Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Theoretical Biology 229 (2004) 523–537
Journal of Theoretical Biology
www.elsevier.com/locate/jtbi

A computational algebra approach to the reverse engineering of gene regulatory networks

Reinhard Laubenbacher*, Brandilyn Stigler

Virginia Bioinformatics Institute at Virginia Tech, 1880 Pratt Drive, Building XV, Blacksburg, VA 24061, USA

Received 15 December 2003; received in revised form 21 April 2004; accepted 28 April 2004

Available online 20 June 2004

Abstract

This paper proposes a new method to reverse engineer gene regulatory networks from experimental data. The modeling framework used is time-discrete deterministic dynamical systems, with a finite set of states for each of the variables. The simplest examples of such models are Boolean networks, in which variables have only two possible states. The use of a larger number of possible states allows a finer discretization of experimental data and more than one possible mode of action for the variables, depending on threshold values. Furthermore, with a suitable choice of state set, one can employ powerful tools from computational algebra, that underlie the reverse-engineering algorithm, avoiding costly enumeration strategies. To perform well, the algorithm requires wildtype together with perturbation time courses. This makes it suitable for small to meso-scale networks rather than networks on a genome-wide scale. An analysis of the complexity of the algorithm is performed. The algorithm is validated on a recently published Boolean network model of segment polarity development in *Drosophila melanogaster*.

Keywords: Reverse engineering; Gene regulatory networks; Discrete modeling; Computational algebra

1. Introduction

Since the advent of DNA microarray technology several techniques from mathematics, statistics, engineering, and computer science have been adapted or newly developed for the purpose of using microarray and other data to reverse engineer the structure of gene regulatory networks. An important goal of systems biology is to discover and model the causal relationships between the components of such networks and the mechanisms that govern their dynamics (Kitano, 2002). Existing reverse-engineering methods can be broadly categorized as continuous vs. discrete and deterministic vs. stochastic. Some methods aim to discover only the network topology, that is, which genes regulate which others, with a directed graph or "wiring diagram" as output, possibly signed to indicate activation or inhibition. The goal of other methods is to describe the dynamics of the network.

We first describe some existing methods in order to place the one proposed in this paper into context. In Section 3 we propose a reverse-engineering approach which draws from the rich theory of computational algebra. We validate the method using a Boolean model of segment polarity genes in *Drosophila melanogaster* with results in Section 4. In Section 5, we discuss an issue of particular importance to our method, that of data discretization. Furthermore, we address the issue of the effect of noise in experimental data on our algorithm in Section 6. We close with a discussion of future work.

2. Approaches to reverse engineering

Several methods have been proposed that reconstruct only the wiring diagram of the network. Bayesian network methods, first proposed in Friedman et al. (2000), and further developed in Hartemink et al. (2001), can be applied to single time points. This approach was applied in Hartemink et al. (2001) to data from 52 *Saccharomyces cerevisiae* Affymetrix chips, in order to

*Corresponding author. Tel.: 540-231-7506; fax: 540-231-2606.
E-mail addresses: reinhard@vbi.vt.edu (R. Laubenbacher), bstigler@vbi.vt.edu (B. Stigler).

0022-5193/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jtbi.2004.04.037

analyze and score models of the regulatory network responsible for the control of genes necessary for galactose metabolism. The network considered involved 7 nodes. In order to derive network structures with high confidence, Bayesian network methods require more data than are typically available for gene regulatory networks. In Pe'er et al. (2001) a bootstrapping technique was developed to find features in networks for which only small data sets are available.

In Filkov et al. (2002) another statistical method was proposed that compares expression profiles from a time series of measurements. As an application the authors analyzed the data sets on *S. cerevisiae* in Cho et al. (1998) and Spellman et al. (1998), which consist of four microarray time series. The same data set was used to validate an interesting method to provide information about regulatory interactions using classifiers, developed in Soinov et al. (2003). In addition to the wiring diagram, this method produces logical relationships between state changes of variables, including compound interactions.

A method adapted from metabolic control analysis was proposed in de la Fuente et al. (2002). It requires input data based on very small perturbations of the rate of expression of each gene. The method was applied to simulated data using a model, published in Mendoza et al. (1999), of the gene network that controls flower morphogenesis in *Arabidopsis thaliana*, involving 10 genes. Using 10% perturbations, the authors were able to completely reconstruct the network of regulatory relations. A similar method was applied in Gardner et al. (2003) to construct a first-order model of regulatory interactions in a nine-gene subnetwork of the SOS pathway in *E coli*.

The most common approach to the modeling of dynamics is to view a gene regulatory network as a biochemical network of gene products, typically mRNA and proteins, and to describe their rates of change through a system of ordinary differential equations. Thus, the modeling framework is that of a time-continuous, deterministic dynamical system. We describe in detail the method in Yeung et al. (2002), as it shares many features with our method, even though it uses a different modeling framework. According to the authors, the method is intended to generate a "first draft of the topology of the entire network, on which further, more local, analysis can be based". The authors make two assumptions.

Firstly, the system is assumed to be operating near a steady state, so that the dynamics can be approximated by a linear system of ordinary differential equations:

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = -\lambda_i x_i(t) + b_i(t) + \xi_i(t) + \sum_{j=1}^{n} w_{ij} x_j(t),$$

for $i = 1, \ldots, n$. Here, $x_1, \ldots, x_n$ are mRNA concentrations for n genes, the λ_i are the self-degradation rates, the b_i are the external stimuli, and the ξ_i represent noise. The (unknown) w_{ij} describe the type and strength of the influence of the j-th gene on the i-th gene. They assemble to a square matrix W of real numbers. The output of the reverse-engineering algorithm is this matrix W. The input is a series of data points obtained by applying the stimulus $(b_1, \ldots, b_n)^{\mathrm{T}}$ and measuring the concentrations $x_1, \ldots, x_n$ m times. Assembling these measurements into a matrix X, neglecting noise, and absorbing self-degradation into the coupling constants w_{ij}, we obtain a matrix equation

$$\frac{\mathrm{d}}{\mathrm{d}t} X = WX + B.$$

Here, X is an $(n \times m)$-matrix, W an $(n \times n)$-matrix, and B an $(n \times m)$-matrix. Using singular value decomposition (SVD) one obtains

$$X^{\mathrm{T}} = UWV^{\mathrm{T}},$$

where U and V are orthogonal to each other. The first step is to obtain a particular solution W_0 to the reverse-engineering problem. One then obtains all possible solutions to the problem as

$$W = W_0 + CV^{\mathrm{T}},$$

where C ranges over the space of all square $(n \times n)$-matrices whose entries are equal to 0 for a certain range of j and arbitrary otherwise. Equivalently, CV^{T} ranges over all matrices that vanish on the given time points.

The second assumption made in the paper is that gene regulatory networks are sparse. This provides a selection criterion on which to base a particular choice of C, and hence of W. That is, the method selects the sparsest connection matrix W. This is accomplished through a particular choice of norm, and robust regression. The algorithm was validated by way of simulated data from three networks.

The other end of the model spectrum takes the view of a gene regulatory network as a logical switching network. First proposed in Kauffman (1969), Boolean network models have the advantage of being more intuitive than ODE models and might be considered as a coarse-grained approximation to the "real" network. They differ from ODE models in that time is taken as discrete and gene expression is discretized into only two quantitative states, as either present or absent. There is increasing evidence that certain types of gene regulatory networks have key features that can be represented well through discrete models or hybrid models; see, e.g., Filkov and Istrail (2002).

Several algorithms for reverse engineering of Boolean network models for gene expression have been proposed. As an example, the algorithm REVEAL (Liang et al., 1998) uses as a criterion for model selection the

concept of so-called mutual information. For a given experimental data set (assumed to be already discretized into a binary data set), that is, a given set of state transitions, the algorithm finds a Boolean function for each node of the network that "optimally" determines the output from the input by using as few variables as possible. In essence, the algorithm finds the sparsest possible Boolean network that is consistent with the data. The search is done by enumeration. In order to make this process feasible, the number of inputs to the functions is restricted to at most three. The algorithm has been tested by the authors on simulated data sets and was found to perform very efficiently. Another algorithm for Boolean network identification was given in Akutsu et al. (1999). It is in essence also an enumeration algorithm, applied to networks in which the in-degree of each node is at most 2.

One of the disadvantages of the Boolean network modeling framework is the need to discretize real-valued expression data into an ON/OFF scheme, which loses a large amount of information. In response to this deficiency, multi-state discrete models and hybrid models have been developed. The most complex one (Thomas, 1991; Thieffry et al., 1995; Thieffry and Thomas, 1998) uses multiple states for the genes in the network corresponding to certain thresholds of gene expression that make multiple gene actions possible. The authors are most interested in the modeling and function of feedback loops. The model includes a mixture of multi-valued logical and real-valued variables, as well as the possibility of asynchronous updating of the variables. While this modeling framework is capable of better capturing the many characteristics of gene regulatory networks than Boolean networks, it also introduces substantially more computational complications from a reverse-engineering point of view. In particular, the ability of asynchronous update adds orders of magnitude to the combinatorial explosion of possibilities. Multiple discrete expression levels were also used in the reverse-engineering method in Repsilber et al. (2002), which uses genetic algorithms to explore the parameter space of multistage discrete genetic network models.

In Brazma and Schlitt (2003) a hybrid modeling framework was introduced that tries to capture discrete as well as continuous aspects of gene regulation. The authors' finite state linear model has a Boolean network type control component, as well as linear functions that represent an environment of substances that change their concentrations continuously. According to the authors, this framework can be generalized to more than two states for the logical variables. The reverse-engineering algorithm described in Brazma and Schlitt (2003) is based essentially on enumeration of all possible functions that fit a given data set, and no algorithmic criterion for model selection is given.

Finally, it is worth mentioning a Boolean network approach that incorporates stochastic features of gene regulation. Probabilistic Boolean networks have been introduced in Schmulevich et al. (2002). So far, no algorithmic reverse-engineering methods have been given for this modeling framework. While this survey of existing reverse-engineering methods is by no means comprehensive, it provides a context for the new method proposed in this paper. For reviews of other existing methods the reader can consult, e.g., Bower and Bolouri (2001) or de Jong (2002).

While continuous and discrete modeling approaches may seem disjoint, it is useful to keep in mind that most ODE models cannot be solved analytically and that numerical solutions of such systems involve the approximation of the time-continuous system by a time-discrete one. Furthermore, when validating an ODE model using microarray data, it is often necessary to utilize thresholds for continuous concentrations. The connection between an ODE system and an associated discrete system that captures information about the continuous dynamics has been explored, e.g., in Lewis and Glass (1991) within an artificial neural network setting. In Muraille et al. (1996) a discrete network model is used to obtain information about the dynamics of an ODE model for the regulation of immune response. So, in the end, the two modeling paradigms might not be as far apart as it appears and could serve complementary roles. The modeling framework we describe in the next section is of the discrete, multi-state variety. We propose here an approach that describes a regulatory network as a time-discrete, multi-state dynamical system, synchronously updated. We make an additional assumption on the number of states each node in the network can take on. This assumption allows us to use techniques and algorithms from computational algebra. As mentioned earlier, it is difficult for any reverse-engineering method to be validated using a real system as a test case, firstly because a suitable data set might not be available and, secondly, model predictions might be hard to verify without extra experiments. For this reason, most methods discussed above have used simulated data for validation. We have chosen to use a recent Boolean network model for segment polarity development in *D. melanogaster* (Albert and Othmer, 2003), consisting of twelve cells with sixteen nodes per cell, representing genes and gene products. The model is sufficiently complex to be of interest, but small enough so we can compare in detail our reverse-engineered network with the real one, thereby illustrating the performance and limitations of our method.

3. An algebraic approach to reverse engineering

As mentioned above, we adopt the modeling framework of time-discrete multi-state dynamical systems. We

start with a network on n nodes. In the context of gene regulatory networks, each node can represent a gene. Let S be the set of possible states of any node of the network, and assume that S is a finite (but possibly large) set. One should think of S as a set of discretized expression levels: for instance $S = \{-1, 0, 1\}$. (We will address the issue of data discretization in Section 5.) Let

$$F : S^n \to S^n$$

be a time-discrete dynamical system over S of dimension n. Then F can be described in terms of its coordinate functions $f_i : S^n \to S$, for $i = 1, \ldots, n$. That is, if $\mathbf{s} = (s_1, \ldots, s_n) \in S^n$ is a state, then $F(\mathbf{s}) = (f_1(\mathbf{s}), \ldots, f_n(\mathbf{s}))$. We refer to time-discrete dynamical systems over finite state sets as *finite dynamical systems*.

The reverse-engineering problem we are focusing on is one of model selection and can be stated as follows.

Given one or more time series of state transitions, generated by a biological system with n varying quantities, say mRNA concentrations, choose a finite dynamical system $F : S^n \to S^n$, which reproduces the time series and "best describes" the biological system. To be precise, we presume that we are given a set of state transitions of the network, in the form of one or more time series:

$$\begin{aligned} &\mathbf{s}_1 = (s_{11}, s_{12}, \ldots, s_{1n}), \quad \ldots \quad , \mathbf{s}_m = (s_{m1}, \ldots, s_{mn}), \\ &\mathbf{t}_1 = (t_{11}, t_{12}, \ldots, t_{1n}), \quad \ldots \quad , \mathbf{t}_r = (t_{r1}, \ldots, t_{rn}). \\ &\qquad\qquad\qquad \ldots \end{aligned}$$

These satisfy the property that, if the unknown transition function of the network is F, then

$$\begin{aligned} F(\mathbf{s}_i) &= \mathbf{s}_{i+1} \quad \text{for } i = 1, \ldots, m-1, \\ F(\mathbf{t}_j) &= \mathbf{t}_{j+1} \quad \text{for } j = 1, \ldots, r-1, \\ &\ldots \end{aligned}$$

where $\mathbf{s}_i, \mathbf{t}_j, \ldots \in S^n$ for $i = 1, \ldots, m$, $j = 1, \ldots, r, \ldots$. Typically, there will be more than one possible choice. In fact, unless **all** state transitions of the system are specified, there will be more than one network that fits the given data set. Since this much information is hardly ever available in practice, any reverse-engineering method has to choose from a large set of possible network models. In the absence of a good understanding of the properties and characteristics of gene regulatory networks, one is limited to some type of Occam's Razor principle for model selection. Before describing the principle we use, we first need to describe our computational framework.

We now make the further assumption that our state set S is chosen so that the number of elements is a power of a prime number. This can be easily accomplished by an appropriate choice of data discretization. One consequence then is that S can be given the structure of a *finite field* (Lidl and Niederreiter, 1997), that is, a finite number system. One possible approach is to choose a prime number p of possible states, in which case the number system can be taken to be $\mathbb{Z}/p$, the integers modulo p. It follows (see, e.g., Lidl and Niederreiter, 1997, p. 369) that each of the coordinate functions of F can be expressed as a polynomial function in n variables, with coefficients in S, and so that the degree of each variable is at most equal to the number of elements in S.

Example 1. Boolean functions can be represented as polynomial functions with coefficients in $\mathbb{Z}/2 = \{0, 1\}$. Indeed, if x and y are Boolean variables, then we have the following translations:

$$\begin{aligned} &x \wedge y \coloneqq xy, \\ &x \vee y \coloneqq x + y + xy, \\ &\neg x \coloneqq x + 1, \end{aligned}$$

where $\wedge$ represents the *AND* operation, $\vee$ the *OR* operation, and $\neg$ the *NOT* operation. Note that addition does not correspond to the logical *OR* function, but rather the exclusive *OR* function, denoted by *XOR*.

As observed above, the model F we are searching for is determined by its coordinate functions $f_i : S^n \to S$. We can reverse engineer each coordinate function independently and thus reconstruct the network one node at a time. Our algorithm is very similar in outline to that in Yeung et al. (2002), described earlier. We first compute the space of all networks that are consistent with the given time series data. We then choose a particular network $F = (f_1, \ldots, f_n)$ that satisfies the following property:

Criterion 2 (Criterion for model selection). *For each i, f_i is minimal in the sense that there are no non-zero polynomials $g, h \in S[x_1, \ldots, x_n]$[1] such that $f_i = h + g$, and g is identically equal to zero on the time series.*

In essence, we exclude terms in the polynomials f_i that vanish on the given data set. No reverse-engineering method is able to detect such terms without prior information, even though they may exist in the real network but be non-functional under the particular experimental conditions. Note that this criterion is different from that used in Yeung et al. (2002) and also by REVEAL. In both of those algorithms the search is for the sparsest network, that is, a network such that each node takes inputs from as few variables as possible. In terms of the coordinate functions, the basic mathematical reverse-engineering problem is then as follows, formulated for one time series. The statement of the problem for several time series is similar.

[1] $S[x_1, \ldots, x_n]$ is the set of polynomials in the variables $x_1, \ldots, x_n$ with coefficients from S. In fact, this set is closed under addition and multiplication of polynomials and has the algebraic structure of a ring.

Problem 3. *Suppose we are given a time series of states in* S^n:

$$\mathbf{s}_1 = (s_{11}, \ldots, s_{1n}), \ldots, \quad \mathbf{s}_m = (s_{m1}, \ldots, s_{mn}).$$

(1) *For each* $i \in \{1, \ldots, n\}$, *find all polynomial functions* $f_i \in S[x_1, \ldots, x_n]$ *that send a state* $\mathbf{s}_j$ *to the i-th coordinate of the next state*; *that is*,

$$f_i(\mathbf{s}_j) = s_{j+1,i} \quad \text{for } j = 1, \ldots, m-1. \tag{1}$$

(2) *From that set of functions*, *choose one that does not contain any terms that are identically equal to zero at all time points* (*which is unique*; *see Appendix* A).

The key advantage of the polynomial modeling framework over a finite field is that there is a well-developed algorithmic theory that provides mathematical tools for the solution of this problem which have polynomial time complexity in the number of network variables. Thus, we can overcome one disadvantage that discrete models have compared to ODE models, for which there is a well developed mathematical theory. This feature has been an important reason for the use of polynomial systems over finite fields in engineering, in particular control theory (see, e.g., Marchand and Le Borgne, 1998).

3.1. Reverse-engineering algorithm

We first outline the algorithm, and then give a detailed description of each step.

Input: *A time series* $\mathbf{s}_1, \ldots, \mathbf{s}_m \in S^n$ *of network states*.

Algorithm.

(1) *Compute a particular solution* f_i^0 *for each node* $i = 1, \ldots, n$.
(2) *Compute the set* I *of all functions that vanish on the time series*.
(3) *Compute the reduction* f_i *of* f_i^0 *with respect to* I.

Output: *A polynomial function* $f_i \in S[x_1, \ldots, x_n]$ *for each node* $i = 1, \ldots, n$, *such that* $f_i(\mathbf{s}_j) = s_{j+1,i}$ *for* $j = 1, \ldots, m-1$, *and such that* f_i *does not contain component polynomials that vanish on the time series*.

We fix one node/gene, say node i, in the network and reverse-engineer its transition function. Given a time series $\mathbf{s}_1, \ldots, \mathbf{s}_m$, as above, we are looking for all polynomial functions $f_i \in S[x_1, \ldots, x_n]$ that satisfy Equation (1). We accomplish this by first computing a particular polynomial f_i^0 that satisfies the equation. There are several methods to do this, Lagrange interpolation being one of them. We use the following formula, based on the so-called Chinese Remainder Theorem (see, e.g., Lang, 1971, p. 63):

$$f_i^0(\mathbf{x}) = \sum_{j=1}^{m-1} s_{j+1,i} r_j(\mathbf{x}),$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ and the polynomials r_j are defined as follows. For each $1 \leqslant j \neq k < m$ such that $\mathbf{s}_j \neq \mathbf{s}_k$, find the first coordinate l in which they differ. Define

$$b_{jk}(\mathbf{x}) = (s_{j,l} - s_{k,l})^{p-2}(x_l - s_{k,l})$$

for every $j \neq k$ and set

$$r_j(\mathbf{x}) = \prod_{k=1}^{m-1} b_{jk}(\mathbf{x}).$$

We note that $r_j(\mathbf{s}_j) = 1$ and $r_j(\mathbf{x}) = 0$ otherwise. It is straightforward to check that this polynomial function does indeed interpolate the given time series.

Suppose there are two polynomial functions $f_i, h_i \in S[x_1, \ldots, x_n]$ that interpolate the state transitions for node i. Then

$$f_i(\mathbf{s}_j) = s_{j+1,i} = h_i(\mathbf{s}_j)$$

for each time step $j = 1, \ldots, m-1$, so that $(f_i - h_i)(\mathbf{s}_j) = 0$ for all j. Therefore, any two such functions differ by a function that is identically equal to 0 on the given time series. So, in order to find all functions that fit the data, we need to find all functions that vanish on the given time points. Let I be the set of all polynomials in the ring $S[x_1, \ldots, x_n]$ which are zero on the time series; we call such polynomials *vanishing polynomials*. See Appendix A for an algorithm to find I. Then we obtain the set of all possible interpolating functions for node i as

$$f_i^0 + I = \{f_i^0 + g : g \in I\}.$$

Note that this representation is very similar to the way one computes all solutions to a non-homogeneous system of linear equations, where f_i^0 represents a particular solution to the system, and I is the space of solutions of the corresponding homogeneous system.

The next step is to reduce the polynomial f_i^0 modulo I; that is, we divide f_i^0 by the vanishing polynomials. So we can write f_i^0 as

$$f_i^0 = f_i + g$$

with $g \in I$. Furthermore, f_i is minimal in the sense that it cannot be further decomposed into $f' + g'$ with $g' \in I$. In other words, g represents the part of f_i^0 that lies in I and hence is identically equal to 0 on the given time series. In fact, f_i is the *reduction* of f_i^0 and is the output of the algorithm for node i.

Example 4. We illustrate the algorithm with a small artificial gene network. Consider mRNA expression

levels for the genes g_1, g_2, and g_3 over the course of 5 time steps:

Time	g_1	g_2	g_3
1	1.6104	1.2042	1.0072
2	1.7073	1.3252	1.0185
3	1.7254	1.4118	1.0336
4	1.7011	1.4616	1.0508
5	1.6601	1.4814	1.0685

Suppose we choose two thresholds for each gene so that the data can be discretized into three categories: 1 for an increase in expression, 0 for no change in expression, and-1 for a decrease in expression. Then we have the following discrete time series of states in $\mathbb{Z}/3$:

Time	g_1	g_2	g_3
1	−1	−1	−1
2	1	0	−1
3	1	0	0
4	0	1	1
5	0	1	1

Using the formula given in Step 1 of the algorithm, a particular solution f_i^0 for each gene represented by x_i, for $i = 1, 2, 3$, is

$$f_1^0 = x_1^2x_3 - x_1^2 + x_1x_3 + x_1,$$
$$f_2^0 = -x_1^2x_3 + x_1^2 - x_1x_3 - x_1 + 1,$$
$$f_3^0 = -x_1^2x_3 - x_1^2 - x_1x_3 + x_1 + 1.$$

The set of functions in $\mathbb{Z}/3[x_1, x_2, x_3]$ which vanish on the discrete time series is given by

$$I = \langle x_1 + x_2 - 1, x_2x_3 - x_3^2 + x_2 - x_3, \\ x_2^2 - x_3^2 + x_2 - x_3 \rangle.$$

(For an explanation of the notation, see Appendix A.) The reduction of each particular solution by the set I yields the model

$$f_1 = -x_3^2 + x_3,$$
$$f_2 = x_3^2 - x_3 + 1,$$
$$f_3 = -x_3^2 + x_2 + 1.$$

For this computation, the variable ordering $x_1 > x_2 > x_3$ was used.

3.2. Complexity of the algorithm

Recall that m is the number of time points, n is the number of nodes, and p is the size of the state set S. For each step of the algorithm in Section 3.1, we analyze its complexity.

(1) According to the above algorithm, the particular solution f_i^0 for each node $i = 1, \ldots, n$ is composed of the r_j polynomials, where each r_j is a product of the b_{jk} polynomials for $1 \leqslant k \neq j < m$. There are $O(m^2)$ b_{jk} polynomials, each requiring $O(n)$ comparisons of network states. We assume that these comparisons require unit time, $O(1)$. Furthermore, there are $O(m)$ polynomials r_j, each a product of $O(m)$ linear factors. So, the complexity of computing f_i^0 is as follows:

$$\begin{aligned} &\textit{complexity of } f_i^0 \\ &\quad = (\textit{complexity of } 1\ r_j) \times (\#r_j\text{'s}) \\ &\quad = (\textit{complexity of } 1\ b_{jk} \times \#b_{jk}\text{'s in } 1\ r_j) \times O(m) \\ &\quad = O(n)O(m)O(m) = O(nm^2). \end{aligned}$$

Hence the complexity of computing a particular solution is $O(n^2m^2)$.

(2) The ideal of vanishing polynomials is computed by an application of the Buchberger–Möller algorithm (Abbott et al., 2000). The complexity of this step is

$$O(m^2(g(m,n) + m)(\log p)^2 + m^2n^2)$$

as reported in Abbott et al. (2000), where $g(m,n)$ is $O(m^{(n-1)/n})$ (Berman, 1981). Since g is sublinear in m, the worst case complexity of this step is quadratic in the number of nodes and cubic in the number of time points (Robbiano, 1998).

(3) Dubé et al. (1986) show that the complexity of reducing a polynomial by an ideal requires $O(2^{cd}L)$ time, where c is a constant, d is the degree of the polynomial being reduced, and L is the number of terms in the polynomial. Because each f_i^0 is a sum of polynomials of degree m (the r_j's) and the computations take place in a finite field, the degree of any particular solution is at most m, the number of time points, reducing the complexity to $O(2^{cm}L)$. To estimate L note that the polynomials r_j are products of $m-1$ linear factors of the form $x_i - c$, up to constants. After multiplying, the number of terms is $O(2^{m-1})$, so that the number of terms in f_i^0 is $O((m-1)2^{m-1})$. To reduce all particular solutions therefore requires $O(n(m-1)2^{cm+m-1})$ time.

Note: While the complexity of the reduction step is exponential in the number of time points m, in practice the number of time points will be small, compared to the number of network nodes.

In summary, the complexity of the algorithm is

$$O(n^2m^2) + O(m^3 + m)(\log p)^2 + m^2n^2) \\ + O(n(m-1)2^{cm+m-1}).$$

It is quadratic in the number n of variables and exponential in the number m of time points.

As explained in Appendix A, Gröbner basis calculations require the choice of an ordering of the terms in the polynomials, in particular an ordering of the variables. This is necessary in order to carry out long

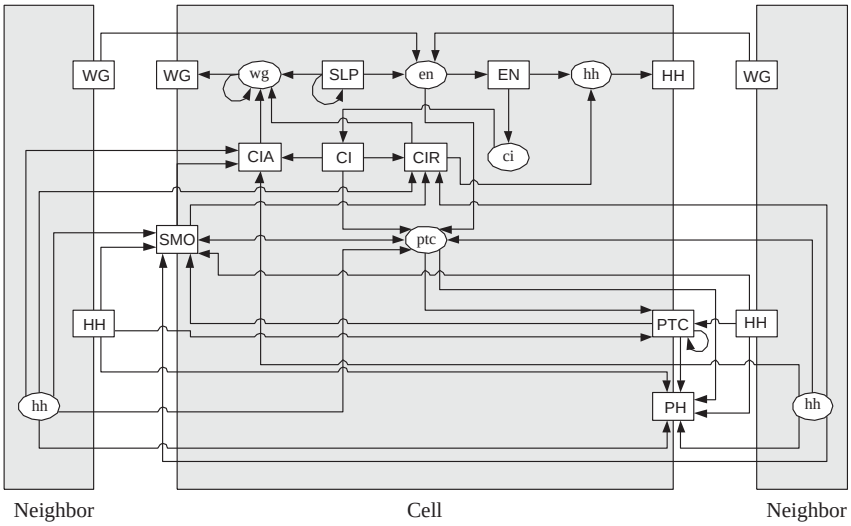

Fig. 1. The graph of interactions in one cell of $\mathcal{N}$, with cellular interactions. Ovals = mRNAs, rectangles = proteins. SLP denotes 2 forkhead domain transcription factors encoded by the gene *sloppy paired* and is believed to activate the segment polarity genes depicted in the model (Cadigan et al., 1994). PH is the protein complex formed by the binding of HH to PTC (Ingham and McMahon, 2001). The protein SMO is encoded by the gene *smoothened*. Because its transcription is not regulated by any molecular species in the model, *smoothened* is not represented.

division of multivariate polynomials. The output of the algorithm depends on the term ordering chosen, in the sense that "cheaper" variables, that is, those that are smallest in the ordering, will be used preferentially in both the interpolation as well as the reduction calculations. With prior information about the existence of certain regulatory relationships this feature can be used to incorporate this prior knowledge into the algorithm. (Likewise, using the Elimination Theorem discussed in Appendix A, one can exclude certain variables from appearing.) In general, however, this dependence makes the interpretation of the precise form of the functions difficult at times. In those cases we resort to building a consensus model using a collection of variable orderings that makes each variable "equally costly on average". We can then extract from these different models common regulatory links and common non-linear terms. In the next section we illustrate this strategy with a detailed discussion of an example.

4. Validation of the method

In this section, we present an application of the proposed method to a well-characterized network of segment polarity genes responsible for pattern formation in the *D. melanogaster* embryo. In Albert and Othmer (2003) a Boolean model, which we will denote by $\mathcal{N}$, was presented for the regulatory interactions in the network of five segment polarity genes and their products, in which the functions governing state transitions are Boolean functions. The network consists of a ring of 12 interconnected cells, grouped into 3 parasegment primordia, in which the genes are expressed in every fourth cell. The model was validated using published gene and protein expression data. Our goal is to reverse engineer the Boolean network $\mathcal{N}$, whose wiring diagram is depicted in Fig. 1, from time series generated by the Boolean functions, including null mutant time series for each of the five genes in the network. A subset of the functions is given in Table 1. Note that for our purposes, it is irrelevant whether the Boolean model is indeed correct.

The genes represented in the Albert–Othmer model are *wingless* (*wg*), *engrailed* (*en*), *hedgehog* (*hh*), *patched* (*ptc*), and *cubitus interruptus* (*ci*). Also included in the Boolean model are the products encoded by these 5 genes, as well as 2 other compounds (*smoothened* protein, denoted by SMO, and *sloppy-paired* proteins denoted as one compound SLP), constituting 15 distinct molecular species. Fig. 1 depicts the graph of connections in the Boolean model. In the graph, nodes represent mRNAs and proteins. An edge between nodes indicates that the node at the tail is involved in the regulation of the head node. For example, an edge $A \to B$ between protein nodes A and B implies that A regulates the synthesis of B, whereas an edge $A \to b$ from protein A to mRNA b implies that A regulates the synthesis of b, that is, the transcription of gene b. Note that edges denote the existence of regulation, not the type, whether activation or inhibition. Table 1 lists some of the Boolean functions that accompany the model in Fig. 1.

We consider a network of 60 nodes (15 molecular species $\times$ 4 cells). The starting point for our reverse-engineering algorithm is a collection of time series of

Table 1
Sample Boolean functions for the network $\mathcal{N}$

$f_6 = hh_i^{t+1} = EN_i^t \wedge \neg CIR_i^t$
$f_7 = HH_i^{t+1} = hh_i^t$
$f_8 = ptc_i^{t+1} = CIA_i^{t+1} \wedge \neg EN_i^{t+1} \wedge \neg CIR_i^{t+1}$
$f_9 = PTC_i^{t+1} = ptc_i^t \vee (PTC_i^t \wedge \neg HH_{i-1}^t \wedge \neg HH_{i+1}^t)$

Superscripts denote time and subscripts location relative to cell i, $i = 1, \ldots, 12$.

Table 2
Polynomial representations of the sample Boolean functions in Table 1

$f_6 = x_5(x_{15} + 1)$
$f_7 = x_6$
$f_8 = x_{13}((x_{11} + x_{20} + x_{11}x_{20}) + x_{21} + (x_{11} + x_{20} + x_{11}x_{20})x_{21})(x_4 + 1)$
$\quad (x_{13}(x_{11} + 1)(x_{20} + 1)(x_{21} + 1) + 1)$
$f_9 = x_8 + x_9(x_{18} + 1)(x_{19} + 1) + x_8x_9(x_{18} + 1)(x_{19} + 1)$

To account for simultaneous updating, we substituted simultaneously updated terms with their function expressions (e.g. in f_8 we replaced CIA_i^{t+1} with $CI_i^t \wedge (SMO_i^t \vee hh_{i-1}^t \vee hh_{i+1}^t)$. See Table 6 for the list of variable names.

discrete expression data. We used the Boolean initializations for the wildtypes published in Albert and Othmer (2003) for the 5 genes and generated times series using the Boolean functions. All the initializations terminate in steady states when evaluated by the Boolean functions in Table 5 in Appendix B. Table 2 contains the translations of the Boolean functions in Table 1 into polynomial functions. (See Table 6 in Appendix B for all other polynomials and a legend of variable names.) Since each cell of the ring is assumed to have the same network of segment polarity genes, we report our findings of the reconstruction of the network in one cell.

The size of the state space of the Boolean network is 2^{60}, involving multiple components. Any single trajectory in that space vastly underdetermines the network. Therefore we include knock-out time series for each gene in the network. Altogether we use 6 time series: one for the wildtype and one for each gene knock-out. As the length of each time series is at most 8 time steps, constituting a total of 42 data points, the data still comprises only a minuscule fraction of the state space, less than $(4 \times 10^{-15})\%$ of 2^{60}.

To simulate an experiment in which node x_i representing a gene is knocked out, we set its corresponding update function f_i in Table 6 to 0 and kept all other functions the same. When applicable, we also set the corresponding functions in neighboring cells equal to 0. For example, to simulate the knocking out of the hedgehog gene, we set $f_6 = 0, f_{20} = 0$, and $f_{21} = 0$, where f_{20} and f_{21} are the functions associated to the gene in neighboring cells. We also set the i-th entry, corresponding to the initial mRNA concentration for x_i, in the wildtype initialization to 0. For each knock-out, we generated a new time series, which also ended in a steady state, by iteration of the modified initialization.

Recall that we first construct all polynomial models (sets of polynomial functions) which fit a given time series. Then we select the one which is minimal with respect to summands in each of the functions which evaluate to 0 on all time points. The algorithm relies on the choice of a total ordering for all possible terms in the given variables, in particular, a total ordering of the variables themselves. The effect of such a variable ordering is that the "cheaper" variables, those that are smaller in this ordering, are used preferentially in the algorithm. In order to counteract this dependency, we use a consensus model extracted from four different choices of variable ordering: the ordering $x_1 < \cdots < x_{21}$, the reverse ordering $x_1 > \cdots > x_{21}$, and two additional orderings which make "interior" variables greatest and least. The variables $x_1, \ldots, x_{21}$ denote all interacting molecular species in a particular cell. There are a number of computational algebra packages which can be used for our purposes, such as *Macaulay 2* (Grayson and Stillman) and CoCoA (CoCoaTeam). For all computations discussed in this section, we used *Macaulay 2*.

Using these 4 orderings, we performed the following experiments. We first ran the algorithm to generate a set of polynomial models that fit the wildtype time series. Then, we incorporated the knock-out time series that we generated from the Boolean functions. In both experiments we intersected the models, one for each term order, to obtain a "consensus" model. If we use the 8 time points of expression data for the wildtype to construct a discrete polynomial model, our method predicts the network in one cell to have 20 links, of which 14 are intracellular (each cell of $\mathcal{N}$ has 44 links). The performance of the algorithm dramatically improves, however, if we incorporate knock-out time series for the five genes, which we demonstrate below.

The consensus model we construct from the wildtype and knock-out mutants results in prediction of 37 of the 44 links in one cell of the Boolean network (see Fig. 2). We correctly identified the molecular species which regulate the transcription of genes *wg* and *ci* and the synthesis of proteins SLP, WG, EN, HH, PTC, and CI. We summarize the model of interactions and compare it to the model in Fig. 1. We will discuss reconstruction of certain dynamic properties at the end of the section.

In determining which species affect the transcription of the gene *hh*, we found a function that involves fewer terms than the polynomial representation of its counterpart in the Boolean model. Specifically, the function is in terms of the gene product EN. It satisfies all time series, including knock-out data, generated by the corresponding Boolean function. The discrepancy lies in the existence of a term in the Boolean function which does

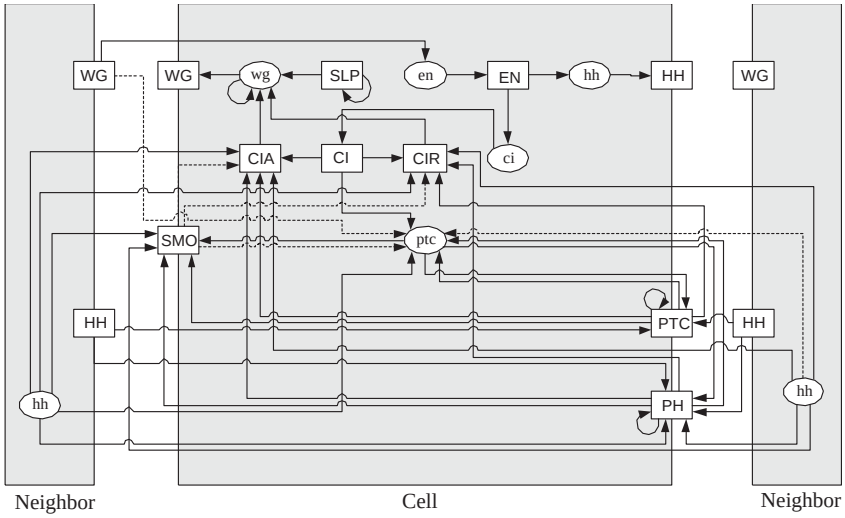

Fig. 2. The graph of the consensus model. Solid lines are links that appear for all 4 variable orders, whereas dashed lines are links that appear for 3 of the 4 variable orders.

not contribute to the updating of *hh*, that is, a term that is constant equal to 0 on all input time points. However, links whose effects are not reflected in the given data are not detectable by any reverse-engineering method unless prior information about the link is given. Similarly, the Boolean function for *en* also contains such terms, which accounts for differences in regulation dependencies. We infer that transcription of the gene *en* is regulated by Wingless protein in an anterior cell.

In this model, transcription of the *patched* gene to synthesize *ptc* mRNA in a given cell is regulated by the following molecular species: the proteins PTC, PH, SMO, CI and WG in the anterior cell, as well as *hh* from neighboring cells. We were not able to conclude that *en* regulates the transcription of the gene.

For the protein complex PH, we detected that its synthesis is regulated by itself and by *ptc*, *hh*, and their corresponding products in adjacent cells. Recall that PH is the molecule formed when HH from adjacent cells binds to the transmembrane receptor PTC. In Albert and Othmer (2003) the authors assume in their model that this binding occurs instantaneously since it is known that the reaction occurs faster than transcription or translation (which they also presuppose to require 1 time unit for completion). Therefore, we infer that the synthesis of PH at a given time step is determined by the expression levels of *ptc*, *hh* and *hh* product, from neighboring cells, in the previous time step. However, we did not identify PTC as a direct regulator.

We identified the gene *ptc*, its product, the complex PH, and extracellular *hh* to affect synthesis of SMO. The binding of PTC to SMO inactivates it via post-translational modification (Ingham, 1998). However, if PTC binds to HH, then SMO is activated (Ingham, 1998). So the translation of *smoothened* mRNA is dependent on the synthesis of the complex PH. (The mRNA *smoothened* was not included in the model since the gene is transcribed ubiquitously throughout the segment (Ingham, 1998) and its transcription is not regulated by any biochemical in the model.) Since this binding of PTC and SMO is assumed to occur instantaneously, as described above, we instead detect the regulation by *hh* in adjacent cells and not its product.

Next, we focus on reverse engineering the dynamics of the Boolean model, that is, the Boolean functions on each of the network nodes. As pointed out above, the functions in the Boolean model contain terms that evaluate to 0 on all input data, and so we are unable to detect the corresponding relationships. In order to compare the dynamics predicted by our method with the Boolean model we reduce the polynomials in Table 6 by removing vanishing terms, as described in the previous section. Note, however, that this reduction depends on the choice of term ordering, as does the output of our algorithm. For each choice of term ordering, the reduced functions of the Boolean model and the functions reverse engineered from the data for each cell agree exactly (see Table 3 for sample polynomials; see Table 7 for all polynomials). This shows that we are able to completely predict the reduced Boolean model's dynamics from the wildtype and knock-out data.

However, due to the dependence of our method on the chosen term ordering, the particular form of the reverse-engineered functions may not be directly interpretable with respect to regulatory relationships. We therefore proceed to extract information about network dynamics from terms common to the reverse-engineered functions for the multiple term orderings used.

Table 3
Sample Boolean functions reduced by the ideal of wildtype and knock-out time series

$f_6 = x_5$
$f_7 = x_6$
$f_8 = x_{12}x_{13} + x_{13}x_{16} + x_{13}x_{20} + x_{18}x_{20} + x_{13}x_{21} + x_{19}x_{21} + x_{13} + x_{18} + x_{19}$
$f_9 = x_{10}x_{14} + x_{14}x_{18} + x_{14}x_{19} + x_9x_{20} + x_{18}x_{20} + x_9x_{21} + x_{19}x_{21} + x_8 + x_9 + x_{10}$

For each term ordering, the model constructed only from the wildtype is linear. Using the 4 term orderings mentioned above, we found 19 terms consisting of a single variable, in which 10 are true positives. These terms, which we call "single interactions", account for 77% of the linear terms in the Boolean network for one cell of the ring. However, the degrees of the polynomial functions in $\mathcal{N}$ range from 1 to 6. Incorporating knock-out data yields more comprehensive results, highlighted in the following discussion.

In all models built from the knock-out time series, there are 18 linear terms. Of these, 12 are in one cell of $\mathcal{N}$, accounting for 92% of the linear terms present. Specifically, the linear terms in the functions for *hh* and for all the proteins, excluding the complex PH and the transcriptional forms CIA and CIR of the protein CI, were completely identified. In three of the four models, we found 21 linear terms, of which 12 are in one cell of $\mathcal{N}$.

As distinct from the models built from wildtype data only, there are non-linear terms in the models from the knock-out data. We call non-linear terms "cooperative interactions". For the protein SMO, we found that its synthesis depends on the cooperative interaction between the genes *ptc* and extracellular *hh*. Specifically, the terms $ptc_i{*}hh_{i-1}$ and $ptc_i{*}hh_{i+1}$ appear in the polynomial function for SMO for all term orders used, of which both appear in the corresponding Boolean function. In the function describing transcription of *wg*, the term $\mathrm{SLP}_i{*}\mathrm{CIA}_i$ is common to all models and $wg_i{*}\mathrm{CIR}_i$ appears in three of the four models. Both of these products are terms in the Boolean function for *wg*. In fact we found 3 non-linear terms common to all models. In three of the four models, there are 11 non-linear terms, of which 8 are in $\mathcal{N}$, accounting for 27% of the quadratic terms. While $\mathcal{N}$ contains polynomial functions of degree as high as 6 involving 77 superquadratic terms, our method did not find interactions of degree higher than 2. A summary of these results is displayed in Table 4.

Our method shows a marked improvement when knock-out data are included. We were able to reconstruct approximately 84% of the topology of the Boolean network, versus only 32% when knock-out data are not included. Further, we correctly identified 12 single interactions and 8 cooperative interactions, versus none in the model constructed with only wildtype data.

Table 4
Performance of dynamics detection for one cell of $\mathcal{N}$

Total single interactions in $\mathcal{N}$		13
Total cooperative interactions in $\mathcal{N}$		30
Single interactions	4 TO	3 TO
Total predicted	18	21
True positives	12	12
False positives	6	9
Cooperative interactions	4 TO	3 TO
Total predicted	3	11
True positives	3	8
False positives	0	3

Single interactions = degree-one terms; cooperative interactions = degree-two terms. 4 TO denotes results for all 4 term orders used, whereas 3 TO denotes results for any 3 of the 4 term orders used.

5. Data discretization

Since gene expression data are real numbers, the first step in any reverse-engineering algorithm using discrete models must be to discretize these real numbers into a finite (typically small) set of possible states. For Boolean networks this simply amounts to the choice of a single threshold for the expression level of each gene, below which a gene is considered inactive. The issue of data discretization has been studied, in particular from the points of view of Bayesian network applications and machine learning; see, e.g., Dougherty et al. (1995) and Friedman and Goldszmidt (1996).

Obviously, the way one discretizes the data plays an important role in what model one obtains. The first important choice is the number of discrete states allowed. In our case, this is the choice of p in the algorithm. It follows from results in Green (2003) that for p large enough the result of our reverse-engineering algorithm does not depend on p, in the sense that the terms in the polynomials remain the same, possibly with different coefficients. While that paper does not give an algorithm for choosing a suitable p, extensive experiments with networks simulated with the biochemical network simulation program *Gepasi* (GEPASI; Mendes, 1997) suggest that for data sets up to 50 nodes an optimal p is in the range between 5 and 13. The effect of the choice of p is demonstrated in Figs. 3 and 4. This example also demonstrates that the availability of more than two discrete states can be very helpful for modeling purposes.

There are various ways to attach a discrete label to real-valued data. Thresholds with biological relevance is one type of labeling that can be used. For example, up-regulation, lack of change, and down-regulation of a gene are two thresholds that can partition a given data

set into 3 groups, with labels $1, 0, -1$, respectively. This maps the real-valued data into values in $\mathbb{Z}/3$. More thresholds can be integrated to refine the partitioning of the data set. Another method of discretization is to normalize the expression of each gene or protein and use the deviation from the mean to discretize the data.

6. Effect of noise on the algorithm

In the previous section, it was shown that different levels of resolution result in sharp differences in the discrete time series. To eliminate this effect, a suitably large prime should be considered (Green, 2003). Furthermore, data collected from experimental processes typically have errors introduced due to deficiencies in methodology or imperfections in instrumentation, as is the case in microarray data. In addition, biological systems, such as biochemical pathways, exhibit variability in population or concentration levels. For our purposes it is important to quantify this noise.

Because our method begins by discretizing real-valued time series, one would expect that discretization will smooth out some of the noise. To test this hypothesis, we incorporated random noise, using a normal distribution, into time series generated from 100 simulated networks generated by the AGN software described in Mendes et al. (2003). We generated 25 networks with 20 nodes for each of the 4 topologies supported by AGN. All time series consist of 11 time steps. We added noise in two ways: first, as a percentage of each data point, simulating biological variation, and second, as a percentage of a fixed value, simulating instrumentation error. In both cases, we added approximately 10% and 25% noise, respectively, where 10% (25%) noise means we switched 1 (2) entries in each state tuple. We chose 3 different thresholds (0.5, 1.0, and 2.5) with which to Booleanize all time series. Then we computed the median number of entries that were changed in the noisy time series for both 10% and 25% noise, as compared to the original time series.

The median number of entries that changed for all experiments is 5 ($\approx$2.3% of 220). In the case when noise is a percentage of each data point, the median is 2 ($\approx$1% of 220), when the threshold was restricted to 0.5 or 2.5. For the experiments described above, we infer that $\approx$10% of the noise added is propagated to the discrete time series.

To test for the effect of noise on our method, we added 1% noise to the discrete time series for the Boolean functions in Albert and Othmer (2003). (We estimate this to correspond to approximately 10% noise in the "real" data, given the above results.) We then applied our method to the noisy time series and used 8 variable orderings to construct a consensus model. For 6 of the nodes (SLP, *wg*, WG, *en*, HH, and CI) we were able to identify all of the correct links, plus some false positives. For 5 of the nodes (*ptc*, PTC, SMO, CIR, and CIA), we identified half of the correct links. For the remaining nodes, we had no conclusive results.

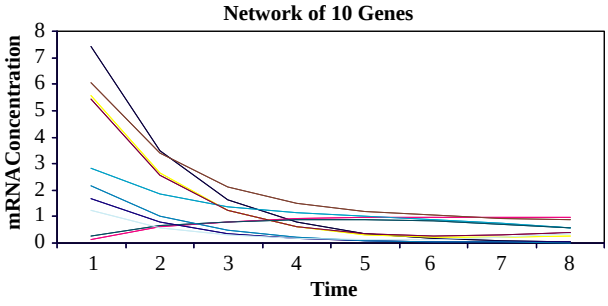

Fig. 3. The graph of a real-valued time series for a network $\mathcal{G}$ of 10 genes, where each curve is a time series of mRNA concentrations for one gene of the network.

7. Discussion and future work

We have described a new method to discover regulatory relationships between the nodes in a gene regulatory network from time series of experimental data. The modeling framework is that of time-discrete

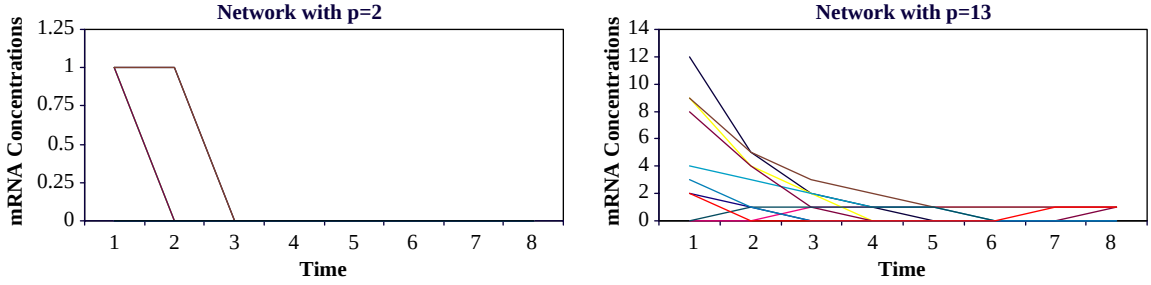

Fig. 4. The graphs of discrete time series for $\mathcal{G}$ for fixed primes: $p = 2$ (left) and $p = 13$ (right).

dynamical systems with a finite (but possibly large) state set for the variables. The crucial further assumption is that the number of possible states for a variable is equal to a power of a prime number, so that one can impose an algebraic structure on the set. As a consequence, we have available to us the very well-developed theory of algorithmic polynomial algebra, with a variety of implemented procedures. It is this machinery that we employ for the task of reverse engineering. In giving up a bit of freedom in the choice of state sets, we gain a mathematical framework well suited for reverse-engineering purposes. It is important to observe, however, that the assignment of states to finite "numbers" is arbitrary, and no order relation between states is maintained by the algebraic operations.

Our method is very similar to that in Yeung et al. (2002), in that we first compute all possible networks that fit the given data. We do this not by enumeration, but rather by a procedure similar to describing all solutions of a non-homogeneous system of linear equations. Then, as in Yeung et al. (2002), we choose a particular network. Our selection criterion is to choose polynomial functions that do not contain any terms which are identically equal to 0 on the given data set. So, we choose a minimal network, not in terms of network connections like in Yeung et al. (2002) and Liang et al. (1998), but rather in terms of the structure of the functions. The rationale is that if a term vanishes on the data input, then no reverse-engineering method should be able to identify it without prior biological knowledge. In fact, one way to interpret the reduced *Drosophila* model in Table 7 is as representative of the network complexity reflected in the available data set. The additional complexity of the "real" system cannot be recovered without additional data.

The advantages of our method are that we can explore the whole model space for a data set, without using enumeration techniques. Furthermore, we have an algorithmic selection criterion to choose a model. And, finally, our models are multi-state, which allows the choice of a resolution to fit the characteristics of the data set.

We contrast our method with the discrete methods described in the introduction. In its essence, the Thieffry-Thomas model can also be represented as a function from a finite set of states to itself, with dynamics generated by iteration of this function. The lack of any further mathematical structure in the model makes its analysis very difficult. Moreover, this modeling framework does not lend itself to mathematical reverse-engineering methods. Even in the much simpler case of Boolean networks, all the algorithms discussed above rely at some point on enumeration of a large number of possible components of the putative network.

For method validation we used the Boolean model of the *D. melanogaster* segment polarity network recently published in Albert and Othmer (2003). We generated time series and perturbed time series from this model and used them as input for the algorithm. Of the 44 links in the model for a particular cell, we correctly identify 37, together with 13 links not present in the model. For some of the links that our method identifies incorrectly, we provide evidence as to possible reasons. Furthermore, we are able to identify some of the key non-linear relations among variables in the model of a cell in the ring.

In the absence of good approximation methods over finite fields, our method does exact fitting of data and is consequently quite sensitive to noise. As shown, the introduction of 10% noise leads to a significant deterioration of algorithm performance. A detailed study of the effect of noise on algorithm performance is beyond the scope of the present paper. Furthermore, in order to incorporate robustness to noisy data, we are developing an expanded method based on a genetic algorithm that optimizes data fit and model complexity. It uses the algorithm presented here to compute a starting model that has exact data fit, but possibly excessive model complexity. An efficient description of the model space makes use of sophisticated results from the theory of Gröbner bases. This method will also remedy another weakness, namely the dependence of the algorithm on the chosen term order, which prevents us from obtaining a full dynamic model.

Another advantage of the computational algebra framework is that it allows for a systematic study of the suitability of a given data set for network identification, and we are in the process of carrying out such a study. Finally, a more systematic study of different data discretization methods and their effect on noise reduction will also be carried out. Further validation will include application to real-valued data generated from both simulated networks and published models that generate real-valued data, before applying it to experimental data, where the exact network is not known a priori.

Acknowledgements

The research in this paper was partly supported by NSF grants DMS-0138323 and DMS0083595, and NIH grant RO1GM068947-01. The authors thank J. Shah for the implementation of some of the algorithms discussed in this paper, and O. Colón-Reyes, A. de la Fuente, L. García, E. Green, P. Mendes, E. Nordberg, J. Snoep, M. Stillman, and B. Sturmfels for helpful discussions. Finally, the authors are grateful to the anonymous referees for going beyond "the call of

duty" in their extensive comments that resulted in significant improvements.

Appendix A. Mathematical background

In this section we give the mathematical details of the reverse-engineering algorithm and some basic facts about computational algebra relevant to this paper. The basic problem that lies at the heart of computational algebra is that long division of multivariate polynomials differs from that of univariate polynomials in that the remainder is not uniquely determined. It depends on several choices that need to be made along the way. Univariate division of a polynomial f by another one, g, proceeds by dividing the highest power of the variable in g into the highest power in f. For multivariate polynomials there are many choices of ordering the terms of a polynomial, which affects the outcome of the division. Also, when dividing more than one polynomial into a given one, the outcome typically depends on the order in which the division is carried out. To be precise, let S be a field, e.g. the field of real or complex numbers, or a finite field, and let $f, f_1, \ldots, f_m \in S[x_1, \ldots, x_n]$ be polynomials in the variables $x_1, \ldots, x_n$. The question whether there are polynomials $g_1, \ldots, g_m \in S[x_1, \ldots, x_n]$ such that

$$f = \sum_{i=1}^{m} g_i f_i$$

can in general not be answered algorithmically. In the language of abstract algebra, let $I = \langle f_1, \ldots, f_m \rangle$ be the *ideal* in $S[x_1, \ldots, x_n]$ generated by the f_i, that is, I is the set of all linear combinations $\sum g_i f_i$, with $g_i \in S[x_1, \ldots, x_n]$. Then we are asking whether f is an element of I. This is known as the *ideal membership problem*.

In general, the ideal I can be generated by sets of polynomials other than the f_i, similar to a vector space possessing many different bases, and the solution to the ideal membership problem does not depend on the choice of a particular generating set. It turns out that if one chooses a very special type of generating set, known as a *Gröbner basis*, for the ideal, then the ideal membership problem becomes solvable algorithmically. There is a basic algorithm to compute a Gröbner basis for an ideal, the Buchberger algorithm. Using this algorithm as a foundation, many problems in multivariate computational algebra can be solved algorithmically, including the computation of intersections of ideals, which is the key computation we are using in our algorithm. The central ingredient in this algorithm is the Elimination Theorem (see Cox et al., 1997, p. 113). This theorem is also the source of the very high computational complexity of many algorithms, since it requires the use of a computationally expensive type of term ordering. For details see, e.g., Cox et al. (1997).

As described earlier, the basic idea of our algorithm is as follows. First we choose a term order for $S[x_1, \ldots, x_n]$. This is necessary for all subsequent calculations. To compute the ideal I of all polynomial functions that are identically equal to 0 on a given collection $\{\mathbf{s}_i\}$ of points we proceed as follows. Let I_i be the ideal of all polynomials that take on the value 0 on $\mathbf{s}_i$. It is straightforward to see that I_i contains the ideal

$$\langle x_1 - s_{i1}, \ldots, x_{1n} - s_{in} \rangle.$$

But this ideal is maximal with respect to inclusion, so that it has to be equal to I_i. Then the ideal I is equal to the intersection

$$I = \bigcap_{i=1}^{m} I_i.$$

Finally the reduction of the special solution f_0 modulo I is simply the remainder of f_0 under division by a reduced Gröbner basis of I. This too is a standard algorithm implemented in most computer algebra systems. It is important to observe that, if g_0 is another particular solution to the interpolation problem, then f_0 and g_0 differ by a polynomial in I, as shown before. Therefore, the reduction of f_0 by I is equal to the reduction of g_0. Consequently, it does not matter which method we use to construct a particular solution.

Appendix B. Tables

Boolean functions for the network in one cell of the ring is given in Table 5 and their polynomial representation is given in Table 6. Boolean functions reduced by

Table 5
Boolean functions for the network $\mathcal{N}$ in one cell of the ring

$$f_1 = SLP_i^{t+1} = \begin{cases} 0, & \text{if } i \bmod 4 = 1 \text{ or } i \bmod 4 = 2 \\ 1, & \text{if } i \bmod 4 = 3 \text{ or } i \bmod 4 = 0 \end{cases}$$
$$f_2 = wg_i^{t+1} = (CIA_i^t \wedge SLP_i^t \wedge \neg CIR_i^t) \vee (wg_i^t \wedge (CIA_i^t \vee SLP_i^t) \wedge \neg CIR_i^t)$$
$$f_3 = WG_i^{t+1} = wg_i^t$$
$$f_4 = en_i^{t+1} = (WG_{i-1}^t \vee WG_{i+1}^t) \wedge \neg SLP_i^t$$
$$f_5 = EN_i^{t+1} = en_i^t$$
$$f_6 = hh_i^{t+1} = EN_i^t \wedge \neg CIR_i^t$$
$$f_7 = HH_i^{t+1} = hh_i^t$$
$$f_8 = ptc_i^{t+1} = CIA_i^{t+1} \wedge \neg EN_i^{t+1} \wedge \neg CIR_i^{t+1}$$
$$f_9 = PTC_i^{t+1} = ptc_i^t \vee (PTC_i^t \wedge \neg HH_{i-1}^t \wedge \neg HH_{i+1}^t)$$
$$f_{10} = PH_i^{t+1} = PTC_i^{t+1} \wedge (HH_{i-1}^{t+1} \vee HH_{i+1}^{t+1})$$
$$f_{11} = SMO_i^{t+1} = \neg PTC_i^{t+1} \vee HH_{i-1}^{t+1} \vee HH_{i+1}^{t+1}$$
$$f_{12} = ci_i^{t+1} = \neg EN_i^t$$
$$f_{13} = CI_i^{t+1} = ci_i^t$$
$$f_{14} = CIA_i^{t+1} = CI_i^t \wedge (SMO_i^t \vee hh_{i-1}^t \vee hh_{i+1}^t)$$
$$f_{15} = CIR_i^{t+1} = CI_i^t \wedge \neg SMO_i^t \wedge \neg hh_{i-1}^t \wedge \neg hh_{i+1}^t$$

Table 6
Polynomial representations of the Boolean functions in Table 4, together with the legend of variable names

$$f_1 = x_1$$
$$f_2 = (x_{15}+1)(x_1x_{14}+x_2(x_1+x_{14}+x_1x_{14})+x_1x_2x_{14}(x_1+x_{14}+x_1x_{14}))$$
$$f_3 = x_2$$
$$f_4 = (x_{16}+x_{17}+x_{16}x_{17})(x_1+1)$$
$$f_5 = x_4$$
$$f_6 = x_5(x_{15}+1)$$
$$f_7 = x_6$$
$$f_8 = x_{13}((x_{11}+x_{20}+x_{11}x_{20})+x_{21}+(x_{11}+x_{20}+x_{11}x_{20})x_{21})(x_4+1) + (x_{13}(x_{11}+1)(x_{20}+1)(x_{21}+1)+1)$$
$$f_9 = x_8+x_9(x_{18}+1)(x_{19}+1)+x_8x_9(x_{18}+1)(x_{19}+1)$$
$$f_{10} = (x_8+x_9(x_{18}+1)(x_{19}+1)+x_8x_9(x_{18}+1)(x_{19}+1))(x_{20}+x_{21}+x_{20}x_{21})$$
$$f_{11} = x_8+x_9Y+x_8x_9Y+1+x_{20}+((x_8+x_9Y+x_8x_9Y+1)x_{20})+x_{21} + (x_8+x_9Y+x_8x_9Y+1+x_{20}+(x_8+x_9Y+x_8x_9Y+1)x_{20})x_{21}$$
$$f_{12} = x_5+1$$
$$f_{13} = x_{12}$$
$$f_{14} = x_{13}((x_{11}+x_{20}+x_{11}x_{20})+x_{21}+(x_{11}+x_{20}+x_{11}x_{20})x_{21})$$
$$f_{15} = x_{13}(x_{11}+1)(x_{20}+1)(x_{21}+1)$$

SLP_i	wg_i	WG_i	en_i	EN_i	hh_i	HH_i	ptc_i
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
PTC_i	PH_i	SMO_i	ci_i	CI_i	CIA_i	CIR_i	
x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	
WG_{i-1}	WG_{i+1}	HH_{i-1}	HH_{i+1}	hh_{i-1}	hh_{i+1}	Y	
x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	$(x_{18}+1)(x_{19}+1)$	

The subscript i denotes a particular cell of the ring.

Table 7
Boolean functions reduced by the ideal of wildtype and knock-out time series

$$f_1 = x_1$$
$$f_2 = x_1x_{14}+x_2x_{14}+x_2x_{15}+x_2$$
$$f_3 = x_2$$
$$f_4 = x_{16}$$
$$f_5 = x_4$$
$$f_6 = x_5$$
$$f_7 = x_6$$
$$f_8 = x_{12}x_{13}+x_{13}x_{16}+x_{13}x_{20}+x_{18}x_{20}+x_{13}x_{21}+x_{19}x_{21}+x_{13}+x_{18}+x_{19}$$
$$f_9 = x_{10}x_{14}+x_{14}x_{18}+x_{14}x_{19}+x_9x_{20}+x_{18}x_{20}+x_9x_{21}+x_{19}x_{21}+x_8+x_9+x_{10}$$
$$f_{10} = x_{10}x_{14}+x_{14}x_{18}+x_{14}x_{19}+x_8x_{20}+x_8x_{21}+x_{10}+x_{18}+x_{19}$$
$$f_{11} = x_8x_{20}+x_9x_{20}+x_{18}x_{20}+x_8x_{21}+x_9x_{21}+x_{19}x_{21}+x_8+x_9+x_{18}+x_{19}+1$$
$$f_{12} = x_5+1$$
$$f_{13} = x_{12}$$
$$f_{14} = x_{11}x_{13}+x_9x_{20}+x_{18}x_{20}+x_9x_{21}+x_{19}x_{21}+x_{10}+x_{18}+x_{19}$$
$$f_{15} = x_{11}x_{13}+x_9x_{20}+x_{18}x_{20}+x_9x_{21}+x_{19}x_{21}+x_{10}+x_{13}+x_{18}+x_{19}$$

the ideal of wildtype and knock-out time series are given in Table 7.

References

Abbott, J., Bigatti, A., Kreuzer, M., Robbiano, L., 2000. Computing ideals of points. J. Symbolic Comp. 30, 341–356.

Akutsu, T., Miyano, S., Kuhara, S., 1999. Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. In: Altman, R.B., Lauderdale, K., Dunker, A.K., Hunter, L., Klein, T.E. (Eds.), Proceedings of the Pacific Symposium Biocomputation, Vol. 4, World Scientific, Singapore, pp. 17–28.

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

Berman, D., 1981. The number of generators of a colength N ideal in a power series ring. J. Algebra 73, 156–166.

Bower, J., Bolouri, H., 2001. Computational Modeling of Genetic and Biochemical Networks. The MIT Press, Cambridge, MA.

Brazma, A., Schlitt, T., 2003. Reverse engineering of gene regulatory networks: a finite state linear model. Preprint. Available at http://genomebiology.com/2003/4/6/P5.

Cadigan, K., Grossniklaus, U., Gehring, W., 1994. Localized expression of sloppy paired protein maintains the polarity of Drosophila parasegments. Genes Dev. 8, 899–913.

Cho, R., Campbell, M., Winzeler, E., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T., Gabrielan, A., Landsman, D., Lockhart, D., Davis, R., 1998. A genome-wide transcriptional analysis of the mitotic cell cycle. Mol. Cell 2, 65–73.

CoCoATeam CoCoA: a system for doing Computations in Commutative Algebra. Available at http://cocoa.dima.unige.it.

Cox, D., Little, J., O'Shea, D., 1997. Ideals, Varieties, and Algorithms. Springer, New York.

de Jong, H., 2002. Modeling and simulation of genetic regulatory systems: a literature review. J. Comput. Biol. 9, 67–103.

de la Fuente, A., Brazhnik, P., Mendes, P., 2002. Linking the genes: inferring quantitative gene networks from microarray data. Trends Genet. 18, 395–398.

Dougherty, J., Kohavi, R., Sahami, M., 1995. Supervised and unsupervised discretization of continuous features. In: Prieditis, A., Russell, S. (Eds.), Machine Learning: Proceedings of the 12th International Conference, Morgan Kauffman, San Francisco, CA.

Dubé, T., Mishra, B., Yap, C., 1986. Admissible orderings and bounds on Gröbner normal form algorithm. it NYU Computer Science Technical Report.

Filkov, V., Istrail, S., 2002. Inferring gene transcription networks: the Davidson model. Genome Informatics 13, 236–239.

Filkov, V., Skiena, S., Zhi, J., 2002. Analysis techniques for microarray time-series data. J. Comput. Biol. 9, 317–330.

Friedman, N., Goldszmidt, M., 1996. Discretization of continuous attributes while learning Bayesian networks. In: Saitta, L. (Ed.), Proceedings of the 13th International Conference on Machine Learning, Morgan Kauffman, San Francisco, CA, pp. 157–165.

Friedman, N., Linia, M., Nachman, I., Pe'er, D., 2000. Using Bayesian networks to analyze expression data. J. Comput. Biol. 7, 601–620.

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

GEPASI, . Available at http://www.gepasi.org.

Grayson, D., Stillman, M. Macaulay 2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

Green, E., 2003. On polynomial solutions to reverse-engineering problems. Preprint.

Hartemink, A., Gifford, D., Jaakkola, S., Young, R., 2001. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. Pacific Symposium on Biocomputation, World Scientific, Singapore.

Ingham, P., 1998. Transducing hedgehog: the story so far. EMBO J. 17, 3505–3511.

Ingham, P., McMahon, A., 2001. Hedgehog signaling in animal development: paradigms and principles. Genes Dev. 15, 3059–3087.

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

Kitano, H., 2002. Systems biology: a brief overview. Science 295, 1662–1664.

Lang, S., 1971. Algebra. Addison-Wesley, Reading, MA.

Lewis, J.E., Glass, L., 1991. Steady states, limit cycles, and chaos in models of complex biological networks. Int. J. Bifurcation Chaos 1, 477–483.

Liang, S., Fuhrman, S., Somogyi, R., 1998. REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. Pac. Symposium Biocomputation, Vol. 3, World Scientific, Singapore, pp. 18–29.

Lidl, R., Niederreiter, H., 1997. Finite Fields, Encyclopedia of Mathematics and its Applications 20, 2nd Edition. Cambridge University Press, Cambridge.

Marchand, H., Le Borgne, M., 1998. Partial order control of discrete event systems modeled as polynomial dynamical systems. In: IEEE International Conference on Control Applications, Trieste, Italy.

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

Mendes, P., Sha, W., Ye, K., 2003. Artificial gene networks for objective comparison of analysis algorithms. Bioinformatics 19, 122–129.

Mendoza, L., Thieffry, D., Alvarez-Buylla, E., 1999. Genetic control of flower morphogenesis in *Arabidopsis thaliana*: a logical analysis. Bioinformatics 15, 593–606.

Muraille, E., Thieffry, D., Leo, O., Kaufman, M., 1996. Toxicity and neuroendocrine regulation of the immune response: a model analysis. J. Theor. Biol. 183, 285–305.

Pe'er, D., Regev, A., Elidan, G., Fridman, N., 2001. Inferring subnetworks from expression profiles. Bioinformatics 17, 215–224.

Repsilber, D., Liljenström, H., Andersson, S., 2002. Reverse engineering of regulatory networks: simulation studies on a genetic algorithm approach for ranking hypotheses. Biosystems 66, 31–41.

Robbiano, L., 1998. Gröbner bases and statistics. In: Buchberger, B., Winkler, F. (Eds.), Gröbner Bases and Applications. Cambridge University Press, New York.

Schmulevich, I., Dougherty, E., Kim, S., Zhang, W., 2002. Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. Bioinformatics 18, 261–274.

Soinov, L.A., Krestyaninova, M.A., Brazma, A., 2003. Towards reconstruction of gene networks from expression data by supervised learning. Genome Biol. 4, R6.

Spellman, P., Sherlock, G., Zhang, M., Iyer, V., Anders, K., Eisen, M., Brown, P., Botstein, D., Futcher, B., 1998. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. Mol. Biol. Cell 9, 3273–3297.

Thieffry, D., Thomas, R., 1998. Qualitative analysis of gene networks. Proceedings of the Pacific Symposium on Biocomputing, World Scientific, Singapore, pp. 77–88.

Thieffry, D., Thomas, R., Kaufman, M., 1995. Dynamical behaviour of biological regulatory networks—I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. Bull. Math. Biol. 57, 247–276.

Thomas, R., 1991. Regulatory networks seen as asynchronous automata: a logical description. J. Theor. Biol. 153, 1–23.

Yeung, M., Tegnér, J., Collins, J., 2002. Reverse engineering gene networks using singular value decomposition and robust regression. Proc. Natl Acad. Sci. 99, 6163–6168.