Pp. 269–280 in *Proceedings of Applied Decision Technologies 1995* (London, April 1995).

Mapping Bayesian Networks to Boltzmann Machines

Petri Myllymäki
Department of Computer Science
P.O.Box 26, FIN-00014 University of Helsinki, Finland
Email: Petri.Myllymaki@cs.Helsinki.FI

Abstract

We study the task of finding a maximal a posteriori (MAP) instantiation of Bayesian network variables, given a partial value assignment as an initial constraint. This problem is known to be NP-hard, so we concentrate on a stochastic approximation algorithm, simulated annealing. This stochastic algorithm can be realized as a sequential process on the set of Bayesian network variables, where only one variable is allowed to change at a time. Consequently, the method can become impractically slow as the number of variables increases. We present a method for mapping a given Bayesian network to a massively parallel Bolztmann machine neural network architecture, in the sense that instead of using the normal sequential simulated annealing algorithm, we can use a massively parallel stochastic process on the Boltzmann machine architecture. The neural network updating process provably converges to a state which solves a given MAP task.

1 Introduction

Bayesian networks can be used for defining a joint probability distribution on a set of discrete random variables by first explicitly identifying conditional independencies between the variables and then determining a set of structurally simple local conditional probabilities. Given a a situation where some of the variables are clamped (permanently fixed) to some value, the Bayesian network representation can be used for computing the conditional probabilities for the values of the free variables, given the clampings, or alternatively it can be used for finding the maximal probability variable-value combination among all the instantiations consistent with the clampings. In this study we are only concerned with the latter task, and we call it the *maximal a posteriori* instantiation problem, or shorter the *MAP problem*. As in diagnostic applications this kind of a system would provide the user the best explanation for a given set of symptoms, the solution is sometimes called the *most probable explanation* [24], and the task of obtaining the MAP solution is sometimes referred to as *abductive inference* [23].

It has been shown that the MAP problem for Bayesian networks belongs to the class of NP-hard problems with respect to the size of the corresponding network [1, 28], and hence the MAP problem is probably intractable (in the worst case sense) for Bayesian networks in general. However, for *singly-connected* Bayesian network structures (networks with at most one path between any two variables, disregarding the direction of the connecting arc), the MAP problem can be solved in polynomial time [24, 23]. Consequently, most existing systems for Bayesian reasoning transform first a given multi-connected BN structure to a singly-connected network, and use then the existing polynomial-time algorithms. This transformation can be done either explicitly by clustering several nodes together as in [16, 26] or [24, Ch.4.4.1], or implicitly by blocking multiply connected paths by conditioning a set of variables as in [24, Ch.4.4.2] (for a discussion of different transformation techniques, see [27]). Alternatively, clustering can also be done by introducing new *latent variables*

which represent separate clusters of original variables [4, 22]. The latent variable model offers an interesting opportunity for constructing Bayesian networks from data using unsupervised learning algorithms. However, it is clear that as the problem is NP-hard, any conditioning method may in the worst case take exponential time (assuming $P \neq NP$), and similarly, any clustering method may result in an exponentially large Bayesian network, which causes exponential execution times with respect to the size of the original network.

As the worst-case analysis of the MAP problem has indicated that there is not much hope of finding a polynomial time solution (for unrestricted BN structures), there has been a growing interest to develop algorithms which could be shown to be able to produce good solutions in the *probably approximately correct (PAC)* sense (error in the solution is small with high probability). In *Monte Carlo methods* the goal is to find a stochastic random process $S(t), t = 0, 1, \ldots$ which will converge to a good solution with high probability. For a special class of stochastic processes, *Markov chains*, this kind of behavior can be shown to occur in the limit. The hope is that this kind of a process would also degrade gracefully and give (with high probability) a good approximation of the correct solution in feasible time. *Simulated annealing* is a Markov chain Monte Carlo method that is theoretically capable of finding global optimum points of a given function in exponential time, and has also shown very good results in polynomial-time practical experiments. However, most versions of simulated annealing use a sampling method where only one variable of the given Bayesian network can be updated at a time, and hence when implemented on a conventional serial computer, the method may become impractically slow as the number of variables increases. In this study, we show how a given Bayesian network can be mapped to a massively parallel *neural network (NN)* architecture, in the sense that a given MAP instantiation problem on a Bayesian network can be solved by using a stochastic simulation process on a neural network architecture, instead of using a process on the actual BN structure. The NN architecture proposed here is a *Boltzmann machine* with two layers of stochastic units, where all the units on one layer are updated at the same time. Compared to the models suggested in [21, 19], the mapping proposed here is more general as it allows multivalued attributes, whereas the earlier models assumed the variables to be binary. The resulting NN updating process provably converges to a state which can be projected to a MAP solution on the original BN structure. In a sense, the NN updating process corresponds to a simulated annealing process on a given Bayesian network where all the variables can be updated at the same time. Consequently, if suitable hardware is available, the processing time becomes independent of the size of the Bayesian network. On the other hand, the NN 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 NN process is only an approximation of a simulated annealing process on the Bayesian network. In section 6 we show results of some simulations which show that the speedup gained from parallelization is enough for compensating the loss of accuracy in the stochastic process, provided that suitable massively parallel hardware is available.

2 Bayesian networks

Let $\boldsymbol{U}$ denote a set of N discrete random variables, $\boldsymbol{U} = \{U_1, \ldots, U_N\}$. 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* $\boldsymbol{\Omega}$. Hence our variable base $\boldsymbol{U}$ can also be regarded as a random variable $\vec{U}$, the values of which are the configuration vectors. Generally, if $\boldsymbol{X} \subseteq \boldsymbol{U}$ is a set of variables, $\boldsymbol{X} = \{X_1, \ldots, X_n\}$, 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 $\boldsymbol{\mathcal{F}}$ denote the set of all the possible subsets of $\boldsymbol{\Omega}$, the set of *events*, and let $\mathcal{P}$ be a probability measure defined on $\boldsymbol{\Omega}$. The triple $(\boldsymbol{\Omega}, \boldsymbol{\mathcal{F}}, \mathcal{P})$ now defines a *joint probability distribution* on our

variable base $\boldsymbol{U}$. Having fixed the configuration space $\boldsymbol{\Omega}$ (and the set of events $\boldsymbol{\mathcal{F}}$), 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 model relies on explicitly identifying and exploiting conditional independencies between the variables of $\boldsymbol{U}$:

Definition 2.1 Let $\boldsymbol{X}, \boldsymbol{Y}, and \boldsymbol{Z}$ be sets of variables. Then $\boldsymbol{X}$ is *conditionally independent of* $\boldsymbol{Y}$, *given* $\boldsymbol{Z}$, 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 $\boldsymbol{Z}$ intercept any dependencies between the variables in $\boldsymbol{X}$ and the variables in $\boldsymbol{Y}$: knowing the values of $\boldsymbol{Z}$ renders information about the values of $\boldsymbol{Y}$ irrelevant to determining the distribution of $\boldsymbol{X}$. Using the concept of conditional independence we can now give the definition of Bayesian network models:

Definition 2.2 A *Bayesian (belief) network* (BN) representation for a probability distribution $\mathcal{P}$ on a set of discrete variables $\boldsymbol{U} = \{U_1, \ldots, U_N\}$ is a pair $\{\mathcal{G}, \mathcal{P}_\mathcal{G}\}$, where $\mathcal{G}$ is a directed acyclic graph whose nodes correspond to the variables $\boldsymbol{U} = \{U_1, \ldots, U_N\}$, and whose topology satisfies the following: each variable $X \in \boldsymbol{U}$ is conditionally independent of all its non-descendants in $\mathcal{G}$, given its set of parents $\boldsymbol{F_X}$, and no proper subset of $\boldsymbol{F_X}$ satisfies this condition. The second component $\mathcal{P}_\mathcal{G}$ is a set consisting of the corresponding conditional probabilities $\mathcal{P}\{X \mid \boldsymbol{F_X}\}$.

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* [12]. Rigorous introductions to the Bayesian network theory can be found in [24, 23].

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:

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

where $\boldsymbol{F_{U_i}}$ denotes the predecessors of variable U_i, and the conditional probabilities $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F_{U_i}}} U_j = u_j\}$ 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: $m = \sum_i (|U_i| \prod_{U_j \in \boldsymbol{F_{U_i}}} |U_j|)$. In many natural situations, this number can be several magnitudes smaller than the size of the full configuration space.

Let C_i denote a set consisting of a variable U_i and all its predecessors in a Bayesian network G, $C_i = \{U_i\} + \boldsymbol{F_{U_i}}$. 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,

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

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 [13, 15, 16, 30, 21, 18], 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*

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

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

3 THE MAP problem and simulated annealing

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

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

We can now clearly find a MAP state by maximizing the potential function V. In *simulated annealing (SA)* 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 Metropolis-Hastings version [17, 9] of simulated annealing the acceptance probability is defined as

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

whereas Barker [3] has introduced an alternative method with acceptance probabilities of the form

$$A_{ij}(t) = \frac{1}{1 + \exp(\frac{V(\vec{u}_i) - V(\vec{u}_j)}{T(t)})}, \tag{4}$$

where $T(t)$, the *computational temperature*, is a monotonely decreasing function converging to zero as t approaches infinity. 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 [2]. 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 [6]. 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)* [11] 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

$$C(\vec{s}) = \sum_{j=1}^{n} \sum_{i=j}^{n} w_{ij} s_i s_j \,, \tag{5}$$

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:

$$P(s_i = 1) = \frac{1}{1 + \exp(\frac{-I_i}{T(t)})}, \tag{6}$$

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 $\vec{s}'$ be a new state which is produced by changing the state of node S_i. It is now relatively easy too see [20] that the difference in consensus between the two states is exactly the net input in (6):

$$V(\vec{s}') - V(\vec{s}) = \sum_j w_{ij} s_j \,.$$

it follows that the BM updating process can be regarded as a Gibbs sampling process with acceptance probabilities identical to those given in (4), with respect to a Gibbs distribution

$$P\{\vec{s}\} = e^{V(\vec{s})/T(t)}.$$

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 (6) 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 [7] 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 can be proved [2, 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 [2, 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 as

$$V(\vec{u}) = \sum_{i=1}^{N} V_i(\vec{u}) = \sum_{j=1}^{m} \chi_j(\vec{u})\lambda_j\,, \tag{7}$$

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 (7), 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 (7) by adding a constant λ^* to each of the parameters λ_j. If we set $\lambda^* > -\min_j \ln p_j$, the new potential function becomes strictly positive. 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 (7), 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 (7). 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

$$C(\vec{s}) = C(a_1, \ldots, a_n, b_1, \ldots, b_m) = \sum_{j=1}^{m} s_j \sum_{i=1}^{n} w_{ji} s_i = \sum_{j=1}^{m} s_j I_j\,, \tag{8}$$

where $I_j = \sum_i w_{ji} s_i$ 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 $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in \boldsymbol{F}_{U_i}} U_k = u_k\}$ be the conditional probability used for computing λ_j,

$$\lambda_j = \ln \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in \boldsymbol{F}_{U_i}} U_k = u_k\} + \lambda^*. \tag{9}$$

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

1. The weights of all the arcs connecting the node B_j to feature nodes representing values of variables not in $\{U_i, \boldsymbol{F}_{U_i}\}$ are set to zero.

2. The weight from the 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 (9) 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 the 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 1. 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 $\mathcal{G}_{\rm BN}$, and the corresponding BM_2 network $\mathcal{G}_{\rm BM}$. 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 $\mathcal{G}_{\rm BM}$. Correspondingly, the vector $\vec{b} = (b_1, \ldots, b_m)$, consisting of the states of the pattern nodes is called a *pattern vector*. We say that a feature vector of $\mathcal{G}_{\rm BM}$ is *consistent* (with $\mathcal{G}_{\rm BN}$), if there is exactly one feature node active for each of variables in $\mathcal{G}_{\rm BN}$, representing one possible value of that variable. Consequently, a consistent feature vector can be mapped to an instantiation of $\mathcal{G}_{\rm BN}$. As before, let B_j be a pattern node corresponding to a parameter λ_j, and let $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in \boldsymbol{F}_{U_i}} U_k = u_k\}$ 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 $< U_i = u_i, \bigwedge_{U_k \in \boldsymbol{F}_{U_i}} U_k = u_k >$. A state $\vec{s} = (\vec{a}, \vec{b})$ of $\mathcal{G}_{\rm BM}$ is called consistent if $\vec{a}$ is consistent, and only pattern nodes consistent with $\vec{a}$ are on. It is now easy to proof the following simple lemma:

Lemma 5.1 A BM_2 network converges to a consistent state.
Proof: We know that a BM_2 network converges to a state $\vec{s} = (\vec{a}, \vec{b})$ which maximizes the consensus $C(\vec{s}) = \sum_{j=1}^{m} s_j I_j$. On the other hand, it is easy to see that the net input I_j to a pattern node B_j is positive only if B_j is consistent with the current feature vector $\vec{a}$. If the feature vector $\vec{a}$ is not consistent, no pattern node is consistent with $\vec{a}$, and hence $C(\vec{s}) < 0$. As there are states with positive consensus, $\vec{a}$ cannot be inconsistent. It now follows that $\vec{s}$ must be consistent, since if there are inconsistent pattern nodes active on the second layer, switching them off would increase the consensus of the network. □.

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

Proposition 5.2 Assume we are given a Bayesian network $\mathcal{G}_{\rm BN}$, and a MAP problem with clampings $< \vec{E} = \vec{e} >$. This problem can be solved by the corresponding BM_2 network $\mathcal{G}_{\rm BM}$ by first clamping the feature nodes corresponding to the variables in $\boldsymbol{E}$ to values given by the assignment $< \vec{E} = \vec{e} >$, and then by letting the network run until converged. The feature vector of the final state, when mapped back to $\mathcal{G}_{\rm BN}$, is a MAP state.
Proof: According to lemma 5.1 the network converges to a stable state, where all the active pattern nodes are consistent with the consistent feature vector. It is easy to see that the net input to a consistent pattern node B_j is λ_j. Consequently, the consensus of the final state is

$$C(\vec{s}) = \sum_{j=1}^{m} s_j I_j = \sum_{j=1}^{m} s_j \lambda_j ,$$

which is equal to the maximal potential (7) of the Bayesian network $\mathcal{G}_{\rm BN}$. As this potential has the same maximum points as the distribution function of the network $\mathcal{G}_{\rm BN}$, the MAP problem has been solved. □.

If the parameters λ_j are not scaled as suggested earlier, the potential function is strictly negative, so this theorem could not be used without scaling as then the network would always have a trivial maximum point at zero, corresponding to a state where all the nodes are off.

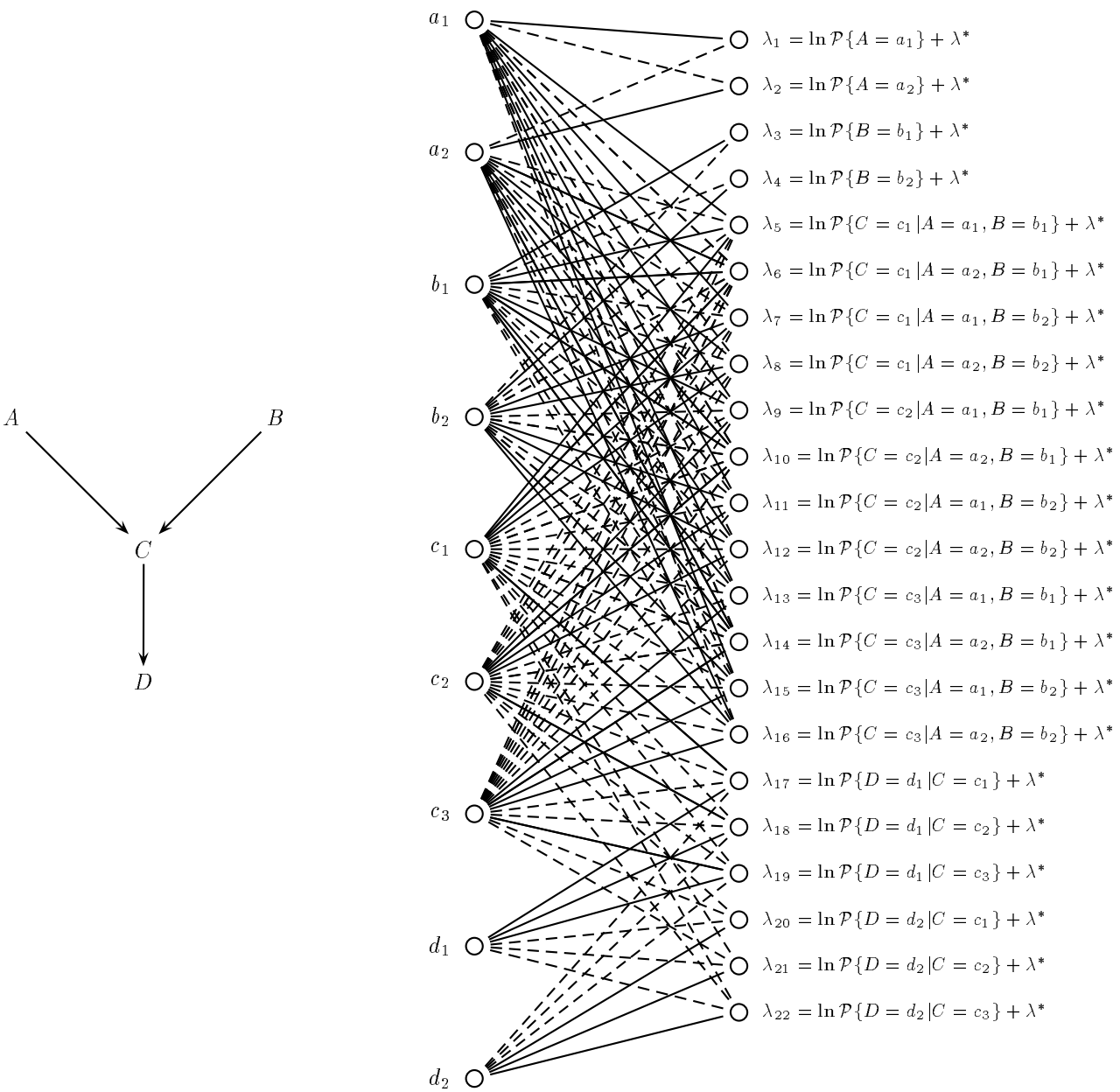

Figure 1: A simple Bayesian network (on the left), and the corresponding BM_2 structure with the parameters λ_j. Solid lines represent arcs with positive weight, negative arcs are printed with dashed lines.

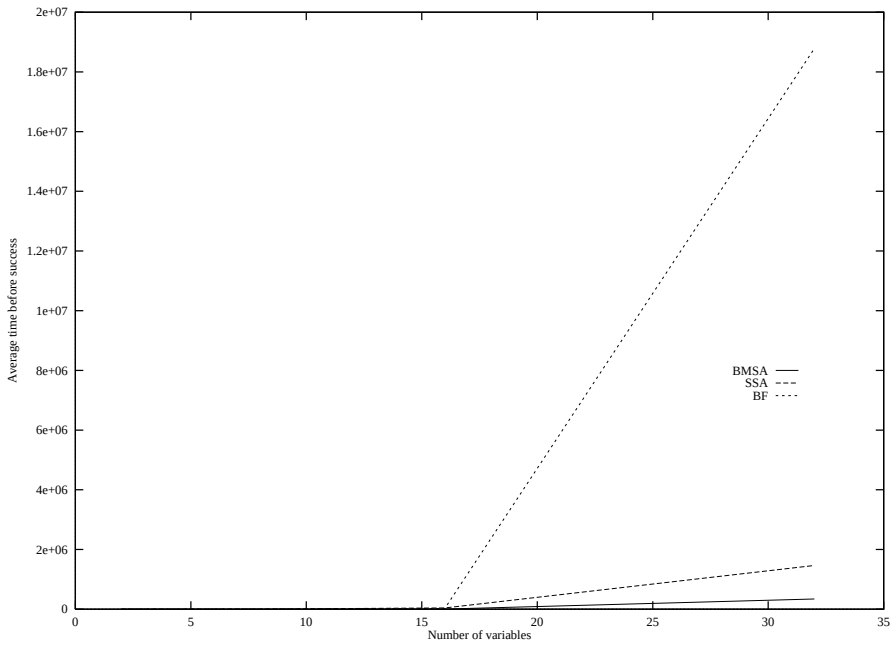

Figure 2: The average time needed to find a MAP state as a function of number of variables.

6 Empirical results

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 32. 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). Both SSA and BMSA used the same parameters for the cooling schedule of the annealing algorithm. 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). After each iteration, the temperature was lowered by multiplying it with the *annealing factor* F, $F < 1.0$. Although simple, this type of cooling schedule is very common, and has proven successful in many applications [2]. It is also empirically observed that more sophisticated annealing methods do not necessarily produce any better results than this simple method [14].

We use the term *trial* for denoting one simulation run, where the temperature goes from its initial value to zero. If the final state after a trial was not the MAP state, we started new trials until the MAP solution was found. As expected, the SSA samples the actual probability distribution more accurately, and hence needs fewer trials to succeed. Consequently, on a serial computer, the SSA algorithm is more efficient than BMSA. However, if we consider the actual number of time units needed to perform all these trials, we see that BMSA is more efficient than SSA, provided that suitable hardware is available (see figure 2).

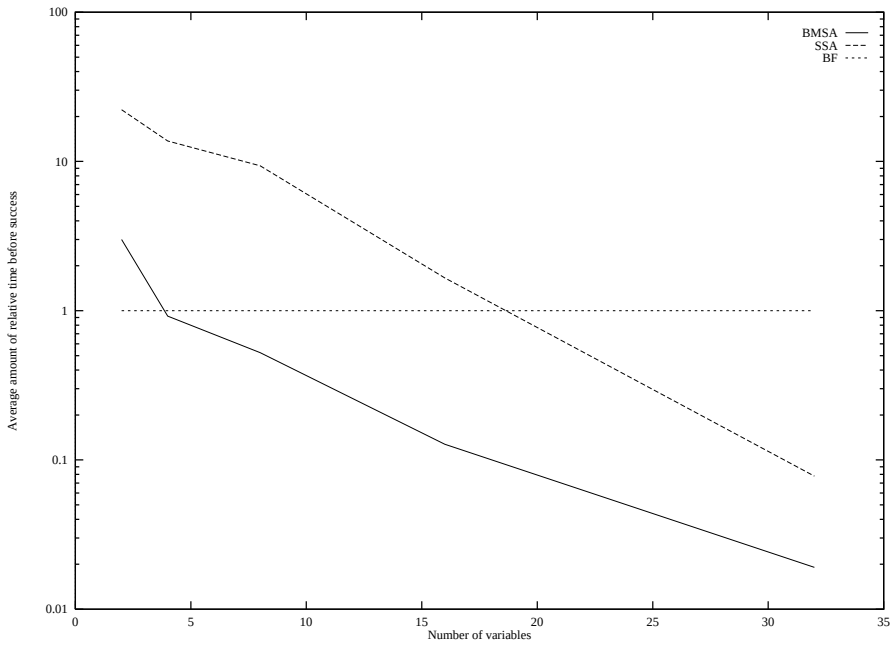

Figure 3: The average relative time needed to find a MAP state as a function of number of variables.

To get a clearer picture of the situation, we plotted in figure 3 the relative number of the time units needed, where the time needed for performing the brute force search is 1.

For a more detailed description of the test results, we refer to [20].

7 Conclusion and future work

We presented a method for solving a given Bayesian network MAP problem by a massively parallel neural network computer. From the Bayesian network point of view, the mapping offers a possibility to speed up a simulated annealing process for solving MAP problems. Our experimental simulations showed that the proposed neural network solution is more efficient than a normal sequential simulated annealing, provided that suitable hardware is available. When evaluating the results, it is most important to keep in mind that as our intention was not to study the efficiency of the simulated annealing algorithm per se, but to compare the massively parallel implementation to the sequential one, we did not spend a lot of time with tuning the parameters of the algorithm, and hence our realization of the simulated annealing is definitely not an optimal one. In spite of this, both the simple SSA algorithm and BMSA seem to scale up very well with the size of the problem domain state space. On the other hand, it should also be kept in mind that the material used in the tests was artificially generated, and results with real data could be different. Nevertheless, we see no reason why the massively parallel implementation would not be faster than the sequential one in this case also. It should also be noted that the method developed here apply only for massively parallel neural network implementations — parallelization of simulated annealing using more conventional computing architectures is discussed in [8].

From the neural network point of view, the proposed mapping from a Bayesian network to a Boltzmann machine offers an interesting opportunity for constructing neural networks from expert knowledge, instead of learning them from raw data. Moreover, the resulting neural network could also be used as a good initial state for existing learning algorithms [11, 5], which would probably

result in speeding up of the learning process. On the other hand, after having learned the neural network parameters, the network could be mapped back to a Bayesian network representation, thus allowing an interesting scheme for knowledge extraction from neural networks.

References

[1] Cooper, G.F., The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42 (1990) 2–3 (March), 393–405.

[2] Aarts, E. and Korst, J., *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Sons, Chichester, 1989.

[3] Barker, A.A., Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. *Aust. J. Phys.* 18 (1965), 119–133.

[4] Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W. , and Freeman, D., AutoClass: A Bayesian Classification System. Pp. 54–64 in *Proceedings of the Fifth International Conference on Machine Learning* (Ann Arbor, June 1988).

[5] Freund, Y., and Haussler, D., Unsupervised learning of distributions on binary vectors using two layer networks. Pp. 912–919 in *Neural Information Processing Systems 4*, edited by J.Moody, S.J.Hanson and R.P.Lippmann. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[6] Geman, S. and Geman, D., Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 6 (1984), 721–741.

[7] de Gloria, A., Faraboschi, P., and Olivieri, M., Clustered Boltzmann Machines: Massively parallel architectures for constrained optimization problems. *Parallel Computing* 19 (1993), 163–175.

[8] Greening, D., Parallel Simulated Annealing Techniques. Physica D 42 (1990), 293–306.

[9] Hastings, W.K., Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57 (1970), 97–109.

[10] Henrion, M., Shachter, R.D., Kanal, L.N., and Lemmer, J.F. (eds.), *Uncertainty in Artificial Intelligence 5.* Elsevier Science Publishers B.V. (North-Holland), Amsterdam 1990.

[11] Hinton, G.E., and Sejnowski, T.J., Learning and relearning in Boltzmann machines. Pp. 282–317 in [25].

[12] Howard, R.A., and Matheson, J.E., Influence diagrams. Pp. 763–771 in *Readings in Decision Analysis*, edited by R.A.Howard and J.E.Matheson. Strategic Decisions Group, Menlo Park, CA, 1984.

[13] Hrycej, T., Gibbs sampling in Bayesian networks. *Artificial Intelligence* 46 (1990), 351–363.

[14] Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C., Optimization by simulated annealing: an experimental evaluation; Part I, graph partitioning. *Operations Research*, 37 (1989) 6 (November-December), 865–892.

[15] Laskey, K.B., Adapting connectionist learning to Bayesian networks. *International Journal of Approximate Reasoning* 4 (1990), 261–282.

[16] Lauritzen, S.L. and Spiegelhalter, D.J., Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Stat. Soc., Ser. B* 50 (1988) 2, 157–224. Reprinted as pp. 415–448 in *Readings in Uncertain Reasoning*, edited by G.Shafer and J.Pearl. Morgan Kaufmann, San Mateo, CA, 1990.

[17] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, M.N. and Teller, E., Equations of state calculations by fast computing machines. *Journal of Chem. Phys.* 21 (1953), 1087–1092.

[18] Myllymäki, P., Bayesian reasoning by stochastic neural networks. Ph.Lic. Thesis, Report C-1993-67, Department of Computer Science, University of Helsinki, 1993.

[19] Myllymäki, P., Using Bayesian networks for incorporating probabilistic a priori knowledge into Boltzmann machines. Pp.97-102 in *Proceedings of SOUTHCON'94* (Orlando, March 1994).

[20] Myllymäki, P., Finding MAP Instantiations of Bayesian Networks by Stochastic Neural Networks. Ph.D. Thesis, University of Helsinki, Department of Computer Science (in preparation).

[21] Myllymäki, P., and Orponen,P., Programming the Harmonium. Pp. 671–677 in *Proceedings of the International Joint Conference on Neural Networks* (Singapore, November 1991), Vol. 1. IEEE, New York, NY, 1991.

[22] Myllymäki, P., and Tirri, H., Constructing computationally efficient Bayesian models via unsupervised clustering. Pp. 237–248 in *Probabilistic Reasoning and Bayesian Belief Networks*, edited by A.Gammerman. Alfred Waller Publishers, Suffolk 1995.

[23] Neapolitan, R.E., *Probabilistic Reasoning in Expert Systems.* John Wiley & Sons, New York, NY, 1990.

[24] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA, 1988.

[25] Rumelhart, D.E. and McClelland, J.L., *Parallel Distributed Processing*, Vol. 1. MIT Press, Cambridge, MA, 1986.

[26] Shachter, R.D., Probabilistic Inference and Influence Diagrams. *Operations Research*, 36 (1988) 4 (July-August), 589–604.

[27] Shachter, R.D., Evidence absorption and propagation through evidence reversals. Pp. 173–190 [10].

[28] Shimony, S.E., Finding MAPs for belief networks is NP-hard. *Artificial Intelligence* 68 (1994), 399-410.

[29] Smolensky, P., Information processing in dynamical systems: Foundations of Harmony Theory. Pp. 194-281 in [25].

[30] Spiegelhalter, D.J., Probabilistic reasoning in predictive expert systems. Pp. 47–67 in *Uncertainty in Artificial Intelligence* edited by L.N.Kanal and J.F.Lemmer. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1986.