Learning Genetic Regulatory Network Connectivity from Time Series Data

Nathan A. Barker, Chris J. Myers, and Hiroyuki Kuwahara

Abstract—Recent experimental advances facilitate the collection of time series data that indicate which genes in a cell are expressed. This information can be used to understand the genetic regulatory network that generates the data. Typically, Bayesian analysis approaches are applied which neglect the time series nature of the experimental data have difficulty in determining the direction of causality, and do not perform well on networks with tight feedback. To address these problems, this paper presents a method to learn genetic network connectivity which exploits the time series nature of experimental data to achieve better causal predictions. This method first breaks up the data into bins. Next, it determines an initial set of potential influence vectors for each gene based upon the probability of the gene's expression increasing in the next time step. These vectors are then combined to form new vectors with better scores. Finally, these influence vectors are competed against each other to determine the final influence vector for each gene. The result is a directed graph representation of the genetic network's repression and activation connections. Results are reported for several synthetic networks with tight feedback showing significant improvements in recall and runtime over Yu's dynamic Bayesian approach. Promising preliminary results are also reported for an analysis of experimental data for genes involved in the yeast cell cycle.

Index Terms—Learning influences, genetic regulatory networks, time series data, graphical models.

1 Introduction

Recent experimental advances allow cellular activities to be studied with a variety of techniques providing vast amounts of data about cellular interactions [1], [2], [3]. A variety of techniques have been developed to reason about genetic regulatory networks that generate this data. One common approach is to use *Bayesian* networks [4], [5], [6], [7]. One recent work by Sachs et al. successfully applied Bayesian networks to intracellular multiparameter flow cytometry [8]. Bayesian networks, however, have several limitations. First, Bayesian approaches often have difficulty determining the direction of causality. Another limitation is that Bayesian networks rely on statistical analysis, so they do not perform well with small amounts of data. Also, they rely primarily on correlational data and only use time to separate the data points. Finally, they can only be applied to networks that are acyclic, which is a major limitation since feedback control is a crucial component of genetic regulatory networks.

Dynamic Bayesian Networks (DBN) are an extension of Bayesian networks which incorporate an aspect of time to allow networks with cycles to be found. They have been used with some success [9], [10], [6], [7]. There have also been extensions which use prior biological knowledge, small data sets, a class of DBNs called state-space models, and even some that incorporate transcription factor binding location data to try and improve the results [11], [12], [13], [14]. One interesting work is that by Yu et al. that combines a DBN search with a local analysis [15]. This method, however, has several limitations. First, it begins with an expensive (exponential in the number of genes) search for a best fit network. This approach also neglects the time series nature of the data. Finally, while this method is evaluated on several networks, only two of the networks have any cycles (i.e., feedback), and these cycles are several genes long. We feel these networks do not fully capture the regulation that occurs in genetic regulatory networks.

This paper describes the `GeneNet` algorithm which improves upon Yu et al.'s method in several ways. First, it avoids the expensive global analysis, and, instead, focuses just on the local analysis to determine influences. Next, it leverages the time series nature of the data by looking at two sequential time points to determine the probability of an increase in gene expression. Finally, it targets learning genetic networks with cyclic or tight feedback behavior.

2 Regulatory Systems Overview

A genetic network for part of the phage λ decision circuit is shown in Fig. 1 [16]. These networks are constructed from DNA. The DNA includes the *genes*, *cI* and *cII*, that are the blueprint for producing the *proteins*, CI and CII. The DNA also includes regions called *promoters*. Our example includes two promoters, P_R and P_{RE}, represented as bent arrows indicating the direction of transcription. *Transcription* begins when an enzyme called *RNA polymerase* (RNAP) binds to the promoter. As the RNAP molecule proceeds, it forms a complementary strand called *messenger RNA* (mRNA). When the mRNA molecule comes into contact

- *N.A. Barker is with the Department of Computer Science and Information Systems, Southern Utah University, Cedar City, UT 84720. E-mail: barkern@suu.edu.*
- *C.J. Myers is with the Electrical and Computer Engineering Department, University of Utah, Salt Lake City, UT 84112. E-mail: myers@ece.utah.edu.*
- *H. Kuwahara is with Microsoft Research, Centre for Systems and Computational Biology, University of Trento, Trento, Italy. E-mail: kuwahara@cosbi.eu.*

Manuscript received 19 Aug. 2008; revised 13 Mar. 2009; accepted 23 Apr. 2009; published online 4 May 2009.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2008-08-0150. Digital Object Identifier no. 10.1109/TCBB.2009.48.

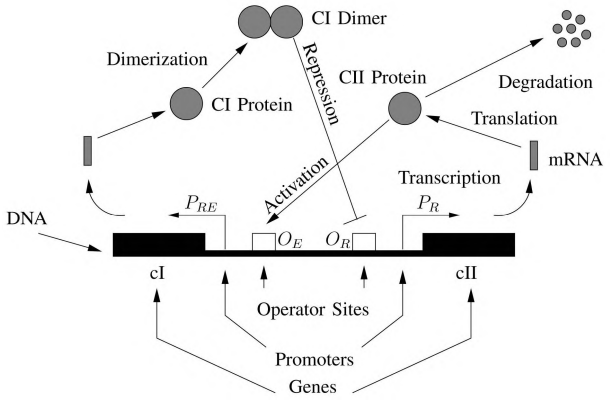

Fig. 1. A portion of a simple genetic network.

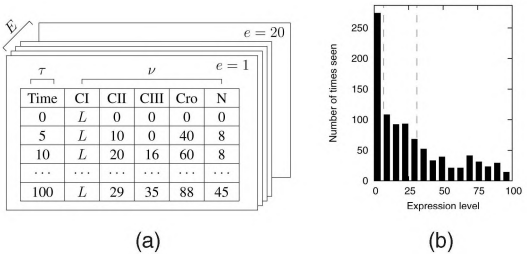

Fig. 2. (a) Example time series data. (b) Time series data values and levels assigned for the CIII species.

with a *ribosome*, it *translates* the mRNA into the protein that is coded for by the gene. A gene is said to be *expressed* if it is being actively transcribed, indicated by the presence of mRNA synthesized from that gene. Proteins, known as *transcription factors*, can bind to operator sites to either *activate* or *repress* transcription. There are two *operator* sites, O_E and O_R, in this example. The CII protein must bind to the O_E operator site to activate transcription of the *cI* gene. Without the CII protein, CI is only produced at a low *basal* rate. The CI protein after dimerizing can bind to the O_R operator to repress production of CII.

3 Time Series Experiments

The goal of this work is to determine the influences between species (typically proteins) in a set S by examining a set of time series data experiments, E, over these species. These data may be collected, for example, using *microarrays* which can monitor gene expression data for thousands of genes simultaneously [17], [18], [19]. Each data point in E, is a 3-tuple $\langle e, \tau, \nu \rangle$, where $e \in \mathbb{N}$ is a natural number representing the experiment number, $\tau \in \mathbb{R}$ is the time at which the species values were measured, and $\nu \in (\mathbb{R} \cup \{L\} \cup \{H\} \cup \{-\})^{|S|}$ is the state of each species $s \in S$. L and H are values that represent that a species is mutated low and high, respectively, in that data point. The symbol "-" represents an unknown value. The notation $\nu(s)$ denotes the value of species s for that data point. The notation $|E|$ is used to indicate the total number of data points within all of the experiments in E. The set of experimental data point successor pairs is defined formally as:

$$SUCC = \{(\langle e, \tau, \nu \rangle, \langle e', \tau', \nu' \rangle) \mid \langle e, \tau, \nu \rangle \in E \wedge \langle e', \tau', \nu' \rangle \in E \wedge (e = e') \wedge (\tau < \tau') \wedge \neg \exists \langle e, \tau'', \nu'' \rangle \in E.(\tau < \tau'') \wedge (\tau'' < \tau')\}.$$

Fig. 2a shows an example set of time series data from the genetic network for the phage λ decision circuit. The set of species in this figure is $S = \{CI, CII, CIII, Cro, N\}$. There are 20 experiments, and each experiment has 21 data points (i.e., $|E| = 420$). An example data point is $\langle 1, 100, \nu \rangle$, where $\nu = \langle L, 29, 35, 88, 45 \rangle$. Note that L represents that species CI is mutated to a low state in that data point.

Our method discretizes the data from the time series experiments into a small number of *bins*, n. Our results indicate that an n of three or four bins per species performs the best. For simplicity, this paper discretizes each species into the same number of bins, but the user can specify different numbers of bins for each species. The bins are defined by the *levels* that separate them. Formally, the levels, θ, for each species, $s \in S$, are $\langle \theta_0(s), \ldots, \theta_n(s) \rangle$, where $\theta_0(s)$ is 0, and $\theta_n(s)$ is ∞. These levels group the data into a small set of bins Φ, where $\Phi_j(s) = [\theta_j(s), \theta_{j+1}(s))$. The lowest bin for a species is $\Phi_0(s)$ and the highest bin is $\Phi_{n-1}(s)$. A *bin assignment*, $b \in \{0, .., n-1, *\}^{|S|}$, assigns each $s \in S$ to a bin. The notation $b(s)$ indicates the bin assignment for species s in b. Note that a bin assignment of "$*$" for s indicates that there is no bin assignment to s. A bin assignment that includes $*$s is called a *partial bin assignment*. The bin for a "$*$" is defined by $\Phi_*(s) = [0, \infty)$. The merger or union of two partial bin assignments, denoted $b \cup b'$, is accomplished by performing the union for each species as follows:

$$b(s) \cup b'(s) = \begin{cases} b(s), & \text{if } b'(s) = *, \\ b'(s), & \text{otherwise.} \end{cases}$$

Note that this is only defined if $\forall s.(b(s) = b'(s) \vee b(s) = * \vee b'(s) = *)$. Also, $b(s)++$ increments the bin assignment of species s to the next highest bin. Likewise $b(s)--$ decrements the bin assignment of species s to the next lowest bin. A set of bin assignments is denoted by B.

Discretizing the data can be done in many ways. The simplest approach is to divide the range of the data into equal size bins. To improve statistical significance, a better approach is to divide the data up into bins containing equal amounts of data. To calculate the amount of data that should be in each bin if the data is evenly divided, one divides the total number of data points by the number of bins. One can then place the levels such that each bin contains nearly the same amount of data. In Fig. 2b, the dashed lines at 7 percent and 31 percent for the CIII species in the phage λ decision circuit divide the data so that each bin contains roughly the same amount of data. Fig. 3 shows the `DetermineLevels` function which divides the data so that each bin contains nearly equal amounts of data. This function first sorts the data by value and puts the lowest and highest levels to 0 and ∞, respectively. Next, it determines how many data points should optimally be in the next bin. It then walks the sorted data until it finds that many data points and assigns a bin to the next highest value. It then redivides the remaining data and starts again. For example, if there are 100 data points, and 50 of them are at 0, then there should be a bin for 0 which includes all

```
DetermineLevels(Species Set S,
    Expts E, Number of levels n)
  foreach s ∈ S
    stack ES := sort(E, s)
    θ_j(s) := 0
    θ_n(s) := ∞
    for j := 1 to n − 1
      θ_j(s) := ∞
      for k := 0 to |ES| / (n−j+1)
        ⟨e, τ, ν⟩ := pop(ES)
      do ⟨e', τ', ν'⟩ := pop(ES)
      while (ν(s) = ν'(s) ∧ |ES| > 0)
      if (|ES| ≠ 0) then
        θ_j(s) := ν'(s)
        push(ES, ⟨e', τ', ν'⟩)
  return θ
```

Fig. 3. The `DetermineLevels` function.

50 data points, and the remaining two bins should optimally have 25 data points in them.

Once the levels are selected, individual time series data points can be assigned to bins using the `BA` function shown in Fig. 4. The `BA` function takes a time series data point, the set of species, the levels for the species, and the total number of bins. It selects a species and sets the bin for that species to 0 if the species is mutated low in that data point, or sets the bin to $n-1$ if the species is mutated high, or sets the bin to the middle bin if the value is unknown. If the species value did not match one of these cases, the bin is determined by first testing if the expression value for that species is below level 1. If the species' expression value is below that level, it falls into bin 0. If the expression value is greater than this value, the `BA` function uses the next highest level, until it finds the correct bin for that species. It does the same thing for each species in S. It then returns the created bin assignment.

Although discretizing the data reduces the range of the data, the algorithms presented in this paper examine this data often, so a more compact and efficient representation is needed. In particular, data points that map to the same bin assignment can be combined. For each combined bin assignment, our method records both the number of data points that map to it as well as the number of times that each species expression value increases in the next time point or has increased from the previous time point. This *compressed data* (CD) structure is defined formally as $CD : S \rightarrow (\{0, .., n-1\}^{|S|} \rightarrow \{\mathbb{N} \times \mathbb{N}\})$. The CD is computed using the `Compress` function shown in Fig. 4. The `Compress` function maps a species $s \in S$ and a bin assignment to a tuple representing the number of times this bin assignment is seen in the data and the number of times species s increases in the next time point. The notation CD_s is shorthand for $CD(s)$ and refers to the mapping for species s. The notation $CD_s(b) = \langle \mathbb{N} \times \mathbb{N} \rangle$ refers to the mapping of a species and bin assignment to the tuple described above. The `Compress` function works as follows: First, for every experimental time point and species, s, the CD_s is initialized. Note that the function does not allocate

```
BA(Data ν, Species set S,
    Levels θ, Number of bins n)
  foreach s ∈ S
    if ν(s) = L then b(s) = 0
    else if ν(s) = H then b(s) = n − 1
    else if ν(s) = − then b(s) = (n−1)/2
    else for i := 1 to n − 1
        if ν(s) < θ_i(s) then
          b(s) := i − 1
          break
  return b

Compress(Expts E, Species Set S,
    Levels θ)
  foreach ⟨e, τ, ν⟩ ∈ E
    foreach s ∈ S
      CD_s(BA(ν, S, θ, n)) := ⟨0, 0⟩
  foreach (⟨e, τ, ν⟩, ⟨e', τ', ν'⟩) ∈ SUCC
    if (ν(s) ∉ {L, H, −} ∧ ν' ∉ {L, H, −}) then
      foreach s ∈ S
        if (τ < τ' ∧ ν(s) < ν'(s)) then
          CD_s(BA(ν, S, θ, n)) :=
            CD_s(BA(ν, S, θ, n)) + ⟨1, 1⟩
        else
          CD_s(BA(ν, S, θ, n)) :=
            CD_s(BA(ν, S, θ, n)) + ⟨0, 1⟩
  return CD
```

Fig. 4. The `BA` and `Compress` functions.

all possible bin assignments, but only records those bin assignments that appear in the data, as, in general, the state space is sparse. This limits the size of the CD to be in the worst case the same size as the experimental data itself (i.e., $|E|$) if, in the unlikely event that every row in the experimental data has a different bin assignment. Next, the algorithm checks for every element of $SUCC$ and species, s, that the data entry for s is not mutated or empty. It then checks if the species value is increasing between time points, and then the seen and increasing values are incremented for the associated bin assignment. If the species expression value is not increasing, then only the seen value is incremented.

The later algorithms often use partial bin assignments. In this case, it is useful to project the compressed data onto the species $S' \subseteq S$, where S' is the set of species with specified values in the partial bin assignment. The *projected compressed data* (PCD) structure is defined formally as $PCD_s : (\{0, .., n-1, *\}^{|S'|} \rightarrow \{\mathbb{N} \times \mathbb{N}\})$. The PCD_s is computed using the `Project` function shown in Fig. 5. The `Project` function maps a bin assignment consisting of a subset of the species, $S' \subseteq S$, to the increasing and occurrence information for a particular species s. The first thing the function does is to look through the domain of the CD_s and initialize each of these entries in the function with zeros. It uses the `PBA` function, shown in Fig. 5, to remove unwanted species from the new bin assignment. The `PBA` function creates a new bin assignment with only the species that occur in the species set S'. The `Project`

```
PBA(BA b, Species Sets S, S')
  foreach s ∈ S
    if s ∈ S' then b'(s) := b(s) else b'(s) := *
  return b'

Project(Comp. Data CD, Species s,
    Species Sets S, S')
  foreach b ∈ domain(CD_s)
    PCD_s(PBA(b, S, S')) := ⟨0, 0⟩
  foreach b ∈ domain(CD_s)
    PCD_s(PBA(b, S, S')) :=
      PCD(PBA(b, S, S')) + CD_s(b)
  return PCD_s
```

Fig. 5. The `PBA` and `Project` functions.

function then looks back through the data and calculates the PCD_s by summing all the entries that correspond to this partial bin assignment for species s.

4 Influence Vectors

Our method represents the connections between species in a genetic network using *influence vectors*. An influence vector, i, is a vector over the species, S, that describes the types of influences that each individual species, $s \in S$, has on a particular species, c. The possible types of influence between species are activation "a," repression "r," no influence "n," or unknown influence "?". An influence is directed from a *parent* to a *child* in that the presence of a parent species has the described influence over the production of a child species. In a genetic network, it is possible (and likely) that multiple parent species may have an influence on a child species. Therefore, the influences between species is represented using a function that returns a vector over the set of species that indicates the type of influence that each species has over a given child species (i.e., $i : S \rightarrow \{a, r, n, ?\}$). A genetic network is represented using a collection of influence vectors, $\mathcal{I}$ (i.e., $\mathcal{I} : S \rightarrow (S \rightarrow \{a, r, n, ?\})$). The notation $\mathcal{I}(c)$ returns an influence vector, i, which indicates the parent species for child species c. The notation $i(s)$ returns the influence that parent s has in influence vector i. The function, $Act(i)$, returns the activating species in i (i.e., those species where $i(s)$ returns "a"). The function, $Rep(i)$, returns the repressing species in i (i.e., those species where $i(s)$ returns "r"). The function $Par(i)$ returns the parents in i (i.e., $Act(i) \cup Rep(i)$). The size of influence vector, i, is indicated by, $|i|$ (i.e., $|Par(i)|$). The merger or union of two influence vectors is denoted $i \cup i'$.

Consider a possible influence vector for the species CIII in the five-species phage λ decision circuit of $i = \langle r, ?, n, r, a \rangle$, where the species ordering is $\langle CI, CII, CIII, Cro, N \rangle$. This influence vector indicates that species CI and Cro repress species CIII's expression, species CII has an unknown influence on CIII, species CIII has no influence on itself, and species N activates CIII's expression.

When first constructing an influence vector, often no information is known. This means that all values in each influence vector, $i \in \mathcal{I}$, are set to "?". For a given network of interest, biologists may have discovered some of the influences between species using precise experiments. This knowledge, however, is likely very incomplete, meaning that very few entries in the influence vectors can be updated from a "?" to either an "a," "r," or "n." Note that the algorithms in this paper currently do not learn self-regulatory effects, but we plan on extending the methods to look for these types of influences.

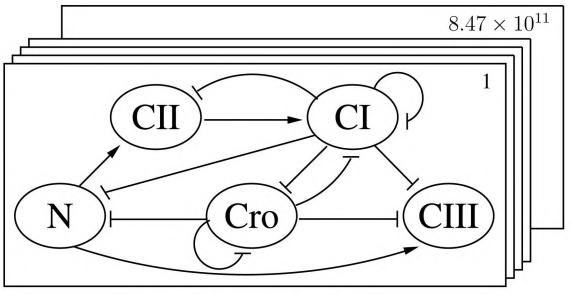

Fig. 6. Graphical representation.

Influence vectors can be represented as a directed graph in which the species are nodes and the influences are edges directed from parent to child species. There are four types of edges. For $i = \mathcal{I}(c)$, there is an *activation edge* from p to c (i.e., $p \rightarrow c$) if $i(p)$ returns "a". There is a repression edge from p to c (i.e., $p \dashv c$) if $i(p)$ returns "r." If the connection is unknown (i.e., $i(s) =$ "?"), a "?" appears as a label on the edge. If there is no connection (i.e., $i(s) =$ "n"), there is no edge in the graph.

Fig. 6 shows the directed graph representation of the phage λ decision circuit that has been discovered by biological experimentation for each of the species in the phage λ decision circuit [16]. This network is one of the 8.47×10^{11} possible networks composed of five species under our setup. In general, the maximum number of potential networks with no "?" influences are $3^{|S|^2}$, since there are $3^{|S|}$ values for the influence vector for each species, and there are $|S|$ species.

5 Scoring Influence Vectors

Given the large number of potential networks, it is crucial to be able to reduce the number of networks considered, which is the subject of Section 6. To evaluate these networks, our method must assign a score to each influence vector considered. A score for an influence vector, i, is determined by calculating probabilities for a species, s, increasing in expression in the next data point given that the current data point is valid and in one of several partial bin assignments, b, i.e.,

$$\mathcal{P}(inc(s) \mid val(s) \cap bin(b)) = \frac{\mathcal{P}(inc(s) \cap val(s) \cap bin(b))}{\mathcal{P}(val(s) \cap bin(b))}.$$

The data point pairs that have valid data, meaning no mutations or unknown value for species s, are:

$$val(s) = \{(\langle e, \tau, \nu \rangle, \langle e', \tau', \nu' \rangle) \in SUCC \mid \nu(s) \notin \{L, H, -\} \wedge \nu'(s) \notin \{L, H, -\}\}.$$

TABLE 1
Bin Assignments, Probabilities, Ratios, and Votes for the Influence Vector $\langle n, r, r, n, n\rangle$ with Species Order $\langle CI, CII, CIII, Cro, N\rangle$, for Child Species N

$\langle CI, CII, CIII, Cro, N\rangle$	Probability	Ratio	Vote
$\langle *, 0, 0, *, *\rangle$	40%		
$\langle *, 0, 1, *, *\rangle$	49%	1.23	v_a
$\langle *, 0, 2, *, *\rangle$	70%	1.75	v_a
$\langle *, 1, 0, *, *\rangle$	58%	1.45	v_a
$\langle *, 1, 1, *, *\rangle$	42%	1.05	v_n
$\langle *, 1, 2, *, *\rangle$	38%	0.95	v_n
$\langle *, 2, 0, *, *\rangle$	66%	1.65	v_a
$\langle *, 2, 1, *, *\rangle$	38%	0.95	v_n
$\langle *, 2, 2, *, *\rangle$	26%	0.65	v_f

This allows us to combine data from multiple experiments with different mutational controls. The $bin(b)$ function returns those data point pairs in the set $SUCC$ where the data values of the first data point in the pair fall within the bins specified in the partial bin assignment, b, i.e.,

$$bin(b) = \{(\langle e, \tau, \nu\rangle, \langle e', \tau', \nu'\rangle) \in SUCC \mid \forall s' \in S.\nu(s') \in \Phi_{b(s')}(s')\}.$$

The probability of a data point pair having valid data for the species of interest and being in a partial bin assignment, b, is:

$$\mathcal{P}(val(s) \cap bin(b)) = \frac{|val(s) \cap bin(b)|}{|SUCC|}.$$

The $inc(s)$ function returns the data point pairs in $SUCC$ where the expression level of species s increases between the first and second data points in the pair, i.e.,

$$inc(s) = \{(\langle e, \tau, \nu\rangle, \langle e', \tau', \nu'\rangle) \in SUCC \mid \nu(s) < \nu'(s)\}.$$

The probability of a data point pair increasing, having valid data for the species of interest, s, and being in a partial bin assignment, b, is:

$$\mathcal{P}(inc(s) \cap val(s) \cap bin(b)) = \frac{|inc(s) \cap val(s) \cap bin(b)|}{|SUCC|}.$$

Using these equations, the probability of species, s, increasing in expression in a data point given that it is a valid data point and in a bin assignment, b, is:

$$\mathcal{P}(inc(s) \mid val(s) \cap bin(b)) = \frac{|inc(s) \cap val(s) \cap bin(b)|}{|val(s) \cap bin(b)|}.$$

In order to score an influence vector, i, a probability of the form above is determined for all possible partial bin assignments using each $s' \in Par(i)$. Consider the influence vector $i = \langle n, r, r, n, n\rangle$ for the child species N, where the order of the species is $\langle CI, CII, CIII, Cro, N\rangle$. Note that $Par(i) = \{CII, CIII\}$. Partial bin assignments are created with a bin assignment for these two species and no bin assignment for the other species. All partial bin assignments, and their associated probabilities, for this influence vector are shown in Table 1 and graphically in Fig. 7.

To determine trends in the data, a ratio is formed of two probabilities of the form of the equation above using two partial bin assignments, b and b'. The partial bin assignment, b', used for comparison against partial bin assignment, b, is called the *base*. The base is a partial bin assignment with the lowest bin assignment for the species in the $Act(i)$ set and the highest bin assignment for the species in the $Rep(i)$ set, unless there are more repressors in the influence vector in which case the roles are reversed. For example, for the influence vector $i = \langle a, r, n, n, n\rangle$, the base bin assignment would be $b' = \langle 0, 2, *, *, *\rangle$, assuming $n = 3$. For the influence vector $i = \langle n, r, r, n, n\rangle$, the base would be $b' = \langle *, 0, 0, *, *\rangle$. The base, b', is constructed as follows:

$$b'(s) = \begin{cases} *, & \text{if } i(s) = \text{'}n\text{'}, \\ 0, & \text{if } (i(s) = \text{'}a\text{'} \wedge |Rep(i)| \leq |Act(i)|) \\ & \quad \vee (i(s) = \text{'}r\text{'} \wedge |Rep(i)| > |Act(i)|), \\ n-1, & \text{otherwise.} \end{cases}$$

To evaluate an influence vector, i, a probability ratio is determined using the probability for the base, b', and the probability calculated for each other partial bin assignment, b, as follows:

$$\frac{\mathcal{P}(inc(s) \mid val(s) \cap bin(b))}{\mathcal{P}(inc(s) \mid val(s) \cap bin(b'))} = \frac{|inc(s) \cap val(s) \cap bin(b')|}{|val(s) \cap bin(b')|} * \frac{|val(s) \cap bin(b)|}{|inc(s) \cap val(s) \cap bin(b)|}.$$

This ratio represents the change in expression between a partial bin assignment and its base which expresses a general trend in the data. Table 1 shows the ratios created for the example.

The development of the base and probability ratios is such that a ratio greater than 1 indicates that the species has more expression over the base, and a ratio less than 1 indicates that the species has less expression over the base. If there are more activating influences in the influence vector (i.e., $|Rep(i)| \leq |Act(i)|$), then a ratio larger than 1 indicates support for the influence vector and a ratio less than 1 does not support the influence vector. If there are more repressing influences in the influence vector (i.e., $|Rep(i)| > |Act(i)|$), then a ratio less than 1 indicates support for the influence vector, and a ratio greater than 1 indicates that the data does not support the influence vector. Each ratio is used to cast a vote for or against an influence vector. Since ratios near 1 are indeterminate, our method also allows a ratio to yield a neutral vote. The votes cast for the example influence vector are shown in Table 1.

The final score is determined using the following equation:

$$\text{score} = \frac{v_f - v_a}{v_f + v_a + v_n}.$$

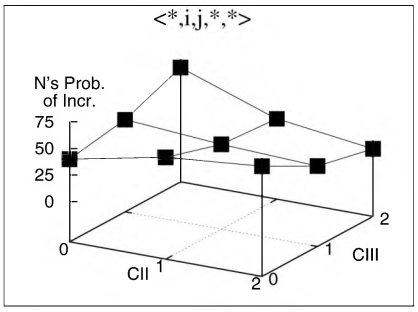

Fig. 7. N's probability (in percentile) of increasing with the influence vector $i = \langle n, r, r, n, n\rangle$, with species order $\langle CI, CII, CIII, Cro, N\rangle$.

TABLE 2
Bin Assignments, Probabilities, Ratios, and Votes for $\langle n, r, r, n, n\rangle$ with Species Order $\langle CI, CII, CIII, Cro, N\rangle$, $G = \{N\}$ and Child Species Is N

$\langle CI, CII, CIII, Cro, N\rangle$	Probability	Ratio	Vote
$\langle *, 0, 0, *, 0\rangle$	40%	base	
$\langle *, 0, 1, *, 0\rangle$	58%	1.45	v_a
$\langle *, 0, 2, *, 0\rangle$	83%	2.08	v_a
$\langle *, 1, 0, *, 0\rangle$	67%	1.66	v_a
$\langle *, 1, 1, *, 0\rangle$	55%	1.37	v_a
$\langle *, 1, 2, *, 0\rangle$	59%	1.47	v_a
$\langle *, 2, 0, *, 0\rangle$	100%	2.5	v_a
$\langle *, 2, 1, *, 0\rangle$	44%	1.09	v_n
$\langle *, 2, 2, *, 0\rangle$	36%	0.90	v_n
$\langle *, 0, 0, *, 1\rangle$	55%	base	
$\langle *, 0, 1, *, 1\rangle$	40%	0.72	v_f
$\langle *, 0, 2, *, 1\rangle$	50%	0.90	v_n
$\langle *, 1, 0, *, 1\rangle$	54%	0.98	v_n
$\langle *, 1, 1, *, 1\rangle$	37%	0.67	v_f
$\langle *, 1, 2, *, 1\rangle$	41%	0.75	v_n
$\langle *, 2, 0, *, 1\rangle$	0%	0.00	v_f
$\langle *, 2, 1, *, 1\rangle$	42%	0.76	v_n
$\langle *, 2, 2, *, 1\rangle$	27%	0.49	v_f
$\langle *, 0, 0, *, 2\rangle$	27%	base	
$\langle *, 0, 1, *, 2\rangle$	22%	0.81	v_n
$\langle *, 0, 2, *, 2\rangle$	50%	1.85	v_a
$\langle *, 1, 0, *, 2\rangle$	30%	1.11	v_n
$\langle *, 1, 1, *, 2\rangle$	28%	1.04	v_n
$\langle *, 1, 2, *, 2\rangle$	28%	1.04	v_n
$\langle *, 2, 0, *, 2\rangle$	100%	3.70	v_a
$\langle *, 2, 1, *, 2\rangle$	30%	1.11	v_n
$\langle *, 2, 2, *, 2\rangle$	24%	0.88	v_n

Note that a score greater than zero indicates support for the influence vector being scored while a negative score indicates there is no support for the influence vector. For our example, the resulting score would be -0.375, which indicates that this is not likely a correct influence vector.

When scoring an influence vector, i, for species s, the probability of increase can be influenced by the level of s. For example, it is often the case that when s is at a high concentration that its rate of increase reduces as degradation begins to dominate. Therefore, considering each level of s separately can give a better picture of how other species influence its rate of increase. Similarly, when comparing two influence vectors, i and i', it is useful to control for the species in i', when evaluating the score for i and vice versa. In both of these cases, our method partitions the set of bins using the species in a control set, G. When controlling for G while evaluating an influence vector, i, the partial bin assignments considered now include those with assignments to all species in $Par(i) \cup G$. Breaking the data up in this way is similar to performing a mutational experiment on the data where our method fixes those species in G to specific levels.

This is illustrated with an example from the phage λ decision circuit, where the child species s is N, the influence vector i is $\langle n, r, r, n, n\rangle$, representing that both CII and CIII repress N, and the set of species $G = \{N\}$, with each species having a total of three bins. The partial bin assignments created for this configuration, along with their probabilities, are shown in Table 2. Fig. 8 represents the probabilities graphically.

The base is also calculated in a slightly different way when controlling for the species in G. The base bin

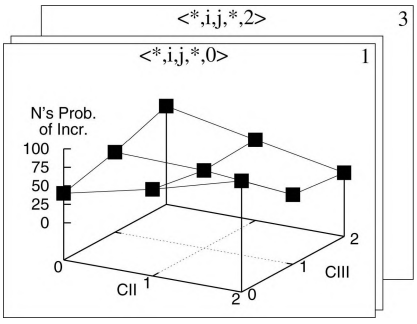

Fig. 8. N's probability (in percentile) of increasing with the influence vector $i = \langle n, r, r, n, n\rangle$, where the species order is $\langle CI, CII, CIII, Cro, N\rangle$ and the set $G = \{N\}$.

assignment, b', agrees with the values in b for each of the members in the G set. This is defined formally as:

$$b'(s) = \begin{cases} b(s), & \text{if } s \in G, \\ *, & \text{if } i(s) = \text{`}n\text{'}, \\ 0, & \text{if } (i(s) = \text{`}a\text{'} \wedge |Rep(i)| \leq |Act(i)|), \\ & \quad \vee (i(s) = \text{`}r\text{'} \wedge |Rep(i)| > |Act(i)|), \\ n-1 & \text{otherwise.} \end{cases}$$

Note that the base determination is now dependent on the partial bin assignment, b, that it is being ratioed against. In Table 2, for example, the base influence vectors are $\langle *, 0, 0, *, 0\rangle, \langle *, 0, 0, *, 1\rangle, \langle *, 0, 0, *, 2\rangle$. Now, each probability ratio calculated must use the proper base (i.e., the one which agrees on the bin assignment for each species in G). One ratio is determined for each of the probabilities in Table 2 excluding the bases. The ratios, and their associated votes, are shown in Table 2. The final score for this vector is -0.16, which indicates that this is not a correct influence vector.

Since the `Score` function is called many times by the other algorithms described in this paper, it is crucial that it is implemented in an efficient manner. A naive implementation would consider each possible partial bin assignment for the species in the $Par(i) \cup G$ set, which has $(n-1)^{|S|}$ entries in the worst case. However, many partial bin assignments may never actually occur in the data. In this case, an optimized strategy for looking through the data would be to only look at the actual bin assignments that occur in the data, thereby avoiding an expensive iteration through bin assignments that are not present. This can be done by compressing and projecting the data using the `Compress` and `Project` functions described in Section 3. The result is that the number of partial bin assignments considered is now $|\text{domain}(PCD)|$, which is $|E|$ in the worst case. While it is possible for $|E|$ to be larger than $(n-1)^{|S|}$, it is typically substantially smaller.

The `Score` function is shown in Fig. 9. In the first step, the function clears the votes. The function then sets all connections of type "?" to type "n." Next, the function checks if all species in i have no connection to s and returns a score of T_i in this case. If this is not the case, then this function constructs the PCD and loops over all partial bin assignments in the domain of PCD. The function then finds the base probability using the `BaseProb` function. If this probability is valid, it then finds the probability for the given bin assignment b. If this probability is also valid, it

```
FindProb(Species s, ProjCompData PCD,
    Threshold T_s, PBA b)
  ⟨incr, occur⟩ := PCD(b)
  if occur < T_s then return −1
  else return incr/occur

Score(Species s, IV i,
    Species Sets G, S, CompData CD,
    Number of Bins n, Thresholds T)
  ⟨v_f, v_a, v_n⟩ := ⟨0, 0, 0⟩
  foreach s ∈ S
    if i(s) = '?' then i(s) = 'n'
  if (i = (n, ..., n)) then return T_i
  PCD := Project(CD, s, S, Par(i) ∪ G)
  foreach b ∈ domain(PCD)
    probB = BaseProb(s, i, G, PCD, n, T_s, b)
    if probB ≠ −1 then
      probA := FindProb(s, PCD, T_s, b)
      if probA ≠ −1 then
        if |Rep(i)| < |Act(i)| then
          if probA/probB > T_a then v_f++
          else if probA/probB < T_r then v_a++
          else v_n++
        else
          if probA/probB < T_r then v_f++
          else if probA/probB > T_a then v_a++
          else v_n++
  if v_f + v_a + v_n = 0 then return 0
  return (v_f − v_a)/(v_f + v_a + v_n)
```

Fig. 9. The `FindProb` and `Score` functions.

constructs the ratio of these probabilites and compares them against the T_a and T_r thresholds to determine the votes, and it calculates the final score. By using the PCD, the scoring function uses only the time series data entries that occur and skips potentially vast empty regions of the state space. Note that the complexity of creating the PCD is $O(|E| * |S|)$.

The use of the PCD structure makes the `FindProb` function very simple, as shown in Fig. 9. As the data has been compressed and projected, there is no need to look through every time series data entry to select those entries that match the partial bin assignment. The probabilities can be calculated from a single entry in the PCD structure. The complexity of the `FindProb` function is $O(1)$.

When data is sparse, as is typically the case, the base bin assignment may have little or no data, making it impossible to calculate a base probability. In this case, no votes are recorded for any ratio that requires this base probability. Therefore, the `BaseProb` function must be capable of determining an alternate base probability in this situation. This process is a bit involved, so the reader is referred to [20] for more details.

Dealing with sparse data increases the complexity of the `Score` function but allows more votes to be computed, which generates better results. The complexity of a `Score` function which uses only the simple definition of base is $O(|E|*|S|)$. With the more sophisticated base probability determination, the probability increases to $O(|E|*|S| + |E|^2)$. Even though $|E|^2$ is usually larger than $|E|*|S|$ in our case studies, this added complexity does not result in increased runtime [20].

```
GeneNet(Species Set S, Expts E,
    Influences I, Number of Bins n,
    Thresholds T)
  θ := DetermineLevels(S, E, n)
  CD := Compress(E, S, θ)
  foreach s ∈ S
    I := CreateIVSet(s, S, CD, I, θ, n, T)
    I := CombineIVs(s, I, S, CD, I, θ, n, T)
    if |I| > 0 then
      I(s) := CompeteIVs(s, I, S, CD, θ, n, T)
  return I
```

Fig. 10. Algorithm to find genetic network connectivity.

6 Learning Influence Vectors

This section describes the `GeneNet` algorithm for learning genetic regulatory networks. The inputs to the `GeneNet` algorithm, shown in Fig. 10, are the set of species of interest, S, the set of experiments, E, the initial influence vector collection, $\mathcal{I}$, which includes any background knowledge, the number of bins used to group the data, n, and the set of threshold values, T, that are used when scoring potential influence vectors. First, the `GeneNet` algorithm determines the levels used to bin the data for each species. The default number of bins used is four. Next, the data is compressed, as explained in Section 3, to form the CD structure to optimize scoring efficiency. Next, each species, s, is considered separately to determine the species that may activate or repress the gene that produces s. The `CreateIVSet` algorithm derives a set of potential influence vectors, I, to assign to $\mathcal{I}(s)$. Each of these vectors differs from the initial value of $\mathcal{I}(s)$ in that at most one unknown entry is set to "a" or "r" while all other unknown entries are set to "n." Next, `CombineIVs` determines if any members of I can be combined to form a larger influence vector with a higher score. Finally, if the $|I|$ is greater than zero, the members of I are competed against each other to determine the final vector of influences on s. The result is a new influence vector, $i = \mathcal{I}(s)$, which includes only those activation or repression influences supported by the data, and it no longer includes any unknown influences. This section describes this algorithm in more detail and uses the learning of influences for the CIII species in the phage λ decision circuit as a running example. Initially, in the running example, the algorithm starts with no information known about the connections to species CIII, except that CIII does not influence itself. To simplify the presentation, only three bins are used in the examples.

6.1 Creating the Influence Vector Set

The `CreateIVSet` algorithm shown in Fig. 11 is used to compute an initial set of potential influence vectors, I, for a species s. In particular, each species s' with an unknown

```
CreateIVSet(Species s, Species Set S,
    CompData CD, Influences I, Levels θ,
    Number of Bins n, Thresholds T)
  I := ∅
  do
    α := Score(s, I(s), {s}, S, n, CD, T)
    foreach s' such that I(s') = ?
      i := I(s)
      i(s') := a
      foreach s'' ∈ (S − {s'})
        if (i(s'') =?) then i(s'') := n
      if (Score(s, i, {s}, S, CD, n, T) ≥ α)
      then I := I ∪ i
      else
        i(s') := r
        if (Score(s, i, {s}, S, CD, n, T) ≥ α)
        then I := I ∪ i
    (T_a, T_r, T_i) := RelaxThresholds(T)
  while |I| < T_n ∧ (T_r < 1 ∨ T_a > 1)
  return I
```

Fig. 11. The `CreateIVSet` algorithm.

influence in the background knowledge, $\mathcal{I}(s)$, is considered as a potential activator or repressor. First, the set of influence vectors, I, is set to the empty set. Next, a score, α, is obtained from $\mathcal{I}(s)$. For every species, s', that has an unknown connection in the background knowledge, a new influence vector, i, is created from $\mathcal{I}(s)$ where the influence for s' is set to "a", and every other species with unknown influence is set to "n." If the score of this new influence vector is greater than or equal to α, then it is added to I. Otherwise, $i(s')$ is changed to "r," and this new vector is scored. If the score is greater than α, then it is retained. If, after considering all species with unknown influence, the size of I is less than T_n, then the `RelaxThresholds` function is called to relax the thresholds. The threshold T_n is the value representing the minimum number of influence vectors that should be found in this algorithm. The T_t threshold is the value used to relax the other thresholds. For example, the T_a threshold is decreased and the T_r threshold is increased by the T_t threshold. If both T_r and T_a have been relaxed to 1, then the T_i threshold is decreased. The entire process is repeated except the newly relaxed thresholds are passed to the *Score* function to allow more influence vectors to be created.

With CIII as the child species, the `CreateIVSet` algorithm considers eight possible influence vectors containing only one influence. The probabilities and ratios determined by the `Score` function are shown in Table 3. The `Score` function uses these ratios to calculate votes and then a final score for the influence vectors. These votes, and the scores calculated from them, are shown in Table 4. For the influence vector $\langle a, n, n, n, n\rangle$, a score of -1.00 is returned from the Score function. As this is less than the T_i threshold, this edge $i(CI)$ is set to "r" and the resulting influence vector $\langle r, n, n, n, n\rangle$ is scored. The score obtained from this vector is 1.00, and it is added to the set of potential influence vectors. The `CreateIVSet` then constructs the other influence vectors in turn. As each influence vector with an "a" edge has a score lower than the T_i threshold, these influence vectors are all discarded and the influence vector with an "r" influence is scored. Of the eight influence vectors shown in Table 4, only three are retained: $\langle r, n, n, n, n\rangle$, $\langle n, n, n, r, n\rangle$, and $\langle n, n, n, n, r\rangle$. As there are three vectors found, the thresholds passed to the `Score` function do not need to be relaxed.

6.2 Combining Influence Vectors

After the initial influence vectors are determined, the `CombineIVs` algorithm shown in Fig. 12 is applied to construct new influence vectors by forming combinations of those found initially. This algorithm computes the set of influences that are of size k, starting at a size of two larger than the background knowledge for species s, by combining two influence vectors in the I set. For each of the combined influence vectors, i', the α_{min} and α_{max} score are computed from the influence vector scores that are a subset of i' and appear in I. An influence vector i is a subset of another vector i' if it has matching "a" or "r" influence in the same spots, or has "n" influences, where "a" or "r" influence appears in i' (i.e., $i \subseteq i'$ if $\forall s.(i(s) = n \vee i(s) = i'(s))$). For example, $\langle a, r, n\rangle \subseteq \langle a, r, a\rangle$ and $\langle r, n, n\rangle \not\subseteq \langle n, r, n\rangle$. Next, the algorithm determines if α_{max} and α_{min} are within T_m of each

TABLE 3
Probabilities and Ratios Found by `CreateIVSet` for Species CIII with Species Order $\langle CI, CII, CIII, Cro, N\rangle$

IV	$P(inc(CIII) \mid bin(b) \cap val(CIII))$				Ratios	
		Φ_0(CI)	Φ_1(CI)	Φ_2(CI)	Φ_1(CI)/Φ_0(CI)	Φ_2(CI)/Φ_0(CI)
$\langle a,n,n,n,n\rangle$ $\langle r,n,n,n,n\rangle$	Φ_0(CIII)	19.0%	1.7%	1.0%	0.09	0.05
	Φ_1(CIII)	17.1%	2.6%	1.2%	0.15	0.07
	Φ_2(CIII)	11.6%	2.7%	1.1%	0.23	0.09
		Φ_0(CII)	Φ_1(CII)	Φ_2(CII)	Φ_1(CII)/Φ_0(CII)	Φ_2(CII)/Φ_0(CII)
$\langle n,a,n,n,n\rangle$ $\langle n,r,n,n,n\rangle$	Φ_0(CIII)	3.1%	13.7%	–	4.32	–
	Φ_1(CIII)	4.4%	7.4%	12.6%	1.65	2.83
	Φ_2(CIII)	19.4%	5.5%	6.8%	0.28	0.35
		Φ_0(Cro)	Φ_1(Cro)	Φ_2(Cro)	Φ_1(Cro)/Φ_0(Cro)	Φ_2(Cro)/Φ_0(Cro)
$\langle n,n,n,a,n\rangle$ $\langle n,n,n,r,n\rangle$	Φ_0(CIII)	11.5%	1.8%	1.5%	0.16	0.13
	Φ_1(CIII)	14.2%	4.7%	3.1%	0.33	0.23
	Φ_2(CIII)	9.7%	5.0%	4.2%	0.52	0.43
		Φ_0(N)	Φ_1(N)	Φ_2(N)	Φ_1(N)/Φ_0(N)	Φ_2(N)/Φ_0(N)
$\langle n,n,n,n,a\rangle$ $\langle n,n,n,n,r\rangle$	Φ_0(CIII)	5.4%	2.9%	3.7%	0.53	0.68
	Φ_1(CIII)	9.3%	7.2%	6.7%	0.78	0.72
	Φ_2(CIII)	8.6%	6.4%	6.1%	0.75	0.71

TABLE 4
Votes and Scores Calculated in the `CreateIVSet` Algorithm for Influence Vectors for Species CIII

$\langle CI, CII, CIII, Cro, N\rangle$	votes$_f$	votes$_a$	votes$_n$	Score
$\langle a,n,n,n,n\rangle$	0	6	0	-1.00
$\langle r,n,n,n,n\rangle$	6	0	0	1.00
$\langle n,a,n,n,n\rangle$	3	2	0	0.20
$\langle n,r,n,n,n\rangle$	2	3	0	-0.20
$\langle n,n,n,a,n\rangle$	0	6	0	-1.00
$\langle n,n,n,r,n\rangle$	6	0	0	1.00
$\langle n,n,n,n,a\rangle$	0	5	1	-0.83
$\langle n,n,n,n,r\rangle$	5	0	1	0.83

other and removes i' from I' if they are not. If the scores are close to each other, the algorithm scores the combined influence vector i'. If this score is less than α_{max}, the influence vector is removed from the set; otherwise, it remains in the set. After all combined influence vectors in I' are considered, the newly created vectors, I', are merged with I and any vectors that are subsets of other vectors are removed from I by the *RemoveSubsets* function. This process repeats until all influence vectors are merged up to size T_j. Finally, any influence vectors that do not have a score better than the background knowledge are filtered from the set using the `Filter` function shown in Fig. 12.

```
Filter(Species s, IV i,  IV Set I,
    Species Set S, Comp. Data CD,
    Number of Bins n, Thresholds T)
  α := Score(s, i, {s}, S, CD, n, T)
  foreach i' ∈ I
    α' := Score(s, i', {s}, S, CD, n, T)
    if α' > α then I' := I' ∪ {i'}
  return I'

CombineIVs(Species s, IV Set I,
    Species Set S, Comp. Data CD,
    Influences 𝓘, Levels θ,
    Number of Bins n, Thresholds T)
  for k = |𝓘(s)| + 2 to T_j
    I' = {i ∪ i' | i ∈ I ∧ i' ∈ I ∧ |i ∪ i'| = k}
    foreach i' ∈ I'
      α_min := ∞
      α_max := −∞
      foreach i ∈ I such that i ⊆ i'
        α := Score(s, i, {s}, S, CD, n, T)
        if α < α_min then α_min := α
        if α > α_max then α_max := α
      if α_min + T_m < α_max then
        I' := I' − {i'}
      else
        α' := Score(s, i', {s}, S, CD, n, T)
        if α' < α_max then I' := I' − {i'}
    I := RemoveSubsets(I ∪ I')
  I := Filter(s, 𝓘(s), I, S, CD, n, T)
  return I
```

Fig. 12. The `CombineIVs` and `Filter` algorithms.

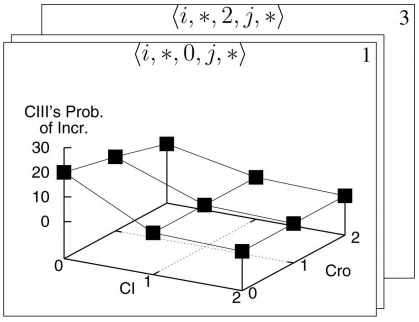

Fig. 13. CIII's probability of increasing with the influence vector $i = \langle r,n,n,r,n\rangle$, where the species order is $\langle CI, CII, CIII, Cro, N\rangle$, and $G = \{CIII\}$.

This function scores the background knowledge and returns only those influence vectors in I that have a better score.

For our example, the `CombineIVs` algorithm starts with the three influence vectors found by the `CreateIVSet` algorithm. The `CombineIVs` algorithm then considers them two at a time. For example, the first two influence vectors $\langle r,n,n,n,n\rangle$ and $\langle n,n,n,r,n\rangle$ are scored, and the scores obtained are 1.0 in both cases. Since these scores are equal, they are within the T_m threshold and are merged to $\langle r,n,n,r,n\rangle$. It then calculates the merged score with the `Score` function, and a score of 1.0 is returned in this case. Fig. 13 represents the 27 probabilities that are calculated during the `Score` function for the merged influence vector. As a score of 1.0 is as good as both single influence vector scores, this vector is added to the list of influence vectors. After all other combinations are tried, the `RemoveSubsets` function removes both $\langle r,n,n,n,n\rangle$ and $\langle n,n,n,r,n\rangle$ vectors. Table 5 shows the scores for influence vectors found by merging vectors. As can be seen from the table, only the $\langle r,n,n,r,n\rangle$ vector mentioned above has a score at least as good as both of the influence vectors merged to create it and is the only one added to the I set. Therefore, the influence vector set I contains only the two influence vectors: $\langle n,n,n,n,r\rangle$ and $\langle r,n,n,r,n\rangle$.

6.3 Competing Influence Vectors

The `CompeteIVs` algorithm shown in Fig. 14 competes the remaining influence vectors until only one remains. It begins by selecting two influence vectors i and i' from I. In general, when choosing a pair of influence vectors, the algorithm ranks the set of influence vectors according to their score. The influence vector with the highest score is paired with the influence vector with the lowest score. Next, it makes a copy of the thresholds used during scoring in case they need to be strengthened. It then obtains a score for i controlling for any

TABLE 5
Votes and Scores Calculated in the `CombineIVs` Algorithm for Influence Vectors for Species CIII

$\langle CI, CII, CIII, Cro, N\rangle$	v_f	v_a	v_n	Score
$\langle r,n,n,r,n\rangle$	24	0	0	1.0
$\langle r,n,n,n,r\rangle$	21	0	3	0.875
$\langle n,n,n,r,r\rangle$	23	0	1	0.958
$\langle r,n,n,r,r\rangle$	71	1	6	0.897

```
CompeteIVs(Specie s, IV Set I,
    Species S, CompData CD, Levels θ,
    Number of Bins n, Thresholds T)
  while |I| > 1
    (i, i') = selectPair(I)
    do
      G = (Par(i') − Par(i)) ∪ {s}
      α := Score(s, i, G, S, CD, n, T)
      G = (Par(i) − Par(i')) ∪ {s}
      α' := Score(s, i', G, S, CD, n, T)
      (T_a, T_r) := StrengthenThresholds(T)
    while (α = α' ∧ T_r > 0 ∧ T_a < 5)
    if (α > α') ∨ (α = α' ∧ |i| < |i'|) then
      I := I − {i'}
    else
      I := I − {i}
  i := select(I)
  return i
```

Fig. 14. The `CompeteIVs` algorithm.

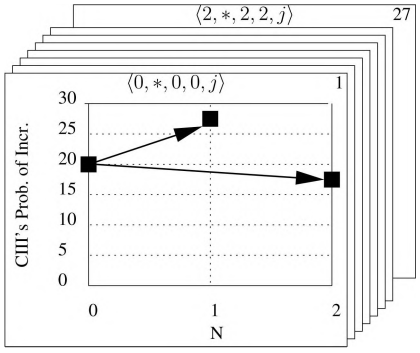

Fig. 15. CIII's probability of increasing with the influence vector $i = \langle n, n, n, n, r\rangle$, where the species order is $\langle CI, CII, CIII, Cro, N\rangle$, and $G = \{CI, CIII, Cro\}$.

species in i' and not in i by performing the following set operation $(Par(i') - Par(i)) \cup \{s\}$. For example, with $i = \langle a, a, n, n, n\rangle$ and $i' = \langle a, n, n, n, a\rangle$, $(Par(i') - Par(i)) \cup \{s\} = (\{CI, N\} - \{CI, CII\}) \cup \{CIII\} = \{CIII, N\}$. Similarly, the `CompeteIVs` algorithm computes a score for i' controlling for those species with influence in i but not in i'. If both influence vectors scores are tied initially, it then computes a stronger set of thresholds which it uses to rescore the influence vectors. The thresholds are strengthened using the `StrengthenThresholds` function which increases the T_a threshold and decreases the T_r threshold by the T_t threshold until the scores are no longer tied or the T_r is less than zero, and T_a is greater than 5. If the thresholds are strengthened too much, then the influence vector with the most influences is removed. If there is a clear winner, then the influence vector with the lowest score is removed from I. This process continues until only one influence vector remains which becomes the final result for this species.

In our example, the `CompeteIVs` algorithm competes $\langle n, n, n, n, r\rangle$ against $\langle r, n, n, r, n\rangle$. The `Score` function is first called controlling for those species in the second influence vector. Fig. 15 represents the probabilities and ratios that would be found in the `Score` function during this competition step. Next, the second influence vector is scored, controlling for the first influence vector. Fig. 16 represents the probabilities and ratios that would be found during this competition step. The votes and scores obtained while competing the two influence vectors are shown in Table 6. The result is that the `GeneNet` algorithm has determined that CIII is repressed by both Cro and CI as it has the higher score.

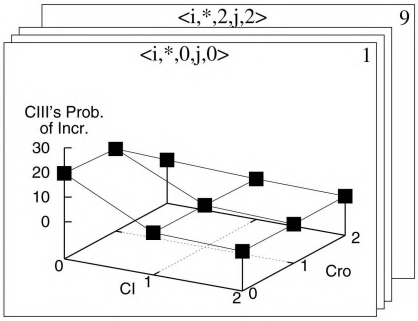

Fig. 16. CIII's probability of increasing with the influence vector $i = \langle r, n, n, r, n\rangle$, where the species order is $\langle CI, CII, CIII, Cro, N\rangle$, and $G = \{CIII, N\}$.

TABLE 6
Scores from the `CompeteIVs` Algorithm for Species CIII

$\langle CI, CII, CIII, Cro, N\rangle$	v_f	v_a	v_n	Score
$\langle n, n, n, n, r\rangle$	12	27	15	0.27
$\langle r, n, n, r, n\rangle$	71	0	1	0.98

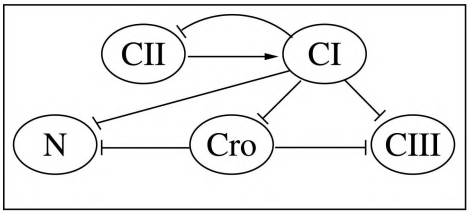

Fig. 17. The influence vectors that `GeneNet` learned for the species in the phage λ decision circuit.

The `GeneNet` algorithm also learns influence vectors for the other species in the phage λ decision circuit network. These influence vectors are shown in Fig. 17. All the arcs reported are in the actual phage λ decision circuit, shown in Fig. 6. Our method, however, does not find two activation arcs, one repression arc, and the two self-arcs.

6.4 Complexity Analysis

Theoretically, one way to approach the problem of learning a genetic network would be to look at every possible network and select the one with the best score. The cost of performing this action would be $O(|\mathcal{I}| * score)$ where $|\mathcal{I}|$ is the total number of possible networks, and $score$ is the complexity to determine a score for each network. By assuming that $\mathcal{I}$ contains influence vectors, the maximum number of potential networks with $|S|$ species can be determined by assuming that a network is a fully connected graph with three types of edges. These edges are activation, repression, or no influence. As a fully connected directed graph with self-arcs has a total of $|S|^2$ edges, and each edge

TABLE 7
Influence Vectors Considered for the Species CIII with Species Order $\langle CI, CII, CIII, Cro, N\rangle$

$\langle i_c, \alpha_c\rangle$	Case	$\langle i_r, \alpha_r\rangle$	Contender
$\langle\langle a,n,n,n,n\rangle$, -1.0$\rangle$	1	$\langle\langle n,n,n,n,n\rangle, 0.5\rangle$	no
$\langle\langle n,a,n,n,n\rangle$, 0.2$\rangle$	1	$\langle\langle n,n,n,n,n\rangle, 0.5\rangle$	no
$\langle\langle n,r,n,n,n\rangle$, -0.2$\rangle$	1	$\langle\langle n,n,n,n,n\rangle, 0.5\rangle$	yes
$\langle\langle n,n,n,a,n\rangle$, -1.0$\rangle$	1	$\langle\langle n,n,n,n,n\rangle, 0.5\rangle$	no
$\langle\langle n,n,n,n,a\rangle$, -0.833$\rangle$	1	$\langle\langle n,n,n,n,n\rangle, 0.5\rangle$	no
$\langle\langle r,n,n,n,r\rangle$, 0.875$\rangle$	3	$\langle\langle n,n,n,n,r\rangle, 1.0\rangle$	yes
$\langle\langle n,n,n,r,r\rangle$, 0.958$\rangle$	3	$\langle\langle n,n,n,r,r\rangle, 1.0\rangle$	yes
$\langle\langle r,n,n,r,r\rangle$, 0.897$\rangle$	3	$\langle\langle n,n,n,r,r\rangle, 1.0\rangle$	yes
$\langle\langle r,n,n,n,n\rangle$, 1.0$\rangle$	4	$\langle\langle r,n,n,r,n\rangle, 1.0\rangle$	yes
$\langle\langle n,n,n,r,n\rangle$, 1.0$\rangle$	4	$\langle\langle n,n,n,r,r\rangle, 1.0\rangle$	yes
$\langle\langle n,n,n,n,r\rangle$, 0.27$\rangle$	7	$\langle\langle r,n,n,r,n\rangle, 0.99\rangle$	no

can be one of three different influences, this gives a total of $3^{|S|^2}$ unique network topologies which changes the complexity to $O(3^{|S|^2} * score)$. By performing a local analysis for each species in S instead of looking at the global structure, the size of the search space can be reduced to $O(|S| * 3^{|S|} * score)$ as each of the species in S has potentially $3^{|S|}$ ways in which it can be influenced. The `CreateIVSet` algorithm looks at the connections between individual species and then selects either, activation, repression, or no influence for each individual influence. This further reduces the complexity to $O(|S| * (|S| + 2^{|S|}) * score)$, or equivalently, $O(|S| * 2^{|S|} * score)$, since each call to the `CreateIVSet` algorithm looks at $|S|$ influence vectors but reduces the remaining search space to $2^{|S|}$. The total number of influences can be expressed using the choose function:

$$O\left(|S| * \sum_{k=0}^{|S|} \binom{|S|}{k} * score\right).$$

The T_j threshold, used in the `CombineIVs` algorithm, sets a limit as to the maximum number of influences an influence vector can contain. Using this threshold, the complexity is bounded by $O(|S| * (|S| + \cdots + |S|^{T_j}) * score)$, which is equivalent to $O(|S|^{T_j+1} * score)$. When $|S| \gg T_j$, this threshold results in a significant improvement in complexity. Using the better base probability function, the total complexity of the `GeneNet` algorithm is $O(|S|^{T_j+1} * (|S| * |E| + |E|^2))$.

7 Experimental Suggestion

This section describes how the results from the `GeneNet` algorithm can be used to suggest further experiments. The `GeneNet` algorithm has been designed to choose the best influence vector for each species of interest. There are generally several other influence vectors that are discarded in favor of a higher scoring influence vector during its execution. These influence vectors may be considered as *contenders*. A contender is represented with a tuple of the form:

$$\langle\langle i_c, \alpha_c\rangle, case, \langle i_r, \alpha_r\rangle\rangle,$$

where i_c is a contender influence vector, α_c is its score, $case \in \{1,2,3,4,5,6,7\}$ is the reason it is discarded, i_r is the influence vector causing it to be discarded, and α_r is its score. There are seven cases where the `GeneNet` algorithm discards an influence vector. Case 1 occurs when the `CreateIVSet` algorithm discards an influence vector that has a score that is not greater than the background knowledge score. Case 2 occurs in the `CombineIVs` algorithm when two influence vectors are not merged because their scores are not within the T_m threshold of each other. Case 3 occurs when a merged influence vector score is smaller than one of its subsets. Case 4 occurs when the `RemoveSubsets` algorithm removes influence vectors that have been merged into a larger influence vector. Case 5 occurs when the `Filter` function removes those influence vectors that have a lower score than the background knowledge. For Case 6, the background knowledge is filtered. Case 7 occurs when the `CompeteIVs` algorithm removes an influence vector that has lost a competition.

`SuggestExpts`($\langle$ IV i_c, Score $\alpha_c\rangle$, Case c, $\langle$ IV i_r, Score $\alpha_r\rangle$)
Case c
1, 5: $Suggest(Par(i_c) - Par(i_r)) := Suggest(Par(i_c) - Par(i_r)) + 1$
2, 3: $Suggest(Par(i_r)) := Suggest(Par(i_r)) + 1$
4, 6: $Suggest(Par(i_c)) := Suggest(Par(i_c)) + 1$
7: $A := (Par(i_c) - Par(i_r)) \cup (Par(i_r) - Par(i_c))$
$Suggest(A) := Suggest(A) + 1$

Fig. 18. The `SuggestExpts` algorithm.

In order to suggest experiments, only a limited number of contenders should be considered. For the phage λ decision circuit example, the learned influence vector for species CIII is $\langle r,n,n,r,n\rangle$ and the other influence vectors considered are shown in Table 7. For Case 1, the influence vector $\langle n,r,n,n,n\rangle$ would likely be retained as a contender. All influence vectors removed by Cases 3 and 4 would likely be considered contenders. Finally, the influence vector removed by Case 7 would not likely be considered a contender.

The `SuggestExpts` algorithm shown in Fig. 18 is utilized to determine which experiments would be the most useful to gather information about the contenders. In particular, it computes a score stored in the `Suggest` array which indicates which gene mutation would gather information about the most contenders. Using the `Suggest` array, the gene for the species that has been suggested for mutation the largest number of times can then be mutated low by removing or crippling the gene. Note that if several sets have the same value, it should select the set with the least amount of species. If there are still ties, the mutation can be selected randomly.

The `SugggestExpts` algorithm considers each contender in turn. For Cases 1 and 5, the `SuggestExpts` algorithm suggests mutating the genes for the differing species between the reason influence vector and the contender. For Cases 2 and 3, it suggests mutating the genes for the species from the reason influence vector. For Cases 4 and 6, it suggests mutating the genes for the species in the contender influence vector. Finally, for Case 7, it suggests mutating the genes for all nonshared species. To

TABLE 8
Results for Each Binning Method

	Recall	Precision	Total Runtime (s)
Equal data per bin	0.67	0.77	54.12
Equal spacing of bins	0.63	0.61	49.86

understand better why particular mutations are suggested, let us consider the case where the contender is species A represses species C and the reason is an influence vector with no connections between these species. Consider a mutational experiment where species A is mutated low using a gene knockout experiment. In this experiment, there would be more data points added into the case where species A is low. If the real network is $\langle r, n, n \rangle$ the probability of species C increasing in expression in this base case would increase, as there would be no species A to repress it, thus adding support for this influence vector to hopefully increase its score above that of the background knowledge. Similar reasoning can be used to understand the other cases [20].

8 Case Studies

The algorithms in this paper have been implemented in the `GeneNet` tool within `iBioSim` [21]. To evaluate our method, it must be tested on known networks. As there are currently very few known genetic networks, and even fewer with available time series data, we first generate synthetic data for several networks of various sizes and types. The evaluation procedure begins by translating the influence vectors for the original network into a very detailed reaction-based model expressed in the *Systems Biology Markup Language* (SBML). The SBML model is then simulated using a version of Gillespie's *stochastic simulation algorithm* [22] implemented in `iBioSim`. The `GeneNet` algorithm then uses a limited amount of this data to construct a network. At this point, the original network is compared to the result of our method.

Our evaluation uses 68 networks. There are 48 four-gene networks inspired by the synthetic networks in Guet et al. [23]. These networks each have the four species: TetR, LacI, CI, and GFP. The GFP promoter is always repressed by CI. Each of the other three promoters can be either repressed by TetR, LacI, or CI, or activated by CI. We have extended this formalism to allow a fourth option, in which a promoter can be repressed by either LacI or CI. There are also 10 randomly connected 10-gene networks. Finally, there are ten 20-gene networks from Yu et al. [15]. Recall, precision, and runtime are measured for each example. Recall is the number of correct arcs divided by the number of total arcs in the actual network, and precision is the number of correct arcs divided by the number of total arcs reported by the learning algorithm.

The method used to bin data and the number of bins used determine how sparse the data are divided. Table 8 shows results for the two binning methods described earlier. These results indicate that dividing the data equally between the bins does a little better in recall, a lot better in precision, and only has a small cost in runtime. The number

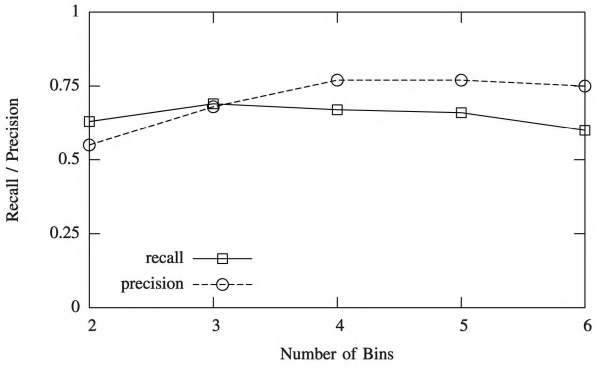

Fig. 19. Varying the number of bins.

of bins is also quite important. With fewer bins, there is more data per bin and the probabilities of a species increasing in expression can be more easily calculated, but there are fewer probabilities. With more bins, the data is more sparsely distributed; however, potentially more information as to the ability of a species to increase in expression can be calculated. Fig. 19 shows how the `GeneNet` algorithm performs given various bin sizes. As the figure shows, precision increases significantly between two to four bins while recall slightly increases between bins two and three. Four bins gives the best recall and precision. Larger than four bins, both recall and precision fall, likely due to the amount of data per bin not being statistically significant. These results are consistent with those found in [15] which also show that binary models (i.e., two bins) give imprecise results. This is likely due to the fact that species in genetic networks often have different effects at different levels. For example, a moderate concentration of a species may activate a gene while a high concentration may repress the gene. In general, we believe that at least three bins should be used, if there is sufficient data. We also believe that for larger experimental data sets, benefit can be gained using four bins. More than four is likely not useful even for very large data sets.

Next, we compare the `GeneNet` algorithm against Yu's DBN tool [15] using the same data for each tool. Given the 68 networks used in this section, the `GeneNet` algorithm performs better in recall in 56 of the 68 cases, ties in 7 cases, and loses in 5 cases. These results are shown as a scatter plot in Fig. 20. The `GeneNet` algorithm performs better in

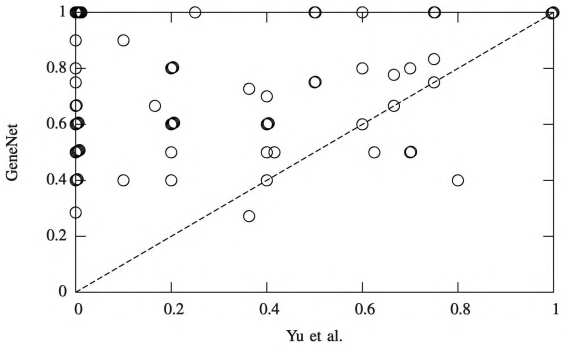

Fig. 20. Recall scatter plot results.

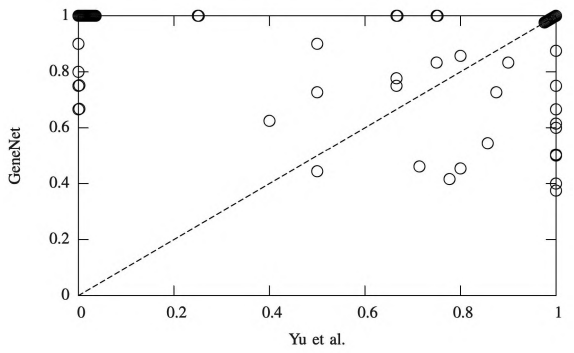

Fig. 21. Precision scatter plot results.

precision in 38 cases, ties in 13, and loses in 17. Note that groups of points with the same value are shown in an offset fashion. Also note that in 21 of the networks, the DBN tool reports no arcs for the entire network, and a precision score of 0 is assumed in these cases. These results are shown as a scatter plot in Fig. 21. Runtime results broken down by example type are shown in Fig. 22. These results show that GeneNet is an order of magnitude faster than the DBN tool.

Our final case study uses Spellman et al.'s gene expression measurements of mRNA levels from *S. cerevisiae* (yeast) [24]. This data is made up of 76 data points broken up into six time series experiments that each use a different cell cycle synchronization method. In particular, we selected eight species involved in cell cycle regulation of which four are known to act as transcription factors (FKH2, SWI5, MCM1, and ACE2). We seed the learning method with this background knowledge. Namely, the initial influence vectors indicate that there is no influence from the other species (CDC20, CTS1, SIC1, and CLN3). A network for these eight genes derived from published literature and experimentally using genomewide location analysis is shown in Fig. 23a (see Tables S1 and S2 from the supplemental information for [25]). Note that location analysis does not yield the type of influence, so the arrows in this graph only indicate that some type of influence is suspected. The network found by our learning method is shown in Fig. 23b. The GeneNet algorithm is able to find 8 of the 11 known influences, and 8 of the 11 arcs that it reports are correct. Two of the missing arcs and one of the incorrect arcs deal with the influence on SWI5. Looking at the list of

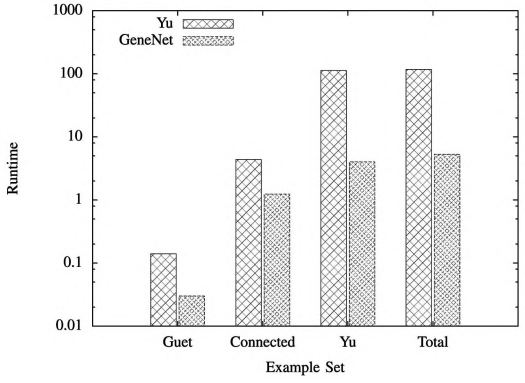

Fig. 22. Average runtime by example type.

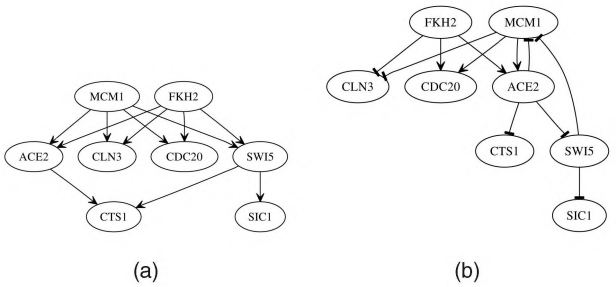

Fig. 23. Genetic network for eight genes from the yeast cell cycle (a) based on published literature and (b) learned by GeneNet.

contenders, we find that both MCM1 and FKH2 (the correct influences) narrowly lose out to ACE2 during the competition step. The third missing arc from SWI5 to CTS1 also narrowly loses a competition after having been combined with MCM1. Finally, the last two incorrect arcs would be correct, if their directions are reversed.

9 Conclusions

This paper describes a new method to learn genetic regulatory networks from time series data. This method can also suggest new experiments to improve the models learned. The results as compared with Yu's et al.'s DBN method are very promising. This result is perhaps somewhat surprising as one may expect that a global analysis would have better recall and precision than a local analysis. However, due to the substantial computational complexity of the global analysis, it is impossible to search the entire global search space. A limited search can often result in getting stuck in a local minimum. A local analysis, however, divides the problem in such a way that prevents poor choices for one part of a network from affecting the quality of the results for the other parts of the network.

There are still several areas in which this method can be improved. First, filtering and interpolation can be used to improve the probability calculations. Second, the GeneNet algorithm can be extended to learn more types of influences. Finally, while the GeneNet algorithm has been applied with interesting results to data from the yeast cell cycle, this work is still quite preliminary, and more study of this data set and other real experimental data sets is planned for the future.

Acknowledgments

This material is based upon work supported by the US National Science Foundation under Grant No. 0331270.

References

[1] P.A. Brown and D. Botstein, "Exploring the New World of the Genome with DNA Microarrays," *Nature Genetics*, vol. 21, pp. 33-37, 1999.

[2] P. Baldi and G.W. Hatfield, *DNA Microarrays and Gene Expression.* Cambridge Univ. Press, 2002.

[3] J. Szustakowski, "Initial Sequencing and Analysis of the Human Genome," *Nature*, vol. 409, pp. 860-921, Feb. 2001.

[4] D. Heckerman, "A Tutorial on Learning with Bayesian Networks," technical report, Microsoft Research, Microsoft Corporation, Nov. 1996.

[5] D. Pe'er, "Bayesian Network Analysis of Signaling Networks: A Primer," *Science's Signal Transduction Knowledge Environment*, vol. 2005, no. 281, p. l4, Apr. 2005.
[6] N. Friedman, K. Murphy, and S. Russell, "Learning the Structure of Dynamic Probabilistic Networks," *Proc. Uncertainty in Artificial Intelligence Conf.*, pp. 139-147, 1998.
[7] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using Bayesian Networks to Analyze Expression Data," *J. Computational Biology*, vol. 7, nos. 3/4, pp. 601-620, 2000.
[8] K. Sachs, O. Perez, D. Pe'er, D. Lauffenburger, and G. Nolan, "Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data," *Science*, vol. 308, pp. 523-529, Apr. 2005.
[9] I.M. Ong and D. Page, "Inferring Regulatory Pathways in E. coli Using Dynamic Bayesian Networks," technical report, Univ. of Wisconsin-Madison, May 2001.
[10] I.M. Ong, J.D. Glasner, and D. Page, "Modelling Regulatory Pathways in E. coli from Time Series Expression Profiles," *Bioinformatics*, vol. 18, pp. s241-s248, 2002.
[11] I. Nachman, A. Regev, and N. Friedman, "Inferring Quantitative Models of Regulatory Networks from Expression Data," *Bioinformatics*, vol. 20, suppl. 1, pp. i248-i256, Aug. 2004.
[12] D. Husmeier, "Sensitivity and Specificity of Inferring Genetic Regulatory Interactions from Microarray Experiments with Dynamic Bayesian Networks," *Bioinformatics*, vol. 19, no. 17, pp. 2271-2282, Nov. 2003.
[13] M.J. Beal, F. Falciani, Z. Ghahramani, C. Rangel, and D.L. Wild, "Bayesian Approach to Reconstructing Genetic Regulatory Networks with Hidden Factors," *Bioinformatics*, vol. 21, no. 3, pp. 349-356, Feb. 2005.
[14] A. Bernard and A.J. Hartemink, "Informative Structure Priors: Joint Learning of Dynamic Regulatory Networks from Multiple Types of Data," *Proc. Pacific Symp. Biocomputing*, pp. 459-470, 2005.
[15] J. Yu, V.A. Smith, P.P. Wang, A.J. Hartemink, and E.D. Jarvis, "Advances to Bayesian Network Inference for Generating Causal Networks from Observational Biological Data," *Bioinformatics*, vol. 20, pp. 3594-3603, Dec. 2004.
[16] M. Ptashne, *A Genetic Switch*. Cell Press & Blackwell Scientific Publishing, 1992.
[17] H. Zhu and M. Snyder, "Protein Arrays and Microarrays," *Current Opinion in Chemical Biology*, vol. 5, pp. 40-45, 2001.
[18] D.J. Lockhart and E.A. Winzeler, "Genomics, Gene Expression and DNA Arrays," *Nature*, vol. 405, pp. 827-836, June 2000.
[19] R.J. Lipschutz, S.P.A. Fodor, T.R. Gingeras, and D.J. Lockhart, "High Density Synthetic Oligonucleotide Arrays," *Nature Genetics*, vol. 21, pp. 20-24, 1999.
[20] N. Barker, "Learning Genetic Regulatory Network Connectivity from Time Series Data," PhD dissertation, Univ. of Utah, 2007.
[21] "ibioSim," http://www.async.ece.utah.edu/iBioSim/, 2010.
[22] D.T. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions," *J. Physical Chemistry*, vol. 81, no. 25, pp. 2340-2361, 1977.
[23] C.C. Guet, M.B. Elowitz, W. Hsing, and S. Leibler, "Combinatorial Synthesis of Genetic Networks," *Science*, vol. 296, pp. 1466-1470, 2002.
[24] P.T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B. Eisen, P.O. Brown, D. Botstein, and B. Futcher, "Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast *Saccharomyce Cerevisiae* by Microarray Hybridization," *Molecular Biology of the Cell*, vol. 9, pp. 3272-3297, Dec. 1998.
[25] T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, G.K. Gerber, N.M. Hannett, C.T. Harbison, C.M. Thompson, I. Simon, J. Zeitlinger, E.G. Jennings, H.L. Murray, D.B. Gordon, B. Ren, J.J. Wyrick, J.B. Tagne, T.L. Volkert, E. Fraenkel, D.K. Gifford, and R.A. Young, "Transcriptional Regulatory Networks in *Saccharomyces Cerevisiae*," *Science*, vol. 298, no. 5594, pp. 799-804, Oct. 2002.

Nathan A. Barker received the BA degree in computer science and German from Southern Utah University, Cedar City, in 2002, and the PhD degree from the University of Utah, Salt Lake City, in 2007. Currently, he is a professor in the Computer Science and Information Systems Department at Southern Utah University, Cedar City. He is a member of the Association for Computing Machinery. His research interests include learning, modeling, and analyzing genetic regulatory networks and other applications of machine learning and data mining.

Chris J. Myers received the BS degree in electrical engineering and Chinese history in 1991 from the California Institute of Technology, Pasadena, and the MSEE and PhD degrees from Stanford University, Stanford, California, in 1993 and 1995, respectively. Currently, he is a professor in the Department of Electrical and Computer Engineering, University of Utah, Salt Lake City. He is the author of more than 80 technical papers and the textbooks *Asynchronous Circuit Design* and *Engineering Genetic Circuits*. He is also a co-inventor on four patents. His research interests include algorithms for the analysis of real-time concurrent systems, analog error control decoders, formal verification, asynchronous circuit design, and the modeling and analysis of genetic regulatory circuits. He received a National Science Foundation (NSF) Fellowship in 1991, an NSF CAREER Award in 1996, and Best Paper Awards at Async1999 and Async2007. He is a senior member of the IEEE.

Hiroyuki Kuwahara received the BS and PhD degrees in computer science from the University of Utah, Salt Lake City, in 2001 and 2008, respectively. He currently holds a junior researcher position at The Microsoft Research—University of Trento Centre for Computational and Systems Biology, Italy. His research interests include development of computational methodologies to analyze dynamics of biochemical networks and computational analysis of system-level properties in biochemical networks to offer new insights.