Using Bayesian Network Inference Algorithms to Recover Molecular Genetic Regulatory Networks

Jing Yu[1,2], V. Anne Smith[1], Paul P. Wang[2], Alexander J. Hartemink[3], Erich D. Jarvis[1]
[1]*Duke University Medical Center, Department of Neurobiology, Box 3209, Durham, NC 27710*
[2]*Duke University, Department of Electrical Engineering, Box 90291,Durham, NC 27708*
[3]*Duke University, Department of Computer Science, Box 90129, Durham, NC 27708*

Recent advances in high-throughput molecular biology has motivated in the field of bioinformatics the use of network inference algorithms to predict causal models of molecular networks from correlational data. However, it is extremely difficult to evaluate the effectiveness of these algorithms because we possess neither the knowledge of the correct biological networks nor the ability to experimentally validate the hundreds of predicted gene interactions within a reasonable amount of time. Here, we apply a new approach developed by Smith, *et al.* (2002) that tests the ability of network inference algorithms to accurately and efficiently recover network structures based on gene expression data taken from a simulated biological pathway in which the structure is known *a priori*. We simulated a genetic regulatory network and used the resultant sampled data to test variations in the design of a Bayesian Network inference algorithm, as well as variations in total quantity of available data, length of sampling interval, method of data discretization, and presence of interpolated data between observed data points. We also advanced the inference algorithm by developing a heuristic influence score that infers the strength and sign of regulation (up or down) between genes. In these experiments, we found that our inference algorithm worked best when presented with data discretized into three categories, when using a greedy search algorithm with random restarts, and when evaluating networks using the BDe scoring metric. Under these conditions, the algorithm was both accurate and efficient in recovering the simulated molecular network when the sampled data sets were large. Under more biologically reasonable small amounts of sampled data, the algorithm worked best only when interpolated data was included, but had difficulty recovering relationships describing genes with more than one regulatory influence. These results suggest that network inference algorithms and sampling methods must be carefully designed and tested before they can be used to recover biological genetic pathways, especially in the context of highly limited quantities of data.

INTRODUCTION

The advent of novel technologies for collecting high-throughput data in molecular biology has led to the concurrent development of bioinformatics tools for analyzing this data. Computer scientists and bioinformaticians soon realized that common inference algorithms used in other fields can be applied to these large amounts biological data, such as those from microarrays, to statistically predict causal molecular pathways. However, these potentially powerful algorithms are limited by our inability to evaluate their accuracy, as we do not know the true biological network in which to compare them with and experimenters can not physically perform in reasonable time the multiple gene knockouts or other types of interventions required to systematically test the predicted networks.

As part of an ongoing project dedicated to integrating the songbird brain (Jarvis et al. 2002), Smith, et al. (2002) developed a novel approach for evaluating the accuracy and efficiency of network inference algorithms in a reasonable amount of time. This approach requires the creation of a biologically reasonable simulation on a computer in which the experimenter makes and knows all the rules. As the simulation runs, data is sampled from it as one would sample data from a real biological system. The sampled data is then passed to an inference algorithm to evaluate the algorithm's ability to recover the simulated system. The inference algorithm can then be modified and made more robust to recover a network that closely matches the simulated system. After confident recovery of the system from limited simulated data is achieved, the algorithm can be applied to real data. The recovered system can then be used to guide further biological experimentation for verifying the predicted regulatory relationships.

In our first use of this approach (Jarvis et al. 2002; Smith et al. 2002), we incorporated multiple levels of biological organization, from the molecular to the behavioral. Here, we attempt to look more closely at a single level of biological organization, the molecular level. We developed a simulator, which we named GeneSimulator, that models genetic regulatory networks and generates correlational data similar to that collected from high-density gene microarrays. We then evaluated various Bayesian network (BN) inference algorithm designs for their ability to recover the underlying genetic regulatory network. We chose to use a BN algorithm, because compared with other common algorithms (Somogyi and Sniegoski 1996; D'haeseleer et al. 1999; Weaver et al. 1999), BN have the ability to simultaneously model non-linear combinatorial relationships, robustly handle noisy data sets, and guard against over-fitting. BN can not handle networks with cyclic structures, such as regulatory feedback loops; however, dynamic Bayesian networks (DBN) can handle cyclic structures (Friedman et al. 1999; Murphy and Mian 1999). We used DBN, and when so configured, they are also capable of coping with hidden variables that are not observed in the data, such as protein levels or protein interactions that affect the measured gene expression data. In the DBN inference algorithm we developed here, we tested different scoring metrics and heuristic search methods, as well as different aspects of data collection and discretization, in order to determine the best configuration for recovering the simulated system. Our analysis provides insight on how to more efficiently use BN inference algorithms for discovering genetic networks from correlational data.

METHODS AND LOGIC

GeneSimulator

GeneSimulator is programmed in Matlab (MathWorks, Inc.). It models genetic regulatory pathways of arbitrary structure (topology) and produces values of gene expression levels at discrete time points. Updates to values at each time step are governed by a simple stochastic process:

$$Y_{t+1} = f(Y_t) = A(Y_t - T) + \varepsilon$$

where Y_t is a vector representing the expression levels of all genes at time t, with expression levels ranging from 0 to 100 (arbitrary units). A is a matrix that represents the relationships of gene interactions in the underlying regulatory pathway. For every entry of A, the magnitude of the entry represents the strength of the regulation that a regulator gene exerts upon a target gene; the sign indicates the type of regulation, with positive values indicating up-regulation and negative values indicating down-regulation. T is a vector of *threshold regulating values* for each gene: a regulatory gene exerts an influence on its target gene only to the extent that it differs from its threshold value. In this study all gene thresholds have been set to half of the maximum value; *i.e.*, every entry of T is exactly 50. If the regulator gene is present at a level above its threshold value, then its regulatory effect on its target genes occurs as specified in A. Contrarily, if the regulator gene is present at a level below its threshold value, then its effect is in the opposite direction of that specified in A to return the gene to its basal level. The ε term is white noise, drawn uniformly at random from the interval –10 to 10. This term is added for stochasticity and is meant to capture all sources of noise, especially inherent biological noise. If a gene has no regulator (the corresponding entries in A are all zero), then it will move in a random walk, with steps taken according to the values of ε. As the simulation runs, the data is sampled in pre-specified intervals as one would do in an actual biological experiment, and the samples are exported to a text file. For example, if we collect data every five time points, then we define the sampling interval to be 5, and the sampled output is the series of expression level vectors ($Y_0, Y_5, Y_{10}, \ldots\ldots$), analogous to data gathered in a microarray time course experiment.

Data Processing and Collection Methods

Discretization: Before being passed to our DBN inference algorithms, the data we collect needs to be discretized. Discrete data allows us to model complex non-linear interactions between genes without resorting to computationally prohibitive calculations over continuous distributions. In this study, we discretized the sampled expression levels generated by GeneSimulator from continuous values into various numbers of categories to determine if finer or coarser discretization improves recovery accuracy. We also evaluated two general types of discretization strategies: hard and soft. Hard discretization employs firm boundaries between categories, requiring a given expression level to belong to only a single category. Soft discretization employs fuzzy boundaries between categories, allowing a given expression level to belong to two or more categories with different percentages each. Other data processing and collection methods are described in the results.

Bayesian Network Inference Algorithms

Our DBN inference algorithms are written in C++ and are designed to search for high-scoring graphical models (networks) that describe probabilistic relationships between variables. The score that is computed for a graph generated from the data collected and discretized is a measure of how successfully the graph explains the relationships in the data and also how simply it does so. Graphs are penalized for over-complexity or over-generality so there is a resultant bias towards simpler graphs. This guards against over-fitting the model to the data.

Every node in the BN graph represents a single variable, here one gene. Every directed edge, or lines with arrows, between two nodes represents a conditional statistical dependence of the child node on the parent node. In the context of a DBN fro recovering a genetic regulatory network, each edge indicates a regulatory relationship in which the parent gene regulates the child gene at a later time.

***Basic Theory of BN*:** A static BN (Friedman et al. 2000) is an acyclic directed graph that encodes a joint probability distribution over χ, where $\chi = \{X_1, \ldots, X_n\}$ is a set of discrete random variables X_i. The BN for χ is specified as a pair $< G, \Theta >$. The variable G represents a directed graph whose vertices correspond to the random variables $X_1, \ldots, X_n$. In this graphical representation, each variable X_i is independent of its non-descendants given its parents in G. The variable Θ represents a set of parameters that collectively quantify the probability distributions associated with the variables in the graph. Each parameter of Θ is specified by $\theta_{x_i|pa(X_i)} = P(x_i \mid pa(X_i))$, for each possible value x_i of X_i, and each possible value $pa(X_i)$ of $Pa(X_i)$. $Pa(X_i)$ denotes the set of parents of X_i in G and $pa(X_i)$ denotes a particular instantiation of the parents. Thus, a BN specifies a unique joint probability distribution over χ given by:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa(X_i))$$

These notions extend quite naturally to DBN, which we exploit here in the context of time series data (for more details, see Murphy and Milan 1999).

The problem of discovering a BN from a collection of observed data can be stated as follows. Given a data set $D = \{Y_1, Y_2, Y_3, \ldots\ldots Y_n\}$ of observed instances of χ, find the most probable graph *G* for explaining the data contained in *D*. One common approach to this problem is to introduce a scoring metric that evaluates how probable each graph *G* explains the data in *D*. In the presence of such a scoring metric, the problem of discovering a BN then reduces to the problem of searching for a graph that yields a high score, given the observed data in *D*. To search the highest scoring graph, a particular search method needs to be used.

***Bayesian Scoring Metrics*:** The Bayesian scoring metric can be generally described (Heckerman 1996) as:

$$Score(G : D) = \log P(G \mid D) = \log P(D \mid G) + \log P(G) - \log P(D)$$

Which states that the score of the graph G given data in D is equivalent to the log of the probability of G given D. In this study, we investigated two types of scoring metrics that employ different assumptions: the BDe (Bayesian Dirichlet equivalent) and the BIC (Bayesian Information Criterion) scoring metrics. Both scoring metrics have an inherent penalty for over-complexity to guard against over-fitting of data. The BDe score captures the full Bayesian posterior probability *P(G|D)*. In this

setting, the prior over graphs needs to be specified (we use a uniform prior) and the prior over parameters is Dirichlet, i.e. a multinomial distribution describing the conditional probability of each variable in the network. The BIC score instead of capturing, is an asymptotic approximation to the full posterior probability *P(G|D)*. This approximation is based on a penalized maximum likelihood estimate. With large amounts of data, the BIC is a good approximation to the full posterior (BDe) score and is faster to compute; however, it is known to over-penalize with small amounts of data.

Score metrics also involve the generation of a conditional probability table (CPT) for each node. The tables store the occurrences for all combinations of parent-child values extracted from the discretized data (Heckerman 1996). The occurrence values in the tables are called sufficient statistics. For a given graph, a CPT table is made for each node. The occurrence values in the tables are then used to calculate the score for each node. Scores for all nodes are then summed to generate the score for the entire graph.

Search Methods To identify BN structures with high scores, search methods are employed that search for the highest scoring graph among a set of graphs using different heuristic methods. The reason for heuristic search methods is that identifying the highest scoring network using scoring metrics is NP complete. As such, heuristic searches are iterative and thus can be run indefinitely and stopped at any time to reveal the highest scoring graph visited thus far. The longer the search, the likely of finding a higher-scoring graph. A suitable cutoff for running time is found empirically, where longer running times do not result in significant changes to the highest scoring graph found. In our study, we tested three heuristic search methods: 1) greedy search with random restarts (Heckerman 1996), 2) simulated annealing (Heckerman 1996), and 3) a genetic algorithm (developed in this study). As in Heckeman (1996), for each type of search we used E to denote the set of eligible changes to a graph and $\Delta(e)$ to denote the change in score of a graph resulting from the modification $e \in E$, where stands for every eligible change. In addition, we created a hash table (a look up table) to store the calculated scores for each node with a certain parent set. We found that this improved the performance of the search significantly, saving computational time by avoiding the recalculation of sufficient statistics in the CPTs when revisiting a previous set of parents for a given node.

Greedy search with random restarts initializes itself by choosing a random graph, then evaluates the change in score $\Delta(e)$ associated with every possible change $e \in E$, and finally selects the change for which $\Delta(e)$ is maximized, provided the maximal $\Delta(e)$ is positive. It proceeds in this fashion until all $\Delta(e)$ are negative and no score improvement can be made. To escape this local maximum, the algorithm then restarts from another random graph, and the entire process is repeated until the total number of iterations is reached.

Simulated annealing also initializes itself by choosing a random graph, but is given an initial temperature T_0, a search parameter. An eligible change $e \in E$ is selected at random and the probability expression $p = \exp(\Delta(e)/T_0)$ is evaluated. If $p > 1$ (which occurs whenever $\Delta(e)$ is positive), then the change e is made; otherwise, the change e is only made with probability p. The procedure begins at a very high temperature so that almost every eligible change in the graph can be made. As the search progresses, the temperature gradually decreases until a very low temperature is reached where very little change is made in the graph. The search then performs similarly to the local searches of the random greedy method.

A genetic algorithm (GA) (Goldberg 1989) is a search method using three operators to explore a space of solutions or, in our case, a set of graphs. The three operators are: *reproduction*, which promotes the best graphs to the next generation, *mutation*, which explores new graphs by introducing variation in the population to avoid local optima, and *crossover*, which selects a swapping point in the parents and exchanges information between them to generate two new graphs, thereby increasing the average quality of a population. We are not aware of a GA being applied to BN searches, and thus explain the operations we implemented in more detail. In our genetic algorithm, we generated a mutation operation that makes a local change in any possible edge in the graph structure. We generated a crossover operation that swaps parts of two graphs. Graph structures are specified as the set of parents for every node, where graph i is donated as $\{Pa_i(X_1), Pa_i(X_2), ..., Pa_i(X_n)\}_i$ and graph j as $\{Pa_j(X_1), Pa_j(X_2), ..., Pa_j(X_n)\}_j$. To crossover, a randomly chosen variable X_k becomes the swap point leading to two new structures, graph i' $\{Pa_i(X_1), ..., Pa_i(X_k), Pa_j(X_{k+1}) ..., Pa_j(X_n)\}_{i'}$ and graph j' $\{Pa_j(X_1), ..., Pa_j(X_k), Pa_i(X_{k+1}) ..., Pa_i(X_n)\}_{j'}$. For each GA iteration, either a mutation or a crossover operation is chosen at random and the newly created graphs are reproduced in the next generation if they have higher scores than the current graphs in the stored population. As it is possible for crossovers to create bi-directional edges, we check for and eliminate such graphs.

***Influence Score*:** Many BN inference algorithms applied to molecular biology are useful for predicting which genes regulate which others, but often do not predict the magnitude or even the sign of regulation (an exception is Hartemink et al 2001). Here using a different approach, we took advantage of the CPT table logic. The occurrence numbers in CPT tables suggest relationships between nodes. Here, we generated CPT-like tables, which we call influence tables, for each pair of connected nodes in the highest scoring graph. The values in the influence tables are occurrences from the discretized data according to the highest graph. An example of an influence table, when there is only one parent node per child node, is shown in **Table 1**. Here, three category discretization was used: low (L), medium (M), and high

		Parent		
		L	**M**	**H**
Child	**L**	LL	ML	HL
	M	LM	MM	HM
	H	LH	MH	HH

Table 1: An example influence score table. The table is a modified version of a CPT without prior knowledge information. Both the parent and child states are discretized as L, M, or H. The values in the cells represent the occurrence of each parent-child combination where the first descriptor (L, M, or H) is the parent and the second (L, M, or H) is the child. For clarity, this table contains the simple case where there is only one parent per child. In practice, the table has more dimensions to account for multiple parents.

(H). The occurrence values in the table are then described as combinations, for example, where HL is the number of times in the data that the parent is high and the child is low. When for a given node the occurrence of LL and HH is greater than the occurrence of LH and HL, then parent gene can be said to be an activator of the child gene. And vice-versa, for the parent being a repressor. This is even the case without considering MX values. Thus, to calculate an influence score from these occurrence values, we used the following formulas:

$$LL\_HL = \frac{LL - HL}{LL + HL} \quad \text{and} \quad HH\_LH = \frac{HH - LH}{HH + LH}$$

$$LL\_LH = \frac{LL - LH}{LL + LH} \quad \textit{and} \quad HH\_HL = \frac{HH - HL}{HH + HL}$$

The signs (+ or -) of the numerators determines the sign of the gene regulation. When LL-HL and HH-LH is greater than 0, and LL-LH and HH-HL is greater than 0, then the parent is considered an activator and gets a + sign. The reverse is the case when both of these values are less than 0, then the parent receives a – sign. Different combinations of these values, such that one is greater and the other is less than 0, the score changes to less positive or less negative, or to just 0 (no sign can be determined). The numerator scales the denominator to generate values between -1 and 1. With four such values, the final sign and magnitude of the influence is calculated with the next set of rules and formulas:

First set the influence score to 0.

If both of LL_HL and HH_LH are greater than 0 (LL and HH dominate), then add the magnitude (LL_HL*HH_LH)/2 to the influence score 0. If both of LL_HL and HH_LH are less than 0 (HL and LH dominate), then subtract (LL_HL*HH_LH)/2 from the influence score 0. Otherwise, keep the influence score 0.

Perform similar operations for LL_LH or HH_HL. After, add both values from both sets of operations and sum them to obtain the final influence score.

These two symmetric steps are used to average the effects from different directions. If several parents are present, the same calculation is done with the other parents fixed in each possible configuration; the average of these influence scores is then used. Regardless of the number of levels of discretization, only the lowest and highest categories are included into the calculation.

In the above manner, the influence score is mapped to a range between –1 and 1. Stated in terms of gene expression, positive numbers reflect activating relationships of a parent on a child gene, while negative numbers reflect repressing relationships. The magnitude of the influence score is suggestive of the gene regulation strength, which means the more positive the score is, the stronger the up-regulation is; the more negative the score is, the stronger the down-regulation is. When the influence score is close to 0, it is difficult to infer the type of regulation (up or down).

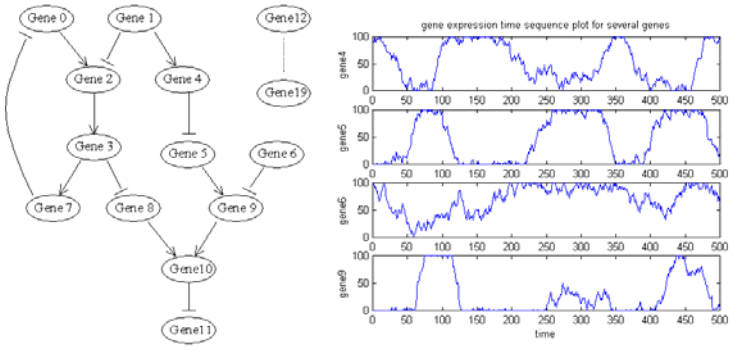

Figure 1: Regulatory network and gene expression time sequence plot.

RESULTS

We first present results generated by the simulator, then examine different configurations for the BN inference algorithm using the same simulated data set. We then examine different sampling and data processing methods using different data sets.

GeneSimulator

Using GeneSimulator, we simulated a genetic regulatory system defined in a matrix A of relationships with the structure shown in the left half of **Figure 1**. Twelve of twenty genes (genes 0 to 11) were connected in a regulatory pathway. In addition a feedback connection, from gene 7 to gene 0, was included. The other eight genes (genes 12 to 19) were not connected to other genes; *i.e.*, they were independent of all other genes. The absolute regulation strength for each connection was set to be the same: 0.1. Consequently, in the matrix A, for every up-regulation of gene y from gene x, $A(y,x) = 0.1$; if the relationship is one of down-regulation, then $A(y,x) = -0.1$. If there is no connection between genes x and y, the corresponding entry $A(y,x) = 0$. To show how GeneSimulator works, we ran it for 500 time points and plotted expression levels for four of the regulated genes in the right half of **Figure 1**. The results are consistent with the relationships specified by the original structure: when gene 4 increase gene 5 decreases, consistent with gene 4 down-regulating gene 5; gene 6 engages in a random walk, consistent with it having no regulator; when gene 5 is high and 6 is low, gene 9 is high, and this is consistent with gene 9 being both up-regulated by 5 and down-regulated by 6. Moreover, the changes in expression occur over a series of time steps. Although our time steps are unitless, if each is considered to be one minute, they match well the time-scale noted for gene expression in actual biological systems (Jarvis and Nottebohm 1997). This underlying regulatory structure was used for all of experiments, except for the complexity of network section. As such, throughout the paper **Figure 1** (left) is to be compared with all other figures.

Bayesian Network Inference Algorithms

Initially we sampled data from GeneSimulator output on a small scale—20 or fewer sampled time points—matching what would be done experimentally using microarrays or other approaches. However, we found that such small sample sizes, without further data processing, as described in the *Data Collection Methods* section below, were not sufficient for recovering the simulated structure. In this section, to more effectively evaluate the BN and the different configurations we made, we ran GeneSimulator with the system of **Figure 1** (left) for 10,000 time points and sampled at an interval of every 5 time points, yielding a total of 2000 sampled time points.

Bayesian Scoring Metrics: To compare the BDe and

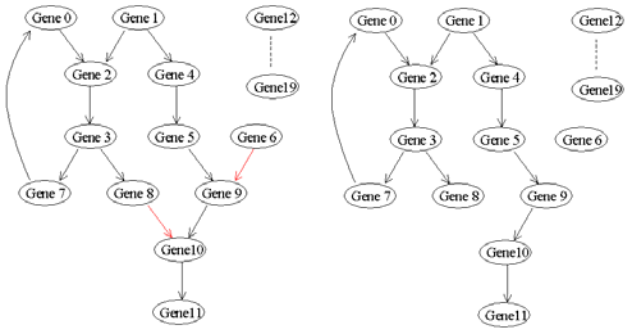

Figure 2: Comparison of graphs found using BDe (left) and BIC (right) scores methods. Shown are the top graphs found. Red – correct edges found by one method and not the other. The arrows in these graphs only specify direction, not positive or negative interactions.

BIC scoring metrics, we chose to use a 3-category hard discretization of the data and a greedy search method with 1000 random restarts. Compared with the original simulated structure in **Figure 1** (left), the BN inference algorithm under these conditions had remarkably good recovery, using either scoring metric (**Figure 2**). All genes (nodes) in the connected networks and most interactions (edges) were recovered. For the BDe scoring metric, the top graph (highest scoring graph) had exactly the same regulatory structure as the simulated system. For the BIC scoring metric, the top graph had two edges missing. We conclude that the BDe score works better than the BIC score in recovering the underlying simulated genetic regulatory pathway given this quantity of sampled data. The missing edges under the BIC score are consistent with its known over-penalization of model complexity.

***Influence Score*:** We found that our heuristic influence score function worked, and it gave reasonable results when compared with the type of regulation (up or down) in the simulated system (**Figure 3**). The signs of the interactions (+ and –) were all correctly identified (compare **Figure 1** with **Figure 3**). The absolute magnitudes of the influences scores in the recovered networks (0.11-0.63) and the regulatory strengths specified in the simulated system (0.1) although in the same range, were not directly comparable. Furthermore, in the recovered network when a node had more than one parent, the edges from those parents had lower influence score magnitudes than if the node had just one parent. This is the case because the influence of one parent to its child is obscured by the influences from the other parent, and the resulting difference is an artifact of how the influence score is calculated. Given these weaknesses, the influence scores still reflected relatively well the gene relationships. For genes with one versus two parents, the influence scores of the recovered edges are similar within each case (~0.6 for single parents; ~0.3 for two parents, **Figure 3**) as in the simulated network. We conclude that the influence score is capable of recovering the sign of the interaction, but its relative magnitude depends upon the number of parents.

Search Methods: To compare heuristic search methods, we used BDe score with 3-category hard discretization. Because it is difficult to set a fair stopping criterion for different search methods, we let each of them run long enough that they did not make any improvement for many iterations (determined empirically). We compared three search methods: greedy search with random restarts, simulated annealing, and a genetic algorithm. All three yielded the same top graphs with exact matches to the simulated system, as in the left of **Figure 2**. However, we found that with this data set greedy search took the least time (minutes) to find the correct graph with the highest score; simulated annealing a longer time (10s of minutes); and genetic algorithm took the greatest amount of time (~hrs). We conclude that the three search methods worked equally well in terms of the top graph found, but the greedy search is best as it can find the top graph in the least amount of time.

Data Processing and Collection Methods

For this section, we used the BDe scoring metric and the greedy search method with 1000 random restarts.

Discretization: Using the same data set as above, we compared the performance of the BN inference algorithm with discretization into different numbers of categories: hard discretization into 2, 3, and 4 categories. The top graph found for each is shown in **Figure 4**. All of these discretization approaches allowed recovery of relatively similar graphs to the simulated system. Only the 3-category discretization found exactly the same regulatory graph as the simulated system. In the case of the 2-category discretization, extraneous edges were found. These likely emerged because too much of the information contained in the data was lost in the overly-coarse discretization. On the other hand, the 4-category discretization led to some difficulty recovering edges whose children had

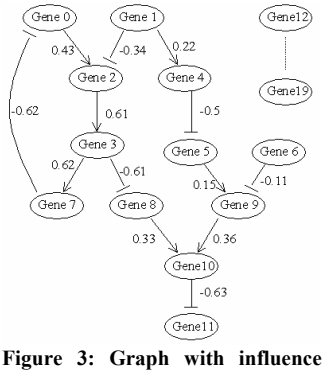

Figure 3: Graph with influence scores. Numbers besides the edges are influence scores from the top BDe generated graph of Figure 2.

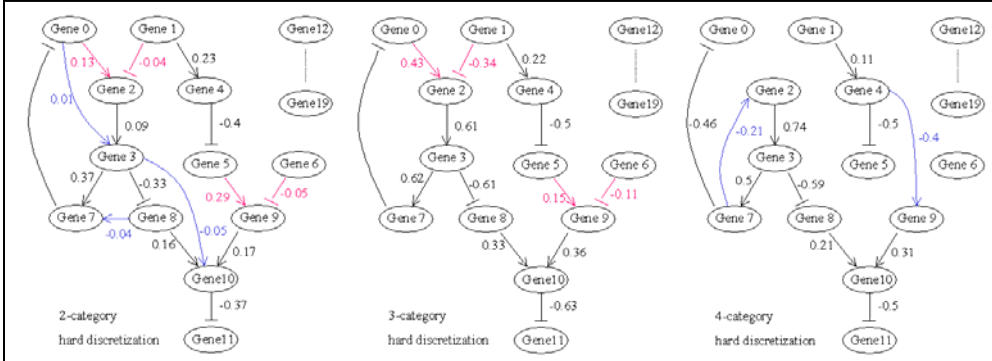

Figure 4: Comparison of number of discretization categories, with hard boundaries. Black – correct edges in common between all three graphs. Red – correct edges found in only one or two of the graphs. Blue – incorrect edges. Numbers beside edges are influence scores.

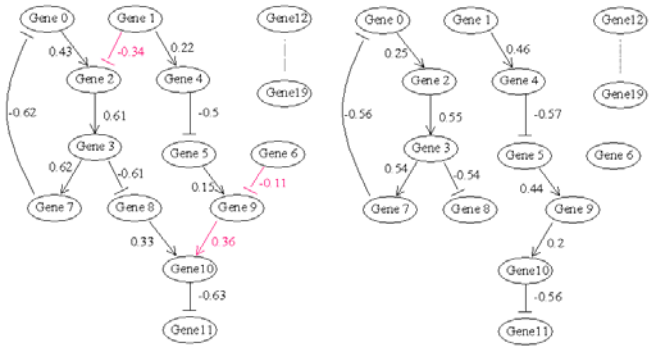

Figure 5: Comparison for hard (left) and soft discretization (right). Red – correct edges found in the top graph by one method and not the other. Numbers beside edges are influence scores.

multiple parents, and instead it found incorrect edges from their grandparents. We believe this occurs because increasingly fine levels of discretization spreads out the data entered in the CPTs, such that individual occurrence values are weakened. To strengthen these values, intuitively, would require more data.

We next compared hard with soft discretization using 3-categories (**Figure 5**). Compared with hard discretization, soft discretization missed edges to genes with 2 parents; only one of the parents could be found. We conclude that with our simulated data the 3-category hard discretization works best in allowing recovery of the simulated genetic pathway. All subsequent analysis below uses 3-category hard discretization.

Sampling Amount, Intervals, and Coverage: Since sampling amount, intervals, and coverage are dependent on each other, it is not possible to test their effects on BN recovery independently. however, testing their effects, can be done by multi-comparisons. First, we sampled different data amounts at the same interval, 5, which effectively changes time coverage across the simulated output (**Figure 6A**). Second, we sampled different data amounts with the same overall coverage, 10,000 time points, which effectively this changes the sampling interval (**Figure 6B**). Third, we sampled at different intervals with the same data amount, 500 data points, which effectively changes the overall coverage (**Figure 7**). In each case, two out of three variables change at the same time. Thus, when the recovery result is different between graphs of the different comparisons (**Figures 6A, 6B, and 7**), the responsible variable is one or both of the changing variables. To decided with a degree of confidence which one it is, the common differences between graphs of two different changing situations (for example the common edges in first graph of **Figure 6A** and first of **6B** compared with their differences in the second graph of **6A** and **6B**) are due to the effect of the common variable that changes in both. As expected, the more data collected, the better the recovery of the simulated system (compare within both **Figures 6A** and **6B**). Increased coverage does not appear to significantly improve recovery (compare within both **Figures 6A** and **7**). This counterintuitive result, implies that at the coverage across the simulation is already well represented in our sampling ranges. Interestingly, when changing the sampling interval, there appears to be an optimum (interval of ~5) that led to the best recovery (compare within both **Figures 7** and **6B**). At interval of 1 (taking every time point, but with a small coverage time of 500) fewer correct edges were found. Increasing the sampling interval to 5, more correct edges were found. But as the sampling interval increased more (to 10 and 20) incorrect edges appeared (**Figure 7** and **6B**), even though in one there is more coverage (**Figure 7**). The explanation for these differences in Figure 7 is that the small sampling interval yields small coverage, not giving much information between any two points spread distantly in time, while the large sampling interval, although it yields large coverage, also loses a large amount of information between any two points.

In all types of comparisons, it was difficult recover both edges of genes with multiple parents (genes 0 and 1 to 2; genes 5 and 6 to 9; and genes 8 and 9 to 10). The main variable that had the strongest effect on the recovery of edges from both parents was increasing data amount sampled (compare **Figures**

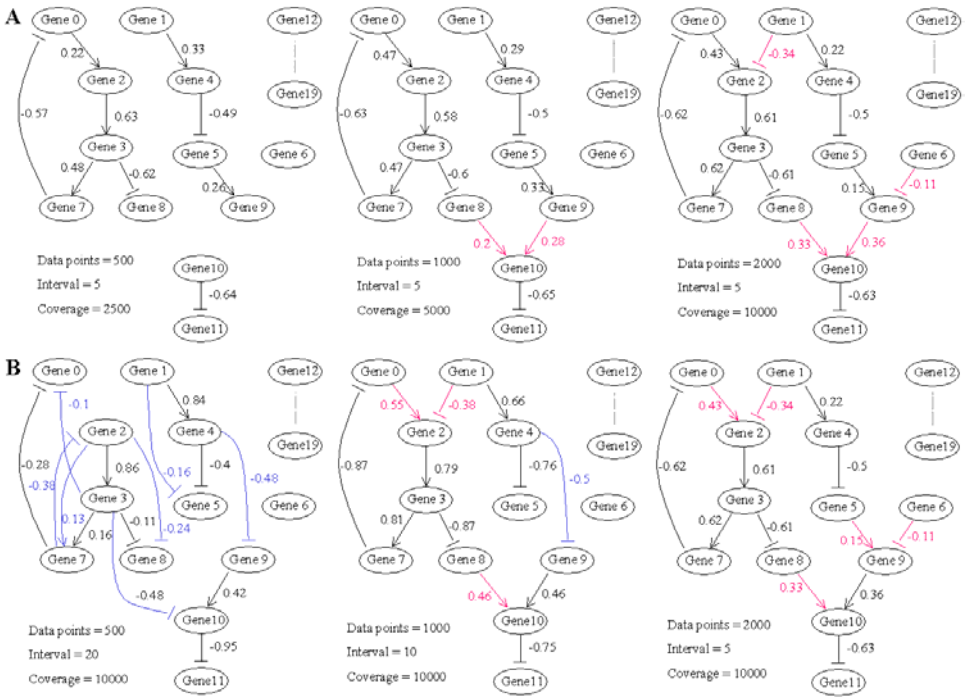

Figure 6: Comparison for changes in data amounts and coverage with same intervals (A) or changes in data amount and intervals with same coverage time (B). From left to right, the number of data points increases. Top graphs are shown. Numbers beside edges are influence scores. Red – correct edges found in only one or two graphs. Blue – incorrect edges.

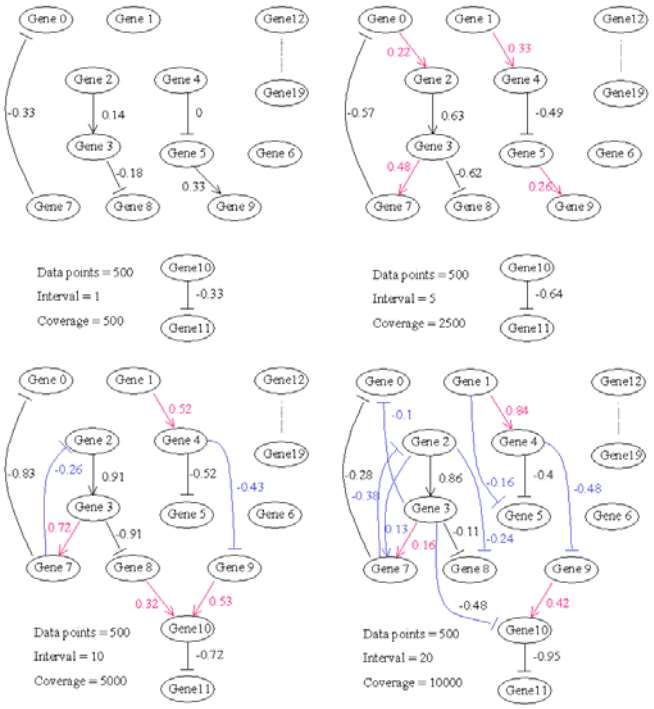

Figure 7: Comparison for sampling intervals. Labeling is the same as is described in Figure 7 legend. Numbers beside edges are influence scores.

6A, 6B, 7). The type of errors found with large sampling intervals were not completely erroneous, but were all incorrect edges from grandparents (**Figure 6B and 7**). There is also an interval effect on our influence score, such that the score increased as sampling intervals increased, until the scores was reduced by the appearance of incorrect multiple parents at very high sampling intervals (**Figures 6A** and **7**). The initial increased interval effect on the influence score makes sense because the magnitude of the difference in expression of a gene between two time points increases as the interval increases. We conclude that with our data set, sampling amount and interval has the largest effect on BN recovery. There is an optimum interval, which will depend upon the underlying timing of the gene regulatory pathway under study.

Biological Sampling and Interpolation of Data Points: In an actual biological gene expression experiment, the amount of data collected is much smaller than that used in the above tests. To mimic the maximum amount of data that could be reasonably collected in a microarray experiment, we limited the total number of data points to be from 100 cDNA microarray slides. With this pre-determined total number, we investigated the best possible way to allocate the animal samples. For example, should we sample 100 time points with one animal each time point or 25 time points with four animals each time point. For this amount of data, we found that the recovery results changed with different data sets from repeated experiments, unlike the bigger data sets above which yielded quite stable results from repeated experiments. To better understand this variation, we collected 10 different data sets for each of these tests, passed each one separately through the BN algorithms and obtained the average results (**Figure 8A**). The edges that appear frequently across the 10 data sets are of

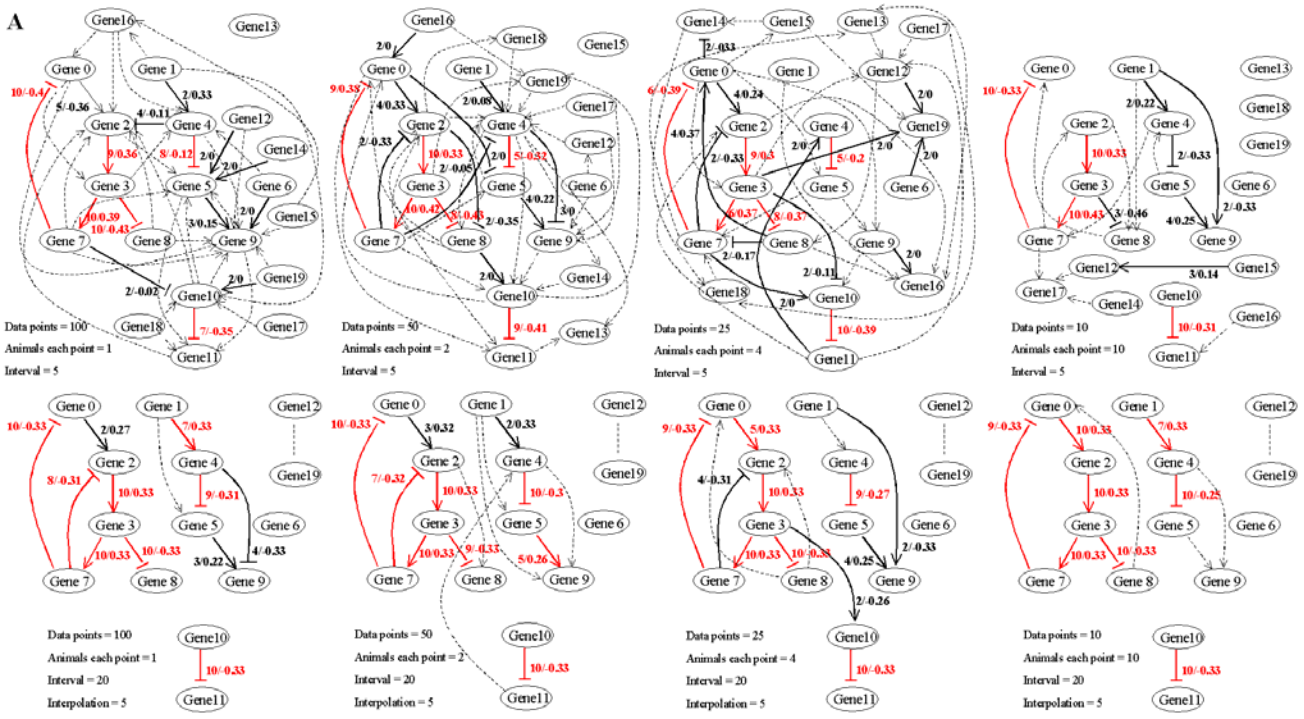

Figure 8: More biologically reasonable sampling with different strategies (A) and performance of interpolating data (B). Shown are the average top graph results from 10 datasets each. Black dashed edge – only found once in 10 recovery results. Black solid edge – find more than once but less than 5 times in 10 recovery results. Red solid edge – find more than or equal to 5 times in 10 recovery results. Numbers beside edges – occurrence (left of slash) and average of influence score (right of slash); these only appear beside the edges found more than once.

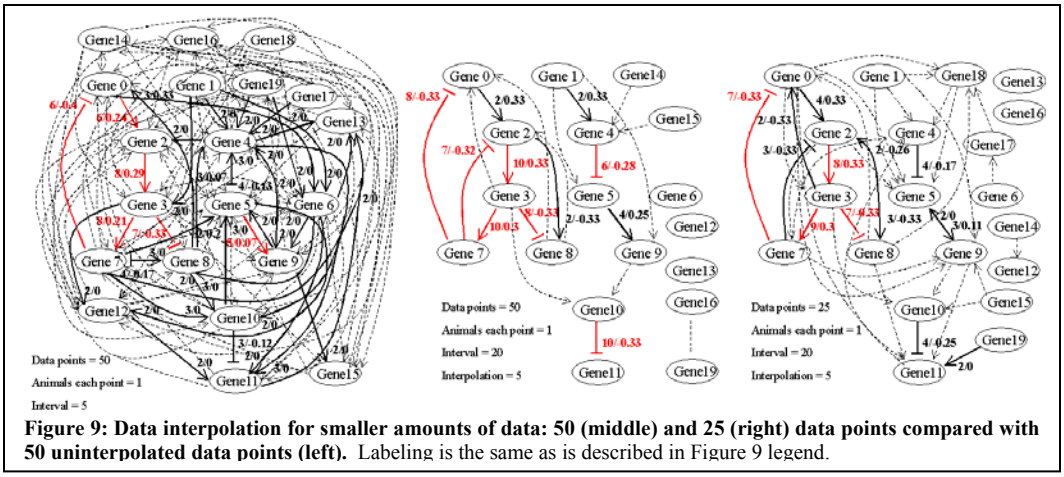

Figure 9: Data interpolation for smaller amounts of data: 50 (middle) and 25 (right) data points compared with 50 uninterpolated data points (left). Labeling is the same as is described in Figure 9 legend.

greater confidence. Here we used a sampling interval of 5 because this worked best in our previous tests.

With this amount of data, only a partial understanding of the regulatory system was recovered, and many originally independent genes were incorrectly included the resultant graph (**Figure 8A**). As the number of animals increased, there was no important change until 10 animals per time point in 10 time points. The number of edges found were dramatically decreased, but for both correct and incorrect edges. The number genes included in the graph also decreased, but in this case mostly for those that were not part of the original regulatory system. Another interesting result is that most of the incorrect edges revealed (both solid and dashed lines) had influence scores of 0 (**Figure 8A,** scores not shown for dashed lines), and this can be used to eliminate such edges.

We tested whether the recovery performance would be improved by interpolating data points. We kept the sampling method the same, except for the sampling interval. We increased the sampling interval, to 20, to have sampling time between points for interpolation. This resulted in increased coverage, but kept the number of data sampling amount the same. We linearly interpolated five data points between every two sampled data points, resulting in a six-fold increase in the amount of available data, though only 20% of it corresponds to actual sampled data. The corresponding results are shown in **Figure 8B**. Regardless of the sampling approach, we found that recovery results much improved compared with the results from the raw data points without interpolation. The genes in the connected network were correctly identified and no independent genes were implicated. Most of the correct edges were found with high occurrence and all with the correct signs of the influence score. The principal missing connections still involved nodes with multiple parents. Some of the incorrect edges were actually found from a grandparent, such as from gene 7 to 2, gene 4 to 9, and gene 3 to 10. Though technically incorrect, they can supply useful information. We conclude that when using biologically reasonable expression data sampling, BN inference algorithms appear to be able to partially recover the nodes of the underlying genetic pathway and their single parent interactions, especially when the data is interpolated. Our BN inference algorithm appears to be less good at recovering multiple parents per node within the constraints of biologically reasonable amounts of data.

To investigate this more closely, we tested the data interpolation performance with an even smaller data amount (**Figure 9**). Using 50 data points without interpolation, the recovered genetic system was a mess (**Figure 9,** left). However, when using 50 data points with interpolation (**Figure 9,** middle) the recovery was much better, even better than that from 100 data points without interpolation (**Figure 8A**). From 25 data points without interpolation, the recovery result was too messy to be shown, but with data interpolation (**Figure 9,** right), the recovery is comparable with that from 100 data points without interpolation. All the results presented in this section are computed using the BDe score function; when we tried using the BIC score function, no edges were found, again confirming that the BIC over-penalizes for over-complexity and that this excessive penalty is most apparent in the limit of small amounts of data.

Non-uniformly Sampled Data: Many biological experiments sample gene expression data in a non-uniform manner. We tested the effect of non-uniform sampling on BN recovery. Here the intervals between successive samples is a number chosen at random between 15 and 25. With a small amount of data, we found that non-uniform interval datasets can recover part of the simulated system only when data interpolation was used (**Figure 10**). The recovery results are comparable with that from uniformly sampled datasets.

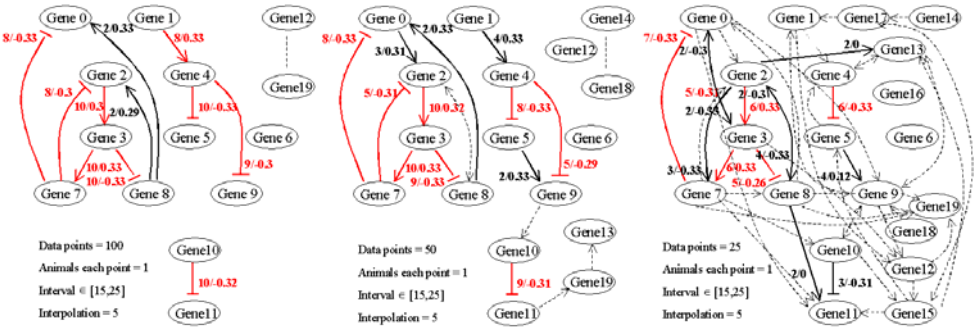

Figure 10: Recovery results from non-uniformly sampled dataset. Labeling is the same as is described in Figure 8 legend. The intervals for successive samples were chosen at random between 15 and 25.

Complexity of the Network

The pathway that we simulated has perhaps less complexity than is likely to be found in biological systems. To test some additional complexity, we used GeneSimulator to generate data for a pathway that includes one node with three parents instead of two (gene 2), as well as two independent genetic pathways (**Figure 11**, left).

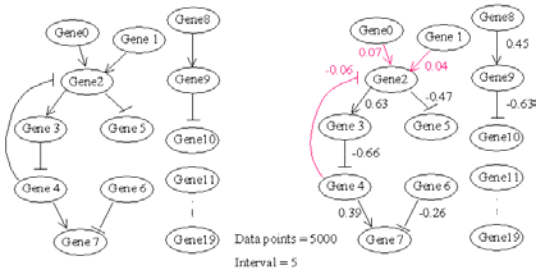

Figure 12: A more complex network. The underlying regulatory network (left) and the recovered one augmented with influence scores (right).

Using our standard conditions of small-data sampling, we were able to recover most of both independent pathways (not shown). However, to recover the three parents to one child, we found that we needed to collect at least 5000 data points (**Figure 12**, right), and the recovered edges had very low influence scores (red). We conclude that BN inference algorithms are able to recover independent genetic pathways inherent in one data set, but that as the number of parents per node grows, the amount of expression data needed to recover the entire network will dramatically grow, perhaps to an extent that the collection of this data becomes impractical.

DISCUSSION

In this study, we tested and modified the ability of BN inference algorithms to recover genetic pathways from simulated gene expression data. We found that the structure of the underlying network and the methods used for data processing and sampling had a critical impact on the ability to recover accurate pathways. This conclusion is important to consider when using our algorithm on actual gene expression data, such as that collected using high-density microarrays.

We found that the best BN inference algorithm for recovering the simulated genetic pathways was a greedy search method with random restarts, employing the BDe scoring metric and being given data discretized using a 3-catergory hard discretization. In addition, our influence scores were very useful in determining up or down regulation and the relative magnitude of regulation. We also found that there was an optimal sampling interval and efficient amount of data that allowed our BN inference algorithm to recover the most accurate pathway. We found that the BIC score does not work as well as the BDe score because the BIC score yields fewer edges or no edge with biologically reasonable sampling. This happens because BIC applies a strong penalty for more complex structure.

One of the key improvements we made to our inference algorithm for use with gene expression data is the generation of influence scores. These scores can determine positive or negative interactions and select against low-scoring incorrect interactions. This improves the accuracy of the network and increases the amount of information recovered.

Three-category discretization was optimal for the simulated dataset we examined. This may seem counterintuitive, as discretization leads to information loss from the data: the more coarse the discretization, the more the information loss. While this is true, it is not however the case that finer discretization results in better BN recovery. This is because with more discretization levels, the more CPT values are spread and also the more BN parameters must be estimated, and as a result, more data points are needed to confidently predict edges between nodes. In addition, discretization with hard boundaries outperformed discretization with soft boundaries. It is possible that because soft discretization also spreads out the distribution of the data, some of the BN relationships are not significant enough under soft discretization to warrant inclusion of an edge.

Based on the results of this study, we believe the major limitation in discovering regulatory networks from gene expression experiments is collecting a sufficient quantity of data to effectively recover all the interactions in the genetic network, especially those associated with genes with multiple parents. Data collection in the context of complex organisms is limited by the physical constraints of the experimenter and the number of animals that can be sacrificed for any one given experiment. Although we found here that our BN inference algorithm will discover genetic regulatory pathways from biologically reasonable amounts of data, the generated pathways can be misleading. Fortunately, interpolation of data from these smaller samples sizes with good coverage of the simulated gene expression pathway can at least yield a significant number of correct genes and their interactions. Moreover, many of the incorrect interactions found are from grandparents, which at least places genes only one gene removed from their true regulator. However, our BN inference algorithm had difficulty identifying more than one parent for genes with multiple parents. This is a serious problem for genetic pathway recovery, as combinatorial regulatory control is a basic property of genetic pathways. However, the solution to this problem lies in not attempting to recover complex pathways from limited amounts of expression data alone. Other types of data can be brought to bear and, when used in conjunction with expression data, can significantly enhance the ability to accurately recover regulatory network structures (Hartemink et al 2002).

Finally, we suspect that the performance of our BN inference algorithm depends in part on how the networks are simulated. Though we included noise in generating our simulated data, the data are clearly not a perfect simulation of true experimental data; some meaningful biological information is always lost in mathematical modeling. Future efforts need to be focused on using more real data to improve how well the output of the simulator matches real world data. Effort also needs to be on simulating interaction of genes products within the same cell versus across cells. As such, the recovery results produced by our BN inference algorithm are not intended to serve as a substitute for gene intervention experiments, but rather as a guide for optimally designing easier to perform correlational experiments for use with inference algorithms and to approximate the accurate genetic pathway leading to easier to test intervention experiements.

ACKNOWLEDGMENTS

We thank Kurt Grandis and Derek Scott of Duke University for assistance in the beginning stage of this project. This research is supported by a Whitehall Foundation grant to EDJ.

REFERENCES

D'haeseleer, P., Wen, X., Furham, S. and Somogyi, R., *Linear modeling of mRNA expression levels during CNS development and injury. (1999)* Proceeding of the Pacific Symposium on Biocomputing. 4: 41-52.

Friedman, N., Murphy, K. and Russell, S., *Learning the Structure form Massive Networks. (1998)* Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence. 139-147.

Friedman, N., Nachman, I. and Pe'er, D., *Learning Bayesian Network Structure of Dynamic Probabilistic Network. (1999)* Proceeding of the Fifteenth Conference on Uncertainty in Artificial Intelligence. 206-215.

Friedman, N., Linial, M., Nachman, I. and Pe'er, D., *Using Bayesian Networks to analyze expression data (2000)* Journal Computational Biology. 7: 601-620.

Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning. (1989)* Wesley, MA.

Hartemink, A. J., Gifford, D., Jaakkola, T. and Young, R., *Using Graphical Models and Genomic Expression Data to Statistically Validate Modeals of Genetic Regulatory Networks. (2001)* Pacific Symposium on Biocomputing.

Hartemink, A. J., Gifford, D., Jakkoola, T. and Young, R., *Combining location and expression data for principled discovery of genetic regulatory network models. (2002)* Pacific Symposium on Biocomputing.

Heckerman, D., *A Tutorial on Learning with Bayesian Networks. (1996)* Technical Report MSR-TR-95-06, Microsoft Research, March, 1995 (revised November, 1996).

Jarvis, E. D. and Nottebohm, F., *Motor-driven gene expression. (1997)* Proceedings of National Academy of Sciences of the United States of America.

Jarvis, E. D. et al, *Integrate Songbird Brain.(2002)* Journal of Comparative Physiology, A (in press).

Murphy, K. and Mian, S., *Modeling Gene Expression Data Using Dynamic Bayesian Networks. (1999)* Technical Report, University of California, Berkeley.

Smith, V. A., Jarvis, E. D. and Hartemink, A. J., *Evaluating Functional Network Inference Using Simulations of Complex Biological Systems. (2002)* Accepted by The 10th international conference on Intelligent Systems for Molecular Biology.

Smith, V. A., Jarvis, E. D. and Hartemink, A. J., *Influence Of Network Topology and Data Collection On Functional Network Influence. (2002)* Pacific Symposium on Biocomputing (in press).

Somogyi, R. and Sniegoski, C. A., *Modeling the complexity of genetic networks: understanding multigenic and pleiotropic regulation. (1996)* Complexity. 1:45-63.

Weaver, D. C., Workman, C. T. and Stormo, G. D., *Modeling regulatory networks with weight matrics. (1999)* Proceedings of the Pacific Symposium on Biocomputing. 4: 112-123.