Applied Intelligence, 11, 31–44 (1999)

Massively Parallel Probabilistic Reasoning with Boltzmann Machines

PETRI MYLLYMÄKI
Complex Systems Computation Group (CoSCo), Department of Computer Science, P.O. Box 26, FIN-00014, University of Helsinki, Finland
Petri.Myllymaki@cs.Helsinki.FI, http://www.cs.Helsinki.FI/∼myllymak/

Abstract. We present a method for mapping a given Bayesian network to a Boltzmann machine architecture, in the sense that the the updating process of the resulting Boltzmann machine model provably converges to a state which can be mapped back to a maximum a posteriori (MAP) probability state in the probability distribution represented by the Bayesian network. The Boltzmann machine model can be implemented efficiently on massively parallel hardware, since the resulting structure can be divided into two separate clusters where all the nodes in one cluster can be updated simultaneously. This means that the proposed mapping can be used for providing Bayesian network models with a massively parallel probabilistic reasoning module, capable of finding the MAP states in a computationally efficient manner. From the neural network point of view, the mapping from a Bayesian network to a Boltzmann machine can be seen as a method for automatically determining the structure and the connection weights of a Boltzmann machine by incorporating high-level, probabilistic information directly into the neural network architecture, without recourse to a time-consuming and unreliable learning process.

Keywords: Boltzmann machines, probabilistic reasoning, Bayesian networks, simulated annealing

1. Introduction

Neural networks are massively parallel computational models consisting of a large number of very simple processing units (for a survey of neural models, see e.g. the classic collections [1–4]. These models can perform certain computational tasks extremely fast when run on customized parallel hardware, and hence they have been suggested as a computationally efficient tool for solving NP-hard optimization problems approximatively [5,6]. Especially suitable for this type of tasks are stochastic neural network architectures, as these models are based on a stochastic updating process which converges to a state maximizing an objective function determined by the variable states of the binary nodes of the network, and by the (constant) parameters ("weights") attached to the connecting arcs. A well-known example of a stochastic neural network architecture is the *Boltzmann machine (BM)* model [7,8].

Boltzmann machines can be used as a computationally efficient tool for finding maximum points of the objective function corresponding to a given network structure, provided that suitable massively parallel hardware is available. In order to apply these models for solving combinatorial optimization problems, we need also an efficient, theoretically justifiable method for constructing a neural network structure where the maximum point of the objective function corresponds to the solution to the optimization problem to be solved. However, Boltzmann machines, and neural network models in general, are typically constructed by inefficient, partly quite ad-hoc methods. In particular, the neural network architecture is usually selected in a more or less arbitrary manner by fixing the number of nodes and the connections between the nodes manually. The weights are then determined by a learning algorithm which first assigns (randomly chosen) initial weights to the connections, and uses then some gradient-based greedy algorithm for changing the weights until the behavior of the network seems to be consistent with a sample of training data [8–10].

There are, however, several serious pitfalls with this approach, which relate to the ”black box” nature of the functioning of the resulting models: normally there is no way of finding out what kind of knowledge a neural network contains after the learning, nor is it possible to explain the behavior of the model. In particular, as the learning algorithms start with a randomly chosen initial state, “tabula rasa”, they are unable to use any prior knowledge of the problem environment, although in many cases this kind of information would be readily available. This results in a very slow and unreliable learning process. Besides, as most of the learning algorithms are “steepest-descent” type greedy algorithms, they are very likely to get stuck in local maximum points.

It follows that although Boltzmann machines provide us with an efficient computational mechanism for finding maximum points of the objective function, finding a Boltzmann machine structure with a suitable objective function can be very difficult in practice. In this paper we present one possible solution to this problem, focusing on a combinatorial optimization problem of great practical importance, the problem of finding the *maximum a priori (MAP)* probability value assignment of a probability distribution on a set of discrete variables. More precisely, we show how a Bayesian (belief) network model [11–13], a graphical high-level representation of a probability distribution over a set of discrete variables, can be mapped to a Boltzmann machine architecture so that the global maximum of the Boltzmann machine objective function (the *consensus* function) has the same maximum points as the probability distribution represented by the Bayesian network model.

Intuitively speaking, a Bayesian network model of a probability distribution is constructed by explicitly determining all the direct (causal) dependencies between the random variables of the problem domain: each node in a Bayesian network represents one of the random variables of the problem domain, and the arcs between the nodes represent the direct dependencies between the corresponding variables. In addition, each node has to be provided with a table of conditional probabilities, where the variable in question is conditioned by its predecessors in the network.

The importance of Bayesian network representations lies in the way such a structure can be used as a compact representation for many naturally occurring distributions, where the dependencies between variables arise from a relatively sparse network of connections, resulting in relatively small conditional probability tables. In these cases, a Bayesian network representation of the problem domain probability distribution can be constructed efficiently and reliably, assuming that appropriate high-level expert domain knowledge is available. There exists also several interesting approaches for constructing Bayesian networks from sample data, and moreover, theoretically solid techniques for combining domain expert knowledge with the machine learning approach (see, e.g., [14]).

The Bayesian network theory offers a framework for constructing algorithms for different probabilistic reasoning tasks (for a survey of probabilistic reasoning algorithms, see [15]). In this paper, we focus on the problem of finding the MAP state in the domain modeled by a given Bayesian network. Unfortunately, for a general network structure, the MAP problem can be shown to be NP-hard [16, 17], which means that very probably it is not possible for any algorithm to solve this task (in the worst case) in polynomial time with respect to the size of the network. Consequently, recently there has been a growing interest in developing stochastic algorithms for solving the MAP problem, where instead of aiming at an accurate, deterministic solution, the goal is to find a good approximation for a given problem with high probability.

A prominent example of such stochastic methods is the *simulated annealing* algorithm [18–21]. This global optimization method can be used as the computational procedure for performing probabilistic reasoning on a Bayesian network, but the method can be excruciatingly slow in practice. The efficiency of simulated annealing depends crucially on a set of parameters which control the behavior of the algorithm. Elaborate techniques for optimizing these parameters can be found in [22–29]. An alternative approach for speeding up the simulated annealing method is to develop parallel variants of the algorithm. Techniques for implementing parallel forms of simulated annealing on conventional hardware can be found in [30, 21, 31]. On the other hand, as suggested in [21], since the Boltzmann machine updating process produces a stochastic process similar to simulated annealing, this neural network model offers an interesting possibility for an implementation on a massively parallel architecture.

In this paper we argue that we can achieve “the best of both worlds” by building a hybrid Bayesian-neural system, where the model construction module uses Bayesian network techniques, while the probabilistic reasoning module is implemented as a massively parallel Boltzmann machine. For constructing such a

hybrid Bayesian-neural system, we need a way to map a given Bayesian network to a Boltzmann machine architecture, so that the consensus function of the resulting Boltzmann machine has the same maximum points as the probability measure corresponding to the original Bayesian network. Although in some restricted domains this kind of a transformation is fairly straightforward to construct [7,32–35], the methods presented do not apply to general Bayesian network structures (see the discussion in [36]). In [37, 38] we presented a mapping from a binary-variable Bayesian network to a harmony network [39], which can be regarded as a special case of the Boltzmann machine architecture. Following the work presented in [40, 41, 36], in this paper we extend this framework to cases where the Bayesian network variables can have an arbitrary number of values, and the neural network module uses the standard Boltzmann machine structure.

Compared to other neural-symbolic hybrid systems (see e.g. [42–45]), the Bayesian-neural hybrid system suggested here has two clear advantages. First of all, the mathematical model behind the system is the theoretically sound framework of Bayesian reasoning, compared to the more or less heuristic models of most other hybrid systems (for our earlier, heuristic attempts towards a hybrid system, see [46–48]). Secondly, although some hybrid models provide theoretical justifications for the computations (see e.g. Shastri's system for evidential reasoning [49]), they may require fairly complicated and heterogeneous computing elements and control regimes, whereas the neural network model behind our Bayesian-neural system is structurally very simple and uniform, and confirms to an already existing family of neural architectures, the Boltzmann machines. In addition, the mapping presented here creates a two-way bridge between the symbolic and neural representations, which can be used to create a "real" modular hybrid system where two (or more) separate (neural or symbolic) inference modules work together.

The mapping described in this paper produces a Boltzmann machine updating process which corresponds in a sense to a simulated annealing process where all the Bayesian network variables are updated simultaneously. Consequently, with suitable massively parallel hardware, processing is quite efficient and becomes independent of the size or the structure of the Bayesian network. On the other hand, the BM updating process works in a state space much larger than the state space of the original Bayesian network, and in terms of accuracy in sampling the probability distribution, the BM process is only an approximation of a simulated annealing process on the Bayesian network. It is therefore an interesting question whether the speedup gained from parallelization compensates for the loss of accuracy in the stochastic process. In the empirical part of the paper, we study this question by using artificially generated test cases.

The paper is organized as follows. First in Section 2 we give the formal definition of Bayesian network models. The probabilistic reasoning task studied in this paper, the MAP problem, is discussed in Section 3. In this section we also describe how simulated annealing can be used for solving this problem. The basic concepts of Boltzmann machine models are reviewed briefly in Section 4, and in Section 5 we describe the suggested mapping from a Bayesian network to a Boltzmann machine. In Section 6 we show results of simulation runs which demonstrate that the massively parallel SA scheme provided by the Boltzmann machine model clearly outperforms the corresponding traditional sequential SA scheme, provided that suitable massively parallel hardware is available.

2. Bayesian Networks

Let U denote a set of N discrete random variables, $U = \{U_1, \ldots, U_N\}$. We call this set our *variable base*. In the sequel, we use capital letters for denoting the actual variables, and small letters $u_1, \ldots, u_N$ for denoting their values. The values of all the variables in the variable base form a *configuration vector* or a *state vector* $\vec{u} = (u_1, \ldots, u_N)$, and all the M possible configuration vectors $(\vec{u}_1, \ldots, \vec{u}_M)$ form our *configuration space* Ω. Hence our variable base U can also be regarded as a random variable $\vec{U}$, the values of which are the configuration vectors. Generally, if $X \subseteq U$ is a set of variables, $X = \{X_1, \ldots, X_n\}$, by $\vec{X} = \vec{x}$ we mean that $\vec{x}$ is a vector $(x_1, \ldots, x_n)$, and $X_i = x_i$, for all $i = 1, \ldots, n$.

Let $\mathcal{F}$ denote the set of all the possible subsets of Ω, the set of *events*, and let $\mathcal{P}$ be a probability measure defined on Ω. The triple $(\Omega, \mathcal{F}, \mathcal{P})$ now defines a *joint probability distribution* on our variable base U. Having fixed the configuration space Ω (and the set of events $\mathcal{F}$), any probability distribution can be fully determined by giving its probability measure $\mathcal{P}$, and hence we will in the sequel refer to a probability distribution by simply saying "the probability distribution is $\mathcal{P}$".

Bayesian networks are a formalism for storing and retrieving the configuration probabilities $\mathcal{P}\{\vec{u}\}$ in a compact and efficient manner. The theoretical foundations of such models were set up in [50, 11, 12, 13], where it was shown how probability distributions can be defined efficiently by explicitly identifying and exploiting conditional independencies between the variables of U:

Definition 1. Let X, Y *and* Z be sets of variables. Then X is *conditionally independent of* Y, *given* Z, if

$$\mathcal{P}\{\vec{X} = \vec{x} | \vec{Y} = \vec{y}, \vec{Z} = \vec{z}\} = \mathcal{P}\{\vec{X} = \vec{x} | \vec{Z} = \vec{z}\}$$

holds for all vectors $\vec{x}, \vec{y}, \vec{z}$ such that $\mathcal{P}\{\vec{Y} = \vec{y}, \vec{Z} = \vec{z}\} > 0$.

Consequently, the variables in Z intercept any dependencies between the variables in X and the variables in Y: knowing the values of Z renders information about the values of Y irrelevant to determining the distribution of X. Using the concept of conditional independence we can now give the definition of Bayesian network models:

Definition 2. A Bayesian (belief) network (BN) representation for a probability distribution $\mathcal{P}$ on a set of discrete variables $U = \{U_1, \ldots, U_N\}$ is a pair $\{\mathcal{G}, \mathcal{P}_\mathcal{G}\}$, where $\mathcal{G}$ is a directed acyclic graph whose nodes correspond to the variables $U = \{U_1, \ldots, U_N\}$, and whose topology satisfies the following: each variable $X \in U$ is conditionally independent of all its non-descendants in $\mathcal{G}$, given its set of parents F_X, and no proper subset of F_X satisfies this condition. The second component $\mathcal{P}_\mathcal{G}$ is a set consisting of the corresponding conditional probabilities $\mathcal{P}\{X \mid F_X\}$.

As the parents of a node X can often be interpreted as direct causes of X, Bayesian networks are also sometimes referred to as *causal networks*, or as the purpose is Bayesian reasoning, they are also called *inference networks*. In the field of decision theory, a model similar to Bayesian networks is known as *influence diagrams* [51]. Rigorous introductions to Bayesian network modeling can be found in [11, 13, 14, 52, 53].

The importance of Bayesian network structures lies in the way such networks facilitate computing the joint configuration probabilities $\mathcal{P}\{\vec{u}\}$ as a product of simple conditional probabilities:

$$\mathcal{P}\{\vec{u}\} = \prod_{i=1}^{N} \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in F_{U_i}} U_j = u_j\}, \qquad (1)$$

where F_{U_i} denotes the predecessors of variable U_i, and the conditional probabilities $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in F_{U_i}} U_j = u_j\}$ can be found in the set $\mathcal{P}_\mathcal{G}$. Consequently, having defined a set of conditional independencies in a graphical form as a Bayesian network structure $\mathcal{G}$, we can use the conditional probabilities $\mathcal{P}_\mathcal{G}$ to fully determine the underlying joint probability distribution. The number of parameters needed, $m = |\mathcal{P}_\mathcal{G}|$, depends on the density of the Bayesian network structure: $m = \sum_i (|U_i| \prod_{U_j \in F_{U_i}} |U_j|)$. In many natural situations, this number can be several magnitudes smaller than the size of the full configuration space. An example of a simple Bayesian network is given in Figure 1.

Let C_i denote a set consisting of a variable U_i and all its predecessors in a Bayesian network G, $C_i = \{U_i\} + F_{U_i}$. We call these N sets the *cliques* of G. For each clique C_i, we define a *potential function* V_i which maps a given state vector to a real number,

$$V_i(\vec{u}) = \ln \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in F_{U_i}} U_j = u_j\}. \qquad (2)$$

The value of a potential function V_i depends only on the values of the variables in set C_i. As has been noted in several occasions [33, 37, 38, 54–56], these cliques can be used to construct an undirected graphical *Markov random field* model of the probability distribution $\mathcal{P}$, which means that the joint probability distribution (1) for a Bayesian network representation can also be expressed as a *Gibbs distribution*

$$\mathcal{P}\{\vec{u}\} = \frac{1}{Z} e^{-\sum_i - V_i(\vec{u})} = \frac{1}{Z} e^{\sum_i V_i(\vec{u})}, \qquad (3)$$

where the clique potential functions V_i are given in (2), and $Z = 1$.

3. The MAP Problem and Simulated Annealing

Let $X \subseteq U$ be a set of variables, $X = \{X_1, \ldots, X_n\}$. By an event $\{\vec{X} = \vec{x}\}$ we mean a subset of $\mathcal{F}$ which includes all the configurations $\vec{u} \in \Omega$ that are consistent with the assignment $\langle X_1 = x_1, \ldots, X_n = x_n \rangle$. Now assume we are given a partial value assignment $\langle \vec{E} = \vec{e} \rangle$ on a set $E \subset U$ of variables as an input, and let $\mathcal{P}_{\max}$ denote the maximal marginal probability in the set $\{\vec{E} =$

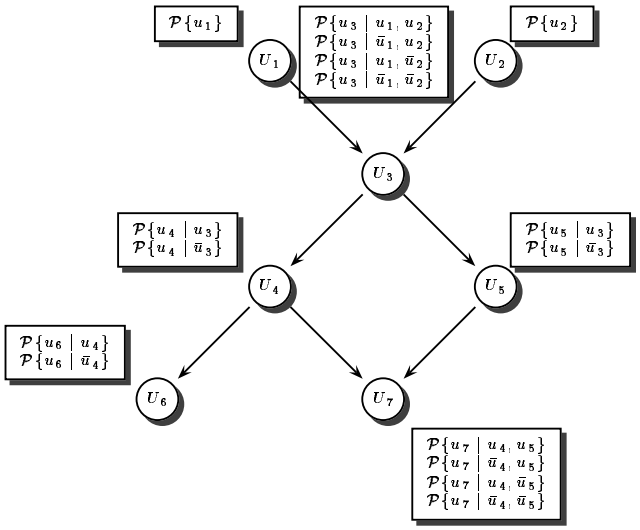

Fig. 1. A simple Bayesian network structure with 7 binary variables $U_1, \ldots, U_7$, and all the conditional probabilities required for determining the corresponding probability distribution exactly. By "u_i" we mean here the value assignment $\langle U_i = 1 \rangle$, and by "$\bar{u}_i$" the value assignment $\langle U_i = 0 \rangle$.

$\vec{e}\}$ consisting of all the configurations consistent with the given value assignment, $\mathcal{P}_{\max} = \max_{\vec{u} \in \{\vec{E}=\vec{e}\}} \mathcal{P}\{\vec{u}\}$. A state $\vec{u}_i \in \{\vec{E} = \vec{e}\}$ with the property $\mathcal{P}\{\vec{u}_i\} = \mathcal{P}_{\max}$ is now called a *maximum a posteriori probability (MAP)* state. In this study, we are interested in the task of finding a MAP state of the configuration space Ω, given some evidence $\langle \vec{E} = \vec{e} \rangle$, and we call this problem the *MAP problem*.

Let us consider a Bayesian network with a joint probability distribution of the form (3), and let us define the potential of a state $\vec{u}$ as the sum of clique potentials,

$$V(\vec{u}) = \sum_i V_i(\vec{u}). \tag{4}$$

We can now clearly find a MAP state by maximizing the potential function V.

In *simulated annealing (SA)* [18–21] this task is solved by producing a stochastic process $\{\vec{u}(t) \mid t = 0, 1, \ldots\}$, which converges to a MAP solution with high probability. The stochastic process used in SA is a Markov chain, which means that, at time t, the next state $\vec{u}(t+1)$ depends only on the current state $\vec{u}(t)$. Assuming that the current state $\vec{u}(t)$ is $\vec{u}_i$, the new state $\vec{u}(t+1)$ is produced by first generating a candidate $\vec{u}_j$ for the next state, and the candidate is then accepted by an *acceptance probability* A_{ij}, otherwise it is rejected, and we set $\vec{u}(t+1) = \vec{u}(t)$. For generating a new candidate state $\vec{u}_j$, most versions of simulated annealing use a simple scheme called *Gibbs sampling* where only one randomly chosen variable is allowed to change its value to a new randomly chosen value.

In the Metropolis-Hastings version [18, 57] of simulated annealing, the acceptance probabilities are defined by

$$A_{ij}(T) = \begin{cases} 1, \text{if } \exp(\frac{V(\vec{u}_j) - V(\vec{u}_i)}{T}) \geq 1, \\ \exp(\frac{V(\vec{u}_j) - V(\vec{u}_i)}{T}), \text{otherwise}, \end{cases}$$

where $T(t)$, the *computational temperature*, is a monotonely decreasing function converging to zero as t approaches infinity. Barker [58] has introduced an alternative method with acceptance probabilities of the form

$$A_{ij}(t) = \frac{1}{1 + \exp(\frac{V(\vec{u}_i) - V(\vec{u}_j)}{T(t)})}. \tag{5}$$

In both cases it can be shown that the resulting stochastic process will converge to a MAP solution with probability p, where p approaches one as the number of iterations approaches infinity [21]. Moreover, it can be shown that if the temperature $T(t)$ decreases towards zero slowly enough, the convergence is almost certain even with a finite time process [20]. Unfortunately, for a theoretically guaranteed convergence, a computationally infeasible exponential number of iterations is needed. Although in practice good results are sometimes obtained with a

relatively small number of iterations, the method can be excruciatingly slow. In the following, we show how to create a massively parallel implementation of simulated annealing by using the Boltzmann machine architecture.

4. Boltzmann Machines

A *Boltzmann machine (BM)* [8] is a neural network consisting of a set of binary nodes $\{S_1,\ldots,S_n\}$, where the *state* s_i of a node S_i is either 1 ("on"), or 0 ("off"). Each arc from a unit S_i to unit S_j is provided with a real-valued parameter w_{ji}, the *weight* of the arc. The arcs are symmetric, so $w_{ij} = w_{ji}$. Each unit S_i is also provided with a real-valued parameter θ_i, the *bias* of the unit.

Let $\vec{s} = (s_1,\ldots,s_n) \in \{0,1\}^n$ denote a global state vector of the nodes in the network. We define the *consensus* of a state $\vec{s}$ as

$$C(\vec{s}) = \sum_{j=1}^{n}\sum_{i=j}^{n} w_{ij}s_i s_j, \tag{6}$$

where w_{ii} denotes the bias θ_i of node S_i, and the weight w_{ij} is defined to be 0 if S_i and S_j are not connected (disregarding the sign reversal, the consensus function is equal to the the energy of a BM model used in most definitions).

The nodes of a BM network are updated stochastically according to the following probabilistic rule:

$$P(s_i = 1) = \frac{1}{1+\exp(\frac{-I_i}{T(t)})}, \tag{7}$$

where $I_i = \sum_j w_{ij}s_j$ is the *net input* to node S_i, and $T(t)$ is a real-valued parameter called temperature, which decreases with increasing time t towards zero. Consequently, each node is able to update its state locally, using the information arriving from the connecting neighbors as the input for a sigmoid function, thus offering a possibility for a massively parallel implementation of this algorithm.

Let us consider a state $\vec{s}$, and let $\vec{r}$ be a new state which is produced by changing the state of node S_i. It is now relatively easy too see [36] that the difference in consensus between the two states is exactly the net input in (7):

$$C(\vec{r}) - C(\vec{s}) = \sum_j w_{ij}s_j = I_i.$$

It follows that the BM updating process can be regarded as a Gibbs sampling process with acceptance probabilities identical to those given in (5), with respect to a Gibbs distribution

$$P\{\vec{s}\} = \exp(C(\vec{s})/T(t)).$$

Consequently, in principle the BM model can be used as a massively parallel tool for finding the maximum of the consensus function. However, the acceptance probability (7) sets here implicitly some additional requirements to the generation probabilities, since the difference in consensus is calculated by keeping all the nodes except one constant. For this reason, if two or more adjacent nodes of the BM network are to be updated at the same time, the corresponding transition probability matrix of the resulting process is no more stochastic, and hence convergence of the algorithm can not be guaranteed. On the other hand, if we allow only one node to be updated at a time, convergence can be guaranteed, but then the parallel nature of the algorithm is lost.

To solve this dilemma, it has been suggested [59] that the nodes of a BM network should be divided into clusters, where no two nodes inside of a cluster are connected to each other. Using this kind of *clustered BM* models we can maintain some parallelism and update all the nodes in one cluster at the same time, while a convergence theorem similar to the convergence theorem of simulated annealing can be proved [21, p. 139]. Obviously, the degree of parallelism depends on the number of clusters in the network. Unfortunately, the problem of finding a minimal set of clusters in a given network is NP-complete [21, p. 141]. In the next section we deal with a special class of BM architectures which has by definition only two clusters, being in this sense an optimal BM architecture.

5. Solving the MAP Problem by Boltzmann Machines

Let us consider a Bayesian network $\{\mathcal{G},\mathcal{P}_\mathcal{G}\}$ for a probability distribution $\mathcal{P}$ of the form (3), and let m denote the number of conditional probabilities in $\mathcal{P}_\mathcal{G}$, $\mathcal{P}_\mathcal{G} = \{p_1,\ldots,p_m\}$. Furthermore, let $\lambda_1,\ldots,\lambda_m$ denote the natural logarithms of the conditional probabilities p_i, $\lambda_i = \ln p_i$. We can now equally well forget about the cliques altogether, and express the potential func-

tion (4) as

$$V(\vec{u}) = \sum_{i=1}^{N} V_i(\vec{u}) = \sum_{j=1}^{m} \chi_j(\vec{u})\lambda_j, \qquad (8)$$

where the function $\chi_j(\vec{u})$ has value one, if $\vec{u}$ is consistent with the value assignment corresponding to probability p_j, otherwise it is zero. It is clear that if we find the maximum of the potential function V in (8), we have found the maximum of the probability distribution $\mathcal{P}$.

For our purposes, we need the function to be maximized to be unbounded above, so we scale the strictly negative potential function (8) by adding a constant λ^* to each of the parameters λ_j. If we set $\lambda^* \geq -\min_j \ln p_j$, the new potential function becomes non-negative. In our empirical tests described in Section 6, we set λ^* to be equal to $-\min_j \ln p_j$.

As the number of non-zero parameters χ_j is always exactly N, the number of cliques, the resulting new potential function is equal to the original function plus a constant $N\lambda^*$, and hence it has the same maximum points as the original function. In the following, we consider maximizing a potential function of the form (8), with the parameters λ_j scaled to positive numbers as described above. If we manage to map this function to a consensus function of a Boltzmann machine, we have accomplished our goal: massively parallel solution to the MAP problem.

Let us now consider a two-layer Boltzmann machine, where the first layer consists of $n = \sum_i |U_i|$ *feature nodes* $A_1, \ldots, A_n$, one for each value for each of the variables of the problem domain. The second layer has altogether m *pattern nodes* $B_1, \ldots, B_m$, one node for each of the parameters λ_j in the sum in (8). Initially, let us assume that each pattern node is connected to all the feature nodes in the first layer, but no two nodes in the same layer can be connected to each other. The consensus function of such a network can be expressed as

$$\begin{aligned} C(\vec{s}) &= C(a_1, \ldots, a_n, b_1, \ldots, b_m) \\ &= \sum_{j=1}^{m} b_j \sum_{i=1}^{n} w_{ji} a_i \\ &= \sum_{j=1}^{m} b_j I_j, \end{aligned} \qquad (9)$$

where $I_j = \sum_i w_{ji} a_i$ is the net input to pattern node B_j.

Let us now consider a pattern node B_j corresponding to a parameter λ_j, and let $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in F_{U_i}} U_k = u_k\}$ be the conditional probability used for computing λ_j,

$$\lambda_j = \ln \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in F_{U_i}} U_k = u_k\} + \lambda^*. \qquad (10)$$

We now set the weights of the arcs between pattern node B_j and feature nodes $A_1, \ldots, A_n$ in the following way:

1. The weights of all the arcs connecting node B_j to feature nodes representing values of variables not in $\{U_i, F_{U_i}\}$ are set to zero.
2. The weight from pattern node B_j to a feature node corresponding to the value u_i is set to λ_j, and the weights of the arcs to feature nodes corresponding to other values of the variable U_i are set to $-\lambda_j$.
3. The weight from node B_j to feature nodes corresponding to the values u_j appearing on the right hand side in (10) are set to λ_j, and arcs to feature nodes representing other values of the predecessors of the variable U_i are set to $-\lambda_j$.
4. The bias θ_j of pattern node B_j is set to $(n_j^+ - 1)\lambda_j$, where n_j^+ is the number of positive arcs leaving from node B_j. However, if $n_j^+ = 0$, we set $\theta_j = 0$.

Following this construction for each of the pattern nodes B_j, we get a structure which we call a *two-layer Boltzmann machine (BM$_2$)*. An example of a BM$_2$ structure in a case of a simple Bayesian network is shown in Figure 2. As arcs with a zero-valued weight are irrelevant for the computations, they are excluded from the network.

Let us now consider a Bayesian network $\mathcal{G}_{\mathrm{BN}}$, and the corresponding BM$_2$ network $\mathcal{G}_{\mathrm{BM}}$. In the sequel, we use the term *feature vector* to denote the vector $\vec{a} = (a_1, \ldots, a_n)$, consisting of the states of the feature nodes of $\mathcal{G}_{\mathrm{BM}}$. Correspondingly, the vector $\vec{b} = (b_1, \ldots, b_m)$, consisting of the states of the pattern nodes is called a *pattern vector*.

Definition 3. The feature vector of a two-layer Boltzmann machine $\mathcal{G}_{\mathrm{BM}}$ is *consistent* (with respect to the corresponding Bayesian network $\mathcal{G}_{\mathrm{BN}}$), if there is exactly one feature node active for each of variables in $\mathcal{G}_{\mathrm{BN}}$, representing one possible value of that variable.

From Definition 3 it follows directly that a consistent feature vector can be mapped to an instantiation of $\mathcal{G}_{\mathrm{BN}}$. As before, let B_j be a pattern node corresponding to a parameter λ_j, and let $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in F_{U_i}} U_k = u_k\}$

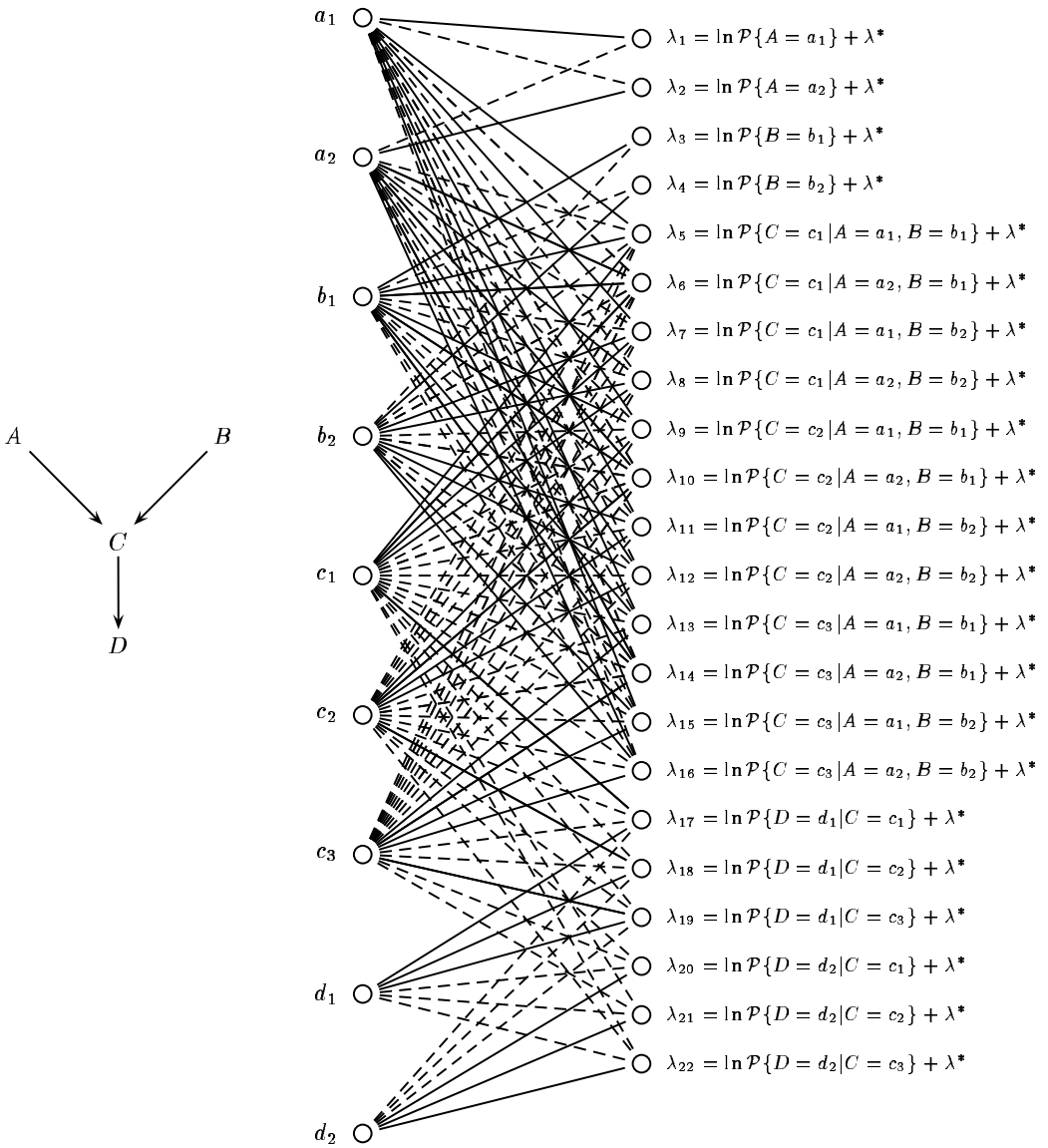

Fig. 2. A simple Bayesian network (on the left), and the corresponding BM_2 structure with the parameters λ_j. Solid lines represent arcs with positive weight, negative arcs are printed with dashed lines.

be the conditional probability used for computing λ_j. We now say that the pattern node B_j is consistent with a feature node $\vec{a}$ if $\vec{a}$ is consistent with the assignment $\langle U_i = u_i, \bigwedge_{U_k \in F_{U_i}} U_k = u_k \rangle$. Moreover, a pattern activation vector $\vec{b}$ is said to be consistent, if all the consistent pattern nodes are on, and all the inconsistent pattern nodes are off. Finally, a state $\vec{s} = (\vec{a}, \vec{b})$ of $\mathcal{G}_{\mathrm{BM}}$ is called consistent if both $\vec{a}$ and $\vec{b}$ are consistent.

It is now easy to proof the following simple lemma:

Lemma 1. *A BM_2 network converges to a consistent state.*

Proof: Let $\vec{s} = (\vec{a}, \vec{b})$ be the final state of a BM_2 updating process. We know that $\vec{s}$ is a state which maximizes the consensus $C(\vec{s}) = \sum_{j=1}^{m} b_j I_j$. It is now easy too see that if $\vec{a}$ is a consistent vector, $\vec{b}$ must also be consistent, since if there were inconsistent pattern nodes active on the second layer in this case, switching them off would increase the consensus of the network, and correspondingly, switching any inactive consistent

pattern node on would increase the consensus. On the other hand, $\vec{a}$ can not be inconsistent, since in this case all the pattern nodes connected to inconsistent feature nodes would be inactive, which means that removing inconsistencies would increase the number of active pattern nodes, thus increasing the consensus. It now follows that $\vec{s} = (\vec{a}, \vec{b})$ must be a consistent state. □

Using this lemma, we can show that the BM_2 structure can be used for solving the MAP problem:

Proposition 1. *The updating process of the BM_2 network converges to a state which maximizes the potential function V of the corresponding Bayesian network $\mathcal{G}_{BN}$, provided that all the parameters λ_j are nonnegative.*

Proof: According to Lemma 1, the network converges to a maximum consensus state where all the active pattern nodes are consistent with the consistent feature vector. This means that each final feature vector $\vec{a}$ corresponds to an instantiation $\vec{u}$. It is easy to see that the net input I_j to a consistent pattern node B_j is λ_j, and hence it follows that

$$\begin{aligned}
\max_{\vec{s}} C(\vec{s}) &= \max_{\vec{a}} \max_{\vec{b}} \sum_{j=1}^{m} b_j I_j \\
&= \max_{\vec{a}} \sum_{j=1}^{m} \max_{b_j \in \{0,1\}} (b_j I_j) \\
&= \max_{\vec{u}} \sum_{j=1}^{m} \chi_j(\vec{u}) \lambda_j \\
&= \max_{\vec{u}} V(\vec{u}).
\end{aligned}$$

□

Please note that if the parameters λ_j were not scaled to nonnegative numbers as suggested earlier, the potential function would be strictly negative, and the BM_2 construction would be useless, since the network would then always have a trivial maximum point at zero, corresponding to a state where all the nodes are off.

6. Empirical Results

6.1. The Setup

To empirically test the effectiveness of the BM_2 construction, we generated several MAP instantiation problems, with N, the number of variables, ranging from 2 to 24. For each N, we generated 10 random Bayesian networks, and attached to each network a random MAP instantiation problem by clamping half of the variables to randomly chosen values. This same test was performed with three different algorithms. As a benchmark, we used a brute force search algorithm (denoted by BF), which is guaranteed to find the MAP solution in $|\Omega_E| * m$ time steps, where $|\Omega_E|$ is the number of possible solutions, and m is the number of conditional probabilities attached to the Bayesian network in question. Secondly, we used a sequential simulated annealing algorithm on the Bayesian network (denoted by SSA), and the corresponding massively parallel BM_2 updating process (denoted here by BMSA). As we did not have access to real neural hardware, the BMSA model was tested by running simulations on a conventional Unix workstation. The time requirement for one iteration of the SSA was assumed to be $O(m)$, whereas the time requirement for one iteration for the BMSA algorithm was assumed to be $O(1)$ (all the nodes on one layer were updated at the same time, so updating the whole network took 2 time steps).

When considering the performance of the SSA or the BMSA algorithm, it is clear that the most critical issue is finding a suitable cooling scheme. Unfortunately, the theoretically correct logarithmic cooling scheme described in [20] can not be used in practice, since the number of iterations required grows too high even with relatively low starting temperatures: for instance, starting with the initial temperature of 2, annealing down to 0.1 would require more than 485 million iteration steps. The problem with heuristic annealing schemes is that if the annealing is done too cautiously, an unnecessarily large amount of computing time may be spent. On the other hand, if the annealing is done too quickly, the theoretical convergence results do not apply, and the results are unreliable. Recent theoretical modifications applied in *fast annealing (FA)* [24] and *adaptive simulated annealing (ASA)* [22, 23] offer interesting alternatives to the exponential Metropolis form of simulated annealing used in this study, but it is currently not clear whether the framework presented here can be extended to these forms of simulated annealing as well. This poses an interesting research problem that deserves further study.

In the experiments performed, our primary objective was not to study the efficiency of the sequential and massively parallel algorithms per se, but to see whether the speedup gained from parallelization

would be enough to compensate for the loss of accuracy in sampling — in other words, whether the BM_2 construction would be computationally practical to use in principle, if suitable hardware was available. It should be noted that as SSA and BMSA use the same parameters for determining the cooling schedule of the annealing algorithm, it seems reasonable to assume that use of the more sophisticated variants of simulated annealing [22–29] would demonstrate roughly equal speedup for both SSA and BMSA.

In our empirical tests, the temperature was lowered according to a simple cooling schedule, where the temperature was multiplied after each iteration by a constant *annealing factor F*, $F < 1.0$. A more detailed study of the behavior of SSA and BMSA with different annealing factors can be found in [36]. Based on this study, in the tests reported here, the annealing factor was set to 0.66. Although simple, this type of cooling schedule is very common, and has proven successful in many applications [21]. It is also empirically observed that more sophisticated annealing methods do not necessarily produce any better results than this simple method [60].

The main goal of the experiments was to study the behavior of the SSA and BMSA algorithms with increasingly complex MAP problems. The complexity of these problems can be increased in two ways: by changing the properties of the probability distribution on the configuration space in question, or by changing the size of the configuration space. We experimented with three methods for changing the configuration space probability distribution: by restricting the conditional probabilities of the Bayesian networks to small regions near zero or one, by changing the density of the Bayesian network structure, and by changing the size of the evidence set E, i.e. by changing the number of clamped variables. The size of the configuration space was increased in two ways: by allowing more variables in the Bayesian networks, and by allowing the variables to have more values.

6.2. The Results

It has been noted [61] that solving certain types of probabilistic reasoning tasks can become very difficult if the Bayesian network contains a lot of extreme probabilities (probabilities with values near zero or one). However, as already noted in [38], in our MAP problem framework this does not seem to be true. We experimented by restricting the randomly generated conditional probabilities of the Bayesian network model in the regions $[0.0, \delta], [1.0 - \delta, 1.0]$, and varied the value of δ between 0.5 and 0.0001, but observed no significant effect on the results with either SSA or BMSA. It would be an interesting research problem to study (analytically or empirically) how the Bayesian network probability distribution $\mathcal{P}$ changes with the parameter ε, but this question was not addressed here.

Increasing the density of Bayesian networks not only changes the shape of the probability distribution on the configuration space, but it also imposes a computational problem as it increases m, the number of the conditional probabilities that need to determined for the model. Since the BMSA algorithm (or actually its simulated massively parallel implementation) is independent of m, increasing the density does not affect the BMSA solution time very much, whereas the solution time of SSA increases significantly (see Figure 3). Nevertheless, it should be noted that we have here extended our experiments to very dense, and even fully connected networks. Naturally, this does not make any sense in practice, since the whole concept of Bayesian networks relies on the networks being relatively sparse. For this reason, in the sequel we use in our experiments relatively sparse networks only (which does not, however, mean that the networks were singly-connected or otherwise structurally simple).

The test set corresponding to Figure 3 consisted of 100 MAP problems on 10 Bayesian networks with only 8 binary nodes. When considering the results, it should be kept in mind that although the BF algorithm seems to work relatively well with these small networks, it does not scale up with increasing size of the networks (as we shall see in Figure 6). For the same reason, the exhaustive BF algorithm performs well with a small number of unclamped variables (in which case the search space is small), but from Figure 4 we see that as the number of unclamped variables increases, the time required for running BF grows rapidly. Both SSA and BMSA appear to be quite insensitive to the number of instantiated variables. In these tests, we used 100 MAP problems on 10 Bayesian networks with 16 binary variables.

In Figure 5, we plot the behavior of the algorithms as a function of the increasing configuration space, when the maximum number of variable values is increased. The test set consisted of 100 MAP problems on 10 10-node Bayesian networks with half of the variables clamped in advance. With networks of this size, the SSA algorithm seems to perform only

comparably to the BF algorithm. However, when the size of the networks is increased, the general tendency is clear: the exhaustive BF algorithm starts to suffer from combinatorial explosion, and fails to provide a computationally feasible solution to the MAP problem (see Figure 6). The SSA and BMSA algorithms, on the other hand, seem to scale up very well. In Figure 6, each data point corresponds to a test set consisting of 100 MAP problems on 10 Bayesian networks with binary nodes, and as before, half of the variables were clamped in advance.

7. Conclusion and Future Work

We presented a method for solving probabilistic reasoning tasks, formulated as finding the MAP state of a given Bayesian network, by a massively parallel neural network computer. Our experimental simulations strongly suggest that the massively parallel BMSA algorithm outperforms the sequential SSA algorithm, provided that suitable hardware is available. As can be expected, the proportional speedup gained from parallelization seems to increase with increasing problem complexity.

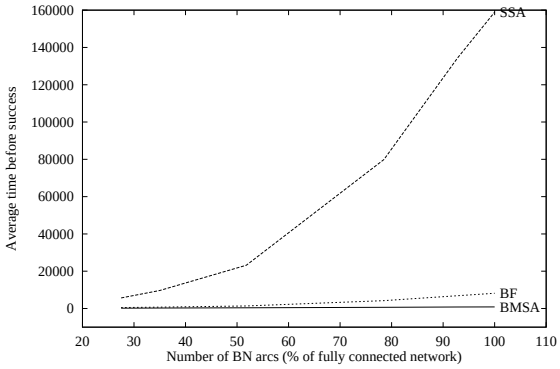

Fig. 3. The behavior of the BMSA and SSA algorithms as a function of the density of Bayesian network.

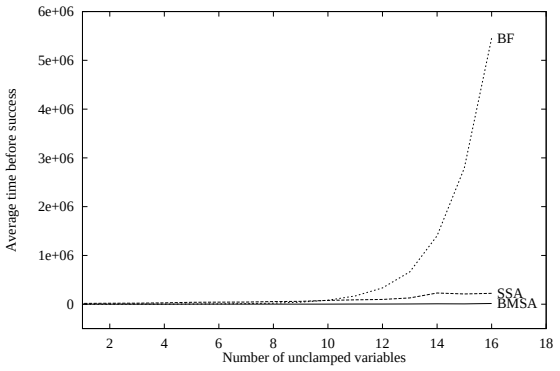

Fig. 4. The behavior of the algorithms as a function of the number of the unclamped variables.

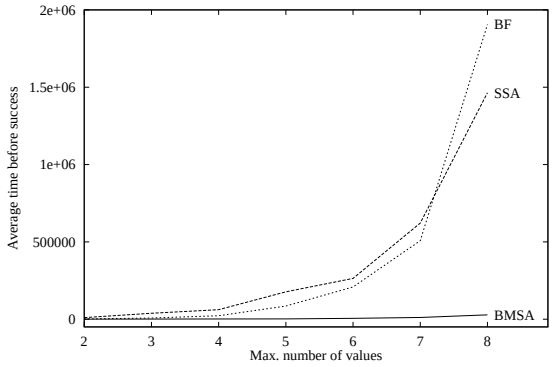

Fig. 5. The behavior of the algorithms as a function of the number of the values of the variables.

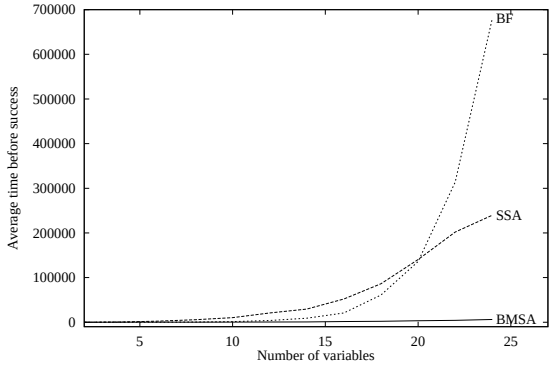

Fig. 6. The behavior of the algorithms as a function of the number of the variables.

It should be noted that our SSA realization of the Gibbs sampling/annealing method uses a heuristic cooling schedule which does not fulfill the theoretical requirements of the convergence theorem of SA. What is more, even as an approximation of the theoretically correct algorithm, the SSA method is not a very sophisticated alternative, and there exist several elaborate techniques for making the algorithm much more efficient than in the experiments here. Nevertheless, it can also be argued that the same techniques would probably yield a similar speedup for the BMSA algorithm as well. Consequently, we believe that these results show that if suitable neural hardware is available, the BMSA algorithm offers a promising basis for building an extremely efficient tool for solving MAP problems.

From the Bayesian network point of view, the mapping presented here provides an efficient implementational platform for performing probabilistic reason-

ing with the stochastic simulated annealing algorithm. From the neural network point of view, the mapping offers an interesting opportunity for constructing neural models from expert knowledge, instead of learning them from raw data. The resulting neural network could also be used as a (cleverly chosen) initial starting point to some of the existing learning algorithms [8–10, 62] for Boltzmann machines, in which case the learning problem should become much easier than with a randomly chosen initial state. Moreover, the resulting "fine-tuned" neural network could also be mapped back to a Bayesian network representation after the learning, which means that the mapping can also be seen as a tool for extracting high-level knowledge from neural networks. From the Bayesian network point of view, this kind of a "fine-tuning" learning process could also be useful in detecting mutually inconsistent probabilities, or other inconsistencies in the underlying Bayesian network representation.

It should also be noted that the results presented here can in principle be used as a computationally efficient, massively parallel tool for solving optimization problems in general, and not only for solving MAP problems as formulated here. Naturally, the efficiency of such an approach would largely depend on the degree to which the function to be maximized can be decomposed as a linear sum of functions, each depending only on a small subset of variables (corresponding to the cliques in the Bayesian network formalism). Further studies on this subject are left as a goal for future research.

Acknowledgements

This research has been supported by the Academy of Finland, Jenny and Antti Wihuri Foundation, Leo and Regina Wainstein Foundation, and the Technology Development Center (TEKES).

References

1. J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA, 1988.
2. J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing 2: Directions for Research*. MIT Press, Cambridge, MA, 1991.
3. D.E. Rumelhart and J.L. McClelland, editors. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA, 1986.
4. J.L. McClelland and D.E. Rumelhart, editors. *Parallel Distributed Processing*, volume 2. MIT Press, Cambridge, MA, 1986.
5. J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.
6. E.B. Baum. Towards practical 'neural' computation for combinatorial optimization problems. In Denker, J., editor, *Proceedings of the AIP Conference 151: Neural Networks for Computing*, pages 53–58, Snowbird, UT, 1986. American Institute of Physics, New York, NY.
7. G.E. Hinton and T.J. Sejnowski. Optimal perceptual inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 448–453, Washington DC, June 1983. IEEE, New York, NY.
8. G.E. Hinton and T.J. Sejnowski. Learning and relearning in Boltzmann machines. In Rumelhart and McClelland [3], pages 282–317.
9. Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. In J. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Neural Information Processing Systems 4*, pages 912–919. Morgan Kaufmann Publishers, San Mateo, CA, 1992.
10. C. Galland. *Learning in Deterministic Boltzmann Machine Networks*. PhD thesis, Department of Physics, University of Toronto, 1992.
11. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.
12. R.D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36(4):589–604, July-August 1988.
13. R.E. Neapolitan. *Probabilistic Reasoning in Expert Systems*. John Wiley & Sons, New York, NY, 1990.
14. D. Heckerman, D. Geiger, and D.M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, September 1995.
15. M. Henrion. An introduction to algorithms for inference in belief nets. In M. Henrion, R.D. Shachter, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 129–138. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1990.
16. G.F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2–3):393–405, March 1990.
17. S.E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.
18. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, M.N. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chem. Phys.*, 21:1087–1092, 1953.
19. S. Kirkpatrick, D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
20. S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.
21. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Chichester, 1989.
22. L. Ingber. Very fast simulated re-annealing. *Mathematical Computer Modelling*, 8(12):967–973, 1989.
23. L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.
24. H. Szu and R. Hartley. Nonconvex optimization by fast simulated annealing. *Proceedings of the IEEE*, 75(11):1538–1540, November 1987.

25. G. Bilbro, R. Mann, and T. Miller. Optimization by mean field annealing. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 91–98. Morgan Kaufmann Publishers, San Mateo, CA, 1989.
26. C. Peterson and J.R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.
27. J. Alspector, T. Zeppenfeld, and S. Luna. A volatility measure for annealing in feedback neural networks. *Neural Computation*, 4:191–195, 1992.
28. N. Ansari, Rajendra S., and G. Wang. An efficient annealing algorithm for global optimization in Boltzmann machines. *Journal of Applied Intelligence*, 3(3):177–192, 1993.
29. S. Rajasekaran and John H. Reif. Nested annealing: A provable improvement to simulated annealing. *Theoretical Computer Science*, 99:157–176, 1992.
30. D. Greening. Parallel simulated annealing techniques. *Physica D*, 42:293–306, 1990.
31. L. Ingber. Simulated annealing: Practice versus theory. *Mathematical Computer Modelling*, 18(11):29–57, 1993.
32. H. Geffner and J. Pearl. On the probabilistic semantics of connectionist networks. Technical Report R-84, UCLA Computer Science Department, Los Angeles, CA, 1987.
33. K.B. Laskey. Adapting connectionist learning to Bayesian networks. *International Journal of Approximate Reasoning*, 4:261–282, 1990.
34. T. Hrycej. Common features of neural-network models of high and low level human information processing. In T. Kohonen, K. Mäkisara, O. Simula, , and J. Kangas, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN-91)*, pages 861–866, Espoo, Finland, June 1991. Elsevier Science Publishers B.V. (North-Holland).
35. R.M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.
36. P. Myllymäki. *Mapping Bayesian Networks to Stochastic Neural Networks: A Foundation for Hybrid Bayesian-Neural Systems*. PhD thesis, Report A-1995-1, Department of Computer Science, University of Helsinki, December 1995.
37. P. Myllymäki and P. Orponen. Programming the Harmonium. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 671–677, Singapore, November 1991. IEEE, New York, NY.
38. P. Myllymäki. *Bayesian Reasoning by Stochastic Neural Networks*. Ph.Lic. Thesis, Tech. Rep. C-1993-67, Department of Computer Science, University of Helsinki, 1993.
39. P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart and McClelland [3], pages 194–281.
40. P. Myllymäki. Using Bayesian networks for incorporating probabilistic a priori knowledge into Boltzmann machines. In *Proceedings of SOUTHCON'94*, pages 97–102, Orlando, March 1994. IEEE, Piscataway, NJ.
41. P. Myllymäki. Mapping Bayesian networks to Boltzmann machines. In A. Gammerman, editor, *Proceedings of Applied Decision Technologies 1995*, pages 269–280, London, April 1995. Unicom Seminars, London. Also: NeuroCOLT Technical Report NC-TR-95-034.
42. J.A. Barnden and J.B. Pollack, editors. *Advances in Connectionist and Neural Computation Theory, Vol. I: High Level Connectionist Models*. Ablex Publishing Company, Norwood, NJ, 1991.
43. G.E.(ed.) Hinton. Special issue on connectionist symbol processing. *Artificial Intelligence*, 46(1–2), 1990.
44. S. Goonatilake and S. Khebbal, editors. *Intelligent Hybrid Systems*. John Wiley & Sons, Chichester, 1995.
45. R. Sun. *Integrating Rules and Connectionism for Robust Commonsense Reasoning*. John Wiley & Sons, Chichester, 1994.
46. P. Floréen, P. Myllymäki, P. Orponen, and H. Tirri. Neural representation of concepts for robust inference. In F. Gardin and G. Mauri, editors, *Proceedings of the International Symposium Computational Intelligence II*, pages 89–98, Milano, Italy, September 1989. Elsevier Science Publishers B.V. (North-Holland).
47. P. Myllymäki, H. Tirri, P. Floréen, and P. Orponen. Compiling high-level specifications into neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 475–478, Washington D.C., January 1990. IEEE, New York, NY.
48. P. Floréen, P. Myllymäki, P. Orponen, and H. Tirri. Compiling object declarations into connectionist networks. *AI Communications*, 3(4):172–183, December 1990.
49. L. Shastri. *Semantic Networks: An Evidential Formalization and Its Connectionist Realization*. Pitman, London, 1988.
50. J. Pearl. Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.
51. R.A. Howard and J.E. Matheson. Influence diagrams. In R.A.Howard and J.E.Matheson, editors, *Readings in Decision Analysis*, pages 763–771. Strategic Decisions Group, Menlo Park, CA, 1984.
52. F. Jensen. *An Introduction to Bayesian Networks*. UCL Press, London, 1996.
53. E. Castillo, J. Gutiérrez, and A. Hadi. *Expert Systems and Probabilistic Network Models*. Monographs in Computer Science. Springer-Verlag, New York, NY, 1997.
54. T. Hrycej. Gibbs sampling in Bayesian networks. *Artificial Intelligence*, 46:351–363, 1990.
55. S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Stat. Soc., Ser. B*, 50(2):157–224, 1988. Reprinted as pp. 415–448 in [63].
56. D.J. Spiegelhalter. Probabilistic reasoning in predictive expert systems. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 1*, pages 47–67. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1986.
57. W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.
58. A.A. Barker. Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. *Aust. J. Phys.*, 18:119–133, 1965.
59. A. de Gloria, P. Faraboschi, and M. Olivieri. Clustered Boltzmann machines: Massively parallel architectures for constrained optimization problems. *Parallel Computing*, pages 163–175, 1993.
60. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; Part I, graph partitioning. *Operations Research*, 37(6):865–892, November-December 1989.
61. H.L. Chin and G.F. Cooper. Bayesian belief network inference using simulation. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 129–147. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1989.
62. G.E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1–3), September 1989.
63. G. Shafer and J. Pearl, editors. *Readings in Uncertain Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

Petri Myllymäki is one of the founders of the Complex Systems Computation (CoSCo) research group at the Department of Computer Science of University of Helsinki. For the past ten years, the CoSCo group has been studying complex systems focusing on issues related to uncertain reasoning, machine learning and data mining. Research areas include Bayesian networks and other probabilistic models, information theory, neural networks, case-based reasoning, and stochastic optimization methods. Dr. Myllymäki is currently working at the University of Helsinki as a research scientist for the Academy of Finland.