use the network reconstruction method in Laubenbacher (2004). The goal of this validation is to show that the discretized data preserved the information about the network that was contained in the corresponding continuous data generated from the network. We used the *A-Biochem* software system developed by Mendes (2003) to generate the simulated network. *A-Biochem* automatically generates artificial gene networks with particular topological and kinetic properties. These networks are embodied in kinetic models, which are used by the biochemical-network simulator *Gepasi* (Mendes, 1993, 1997) to produce simulated gene expression data. We generated an artificial gene network with five genes and ten total input connections using the Albert-Barabási algorithm (Albert, 2000). In Figure 3A, the arrows represent gene activation in the arrow's direction and the segments stand for inhibition.

Gepasi uses a continuous representation of biochemical reactions, based on ordinary differential equations (ODE). With the parameters we specified, *Gepasi* generated an ODE system that represents the network. Analyzing the dynamics of the ODE system, one finds that it has two stable steady states. As Laubenbacher (2004) demonstrated, the performance of their algorithm dramatically improves if knockout time series for genes are incorporated. For this reason, we supplied seven time series of 11 points each: two wild-type time series and five knockout time series, one for each gene. The wild-type time series are generated by solving the ODE system numerically for two different initial conditions. One can simulate a gene knockout by setting the corresponding variable and initial condition to zero. The time points from each of the seven time series constitute the input vectors. The discretization algorithm chose a state set X of cardinality 5 and, based on the discrete data, the reverse-engineering method generated five polynomials describing the discrete model. That is, the discrete model is given by a time-discrete dynamical system

$$f = (f_1, \ldots, f_5) : X^5 \to X^5.$$

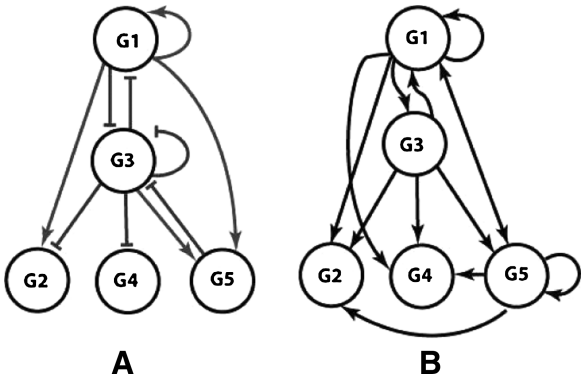


FIG. 3. (**A**) Dependency graph of the *A-Biochem*-generated artificial gene network on five genes G_i. (**B**) Dependency graph of the reverse-engineered system.

Now we compare the dynamics of the two models. First, the discretization maps the ODE's stable steady states to fixed points of the discrete model. To illustrate the matching dynamics of the two models, notice how the discrete model's trajectory in Figure 4B can be superimposed over the continuous trajectory in Figure 4A, both ending at a fixed point/steady state. The same can be observed for the second steady-state.

Further evidence of the ability of SSD to retain information is the fact that the reverse-engineering method in Laubenbacher (2004) was able to extract most of the information about the dependency graph of the network. This can be seen by comparing the inferred diagram in Figure 3B to the diagram of the actual direct interactions in Figure 3A.

3.10. Discretization in the presence of noise

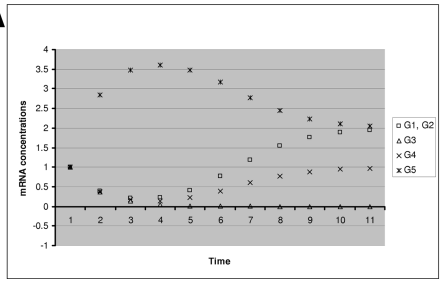
Noise is naturally present in biological data and in microarray data in particular (Hassibi, 2005). Although there are various techniques which increase the accuracy of the microarray measurements, the data inevitably contain errors due to the probabilistic characteristics of the detection process, from sample extraction and mRNA purification to hybridization and imaging. Consequently, it is crucial to study how the proposed discretization algorithm performs in the presence of typical levels of noise in the experimental data.

To study the effect of noise, we considered two types of data. We used the earlier mentioned gene expression measurements generated for a study of rat cervical central nervous system development (Wen, 1998). We selected 18 genes whose nine-point time series have statistical variance of more than 1. Although the data likely contain some noise, here we assume that the measurements are perfect and we add noise of known proportion and distribution assuming it is the only noise in the data. The purpose is to work with data which have the properties of real microarray time series. The second data set is an artificial ten-point time series on 30 ''genes.'' The values are randomly generated real numbers in the range [0, 20] with statistical variance greater than 10.

Two types of noise are added to the data at the same time: overall and point-specific (Hatzimanikatis, 1999). The overall noise is added by sampling from a normal distribution with zero mean and a standard deviation equal to 12.5% of the standard deviation of each gene. The point-specific noise is simulated by adding noise to each time point sampling from a different normal distribution with zero mean and standard deviation equal to 12.5% of the standard deviation of the particular time point value. Thus, the total proportion of added noise amounts to 25%. For each gene 100 noise containing replicates were generated and discretized.

For each gene, we compare the way the original noiseless time series was discretized to the discretization of the noise-containing replicates. We count the number of noise-containing replicates that got discretized exactly the same as the original noiseless time series. For the gene expression data from Wen (1998), 78.72% of all the 18×100 noise-containing replicates were discretized exactly as their corresponding noiseless originals. This number is significantly higher for the randomly generated data of variance 10 or higher: 93.8% of all the 30×100 noise-containing replicates were discretized exactly as their corresponding noiseless originals. The big difference between the two results can be explained by the different statistical variance of the original data in the two cases. Only 5 out of the 18 genes in the expression measurements time series in Wen (1998) have variance greater than 10 while the ''genes'' in the simulated time series had variance of at least 10 with average variance of 34.5. Since the higher the variance of a time series, the

A



B

G1	*G2*	*G3*	*G4*	*G5*
3	3	4	4	1
1	1	3	2	4
1	1	2	2	4
1	2	1	2	4
2	3	1	2	4
3	4	1	2	4
4	4	1	4	4
4	4	1	4	2
4	4	1	4	2
4	4	1	4	2
4	4	1	4	2

FIG. 4. (**A**) Wild-type time series generated by solving numerically the ODE system. (**B**) Corresponding discrete point time series.

better the chance of more distinct and easier to detect discrete states, the discretization of the data with higher variance not surprisingly showed more robustness in the presence of noise.

4. COMPARISON OF DISCRETIZATION METHODS

We now compare the performance of SSD to two other common discretization algorithms: quantile and interval. We point out that data discretization is a crucial step for modeling methods not based on reverse engineering and such methods can also benefit from SSD. To enable the comparison, we use these algorithms to provide discrete data for two reverse-engineering methods: dynamic Bayesian networks (Yu, 2002) and discrete polynomial dynamical systems (Jarrah, 2005). The particular implementations used for the study are outlined next. In 4.1, we describe the *in silico* gene regulatory networks that produced the input data and our comparison metrics are given in 4.4. The data sets used for the comparison are available at http://polymath.vbi.vt.edu/discretization/Discretization Comparison/.

4.1. In silico gene regulatory networks

To evaluate the performance of the different discretization methods as part of a network inference algorithm, we use data from four different artificial gene regulatory networks generated using the software package of Mendes (2003). In these networks the interactions between genes are phenomenological and represent the result of the effect of transcription and translation on the regulation of the genes in the network, implemented as a system of ODEs in *Copasi* (Hoops, 2006).

The first network shown in Figure 5, referred to subsequently as "Network 1," consists of 13 species: ten genes and three different environmental perturbations. The perturbations affect the transcription rate of the gene on which they act directly (through inhibition or activation) and their effect is further propagated through the network by the interactions between the genes. The data used consist of six short time series corresponding to different concentration values for the environmental perturbations, with a total of 144 time steps; this network has been previously used in Camacho (2007).

The second and third networks, "Network 2" and "Network 3," respectively, consist of five genes that interact with each other through inhibition or activation; the data generated consist of seven short time series, with a total of 77 time steps. Similarly, the fourth in *silico* network, "Network 4," consists of ten genes; the data generated consist of seven short time series corresponding to the different given initial conditions, with a total of 147 time steps. The three networks are shown in Figure 6.

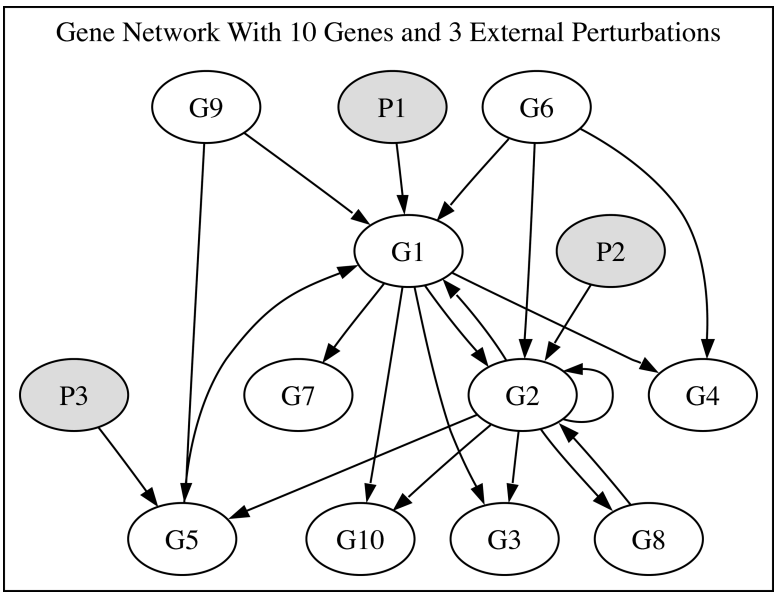


FIG. 5. Network 1: 10 genes and three environmental perturbations. In this network, the three environmental perturbations P1, P2, and P3 directly affect the expression rate of genes G1, G2, and G5, respectively.

4.2. Dynamic Bayesian networks

A Bayesian network is a graphical model that describes the causal relationships between variables. Such relationships are represented by a family of joint probability distributions consistent with the independence assertions embedded in the graph (Hartemink, 2002). A dynamic Bayesian network (DBN) (Friedman, 1998) extends the notion of a Bayesian network to model the stochastic evolution of a set of random variables over time. The structure of a DBN thus describes the qualitative nature of the dependencies that exist between variables in a temporal process (Bernard, 2005).

The software Bayesian Network Inference with Java Objects (BANJO) (Hartemink, 2006) is an implementation of a DBN inference method. The type of data required for this method is in the form of temporal response of the system to a perturbation. BANJO also requires as input a set of parameters, such as search method, time lag (how far back in time each variable depends on the others), type of discretization, etc. The core algorithms in BANJO assume discrete variables, and BANJO provides a discretization functionality using either quantile or interval discretization algorithms with a default upper bound of five states per variable.

The data discretization is handled as follows:

I. *SDD Discretization of Data.* The SDD method returned variables discretized into 5 and 11 states for Network 1 and Network 4, respectively. Similarly, the SSD method returned variables discretized into 7 states for both Network 2 and 3. BANJO handles networks with discretized variables up to 5 states, allowing us to use directly the data from Network 1; in order to use our discretization method for the networks with 5 genes, we extracted the time step measurements that return variables discretized into more than 5 states, to reduced the data to approximately 70 time steps in each data set. Due to the 11 state discretization values return by SDD we were unable to use Network 4, but were able to use it with the discrete polynomial dynamical system reverse-engineering method.

II. *Interval and Quantile Discretization of Data.* BANJO provides a discretization functionality using either quantile or interval discretization algorithms with a default upper bound of five states per variable. Assuming no prior information about the ideal type or number of discretization states, we explore all of the possible discretization methods available in BANJO: quantile and interval into 2, 3, 4 and 5 states to find the highest scored inferred DBN.

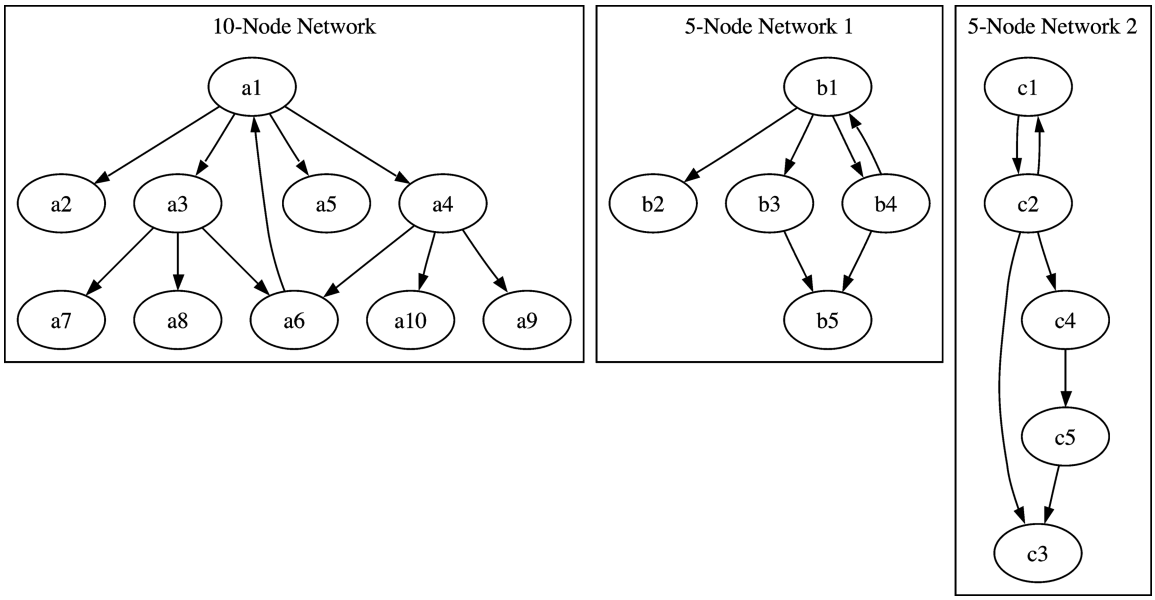


FIG. 6. Three *in silico* networks. The first network consists of 10 genes and the second and third networks consist of 5 genes. In these three networks, the interactions between genes are phenomenological and represent the result of the effect of transcription and translation on the regulation of the genes in the network.

Per each one of these different discretized data sets, we infer a DBN considering two important aspects:

1. As commonly needed for optimization based methods as BANJO, a set of input parameters should be specified in order to infer a DBN (approximately 13 different parameter values), thus we need to explore different combinations of parameter sets for each of the discretized data sets. A set of parameter values is selected based on BANJO's scoring method.
2. Unless we assume an instantaneous time lag (i.e., time lag value 0), it is necessary to build a consensus DBN from the DBNs inferred from each one of the 6 or 7 different input time courses of a given network; this is done in order to avoid having to consider the state values of the first time step of a given time series to depend on the last time step(s) of the previous time course.

Hence, for a given discretization type we first select the combination of sets of parameters that provide a highest score. Then, in order to build a consensus sparse network (i.e., to avoid a high rate of false positives) from the different DBNs obtained from each time series, we select edges that appear in at least two of the output networks.

4.3. Discrete polynomial dynamical systems

We used the reverse-engineering algorithm of Jarrah (2005) together with their ($S1$, $T1$) scoring method[1] for which only the network model(s) with the highest score is generated. In this method, the only input needed for the algorithm is the discretized time series (i.e., unlike BANJO, there are no parameter values to explore).

We compare the network models obtained from the method of Jarrah (2005) when using the following discretizations:

1. **SSD:** As mentioned in section 4.2, the SDD method returned variables discretized into 5 and 11 states for Networks 1 and 4, respectively. Similarly, the SSD method returned variables discretized into 7 states for both networks of five genes (Network 2 and Network 3).
2. **Quantile and Interval discretization** into 5, 11 and 7 states to allow a direct comparison with SSD.

4.4. Metrics for method performance

For an unbiased evaluation of the performance of each reverse-engineering algorithms with the different discretization methods, we quantify:

(a) Correct interactions inferred (true positives, TP)
(b) Incorrect interactions inferred (false positives, FP)
(c) Correct non-interactions inferred (true negatives, TN)
(d) Incorrect non-interactions inferred (false negatives FN)

The values of TPs, TNs, FPs, and FNs are calculated to build a *confusion matrix.* The confusion matrix is a matrix that provides an assessment of the classification procedure, or how "confused" the algorithm might be, by simply counting the interactions discovered. From the confusion matrix we compute the Recall or True Positive Rate $\mathrm{TPR} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$, the False Positive Rate $\mathrm{FPR} = \mathrm{FP}/(\mathrm{FP} + \mathrm{TN})$, Accuracy $\mathrm{ACC} = (\mathrm{TP} + \mathrm{TN})/\mathrm{TotI}$ and Precision or Positive Predictive Value $\mathrm{PPV} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$. It is important to mention that coordinate points (*FPR, TPR*) represent coordinate points within the Receiver Operator Characteristic (ROC) curves, whereas (*TPR, PPV*) represent coordinate points within the Recall Precision Curve; both of these curves are commonly used for the evaluation of reverse-engineering methods. Since we are presenting here pairs of coordinate points in both curves, we decided to compare results in both metrics as well as to add the Accuracy (ACC) score for a broader view of the effect of each discretization method. In particular, it is worth to comment how ROC points can be characterized: a perfect reverse engineering method will ideally have score $(FPR, TPR) = (0, 1)$, whereas the worst possible network will have coordinates $(FPR, TPR) = (1, 0)$ and methods with scores below the identity line (or diagonal) are those that perform no better than a random guess.

The best scored consensus DBNs from BANJO are presented in Table 1. Notice that with respect to any of the four metrics—TPR, FPR, ACC, or PPV—our discretization method performs best. The results for the polynomial dynamical systems method are summarized in Table 2. We observe that using any metric, we

[1]For a Macaulay2 implementation of the ($S1$, $T1$) scoring method, contact R. Laubenbacher at reinhard@vbi.vt.edu.

Table 1. Dynamic Bayesian Network Approach

	TP	*FP*	*TN*	*FN*	*TPR*	*FPR*	*ACC*	*PPV*
SSD Net1	7	9	138	15	.318	.0612	.858	.438
BANJO Net1	2	5	142	20	.091	.0340	.852	.286
SSD Net2	4	2	17	2	.666	.105	.84	.666
BANJO Net2	2	4	15	4	.333	.210	.68	.333
SSD Net3	3	2	17	3	.5	.105	.80	.6
BANJO Net3	2	2	17	4	.333	.105	.76	.5

Best results for the DBNs obtained in BANJO. The first four columns are the four entries in the confusion matrix.

obtained the best results with SSD. From our analysis, we see that with both reverse-engineering algorithms, the SSD performs consistently better, independently of the metric employed.

4.5. Correlation preservation comparison

We calculated the Spearman rank correlation coefficient between each gene's continuous time series and its time series discretized with SSD, interval, and quantile discretizations. 82% of the genes had SSD discretizations that are either more strongly or as strongly correlated to the continuous data as interval and quantile discretizations. Moreover, at level of significance 0.05, the correlation coefficient is large enough to conclude that there is evidence that the continuous time series are strongly correlated to their SSD discretizations.

5. CONCLUSIONS

Discretization of continuous experimental data into finitely many discrete states is important in several contexts, in particular in systems biology when inferring biochemical networks from experimental data such as time series data of DNA microarrays. Almost all methods using dynamic Bayesian networks as well as other discrete methods rely on data discretization as a preprocessing step. A particular characteristic of time series data sets is that they are still quite small, typically consisting of less than 10 time points. It is natural to look at one of the many available clustering methods for this purpose. Upon closer study, however, it becomes clear that discretization of this particular data type with standard clustering methods is problematic. This has led us to develop the new method, SSD, described here. The main result of the paper is a comparison to other commonly used discretization methods by means of their application to two very different network inference methods, both of which give as output a most likely dependency. The comparison shows that network inference using either of these methods results in better predictions with SSD as compared to the other commonly used methods.

Table 2. Polynomial Dynamical Systems Approach

	TP	*FP*	*TN*	*FN*	*TPR*	*FPR*	*ACC*	*PPV*
SSD Net1	12	34	113	10	.5454	.231	.7396	.261
Quantile, 5 states Net1	9	49	98	13	.4090	.333	.6331	.155
Interval, 5 states Net1	7	46	101	15	.3181	.313	.6390	.132
SSD Net2	3	5	14	3	.5	.263	.68	.375
Quantile, 7 states Net2	5	13	6	1	.83	.684	.44	.277
Interval, 7 states Net2	4	11	8	2	.66	.579	.48	.266
SSD Net3	4	7	12	2	.66	.368	.64	.364
Quantile, 7 states Net3	5	13	6	1	.83	.684	.44	.277
Interval, 7 states Net3	5	12	7	1	.83	.632	.48	.294
SSD Net4	7	20	69	4	.63	.225	.76	.259
Quantile, 11 states Net4	10	49	40	1	.90	.551	.50	.169
Interval, 11 states Net4	6	34	55	5	.54	.382	.61	.15

The first four columns are the four entries in the confusion matrix.

It has been our experience that, while SSD performs well in many cases, its output can often be improved with human intervention. This would most likely be the case with any discretization method. Furthermore, there are certain types of data that SSD does not handle well. We do not recommend using SSD whenever long time series are available. In this case, a statistical method, such as Peer (2001), that can take advantage of the statistical properties of the data may be more appropriate. Our method assumes no knowledge of these properties and therefore cannot utilize this information. Another situation when virtually any discretization technique may not produce useful results is in the case when one or several of the system variables change much more rapidly than the rest. In order to avoid inconsistencies in the discretized data, one may need a very large number of discrete states which may be a disadvantage for the modeling process. The authors also acknowledge that discretization possibly introduces spurious dynamic phenomena, such as artificial periodic behavior, due to different time scales in the network. This has been well documented in the literature (Hatzimanikatis, 1999). As a result, the output of any discretization algorithm needs to be carefully examined in light of the original data.

Another problem that is specific for polynomial dynamical systems modeling is that discretization sometimes results in a data set that is inconsistent, in the sense that it cannot be produced by a deterministic dynamical system because a given state that appears more than once in the time series might transition to two different subsequent states. While one can devise ad hoc solutions to this problem, a more systematic way of dealing with this problem remains to be found.

Data discretization represents the transition from the continuous to the discrete data world and therefore plays a crucial role in the construction of discrete models. If this transition is not done well, all subsequent steps are flawed.

ACKNOWLEDGMENTS

We thank A. Jarrah, A. de la Fuente, P. Mendes, and B. Stigler for contributing insights and help with editing, and D. Camacho, I. Hoeschele, and W. Sha for helpful discussions. We are particularly grateful to Pedro Mendes and Diogo Camacho for permitting the use of the simulated gene regulatory networks employed here and that have been constructed by them. The first, second, and fourth authors have been partially supported by the NIH (grant RO1GM068947-01).

DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Akutsu, T., and Miyano, S. 2000. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics* 16, 727–734.

Albert, R., and Barabási, A. 2000. Topology of evolving networks: local events and universality. *Phys. Rev. Lett.* 85, 5234–5237.

Albert, R., and Othmer, H. 2003. The topology of the regulatory interactions predict the expression pattern of the segment polarity genes in *Drosophila melanogaster. J. Theor. Biol.* 223, 1–18.

Andrec, M., and Kholodenko, B.N. 2005. Inference of signaling and gene regulatory networks by steady-state perturbation experiments: structure and accuracy. *J. Theor. Biol.* 232, 427.

Beal, M.J., and Falciani, F. 2005. A Bayesian approach to reconstructing genetic regulatory networks with hidden factors. *Bioinformatics* 21, 349–356.

van Berlo, R., van Someren, E., and Reinders, M. 2003. Studying the conditions for learning dynamic Bayesian networks to discover genetic regulatory networks. *Simulation* 79, 689–702.

Bernard, A., and Hartemink, A. 2005. Informative structure priors: joint learning of dynamic regulatory networks from multiple types of data. *Pac. Symp. Biocomput.*

Camacho, D., Vera-Licona, P., Mendes, P., et al. 2007. Comparison of reverse-engineering methods using an *in silico* network. *Ann. N. Y. Acad. Sci.* 1115, 73–89.

Catlett, J. 1991. Megainduction: machine learning on very large databases [Ph.D. dissertation]. University of Sydney, Australia.

Crescenzi, M., and Giuliani, A. 2001. The main biological determinants of tumor line taxonomy elucidated by of principal component analysis of microarray data. *FEBS Lett.* 507, 114–118.

de la Fuente, A., and Bing, N. 2004. Discovery of meaningful associations in genomic data using partial correlation coefficients. *Bioinformatics* 20, 3565–3574.

Di Camillo, B., and Sanchez-Cabo, F. 2005. A quantization method based on threshold optimization for microarray short time series. *BMC Bioinform.* Suppl 6, S11.

Dojer, N., and Gambin, A. 2006. Applying dynamic Bayesian networks to perturbed gene expression data. *BMC Bioinform.* 7, 249.

Dougherty, J., Kohavi, R., and Sahami, M. 1995. Supervised and unsupervised discretization of continuous features. *Mach. Learn. Proc. 12th Int. Conf.*

Ernst, J., and Bar-Joseph, Z. 2006. STEM: a tool for the analysis of short time series gene expression data. *BMC Bioinform.* 7, 191.

Friedman, N., Murphy, K., and Russell, S. 1998. Learning the structure of dynamic proabilist networks. *14th Conf. Uncertainty Artif. Intell.*

Friedman, N., Linial, M., Nachman, I., et al. 2000. Using Bayesian networks to analyze expression data. *J. Comput. Biol.* 7, 601–620.

Gardner, T.S., and Bernardo, D. 2003. Inferring genetic networks and identifying compound mode of action via expression profiling. *Science* 301, 102–105.

Han, J., and Kamber, M. 2000. *Data Mining: Concepts and Techniques.* Academic Press, San Diego.

Hartemink, A. 2001. Principled computational methods for the validation and discovery of genetic regulatory networks [Ph.D. dissertation]. Massachusetts Institute of Technology, Cambridge, MA.

Hartemink, A., Gifford, D., Jaakkola, T., et al. 2002. Bayesian methods for elucidating genetic regulatory networks. *IEEE Intell. Syst.* 17, 37–43.

Hartemink, A. 2006. BANJO: Bayesian Network Inference with Java Objects. Available at: www.cs.duke.edu/amink/software/banjo/. Accessed December 1, 2009.

Hatzimanikatis, V., and Lee, K.H. 1999. Dynamical analysis of gene networks requires both mRNA and protein expression information. *Metab. Eng.* 1, 275–281.

Hassibi, A., and Vikalo, H. 2005. Probabilistic modeling and estimation of gene expression levels in microarray. *Proc. IEEE Workshop GENSIPS*.

Hoops, S., and Sahle, S. 2006. COPASI—a COmplex PAthway SImulator. *Bioinformatics* 22, 3067–3074.

Jain, A., and Dubes, R. 1988. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ.

Jarrah, A., Laubenbacher, R., Stigler, B., et al. 2007. Reverse-engineering of polynomial dynamical systems. *Adv. Appl. Math.* 39, 477–489.

Johnson, S.C. 1967. Hierarchical clustering schemes. *Psychometrika* 32, 241–254.

Kauffman, S.A. 1969. Metabolic stability and epigenesist in randomly constructed genetic nets. *J. Theor. Biol.* 22, 437–467.

Kim, J., and Bates, D. 2007. Least-squares methods for identifying biochemical regulatory networks from noisy measurements. *BMC Bioinform.* 8, 8.

Knuth, D.E. 1998. *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Addison-Wesley, Reading, MA.

Laubenbacher, R., and Stigler, B. 2004. A computational algebra approach to the reverse engineering of gene regulatory networks. *J. Theor. Biol.* 229, 523–537.

Liang, S., and Fuhrman, S. 1998. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. *Pac. Symp. Biocomput.* 18–29.

MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. *Proc. 5th Berkeley Symp. Math. Statist. Probabil.* 281–297.

Martin, S., and Zhang, Z. 2007. Boolean dynamics of genetic regulatory networks inferred from microarray time series data. *Bioinformatics*.

Mehra, S., and Hu, W-S. 2004. A Boolean algorithm for reconstructing the structure of regulatory networks. *Metab. Eng.* 6, 326.

Mendes, P. 1993. GEPASI: a software package for modeling the dynamics, steady states and control of biochemical and other systems. *Comput. Appl. Biosci.* 9, 563–571.

Mendes, P. 1997. Biochemistry by numbers: simulation of biochemical pathways with Gepasi 3. *Trends Biochem. Sci.* 22, 361–363.

Mendes, P., Sha, W., and Ye, K. 2003. Artificial gene networks for objective comparison of analysis algorithms. *Bioinformatics* 19, ii122–ii129.

Nariai, N., and Tamada, Y. 2005. Estimating gene regulatory networks and protein-protein interactions of *Saccharomyces cerevisiae* from multiple genome-wide data. *Bioinformatics* Suppl 21, ii206–ii212.

Pe'er, D., Regev, A., Elidan, G., et al. 2001. Inferring subnetworks from perturbed expression profiles. *Bioinformatics* 17, S215–S224.

Pemmaraju, S., and Skiena, S. 2003. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica.* Cambridge University Press, New York.

Pournara, I., and Wernisch, L. 2004. Reconstruction of gene networks using Bayesian learning and manipulation experiments. *Bioinformatics* 20, 2934–2942.

Saeed, A., Sharov, V., White, J., et al. 2003. TM4: a free, open-source system for microarray data management and analysis. *BioTechniques* 34, 374–378.

Shannon, C. 1948. A mathematical theory of communication. *Bell Syst. Technical J.* 27, 379–423, 623–656.

Thieffry, D., and Thomas, R. 1998. Qualitative analysis of gene networks. *Proc. Pac. Symp. Biocomput.* 77–88.

Walpole, R., Myers, R., and Myers, S. 1998. *Probability and Statistics for Engineers and Scientists*, 6th ed. Prentice Hall, Englewood Cliffs, NJ.

Wen, X., Fuhrman, S., Michaelis, G., et al. 1998. Large-scale temporal gene expression mapping of central nervous system development. *Proc. Natl. Acad. Sci. USA* 95, 334–339.

Yu, J., Smith, V., Wang, P., et al. 2002. Using Bayesian network inference algorithms to recover molecular genetic regulatory networks. *Int. Conf. Syst. Biol.*

Yu, J., and Smith, V.A. 2004. Advances to Bayesian network inference for generating causal networks from observational biological data. *Bioinformatics* 20, 3594–3603.

Zou, M., and Conzen, S.D. 2005. A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics* 21, 71–79.

Address correspondence to:
Dr. Elena S. Dimitrova
Department of Mathematical Sciences
O-303 Martin Hall
Clemson University
Clemson, SC 29634-0975

E-mail: edimit@clemson.edu