FPGA-ACCELERATED BAYESIAN LEARNING FOR RECONSTRUCTION OF GENE REGULATORY NETWORKS

Iosifina Pournara

Department of Crystallography
Birkbeck College, University of London
Malet Street, London WC1E 7HX, U.K.
email: i.pournara@cryst.bbk.ac.uk

Christos-S. Bouganis, George A. Constantinides

Department of Electrical
& Electronic Engineering
Imperial College London, University of London
Exhibition Road, London SW7 2AZ, U.K.
email: christos-savvas.bouganis@imperial.ac.uk

ABSTRACT

Rapid advances in biological technologies, such as DNA microarrays, have enabled biologists to measure the expression levels of thousand of genes simultaneously under different conditions. This leads to a growing need to find methods that extract valuable information, fast and reliably, from this large amount of data. Recently, the advantages of using Bayesian networks for the reconstruction of gene regulatory networks from microarray data have been shown. However, these methods are very computationally intensive. Here, we explore the inherent parallelism of Bayesian learning and propose a hardware design that can be used for the reconstruction of such networks. The evaluation of the proposed design in a VirtexII demonstrates a speed up of the algorithm by 76 times over a software implementation in a Pentium 4.

1. INTRODUCTION

The expression level of each gene in a multicellular organism is regulated by external stimuli and by the products of other genes, creating a complex *gene regulatory network*. Advances in biological technologies, such as DNA microarrays, enable us to measure the expression levels of thousands of genes simultaneously under different conditions. These measurements can help to unravel the underlying gene regulatory network and to understand complicated biological processes, with application to disease diagnosis.

Recently, various variants of probabilistic graphical models have been used for extracting information of gene regulation from microarray data [1, 2, 3, 4]. Bayesian networks, which are directed acyclic probabilistic graphs, are most widely used for this purpose. They allow the incorporation of prior knowledge about the system and are able to learn a model from a given set of data. Moreover, they can be further used to infer causal relationships between genes and to predict the outcome of specific experiments.

Most research in this area has focused on discrete Bayesian networks [1, 5]. However, all known approaches to optimal Bayesian learning have a complexity that increases exponentially with the number of variables [6]. Moreover, current software heuristics cannot take advantage of the parallel structure of the underlying learning algorithm and are limited to networks with few variables.

In this paper, we introduce the use of FPGAs to tackle the problem of Bayesian learning. The fine grain parallelism provided by the FPGAs in conjunction with the parallel structure of the Bayesian learning algorithm leads to designs that are 76 times faster than the software equivalents, even before pipelining the design. According to the authors' knowledge, this is the first attempt to use FPGAs, or indeed custom hardware, to tackle the computationally intensive problem of Bayesian learning. This paper introduces a design that takes advantage of the reconfigurability of FPGAs and is easily parameterizable to networks with different number of nodes. Related work addressing the problem of linear regression on the gene expression microarray data using FPGAs can be found in [7].

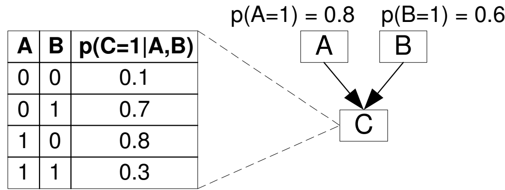
This paper is organized as follows. Section 2 gives a brief introduction to Bayesian learning. Section 3 describes in detail the proposed hardware design for Bayesian learning. Section 4 evaluates the reduction in computational time that is achieved by the proposed design compared to a software implementation, and discusses the area requirements of the hardware design. Finally, the paper concludes in Section 5.

2. BAYESIAN LEARNING

Let us assume that we have a set of discrete random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ where each variable X_i can take on values from a finite set. A Bayesian network encodes the joint probability distribution over the variables $\mathbf{X}$. It is defined by a pair $B =< G, \mathbf{\Theta} >$. G is a directed acyclic graph in which the nodes represent variables and the edges repre-

0-7803-9362-7/05/$20.00 ©2005 IEEE

Authorized licensed use limited to: IEEE Xplore. Downloaded on February 17, 2009 at 11:17 from IEEE Xplore. Restrictions apply.



A	B	p(C=1|A,B)
0	0	0.1
0	1	0.7
1	0	0.8
1	1	0.3

Fig. 1. An example of a Bayesian network with binary variables. The conditional probabilities associated with each variable are also indicated.

sent direct dependencies between the variables. For example, Figure 1 shows a Bayesian network consisting of three variables where nodes A and B are the parents of node C, and C is called the child of A and B. $\mathbf{\Theta}$ represents the set of parameters that quantify the network G. The network G encodes the dependence of C on both A and B.

Bayesian learning involves two steps. The first step is to define a score metric for the fitness of a graph G on the dataset D. The posterior probability of a graph G, $p(G|D)$, given the prior probability of the graph $p(G)$ and the observed data $D = \{C_1, \ldots, C_M\}$ is given by Bayes' rule (1), where the marginal likelihood of the data is given by (2).

$$p(G|D) \propto p(G)p(D|G). \quad (1)$$

$$p(D|G) = \int p(D|\mathbf{\Theta}, G)\, p(\mathbf{\Theta}|G)\, d\mathbf{\Theta}. \quad (2)$$

A Bayesian network encodes the Markov condition which allows the decomposability of the score. This states that a variable X_i is independent of its non-descendents given its parents Π_i. Thus the joint probability can be written more simply in terms of conditional probabilities (3).

$$p(G|D) \propto p(G) \prod_{i=1}^{n} p(X_i^1, \ldots, X_i^M|\Pi_i, G) \quad (3)$$

Figure 1 shows an example of a discrete network and the conditional probabilities associated with each variable. This decomposability property implies that the score of the nodes can be calculated in parallel. A popular scoring metric that has been widely used for discrete networks is the *Bayesian Dirichlet metric* (BD metric) [5]. The main assumption is that the probability distribution of X_i with possible states r_i is a collection of multinomial distributions, one distribution for each possible configuration $\{1, \ldots, q_i\}$ of its parents Π_i. The BD metric for scoring a discrete network is given by (4), where $N'_{ij} = \sum_{k=1}^{r_i} N'_{ijk}$ and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. N'_{ijk} is a positive hyperparameter that denotes the prior observation counts where $X_i = k$ and $\Pi_i = j$, N_{ijk} is the number of cases in D where $X_i = k$ and $\Pi_i = j$, and Γ is the gamma function. For the derivation of this metric, the reader is referred to [5].

$$p(G|D) \propto p(G) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})} \quad (4)$$

The second step of Bayesian learning is to find a network or a set of networks that best fit the data. Each pair of nodes $\{A, B\}$ can have one of three different causal relationships: $A \rightarrow B$, $A \leftarrow B$, or $A \perp B$, the latter indicating the absence of a direct causal relationship. The number of possible directed acyclic networks for a set of variables grows super-exponentially with the number of nodes, making exhaustive searches computationally infeasible [6]. In this paper, we use a simple search heuristic approach, greedy hill climbing, which adds, deletes or reverses edges that result in a maximum increase in the score [5]. Currently we avoid local maxima by starting the hill climbing process with several start-points, although other heuristic searches such as simulated annealing [8] could also be used.

3. HARDWARE DESIGN

The proposed approach to the problem is to design hardware that iteratively considers a change between two nodes *i.e.* addition, deletion or reversal of an edge, checks whether the new structure is acyclic, and if it is, calculates the network score using (4). A comparison between the obtained score and the current one would indicate whether the new network fits the given dataset better and thus should be kept. The whole process is repeated until no further changes lead to a network with better score. The algorithm avoids local maxima by using different starting points.

The nature of the algorithm implies that at each iteration the algorithm has to go through the dataset in order to calculate the score of each node given its parent set. In order to avoid such repeated searches through the dataset, the algorithm is split into two main parts. The first part involves the calculation of the scores for each node for all the possible sets of parents, while the second part focuses on the updating of the network. This approach takes advantage of the reconfigurability of FPGAs by first calculating the scores for each node, and then reconfiguring the FPGA to perform the optimization for the optimum network. The reconfiguration time of the FPGA is not considered since it is insignificant compared to the time that is needed by the rest of the algorithm.

Note that, in this first attempt of using FPGAs for Bayesian learning, we are considering networks of up to 10 nodes, while gene regulatory networks usually involve a large number of genes. However, the reconstruction process of such networks using software implementations also considers small subnetworks of approximately the same size [4]. Moreover, software implementations usually restrict the number of par-

Authorized licensed use limited to: IEEE Xplore. Downloaded on February 17, 2009 at 11:17 from IEEE Xplore. Restrictions apply.

ents that a node could have, *e.g.* to two, in an attempt to minimize the computationally expensive step of searching the network space. The approach presented in this paper has achieved a considerable improvement in the time requirements of the algorithm, without restricting the number of possible parents of a node. Finally, it should be noted that given the current methods reconstruction of large networks is ill-conditioned due to the limited number of existing microarray data [9, 10].

3.1. Hardware for calculating the node scores

The score of a node depends on the set of its parents. This first part of the design calculates the different scores that each node can have given all different possible sets of parents. A top-level diagram of the design for the case of n nodes is illustrated in Figure 2(a). Since the scores of one node can be calculated independently from the scores of another node, the whole design is parallelized by node.

In more detail, the task of the node score calculation module is to calculate the inner product of (4) for each node separately. Due to the fact that multiplication is slow and requires a considerable amount of area compared to addition, the proposed method uses a logarithmic number scheme. Moreover, since the node variables are considered to have two states (in our cases corresponding to over-expression and under-expression of a gene), the inner product of (4) regarding node X_i may be simplified to (5).

$$\log_{10} p(X_i^{1:M}|\Pi_i) = \sum_{j=1}^{q_i} \log_{10}(N_{ij0}!) + \log_{10}(N_{ij1}!) - \log_{10}((1 + N_{ij0} + N_{ij1})!) \quad (5)$$

An outline of the algorithm for the calculation of the scores for a node X_i is illustrated in Figure 3. The main stages of the algorithm are described in more detail below.

3.1.1. Iterating through the parent sets of a node.

This process is responsible to iterate through all the possible combinations of parents that a node can have. This can be easily implemented through an $n - 1$ bit counter for a network with n nodes. Each bit $Parent(k)$ in the counter indicates the presence or absence of the corresponding node X_k from the parent set.

3.1.2. Iterating through the states of a parent set.

The possible parent-set states depend on which nodes are parents. A modified counter is required to obtain these combinations as shown in Figure 2(b). For example, the counter output for the parent set 1001 is the sequence 0000, 0001, 1000, 1001. This was implemented using a counter that counts through selected bits specified by the set of the parents. The hardware implementation is based on a series of one-bit adders where carry bits can be skipped by multiplexors controlled by the parent set register as shown in Figure 2(b).

3.1.3. Calculating the number of cases N_{ij0} and N_{ij1}.

The observational data are stored in a ROM. Each entry of the ROM corresponds to one observational case C_m from the dataset D. In order to produce N_{ij0} and N_{ij1}, the design iterates through the observations and determines the number of matches.

for each possible parent set Π_i of a node X_i
  set the score of the node for this parent set to zero, $S_{\Pi_i} = 0$
  for each possible state j of the parents, $j = 1, \ldots, q_i$
    calculate the number of occurrences of that parent state j in the dataset when the node under question is in state 0 (N_{ij0}) and in state 1 (N_{ij1})
    calculate the value of $1 + N_{ij0} + N_{ij1}$
    calculate $s = \log_{10}(N_{ij0}!) + \log_{10}(N_{ij1}!) - \log_{10}((1 + N_{ij0} + N_{ij1})!)$
    update $S_{\Pi_i} = S_{\Pi_i} + s$
  end
end

Fig. 3. Pseudo-code for calculating the scores of a node.

3.1.4. Calculating the score of a node for a particular set and state of parents.

The calculation of (5) for a particular state of a given set of parents involves the conversion of the variables to the logarithmic equivalents and the use of two adders to calculate the final score. The required precision at the output of the system was determined by measuring the smallest difference between two scores of a node for two different parent sets resulting in a precision requirement of 16 fractional bits.

In order to produce a compact and fast design, the log-Gamma function was tabulated. A ROM lookup table is used, that takes as input a value x and produces the result $\log_{10}(x!)$. The storage of $\log_{10}(x!)$ rather than calculating or storing $x!$ overcomes the dynamic range problems that would otherwise occur with a fixed-point implementation of (5).

In general, the number of bits used for the precision varies with the number of variables. For each extra node, the possible states of the parents are doubled, which corresponds to one extra bit in representing the final result. In summary, the number of rows in ROM lookup should be $M+1$, and the width of the ROM should be $\lceil \log_2(\log_{10}((M+$

Authorized licensed use limited to: IEEE Xplore. Downloaded on February 17, 2009 at 11:17 from IEEE Xplore. Restrictions apply.

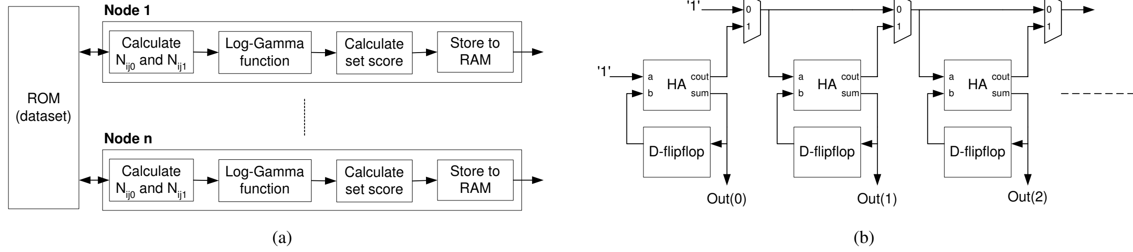


Fig. 2. (a) Top level score calculation. (b) Parent state generator (HA represents a half-adder).

$1)!))\rceil + n + 11$, where M denotes the number of observations in the dataset and n denotes the number of nodes.

3.2. Hardware for updating the network structure

This module is designed to use the previously calculated scores for each node in order to search for an acyclic network structure with the maximum score. The rest of the description focuses on the core module of the hardware, which takes as input two nodes, finds the best causal relationship between them, and finally checks for acyclicity in the resulting network. If the resulting network is not acyclic, the update is rejected, and the previous network is maintained.

A top-level diagram of the hardware design for the core module is illustrated in Figure 4. The design consists of three main modules connected via a network-on-chip structure: the controller module, which is a state machine responsible to generate the control signals for the other modules, a wrapper module that contains the hardware modules for each node in the network, and the node interface module that groups signals from the wrapper module and communicates with the controller. The main advantage of the proposed design is that it can easily accommodate networks with more nodes by adding extra node modules in the wrapper module. Each node module consists of two main parts. A part that calculates the score of the node given a parent set, and a part that checks for acyclicity. Below, the three main functions of the core module are described in more detail.

3.2.1. *Identifying the best causal relationship between two nodes.*

One of the most important functions of the design is to identify the causal relationship ($A \rightarrow B$, $A \leftarrow B$, $A \perp B$) between two nodes, A and B, that leads to the greatest improvement of the score. For each node, two scores are required, a score where the other node is included in its parent set and one where it is not. A further check is performed to assess whether the other node was already in the parent set, which determines whether the resulting network needs to be checked for acyclicity.

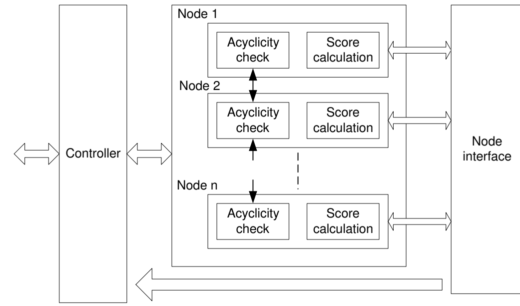


Fig. 4. Top level design for updating the network structure.

Table 1. Update nodes' connection

$A_B < A_{\bar{B}}$	$B_A < B_{\bar{A}}$	$B_A + A_{\bar{B}} < B_{\bar{A}} + A_B$	$AddBtoA$	$AddAtoB$
0	0	0	0	1
0	0	1	1	0
0	1	1 (must be)	1	0
1	0	0 (must be)	0	1
1	1	0	0	0
1	1	1	0	0

The determination of the optimal connection is decided according to Table 1. The two nodes are denoted by A and B. A_B represents the score of node A when node B is in its parent set, and $A_{\bar{B}}$, when it is not. The same convention holds for node B. $AddBtoA$ denotes whether node B should be placed in node A's parent set or not, and similar for $AddAtoB$. The table indicates the type of the update that has to be performed given all the possible connections between the two nodes. In the case where $AddAtoB$ and $AddBtoA$ signals are both low, no connection between the two nodes is inserted and a check for acyclicity is not performed. If either of the signals is high, an edge between the nodes is added accordingly. This triggers a check for acyclicity only if the added connection introduces a new parent in the parent set of one of the nodes.

Authorized licensed use limited to: IEEE Xplore. Downloaded on February 17, 2009 at 11:17 from IEEE Xplore. Restrictions apply.

3.2.2. Acyclicity check.

Given that the starting network is acyclic, deleting an edge retains an acyclic network. A cycle can only be generated by adding an edge or reversing an existing edge. Thus, the acyclicity mode is activated only if either *AddAtoB* or *AddBtoA* signal is high. If the acyclicity mode is triggered, signals from a node are sent out to its parents over the network-on-chip. Then, each parent of that node sends signals out to its own parents in parallel and so on. If a signal reaches back to its originating node, it means that there is a cycle in the graph, and the acyclicity check has failed.

As the signals for acyclicity check are received by each node, they are registered and transmitted in the next clock cycle. This reduces the propagation delay and increases the clock speed. In the worst case, $n-1$ clock cycles are needed for the signals to propagate to all nodes for a network with n nodes.

3.2.3. Control module.

The control module is responsible to generate the appropriate signals to trigger the necessary events. Its control flow is illustrated in Figure 5. It can be seen that the number of clock cycles required for one update can be 5 or $n+5$ for a network with n nodes, depending on whether the system has to perform an acyclicity check, as discussed above.

Cycle	Event
1	*Request* scores A_B and B_A
2	*Receive* scores A_B and B_A and
	Request scores $A_{\bar{B}}$ and $B_{\bar{A}}$
3	*Receive* scores $A_{\bar{B}}$ and $B_{\bar{A}}$ and *Calculate*
	$A_B < A_{\bar{B}}$, $B_A < B_{\bar{A}}$, $A_B + B_{\bar{A}}$ and $A_{\bar{B}} + B_A$
4	*Calculate* $B_A + A_{\bar{B}} < A_B + B_{\bar{A}}$
	if an edge is added or reversed
5	*start* acyclicity check
6 to n+4	*check* for acyclicity
n+5	*update* the connection between the two nodes A and B
	else
5	*update* the connection between the two nodes A and B
	end

Fig. 5. Pseudo-code for updating the network structure. The cycle in which each event takes place is also indicated.

The extra modules that have been omitted in the above description because of their low complexity, but are necessary for the design, include a linear feedback shift register circuit that generates random node identifiers that are given as input to the core module, and a module that is responsible to upload the core module with the initial acyclic network.

4. PERFORMANCE EVALUATION

The design was evaluated regarding the achieved speed and the required area. The circuit was designed in RTL VHDL and placed on a Xilinx Virtex II FPGA, XC2V6000, in package type ff1152 with speed grade 6, using the Xilinx tools and Synplicity's Synplify Pro 7.2.

4.1. Speed

The speed of the hardware is one of the main objectives of this design since the purpose of the hardware is to considerably reduce the execution time for the search of the optimum Bayesian network compared to the designed software models. Two designs are synthesized and tested. The first design can handle networks with 5 nodes, where the second one handles networks with 10 nodes. The speed of the hardware is compared against software running on a Pentium 4 at 2.4GHz.

4.1.1. Calculating the scores of each node's parent set.

For the design that calculates the node scores for all possible parent sets, a maximum clock rate of 124MHz for the case of 5 nodes and 58MHz for the case of 10 nodes is obtained. For a network with n nodes, each node can have $\sum_{k=0}^{n-1} \frac{(n-1)!}{(n-1-k)!k!} = 2^{n-1}$ parent sets, where k denotes the number of parents. A single set with k parents has 2^k possible states. Thus, the number of states of all possible parent sets of a node is $\sum_{k=0}^{n-1} \frac{(n-1)!}{(n-1-k)!k!} 2^k = 3^{n-1}$. For the case of 100 observation data, the total number of clock cycles required to calculate all parent scores for a network with $n = 5$ nodes is $3^4 \times 100$, which at 124MHz takes $65\mu s$. In the case of $n = 10$ nodes, at 58MHz, takes 34ms. It should be noted that the scores for all the possible parent sets are calculated in parallel for all the nodes. The execution times of the software equivalent design are 12.7ms for the case of 5 nodes and 6.9 seconds for the case of 10 nodes. This implies a speed up factor of 195 for the case of 5 nodes and 100 for the case of 10 nodes.

4.1.2. Updating the network.

For this part of the design, a maximum clock rate of 162MHz for the case of 5 nodes and 72MHz for the case of 10 nodes has been achieved. The reason for the reduction in clock frequency with increasing number of nodes is mainly due to the larger data buses that are required, leading to large multiplexors used in the node interface module. In the worst case scenario, where at every update the circuit has to check for acyclicity, the number of updates per second that can

Authorized licensed use limited to: IEEE Xplore. Downloaded on February 17, 2009 at 11:17 from IEEE Xplore. Restrictions apply.

be performed by the design is $16.20 Mupdates/sec$ for the case of 5 nodes and $4.8 Mupdates/sec$ for the case of 10 nodes. The corresponding numbers for the software design are $0.286 Mupdates/sec$ and $0.0625 Mupdates/sec$. The speed up factor achieved by the hardware for the case of 5 nodes is approximately 56, where for the case of 10 nodes is around 76. Note that, while the clock frequency reduces with the number of nodes, the overall speedup increases. This is because the increase in fine grain parallelism is greater than the slowdown in the network-on-chip.

4.2. Area

The resource utilization reported from the synthesis tools when a XC2V6000 FPGA is used is summarized below. The module calculating the node scores consumes 5% of the chip area for the 5 nodes network and 35% for the 10 nodes network. The update the network design occupies 1% of the chip for the 5 nodes network and 14% for the 10 node network. For the case of calculation of the node scores, it can be seen that the network with 10 nodes uses 7 times the resources of the 5 nodes. For the part of the design network updates, the design for the case of 10 nodes is 14 times larger than the design for 5 nodes, indicating a limit to the scalability of the design. This limit results from the partition of the problem into score calculation and network update circuitry, although it may be possible to overcome this limitation through the use of implicit lookup circuitry replacing the RAM blocks.

5. CONCLUSION

This paper presents a hardware approach for the computationally costly problem of Bayesian learning. The proposed design takes advantage of the inherent parallelism that exists in the current Bayesian learning algorithms and can not be exploited by the software approaches. By dividing the design into two main parts, the module for the calculation of the node scores given the parent sets and the module responsible for searching the network space, leads to a fast design easily adaptable to networks with different number of nodes. A comparison against a software approach demonstrated a speed up factor of 56 and 76 for the two modules respectively. Future work will involve the investigation of implicit storage approaches and custom content addressable memories, to aid the mapping of networks with a large number of nodes.

Acknowledgements

The authors would like to thank Andrew James Rothwell and Eranda Jayasinghe for their implementation work on this project. The authors acknowledge the financial support of the EPSRC (EP/C549481/1), The Royal Society (2004/R1), the BBSRC, and the UK Research Council (Basic Technology Research Programme "Reverse Engineering Human Visual Processes" GR/R87642/02).

6. REFERENCES

[1] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using Bayesian networks to analyze expression data," *Journal of Computational Biology*, vol. 7, pp. 601–620, 2000.

[2] A. Hartemink, S. Gifford, J. Jaakkola, and R. Young, "Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks," in *Proceedings of Pacific Symposium on Biocomputing*, 2001.

[3] E. Segal, Y. Barash, I. Simon, N. Friedman, and D. Koller, "From promoter sequence to expression: a probabilistic framework," *In Proceedings of the Sixth Conference on Research in Computational Molecular Biology (RECOMB)*, vol. 6, pp. 263–272, 2002.

[4] I. Pournara and L. Wernisch, "Reconstruction of gene networks using Bayesian learning and manipulation experiments," *Bioinformatics*, vol. 20, pp. 2934–2942, 2004.

[5] D. Heckerman, D. Geiger, and D. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine Learning*, vol. 20, pp. 197–243, 1995.

[6] D. Chickering, "Learning Bayesian networks is NP-complete," in *Lecture Notes in Statistics*, 1995.

[7] T. Van Court, M. C. Herbordt, and R. J. Barton, "Case Study of a Functional Genomics Application for an FPGA-Based Coprocessor," in *Proc. Field Programmable Logic and Applications*, 2003.

[8] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, 1953.

[9] I. Pournara, "Reconstructing gene networks by passive and active Bayesian learning," Ph.D. dissertation, Birkbeck College, University of London, 2004.

[10] D. Husmeier, "Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks," *Bioinformatics*, vol. 19, pp. 2271–2282, 2003.

Authorized licensed use limited to: IEEE Xplore. Downloaded on February 17, 2009 at 11:17 from IEEE Xplore. Restrictions apply.