

# Learning Genetic Regulatory Network Connectivity from Time Series Data <sup>\*</sup>

Nathan Barker, Chris Myers, and Hiroyuki Kuwahara

University of Utah,  
50 S. Central Campus Dr.  
Salt Lake City UT, 84112  
{barkern,myers,kuwahara}@vlsigroup.ece.utah.edu

**Abstract.** Recent experimental advances facilitate the collection of time series data that indicate which genes in a cell are expressed. This paper proposes an efficient method to generate the genetic regulatory network inferred from time series data. Our method first encodes the data into levels. Next, it determines the set of potential parents for each gene based upon the probability of the gene's expression increasing. After a subset of potential parents are selected, it determines if any genes in this set may have a combined effect. Finally, the potential sets of parents are competed against each other to determine the final set of parents. The result is a directed graph representation of the genetic network's repression and activation connections. Our results on synthetic data generated from models for several genetic networks with tight feedback are promising.

## 1 Introduction

High throughput technologies such as cDNA microarrays and oligonucleotide chips can collect *gene expression data* on a large number of genes in parallel which can potentially be used to determine the connectivity of genetic regulatory networks [1]. These techniques measure gene activity and by inference the amount of the protein in the cell produced by each gene. They can be used to gather *time series data* in which a series of measurements of gene activity is taken over a period of time.

Many techniques exist to analyze gene expression data. One approach is *clustering* which groups genes with similar expression patterns [2]. Other approaches modify machine learning algorithms to create Boolean functions that would describe when a gene should be active [3–6]. Many other methods such as the work by Friedman *et al.* use *Bayesian* analysis to learn genetic networks [7]. These methods consider each gene as a variable in a *joint probability distribution*. If knowledge of gene A's expression level yields information about gene B, the genes are said to be *correlated*. In this case, one gene may be a (potential) *parent* of the other. Recently, Sachs *et al.* applied a Bayesian analysis to find causal connectivities in the human T-cell signaling network [8].

---

<sup>\*</sup> This material is based upon work supported by the National Science Foundation under Grant No. 0331270.

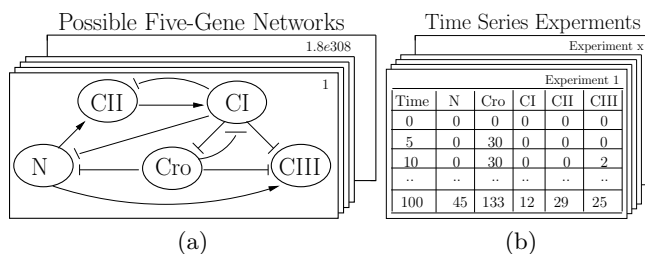
Unfortunately, Bayesian approaches have several limitations. First, Bayesian approaches often have difficulty determining which gene is the parent and which is the *child*. Second, these approaches rely primarily on correlational data and only use time to separate the data points. Third, these approaches can only be applied to networks that are acyclic. This last one we consider a major limitation since feedback control is a crucial component of genetic regulatory networks.

To address some of these limitations, *Dynamic Bayesian Networks* (DBN) have been introduced which incorporate an aspect of time to allow networks with cycles to be found. One example is the work by Yu *et al.* that uses two different scoring functions to find the best matching network to their data [9]. They first generate a DBN to limit the number of parents considered for a gene since the second step is exponential in the number of parents. Next, the algorithm assigns an *influence score* to each potential connection. To find this score, they build a *cumulative distribution function* (CDF) for the child's potential parents. A CDF is basically a set of bins that indicate the number of times the genes are seen in a given configuration or an earlier one. For example, a configuration may be that proteins produced by all genes are at their highest state. If as the parent's value increases the resulting CDF is more positive than the previous one the parent is considered an activator. If the resulting CDF is more negative, then the parent is considered a repressor. As there can be many different parents and the data can be noisy, each case contributes a vote on the likelihood of activation or repression. To evaluate their method, they generate synthetic data for 10 genetic networks that include 20 genes in which 8 to 12 are connected to at least one other gene. Using this synthetic data, their method is shown to be fairly successful at recovering the initial networks given sufficient data. They still do, however, occasionally have problems determining the direction of influence. Time is also still only used to separate data points. Finally, only two of their networks have any feedback (i.e., cycles), and these cycles are several genes long.

This paper, therefore, presents a new method that targets learning genetic networks with feedback and in particular tight feedback. While this method is similar to the work by Yu *et al.* it differs in several key ways. First, while Yu's method begins with an expensive search for a best fit network, our method begins with a local analysis to determine the best potential parents for each gene. Second, our method utilizes the child's own level in the calculations to further help in determining a child's parent genes. Third, Yu's method neglects the time series nature of the data and simply computes a CDF that represents correlation within each individual experimental data point. Our method considers two sequential time series data points to find the percentage of time a child gene rises in a given configuration. We believe that by utilizing the time series nature of the data, our method can potentially better determine activation and repression behavior especially in the case where there is tight feedback in the network. In other words, we feel that not only the context of when a gene is high or low gives information, but more importantly the context in which it rises. Our method is evaluated using synthetic data generated from models of several genetic networks that include tight feedback. Our results indicate that our method is more successful than Yu's in learning these types of networks.

## 2 Formalisms

Our method produces a directed graph model of the genetic regulatory network that indicates which gene products activate and repress other genes. A directed graph model is the tuple  $\langle S, C \rangle$  where  $S$  is the set of species in the network and also the vertices of the graph. Connections between species, or edges in the graph, are represented by  $C \subseteq 2^{(S \times I)} \times S$ . Each connection,  $(P, s) \in C$ , is composed of a set of *parents*,  $P$ , for a child,  $s$ . Each parent in  $P$  is paired with its type of influence (i.e.,  $I = \{a, r\}$  where 'a' represents activation and 'r' represents repression). A directed graph representation of the genes in the phage  $\lambda$  decision network is shown in Figure 1(a) [10]. The 'a' edges are represented by  $\rightarrow$  and 'r' edges are represented by  $\dashv$ . For example,  $N \rightarrow$  CII implies that the proteins produced from gene N activate the gene CII. If there are  $n$  species, there are an astronomical  $2^{4^n}$  possible network topologies.



**Fig. 1.** (a) High level network representation for phage  $\lambda$ . (b) Time series data.

Time series data is assumed to be of the form  $D = \langle S, E \rangle$  where  $S$  is the set of species (typically proteins or mRNA) that is being monitored, and  $E$  is the set of time series data experiments. Each element of  $E$  is of the form  $\Delta = \langle M, \delta \rangle$ , where  $M$  is the set of species mutated in the experiment, and  $\delta$  is the set of data points. A major goal of this work is to integrate results from many different experiments to improve the quality of our predictions. For the best results, time series data should include both wild-type and mutational experiments. A data point  $\alpha \in \delta$  is of the form  $\alpha = \{\tau, \nu\}$  where  $\tau \in \mathbb{R}$  is a time point and  $\nu \in \mathbb{R}^{|S|}$  is a vector over  $S$ , which includes a value for each species. Time series data collected from microarray data, for example, would provide an expression level for each gene monitored by the microarray. Figure 1(b) shows example time series data.

Our method requires time series data as it is difficult to determine the direction of causality from purely correlational data. Consider for example the 5 gene network shown in Figure 1(a) in which the proteins from gene CII activate CI. With only correlational data it would appear that both CII and CI's proteins are either high or low together. It is, however, difficult to determine which gene activates the other. By adding the element of time, the gene that rose first can be detected, and thus an activation pattern can be determined more easily.

### 3 Discovering Genetic Networks

The **GeneNet** algorithm shown in Figure 2 generates a directed graph representation of the connectivity between genes in a genetic network. The main inputs to the **GeneNet** algorithm are the species,  $S$ , and the time series experiments,  $E$ . It also takes three optional inputs. The first is the set of initial network connections,  $C$ , representing background knowledge about known connections between species used to reduce computational time. The next inputs are the probability thresholds,  $T$ , for activation and repression. The final inputs are the encodings, or ranges of time series data values, specified in  $L$  for each molecular species. These encodings can be automatically calculated by taking the highest and lowest level of each species and dividing up the state space evenly. They can also be calculated by partitioning the level of each species into equal size bins, which is the default since it appears to be the most effective. The **GeneNet** algorithm first determines the level encodings if they are not provided and encodes the experimental data into discrete levels for each of the species. Next, each species is considered separately to determine the species (parents) that activate or repress the gene that produces this species (child). If parents are already provided in  $C$  for a species, it is skipped. The experiments where the child species is mutated are discarded as these provide no information for this species.

After initial parent genes are selected, the next step is to determine if parent genes act together to influence the child. The last step of the **GeneNet** algorithm is to determine the most likely parents using the **CompetePossibleParents** function which considers sets of potential parents together. The result of our method is a new network which includes only those activation or repression edges supported by the experimental data. Currently, our method does not look for self activation or self repression though it does allow for cycles resulting from any other type of feedback. The rest of this section uses the 5 gene network shown in Figure 1 as an example. This network is comprised of the 5 genes involved in the Lysis-Lysogeny decision in the phage  $\lambda$  [10]. The running example focuses on gene  $CIII$  as the child gene.

```

GeneNet(Species  $S$ , Expts  $E$ , NetCon  $C$ , Thresholds  $T$ , Encodings  $L$ )
  ( $E, L$ ) := EncodeExpts( $S, E, L$ )
  foreach  $s \in S$ 
    if  $(*, s) \notin C$ 
       $E' := E - \{ \text{experiments in which } s \text{ is mutated} \}$ 
       $C := \text{SelectInitialParents}(s, S, E', C, T, L)$ 
       $C := \text{CreateMultipleParents}(s, S, E', C, T, L)$ 
       $C := \text{CompetePossibleParents}(s, S, E', C, T, L)$ 
  return  $\langle S, C \rangle$ 

```

**Fig. 2.** The **GeneNet** algorithm.

### 3.1 Scoring Parents

Crucial to each step of the **GeneNet** algorithm is a method to evaluate different potential sets of parents for each child gene. This is accomplished with the **ScoreParents** function, shown in Figure 3, which returns a score representing the likelihood of a parent set of genes activating or repressing a child gene. This score is computed by tallying votes for activation ( $votes_a$ ), repression ( $votes_r$ ), and unrelated ( $votes_u$ ) for each potential level assignment. These votes are initially set to 0. Next, all possible level assignments for the other genes of interest,  $G$ , are calculated, and a level assignment,  $l'$ , is selected. The level assignments for the set of potential parents,  $P$ , are also calculated, and a level assignment,  $l$ , not equal to  $l_0$  (the lowest level assignment for the parents) is selected. Next, a probability ratio is calculated. The numerator is the probability of the child rising between two consecutive time points given that the parents are at the levels specified by  $l$  and the other genes of interest are at the level specified by  $l'$ . The denominator is the probability of the child rising when the parents are at their lowest level,  $l_0$ , while the other genes are at the levels specified by  $l'$ . If this probability ratio,  $probRatio$ , is larger than the activation threshold,  $T_a$ , then there is a vote for this parent set activating the child. If the ratio is smaller than a repression threshold,  $T_r$ , then there is a vote for repression. Otherwise, there is a vote for unrelated. The process continues checking each combination of level assignments. When all level assignments have been considered, the score is determined by taking the votes for activation and subtracting the votes for repression and dividing by the total amount of votes obtained. If activation votes outweigh repression votes the score is positive. If repression votes outweigh activation votes the score is negative.

```

ScoreParents( $s, P, G, E, T, L$ )
   $votes_a, votes_r, votes_u := 0$ 
  foreach  $l' \in levelAssignments(G, L)$ 
    foreach  $l \in (levelAssignments(P, L) - \{l_0\})$ 
       $probRatio = \mathcal{P}(s \uparrow | \nu_P = l, \nu_G = l') / \mathcal{P}(s \uparrow | \nu_P = l_0, \nu_G = l')$ 
      if  $probRatio > T_a$  then  $votes_a ++$ 
      else if  $probRatio < T_r$  then  $votes_r ++$ 
      else  $votes_u ++$ 
  return  $(votes_a - votes_r) / (votes_a + votes_r + votes_u)$ 

```

**Fig. 3.** The **ScoreParents** function.

Intuitively, if a parent set truly activates a child, then whenever the parents are low, the child should have a lower rising rate between time points than when the parents are high. This should be true regardless of the level of other species in the system. However, if the selected gene is not a parent and the true parents are in the  $G$  set, then by fixing the true parents to a low state, the probability of the child rising should not change as the false parent's level is increased.

### 3.2 Selecting Initial Parent Sets

The `SelectInitialParents` algorithm, shown in Figure 4, is used to select species from the set  $S$  as potential parents for the child species  $s$ . Every species except the child is considered as a potential parent. The `ScoreParents` function is used to determine if this parent is a potential activator or repressor of the child gene. If the score is higher than the vote threshold,  $T_v$ , (default value is 0.5) the parent gene is included in the set of network connections,  $C$ , as an activator. If the score is lower than the negative of the vote threshold,  $T_v$ , the parent gene is included as a repressor.

```

SelectInitialParents( $s, S, E, C, T, L$ )
  foreach  $p \in S - \{s\}$ 
     $score = ScoreParents(s, \{p\}, \{\{s\}\}, E, T, L)$ 
    if  $score \geq T_v$  then  $C := C \cup \{\{(p, a)\}, s\}$ 
    else if  $score \leq -T_v$  then  $C := C \cup \{\{(p, r)\}, s\}$ 
  return  $C$ 

```

**Fig. 4.** The `SelectInitialParents` algorithm.

Considering the gene *CIII* as the child, the scores obtained for the 4 other genes as parents are shown in Table 1. The result is that *N*, *Cro*, and *CI* are found to potentially repress *CIII* while *CII* potentially activates *CIII*. *CII*, however, is discarded since it does not have a high enough score.

**Table 1.** Rates and values for potential parents of *CIII*.

	Level <sub>0</sub>	Level <sub>1</sub>	Level <sub>2</sub>	L <sub>1</sub> / L <sub>0</sub>	L <sub>2</sub> / L <sub>0</sub>	Votes	Score
N							
CIII Level <sub>0</sub>	5.4%	2.9%	3.7%	0.53	0.68	votes <sub>a</sub> 0	
CIII Level <sub>1</sub>	9.3%	7.2%	6.7%	0.78	0.72	votes <sub>r</sub> 5	
CIII Level <sub>2</sub>	8.6%	6.4%	6.1%	0.75	0.71	votes <sub>u</sub> 1	-0.83
Cro							
CIII Level <sub>0</sub>	11.5%	1.8%	1.5%	0.16	0.13	votes <sub>a</sub> 0	
CIII Level <sub>1</sub>	14.2%	4.7%	3.1%	0.33	0.23	votes <sub>r</sub> 6	
CIII Level <sub>2</sub>	9.7%	5.0%	4.2%	0.52	0.43	votes <sub>u</sub> 0	-1.0
CI							
CIII Level <sub>0</sub>	19.0%	1.7%	1.0%	0.09	0.05	votes <sub>a</sub> 0	
CIII Level <sub>1</sub>	17.1%	2.6%	1.2%	0.15	0.07	votes <sub>r</sub> 6	
CIII Level <sub>2</sub>	11.6%	2.7%	1.1%	0.23	0.09	votes <sub>u</sub> 0	-1.0
CII							
CIII Level <sub>0</sub>	3.1%	13.7%	-	4.32	-	votes <sub>a</sub> 3	
CIII Level <sub>1</sub>	4.4%	7.4%	12.6%	1.65	2.83	votes <sub>r</sub> 2	
CIII Level <sub>2</sub>	19.4%	5.5%	6.8%	0.28	0.35	votes <sub>u</sub> 0	0.2

### 3.3 Creating Multiple Parent Sets

The `CreateMultipleParents` algorithm, shown in Figure 5, is used to test if two or more parents exert a combined effect on the child gene. The current implementation though is limited to parent sets of size two. The algorithm considers two parent sets,  $p_1$  and  $p_2$ , at a time. It first computes the scores for these parent sets (note that this can actually be retrieved from a cache rather than being recomputed and that  $R()$  removes  $a$  or  $r$  from each pair in  $p_1$  and  $p_2$  leaving only the parents). Next, it computes the score for the combination of these two parent sets (i.e.,  $p_1 \cup p_2$ ). If the score for the combination is better than both of the simpler parent sets, then this combination is inserted into  $C$  and potentially combined again. Finally subsets are removed from the set of parents.

```

CreateMultipleParents( $s, S, E, C, T, L$ )
  foreach ( $p_1, s$ )  $\in C$ 
     $score_1 := ScoreParents(s, R(p_1), \{\{s\}\}, E, T, L)$ 
    foreach ( $p_2, s$ )  $\in C - \{(p_1, s)\}$ 
       $score_2 := ScoreParents(s, R(p_2), \{\{s\}\}, E, T, L)$ 
       $score_b := ScoreParents(s, R(p_1) \cup R(p_2), \{\{s\}\}, E, T, L)$ 
      if  $abs(score_b) \geq abs(score_1)$  and  $abs(score_b) \geq abs(score_2)$  then
         $C := C \cup \{(\{p_1\} \cup \{p_2\}, s)\}$ 
   $C := removeSubsets(C)$ 
  return  $C$ 

```

**Fig. 5.** The `CreateMultipleParents` algorithm.

Table 2(a) shows the votes and scores for the three different potential parents of *CIII* as well as the different combinations of parents. For this example, the combination of *N* and *Cro* does not have a better score than *Cro* alone, so it is not retained. The repressor combination of *Cro* and *CI*, however, has a score better or equal to each individually, so this combination is retained.

**Table 2.** Information computed for child CIII.

CreateMultipleParents Algorithm						CompetePossibleParents Algorithm				
Parents	votes <sub>a</sub>	votes <sub>r</sub>	votes <sub>n</sub>	Score	Retained	Parents	votes <sub>a</sub>	votes <sub>r</sub>	votes <sub>n</sub>	score
N	0	5	1	-0.833		N	27	12	15	0.27
Cro	0	6	0	-1.0		Cro, CI	0	71	1	-0.98
CI	0	6	0	-1.0						
N, Cro	0	23	1	-0.958	no					
N, CI	0	21	3	-0.875	no					
Cro, CI	0	24	0	-1.0	yes					

(a)

(b)

### 3.4 Competing Possible Parent Sets

The `CompetePossibleParents` algorithm in Figure 6 competes parent sets until only one parent set remains. The `getContenders` function returns sets of parents to be competed (for example, the sets with the highest and lowest scores in  $C$ ). Next, a score is calculated for each contender,  $q$ , controlling for the levels of the other contenders (i.e.,  $Q - \{q\}$ ). Finally, the scores are compared and the losers, parent sets with scores closest to 0, are removed from  $C$ . Also if a parent set's score represents a change in influence (for example, an activator with a negative score), this parent set is removed. This algorithm is run in a tournament fashion to limit the number of configurations which prevents the data from becoming too sparse. The contenders for  $CIII$  are  $\{N\}$  and  $\{Cro, CI\}$ . Table 2(b) shows the scores for each set of genes. The winner is the set  $\{Cro, CI\}$  so the `GeneNet` algorithm correctly identifies that  $CIII$  is repressed by  $Cro$  and  $CI$ .

```

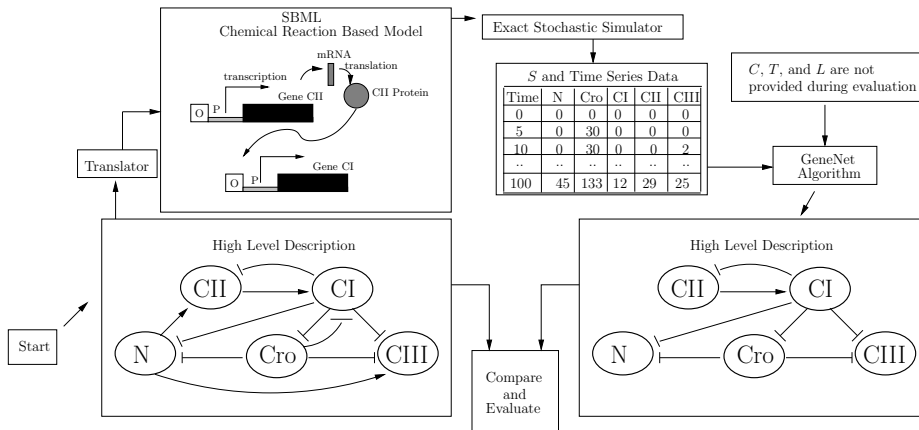
CompetePossibleParents( $s, S, E, C, T, L$ )
  while  $|\{c \in C | c = (*, s)\}| > 1$ 
     $Q = \text{getContenders}(s, S, E, C, T, L)$ 
    foreach  $q \in Q$ 
       $Scores_q = \text{ScoreParents}(s, \{q\}, (Q - \{q\}) \cup \{\{s\}\}, E, T, L)$ 
     $C := \text{removeLosers}(C, Q, Scores)$ 
  return  $C$ 

```

**Fig. 6.** The `CompetePossibleParents` algorithm.

## 4 Results

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 generated synthetic data for the phage  $\lambda$  network and several randomly generated networks. This procedure is shown in Figure 7. The high level description of the gene interactions are translated into a very detailed reaction-based model expressed in the *Systems Biology Markup Language* (SBML). This purely reaction-based model can be simulated stochastically with the U. of Tennessee's *Exact Stochastic Simulator* (ESS). ESS implements an optimized version of Gillespie's *stochastic simulation algorithm* [11] which produces the time evolutions of species in a well-stirred biochemical system. A limited number of time points from this simulation are selected to produce a synthetic time series data run for the original SBML model, which is then used by the `GeneNet` algorithm to construct a network. The original network is compared to the results of our method. As shown in Figure 7, 7 of the 10 arcs in the phage  $\lambda$  network are obtained and no spurious arcs generated. The tool described in [9] recovers 3 of the 10 arcs and reports 3 spurious arcs.



**Fig. 7.** Data flow for the GeneNet algorithm evaluation.

In addition to the phage  $\lambda$  network, our method is examined on 18 synthetic networks inspired by the 4 gene plasmids created by Guet *et al.* [12]. Given 20 experiments with 100 data points generated by ESS for each of these networks, the GeneNet algorithm finds all 72 arcs but reports a total of 74. The DBN tool from [9], only finds 33 of the 72 arcs but reports 43 total arcs.

We also created 10 high level 10 gene randomized networks that followed the Guet *et al.* framework. For this example, we also varied the number of experiments where each experiment includes 200 data points. The results are shown in Table 3 using both our GeneNet algorithm and Yu's DBN algorithm. The runtimes of the two tools are very similar. Precision difference between algorithms are minimal for the first two cases, but increase as more experiments are added. There are larger differences in recall. For example, the GeneNet algorithm finds 26 percent of the arcs using only 1 experiment and nearly 60 percent using only 5 experiments. On the other hand, the DBN algorithm is only able to find 11 percent of the arcs with 5 experiments. With 50 experiments, GeneNet finds 94 percent while the DBN method still only finds 47 percent.

**Table 3.** Results for 10 gene networks.

Number of Experiments	GeneNet		DBN [9]	
	Recall	Precision	Recall	Precision
1	26% (26/100)	19% (26/135)	4% (4/100)	15% (4/26)
5	58% (58/100)	46% (58/126)	11% (11/100)	48% (11/23)
10	75% (75/100)	57% (75/132)	14% (14/100)	42% (14/33)
25	82% (82/100)	65% (82/127)	37% (37/100)	47% (37/79)
50	94% (94/100)	70% (94/135)	47% (47/100)	49% (47/96)

## 5 Discussion

This paper describes a new method for learning connectivities in genetic networks from time-series data using a Bayesian type of analysis. Although the results are promising, there are way to improve them. To reduce noise from the stochasticity of the data a low pass filter can be used. Interpolation can also be used to create intermediate data points to help smooth the data. *Transitive edges*, a special case of extra edges formed from 'shortcuts' when multiple genes are connected by a path, may potentially be removed or marked for later review by graph traversal algorithms. The use of data with interventions (i.e., mutational data) may also significantly improve the results. While our method does allow for cycles as opposed to traditional Bayesian approaches, it does not yet detect self-activation or repression. Finally we plan to apply our method to real experimental data.

### Acknowledgments

The authors thank Jing Yu of Duke for providing his software and Michael Samoilov and Adam Arkin of UC Berkeley for their comments on this work.

### References

1. Brown, P.A., Botstein, D.: Exploring the new world of the genome with DNA microarrays. *Nature Genet.* **21** (1999) 33–37
2. Eisen, M.B., Spellman, P.T., Brown, P.O., Botstein, D.: Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA* **95** (1998) 14863–14868
3. Liang, S., Fuhrman, S., Somogyi, R.: REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. In: *Pacific Symposium on Biocomputing*. Volume 3. (1998) 18–29
4. Akutsu, T., Miyano, S., Kuhara, S.: Identification of genetic networks from a small number of gene expression patterns under the boolean network model (1999)
5. Ideker, T.E., Thorsson, V., Karp, R.M.: Discovery of regulatory interactions through perturbation: Inference and experimental design (2000)
6. Lähdesmäki, H., Shmulevich, I., Yli-Harja, O.: On learning gene regulatory networks under the boolean network model. *Machine Learning* **52** (2003) 147–167
7. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using bayesian networks to analyze expression data. *Journal of Computational Biology* **7** (2000) 601–620
8. Sachs, K., Perez, O., Pe'er, D., Lauffenburger, D.A., Nolan, G.P.: Causal protein-signaling networks derived from multiparameter single-cell data. *Science* **22** (2005) 523–9
9. Yu, J., Smith, V.A., Wang, P.P., Hartemink, A.J., Jarvis, E.D.: Advances to bayesian network inference for generating causal networks from observational biological data. *Bioinformatics* **20** (2004) 3594–3603
10. Ptashne, M.: *A Genetic Switch*. Cell Press & Blackwell Scientific Publishing (1992)
11. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* **81** (1977) 2340–2361
12. Guet, C.C., Elowitz, M.B., Hsing, W., Leibler, S.: Combinatorial synthesis of genetic networks. *Science* **296** (2002) 1466–1470