Department of Computer Science
Series of Publications A
Report A-1995-1

Mapping Bayesian Networks to Stochastic Neural Networks: A Foundation for Hybrid Bayesian-Neural Systems

Petri Myllymäki

University of Helsinki
Finland

University of Helsinki
Department of Computer Science
Series of Publications A, No. A-1995-1

Petri Myllymäki

Mapping Bayesian Networks to Stochastic Neural Networks: A Foundation for Hybrid Bayesian-Neural Systems

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XII, Main Building, on December 5th, 1995, at 2 p.m.

Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland

Helsinki, December 1995

Contact information

Postal address:

Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
Finland

Email address: postmaster@cs.Helsinki.FI (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 0 708 51

Telefax: +358 0 708 44441

ISSN 1238-8645
ISBN 951-45-7211-4
Computing Reviews (1991) Classification: G.3, F.1.1, G.1.6, I.2.5
Helsinki 1995
Yliopistopaino

Mapping Bayesian Networks to Stochastic Neural Networks: A Foundation for Hybrid Bayesian-Neural Systems

Petri Myllymäki

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/

Ph. D. Dissertation, Series of Publications A, No. A-1995-1
Helsinki, December 1995, 93 pages
ISSN 1238-8645, ISBN 951-45-7211-4

Abstract

In this work, we are interested in the problem of finding maximum a posteriori probability (MAP) value assignments for a set of discrete attributes, given the constraint that some of the attributes are permanently fixed to some values a priori. For building a system capable of this type of uncertain reasoning in practice, we need first to construct an accurate abstract representation of the problem domain, and then to establish an efficient search mechanism for finding MAP configurations within the constructed model. We propose a hybrid Bayesian network-neural network system for solving these two subtasks. The Bayesian network component can be used for constructing a compact, high-level representation for the problem domain probability distribution quickly and reliably, assuming that suitable expert knowledge is available. The neural network component provides then a computationally efficient, massively parallel platform for searching the model state space. The main application areas for these kinds of systems include configuration and design problems, medical diagnosing and pattern recognition.

For implementing a hybrid Bayesian-neural system as suggested above, we present here methods for mapping a given Bayesian network to a stochastic neural network architecture, in the sense that the resulting neural network updating process provably converges to a state which can be projected to a MAP state on the probability distribution corresponding to the original Bayesian network. From the neural network point of view, these mappings can be seen as a method for incorporating high-level, probabilistic a priori information directly into neural networks, without recourse to a time-consuming and unreliable learning process. From the Bayesian network point of view, the mappings offer a massively parallel implementation of simulated annealing where all the variables can be updated at the same time. Our empirical simulations suggest that this type of massively parallel simulated annealing outperforms the traditional sequential Gibbs sampling/simulated annealing process, provided that suitable hardware is available.

Computing Reviews (1991) Categories and Subject Descriptors:

G.3	[Probability and statistics]: Probabilistic algorithms
F.1.1	[Models of computation]: Neural networks
G.1.6	[Optimization]: Constrained optimization
I.2.5	[Programming languages and software]: Expert system tools and techniques

General Terms: Algorithms, Theory.

Additional Key Words and Phrases: Hybrid systems, Bayesian belief networks, connectionism, Monte Carlo algorithms, Gibbs sampling, simulated annealing, massive parallelism

Acknowledgments

I am grateful to my advisors, Professor Pekka Orponen and Professor Esko Ukkonen, for their valuable comments and suggestions concerning this thesis. I also wish to thank all the other members of the Complex Systems Computation Group at the Department of Computer Science of the University of Helsinki, especially the "primus motor" of the group, Henry Tirri, for providing a very stimulating and encouraging working environment, and for all the support I have been given during our work together.

The Department of Computer Science at the University of Helsinki, headed by Professor Martti Tienari, has provided me excellent working conditions for this research. I am also very grateful to the Computing Center at the University of Helsinki, headed by Lars Backström, for offering a most pleasant working atmosphere during the time I have worked there, and for the sympathetic attitude towards my requests for leave of absence. I also wish to thank Professor Alex Gammerman, and other members of the staff of the Royal Holloway and Bedford New College, University of London, for the pleasant time I spent at their beautiful campus during the spring of 1995. The financial support of the Jenny and Antti Wihuri Foundation, Leo and Regina Wainstein Foundation, University of Helsinki, and Technology Development Center (TEKES) is gratefully acknowledged.

Contents

Chapter 1

Introduction

This research deals with methods for developing expert system applications in real-world problem domains. In these domains — unlike with most artificial toy problems — the data is typically noisy: imprecise, incomplete or inconsistent. This means that traditional rule-based system relying on pure logic suffer from the brittleness of the resulting software: as the programs are sensitive even to the slightest inaccuracy or incompleteness in their input data [2], the systems tend to grow to have rule bases consisting of tens of thousands of rules, when trying to provide a specific rule for every possible situation [28]. Collecting these kinds of large rule bases can be a very expensive and time-consuming task in practice, and moreover, maintaining and updating large rule bases (while preserving consistency) appears to be extremely difficult [93]. Consequently, it is clear that some kind of a computational mechanism capable of handling uncertain data is necessary for providing expert systems with the robustness required for practical applications.

Several different frameworks for handling noisy data have been developed during the last two decades. In this work, we concentrate on numerical approaches for uncertain reasoning. The first systems of this kind, the most famous example being the MYCIN [15] system for diagnosing bacterial infections, used uncertainty factors and heuristic rules for manipulating these factors. It has been later discovered [48] that MYCIN's original heuristic computational scheme can actually be interpreted as (Bayesian) probabilistic reasoning with certain independence assumptions, similar to those used for constructing the first simple Bayesian schemes for uncertain reasoning, such as the PROSPECTOR [30] system. Unfortunately, in many problem domains these independence assumptions are not valid [74]. Recently, uncertain reasoning systems based on fuzzy logic [119] have gained popularity, especially in Japan [72, 94]. However, it has been shown that any consistent computational framework representing degree of uncertainty as numbers has to be based on axioms of probability theory [20, 77]. Consequently, Bayesian probability theory seems to offer a solid, unifying framework for uncertain reasoning in general [16, 37, 64].

In this work, we study uncertain reasoning in the probabilistic framework. We assume that the problem domain probability distribution is modeled by using a set of discrete attributes (i.e. discrete random variables), and concentrate on studying an *abductive inference* problem, where the goal is to find a *maximum a posteriori probability* (MAP) value assignment for the variables, given the constraint that some of the variables are instantiated to some fixed values in advance (a formal description of the MAP problem can be found in Chapter 2). Obviously, a MAP solver can be used for solving various configuration and design problems, where the goal is to find the best combination of discrete attributes. In addition, many problems in e.g. medical diagnosing, pattern recognition and natural language processing can be formulated in the MAP framework [111].

The structure of a generic MAP solver is shown in Figure 1.1. The system consists of two modules: a *model construction module* and a *query processing module*. The task of the model construction module is to build an accurate model of the problem domain probability distribution, i.e. to select the most suitable instance within the chosen family of mathematical models. This selection is done either by using low-level problem domain sample data (in which case the problem is often referred to as *machine learning*), or by using a priori high-level data provided by domain experts, or by using both types of data. The task of the query processing module is, given a partial attribute value assignment as the input ("query"), to find a MAP attribute value configuration among all the possible (i.e. consistent with the given partial instantiation) complete attribute value configurations within the chosen model.

In Chapters 3 and 4 we describe two different families of models — *Bayesian networks* and *neural networks* — and discuss how they can be used for building MAP solvers of the type described above. In Chapter 5, we show how to build a hybrid Bayesian-neural MAP solver, which offers an interesting possibility to avoid the disadvantages that occur when using either of these two families of models alone.

A Bayesian (belief) network [96, 106] is a graphical high-level representation of a probability distribution over a set of discrete variables. Intuitively speaking, a Bayesian network model 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

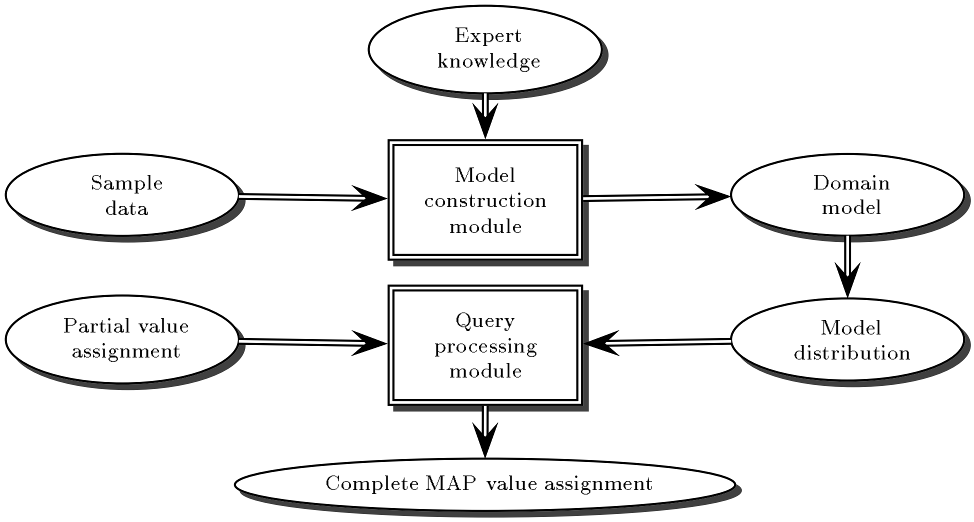


Figure 1.1: Structure of a generic MAP solver.

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 (a formal definition of Bayesian networks can be found Section 3.1). 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 [49].

The Bayesian network theory offers a framework for constructing algorithms for different probabilistic reasoning tasks (for a survey of probabilistic reasoning algorithms, see [52]). In Section 3.2, we discuss the complexity of the MAP problem within the Bayesian network framework, and review briefly some attempts to develop algorithms for solving MAP problems in the Bayesian network framework. Unfortunately, for a general network structure, the MAP problem can be shown to be NP-hard [19, 112], 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 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*. In this study, we shall concentrate on this type of a stochastic, approximative approach.

From the optimization theory point of view, the task of the query processing module of the MAP solver is to solve a constrained global optimization problem, where the constraints are the initial values for a subset of random variables, and the function to be maximized is (within the Bayesian network framework) the probability distribution defined by the given Bayesian network. *Markov Chain Monte Carlo (MCMC)* algorithms [80, 45] are stochastic algorithms that can be used in exponentially large search spaces for finding global maxima in feasible time with high probability. In this study, we are mainly concerned with the most common of the MCMC methods, *Gibbs sampling* [40] (for a short survey of other stochastic simulation methods, see [52]). In Gibbs sampling, a large collection of representative configurations of the problem domain model is generated by iterative local sampling, and the actual variable-value combination occurrence probabilities can be estimated by sample frequencies. On the other hand, combined with a stochastic technique called *simulated annealing (SA)* [80, 71], the Gibbs sampling process will, with high probability, converge to the global maximum of a given function consistent with the initial constraints. This iterative process can be seen as a kind of a stochastic local search, where the probability of finding the globally optimal solution approaches one as the number of iterative steps used approaches infinity [40, 1]. Consequently, simulated annealing can be used for solving the MAP problem stochastically. In Section 3.3 we present the general theoretical framework for MCMC algorithms and the simulated annealing technique.

After being introduced to the optimization theory community in [71], SA has been applied to many different (NP-hard) optimization problems, such as TSP [71], graph partitioning [65], graph coloring [66], number set partitioning [66] and clustering [105, 14] (for an extensive survey of applications, see [73] or [1, pp. 89–90]). One of the main difficulties with the SA algorithm is that the SA theory requires that the objective function to be optimized has to be representable in a specific *Gibbs distribution* form. Usually, a suitable Gibbs distribution has to be constructed manually for each separate optimization problem instance using very low-level concepts of the problem domain. However, there exists a graphical model called a *Markov random field* (MRF) [27, 11, 70, 40], which can be used as a formal tool for constructing Gibbs distributions. Similarly to Bayesian networks, an MRF representation consists of a graphical representation of the dependencies between the variables, together with a set of parameters, *clique potentials*. The values of the clique potentials determine uniquely a Gibbs distribution on the variable value assignment space.

Unfortunately, generally the MRF clique potentials cannot be easily determined directly, as they have no semantically clear probabilistic interpretation. In the Bayesian network framework, however, there is a straightforward mapping from a given Bayesian network to an MRF, which gives a method for determining the MRF clique potentials automatically from the conditional probabilities of the Bayesian network. This means that the problem domain expert can produce an SA process corresponding to the problem domain probability distribution by simply determining a Bayesian network structure and the corresponding conditional probabilities, which can then be mapped to an Markov random field. The Gibbs distribution corresponding to the resulting MRF can then be used for constructing an SA process. This transformation process from the domain expert knowledge to an SA process is described in Section 3.4.

As noted above, Bayesian network theory offers an elegant solution for the model construction module of our MAP solver. Furthermore, the concept of Markov random fields allows us to construct simulated annealing processes for solving MAP problems in the Bayesian network framework, thus providing us with a computational mechanism for the query processing module. The structure of the corresponding Bayesian MAP solver is shown in Figure 1.2. The main difficulty with this approach in practice lies in the inefficiency of the standard sequential simulated annealing: sampling of configurations requires a lot of iterative processing, and consequently implementations of the algorithm on conventional serial computers can be excruciatingly slow with large Bayesian networks.

In Chapter 4 we consider another family of models for building MAP solvers, *neural networks (NN)*. 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 collections [5, 6, 103, 78]). 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 [59, 10]. Especially suitable for these tasks are stochastic neural network architectures, as these models are based on a stochastic updating process very similar to simulated annealing. In Section 4.1 we describe such a network architecture, called the *Boltzmann machine (BM)* [57, 58], and show how the updating process of a BM provably converges to a state which maximizes the *consensus function*, which is an objective function determined by

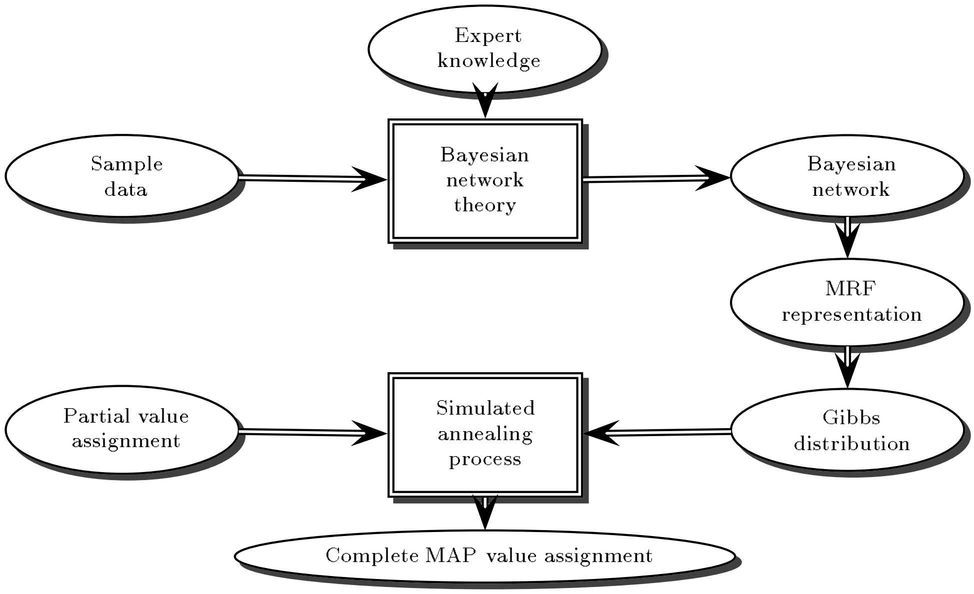


Figure 1.2: Structure of a Bayesian network MAP solver with a stochastic SA query processing module.

the states of the binary nodes of the network, and by the (constant) parameters ("weights") attached to the connecting arcs. In principle the BM model can be regarded as a massively parallel Gibbs sampler, but in order to ensure convergence of the updating process, the nodes have to be updated sequentially, which prevents efficient parallel implementations on neural hardware. In Section 4.2 we present a special case of the BM structure, the *harmony network* [114], which consists of two separate layers of nodes. The two-layer structure of the harmony network model allows us to update all the nodes in one layer simultaneously, making massively parallel implementations possible.

Boltzmann machines can be used as a computationally efficient tool for finding maximum points of the consensus function corresponding to a given network structure (provided that suitable massively parallel hardware is available). In order to apply these models for building MAP solvers, we need also an efficient method for constructing neural network structures with the consensus function having the same maximum points as the probability distribution of the problem domain (see Figure 1.3).

Boltzmann machines, and neural network models in general, are usually constructed from sample data by first selecting (more or less arbitrarily) some network architecture by fixing the number of nodes and the connections between the nodes, assigning (randomly chosen) weights to the connec-

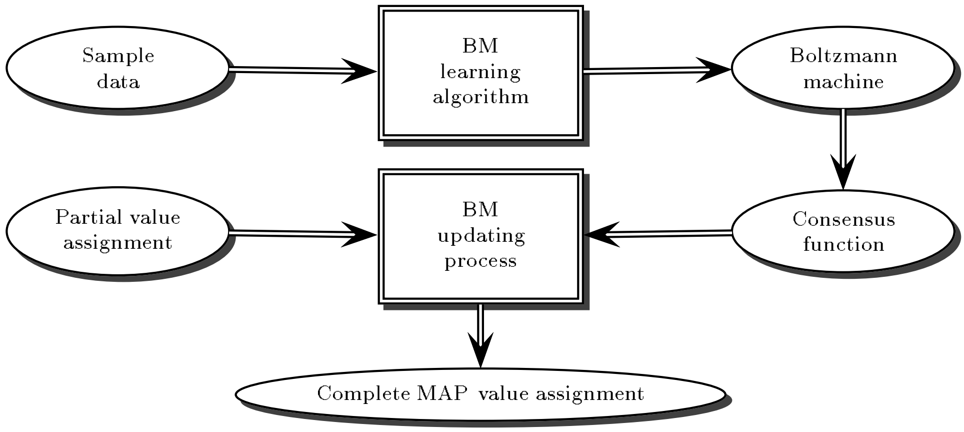


Figure 1.3: Structure of a Boltzmann machine MAP solver.

tions, and then using 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 [58, 35, 36]. 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 even if the chosen network happens to be structurally suitable for representing the problem domain probability distribution, the global maximum will not be reached, unless the initial starting point is chosen near the global maximum point. Even more disturbingly, even in this case the resulting network may be a poor model for the problem domain, as the machine learning algorithms tend to overfit the model with the sample data.

It follows that although Boltzmann machines provide us with an efficient computational mechanism for processing MAP queries, finding a Boltzmann machine structure with a suitable consensus function can be very difficult in practice, so this approach does not offer a satisfactory solution for the model construction problem. Bayesian networks, on the other hand, can be used for constructing models efficiently and reliably from high-level a priori

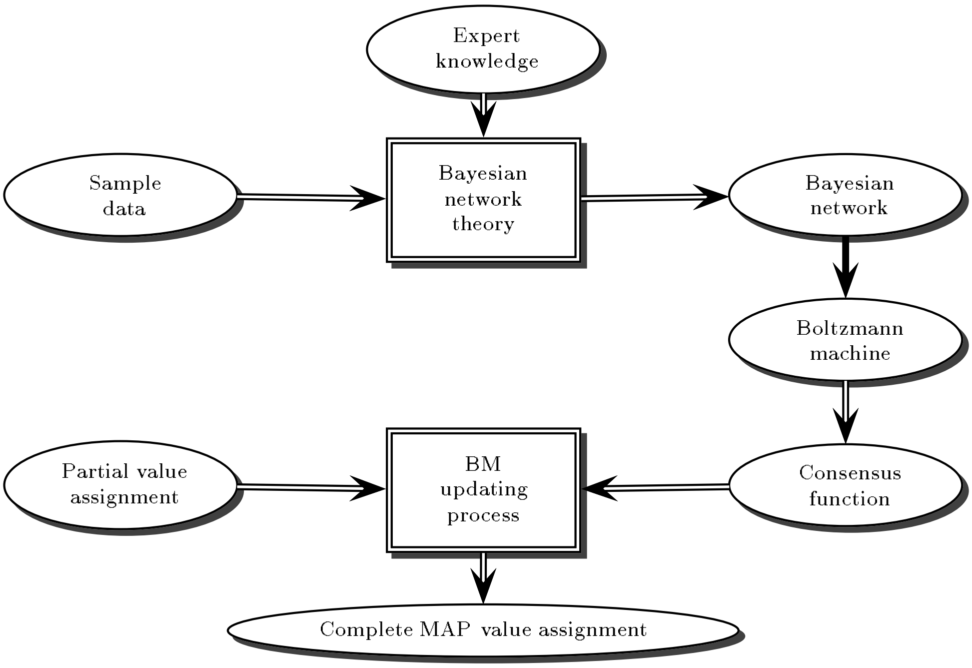


Figure 1.4: Structure of a hybrid Bayesian-neural MAP solver.

information, and moreover, they offer also a very promising framework for constructing models from low-level data, but they fail to provide a computationally attractive solution for the query processing module. We argue that we can achieve "the best of both worlds" by building a hybrid Bayesian-neural MAP solver, where the model construction module uses Bayesian network techniques, while the query processing module is implemented as a neural network.

For constructing a hybrid Bayesian-neural system as suggested above, 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 (see Figure 1.4). Although in some restricted domains this kind of a transformation is fairly straightforward to construct [57, 39, 75, 62, 90], the methods presented do not apply to general Bayesian network structures. In Chapter 5 we present three mappings from a given Bayesian network to a stochastic neural network architecture, in the sense that the updating process of the resulting neural network provably converges to a state which can be projected to a MAP solution on the variable state space. Consequently, a Bayesian network can first be used for constructing a model of the problem domain probability distribution, and the mappings provide a method for constructing a neural network architecture, which can then be used for processing MAP queries efficiently. In Section 5.1 we present a mapping to a harmony network structure with two layers of heterogeneous units, and show in Section 5.2 how a similar mapping can be constructed using a more standard Boltzmann machine architecture, with two layers of homogeneous units. As both of these mappings require the given Bayesian network to consist of only binary variables, we discuss in Section 5.3 possible ways to cope with Bayesian networks with multi-valued variables. In particular, we show a simple extension of the binary-variable case mapping which makes no assumptions about the number of values of the variables. The three constructions presented in these sections are published earlier in reports [84, 81], [82], and [83], respectively.

From the neural network point of view, the mapping from a Bayesian network to a Boltzmann machine can also be seen as a method for incorporating high-level, probabilistic a priori information directly into neural networks, without recourse to the time-consuming and unreliable learning process. Naturally, the resulting neural network could also be regarded as a cleverly chosen starting point to some learning algorithm, but this interesting idea is not studied here further.

Compared to other neural-symbolic hybrid systems (see e.g. [9, 56, 41, 116]), 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 [33, 89, 34]). Secondly, although some hybrid models provide theoretical justifications for the computations (see e.g. Shastri's system for evidential reasoning [109]), 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, as the mappings presented here create a bridge between the symbolic and neural representations, they can be used to create a "real" modular hybrid system, where two (or more) separate (neural or symbolic) inference modules work together. An attempt towards this kind of a hybrid system, consisting of a symbolic Prolog interpreter and a neural network, is described in [85].

In a sense, the updating processes of the NN component of our Bayesian-neural system correspond to simulated annealing processes on the Bayesian network, where all the variables can be updated at the same time. Consequently, with suitable massively parallel hardware, 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 the simulated annealing process on the Bayesian network. In Chapter 6 we compare empirically the performance of the Bayesian MAP solver in Figure 1.2 with the performance of the hybrid MAP solver in Figure 1.4. Our simulation results indicate that the speedup gained from parallelization compensates easily for the loss of accuracy in the stochastic process. This means that the massively parallel SA scheme outperforms the traditional sequential SA scheme, provided that suitable massively parallel hardware is available.

Chapter 2

The MAP problem

Let $\boldsymbol{U}$ denote a set of N discrete[1] 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 number of possible values for variable U_i is denoted by $|U_i|$. 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 $M = \prod_i |U_i|$ possible configuration vectors form our *configuration space* $\boldsymbol{\Omega}$, $\boldsymbol{\Omega} = \{\vec{u}_1, \ldots, \vec{u}_M\}$. 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 $\langle \vec{X} = \vec{x} \rangle$ we mean that $\vec{x}$ is a vector $(x_1, \ldots, x_n)$, and $X_i = x_i$, for all $i = 1, \ldots, n$.

The set of all the possible subsets of $\boldsymbol{\Omega}$, the set of *events*, is denoted by $\boldsymbol{\mathcal{F}}$. 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 $\langle \vec{X} = \vec{x} \rangle$. This event can also be expressed using the logical 'and'-operator:

$$\{\vec{X} = \vec{x}\} = \{ \bigwedge_{i=1,\ldots,n} X_i = x_i \}.$$

If there is no possibility of confusion, we drop the variable names X_i, and refer to an event simply as $\{x_1, \ldots, x_n\}$, or even more briefly as $\{\vec{x}\}$. Furthermore, let $\boldsymbol{U}_X$ denote the set $\{Y_i : Y_i \in \boldsymbol{U} - \boldsymbol{X}\}$, and $\vec{U}_X$ denote the corresponding random variable. If we want to emphasize that in the set $\{\vec{X} = \vec{x}\}$ there are no restrictions on the variables $Y_i \in \boldsymbol{U}_X$, we can write

$$\{\vec{X} = \vec{x}\} = \{ \bigwedge_{X_i \in \boldsymbol{U}_X} X_i, \bigwedge_{X_i \in \boldsymbol{X}} X_i = x_i \},$$

[1] We shall henceforth restrict ourselves to discrete variables, although this restriction is not necessary for the simulated annealing method in general [43].

or more briefly as $\{\vec{U}_X, \vec{x}\}$.

Let $\mathcal{P}$ denote 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 $\mathcal{P}$".

The general problem we are trying to solve is the following: given a partial value assignment $\langle \vec{E} = \vec{e} \rangle$ on a set $\boldsymbol{E} \subset \boldsymbol{U}$ of variables as an input, we wish to determine the values for the variables $U_i \notin \boldsymbol{E}$. In the Bayesian reasoning framework we can distinguish two different approaches to this problem: the *maximum a posteriori probability* assignment approach (MAP), and the *expected value estimation* (EVE) approach. As in [1], the former can be defined formally as follows:

Definition 2.1 (The MAP problem) Let $\mathcal{P}_{\max}$ denote the maximal probability in the set $\boldsymbol{\Omega}_{\mathrm{E}} = \{\vec{E} = \vec{e}\}$ consisting of all the configurations consistent with the given value assignment,

$$\mathcal{P}_{\max} = \max_{\vec{u} \in \boldsymbol{\Omega}_{\mathrm{E}}} \mathcal{P}\{\vec{u}\}.$$

Furthermore, let $\boldsymbol{\Omega}_{\mathrm{opt}}$ denote the set of all the configurations $\vec{u}_i$ in the set $\boldsymbol{\Omega}_{\mathrm{E}}$ with the property $\mathcal{P}\{\vec{u}_i\} = \mathcal{P}_{\max}$. We say that a probabilistic algorithm solves the MAP problem if it gives as the solution a configuration $\vec{u}_i$ with the probability

$$P\{\vec{u}_i\} = \begin{cases} 1/|\boldsymbol{\Omega}_{\mathrm{opt}}| & \text{, if } \vec{u}_i \in \boldsymbol{\Omega}_{\mathrm{opt}}, \\ 0 & \text{, otherwise,} \end{cases}$$

where $|\boldsymbol{\Omega}_{\mathrm{opt}}|$ denotes the number of elements in $\boldsymbol{\Omega}_{\mathrm{opt}}$.

Hence the purpose in the MAP task is to choose one of the maximal probability solutions using a uniform probability distribution on the set $\boldsymbol{\Omega}_{\mathrm{opt}}$, or if $|\boldsymbol{\Omega}_{\mathrm{opt}}| = 1$ (as often is the case), give as the solution the single MAP configuration $\vec{u}_{\mathrm{opt}}$. 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* [96], and the process of obtaining the MAP solution is referred to as *abductive inference* [92].

In the other, expected value estimation (EVE) approach, the goal is to compute (estimates of) the probabilities of the form $\mathcal{P}\{X = x \mid \vec{E} = \vec{e}\}$ for variables $X \notin \boldsymbol{E}$. It is easy to see that these two approaches may produce very different kind of solutions to a given problem. For example, let our variable base consist of only three binary variables, $\boldsymbol{U} = \{U_1, U_2, U_3\}$, and let the probability measure $\mathcal{P}$ be defined on all the eight possible configurations as follows:

$$\begin{array}{lrcl}
1. & \mathcal{P}\{U_1 = 0, U_2 = 1, U_3 = 1\} & = & 0.25 \\
2. & \mathcal{P}\{U_1 = 0, U_2 = 1, U_3 = 0\} & = & 0.0 \\
3. & \mathcal{P}\{U_1 = 0, U_2 = 0, U_3 = 1\} & = & 0.25 \\
4. & \mathcal{P}\{U_1 = 0, U_2 = 0, U_3 = 0\} & = & 0.0 \\
5. & \mathcal{P}\{U_1 = 1, U_2 = 1, U_3 = 1\} & = & 0.0 \\
6. & \mathcal{P}\{U_1 = 1, U_2 = 1, U_3 = 0\} & = & 0.25 \\
7. & \mathcal{P}\{U_1 = 1, U_2 = 0, U_3 = 1\} & = & 0.0 \\
8. & \mathcal{P}\{U_1 = 1, U_2 = 0, U_3 = 0\} & = & 0.25
\end{array}$$

Let us assume that the input assignment set $\boldsymbol{E}$ is empty, and the particular EVE task we are interested in is to compute probabilities $\mathcal{P}\{U_i = 1 \mid \emptyset\}$, for each of the variables U_1, U_2 and U_3. A MAP task solver should now, according to the apriori probabilities listed above, give one of the configurations 1, 3, 6 or 8 as the answer, whereas an EVE task solver would give us the following probabilities:

$$\begin{array}{rcccl}
\mathcal{P}\{U_1 = 1 \mid \emptyset\} & = & \mathcal{P}\{U_1 = 1\} & = & 0.25 + 0.25 = 0.5 \\
\mathcal{P}\{U_2 = 1 \mid \emptyset\} & = & \mathcal{P}\{U_2 = 1\} & = & 0.25 + 0.25 = 0.5 \\
\mathcal{P}\{U_3 = 1 \mid \emptyset\} & = & \mathcal{P}\{U_3 = 1\} & = & 0.25 + 0.25 = 0.5
\end{array}$$

Clearly both of the approaches have their disadvantages: in the MAP approach, we get one and only one configuration as the answer, and may never be aware of other, equally possible answers (unless we repeat the MAP solving process several times). On the other hand, in the EVE approach the user may not get any information of how the variables depend on each other (for instance, in our example $U_1 = 1$ and $U_3 = 1$ never occur at the same time). Naturally, the suitability of the methods for a specific problem depends on the application domain: is it more important to get one definite answer, or is it more important to get a likelihood estimation for a certain variable? It is also important to notice that there is no direct way of mapping the results of one approach to the other, and hence methods developed for one of these tasks do not necessarily apply for the other. In this study, we are only concerned with the MAP approach.

Let us assume for a moment that we are able to store all the M configurations $\vec{u}_1, \ldots, \vec{u}_M$ with the corresponding probabilities $\mathcal{P}\{\vec{u}_1\}, \ldots, \mathcal{P}\{\vec{u}_M\}$ in a huge (imaginary) table structure. To solve the MAP task we need now to search for a table item consistent with the input assignment, and with a maximal marginal probability $\mathcal{P}\{\vec{u}_i\}$. Let $t = 0, 1, \ldots$ be a discrete time scale and let our iterative search algorithm examine one table item in each time step t. The solution found by the search algorithm after t iterations, denoted by $S(t)$, is called the *state* of our iterative process. A "brute force" solution to the search problem is given in Algorithm 2.1.

Algorithm 2.1
Brute force solution to the MAP problem

- ▷ $P_{max} := 0$;
- ▷ for $t := 1$ to M do
  - ▷ if $(\vec{u}_t \notin \mathbf{\Omega}_{\mathrm{E}})$ then $\mathcal{P}\{\vec{u}_t\} := 0$;
  - ▷ if $(\mathcal{P}\{\vec{u}_t\} \geq P_{max})$ then
    - ▷ $P_{max} := \mathcal{P}\{\vec{u}_t\}$;
    - ▷ $S(t) := t$;

    else $S(t) := S(t-1)$;

Obviously, there are two main problems with the brute force approach presented above: first of all, the algorithm needs an exponential size storage space for storing all the M probabilities $\mathcal{P}\{\vec{u}_i\}$,

$$M = \prod_{i=1}^{N} |U_i| \geq 2^N,$$

and secondly it uses an exponential time going through all the M configurations. In many practical situations, the first problem can be solved by using the theory of Bayesian belief networks, as can be seen in the next section. The time complexity of the MAP problem (within the Bayesian network framework) is discussed in more detail in Section 3.2.

Chapter 3

Solving the MAP problem by Bayesian networks

3.1 Bayesian networks

The problem with the brute force approach to the MAP problem (Alg. 2.1) is not only that the number of parameters needed to be stored is exponential, but also that in practice they may be very difficult to obtain. If the probability distribution model of the problem domain is constructed by interviewing domain experts, then estimating probabilities of the form $\mathcal{P}\{U_1, \ldots, U_N\}$ goes very quickly beyond human capacity as the number of variables increases. On the other hand, estimating simple conditional probabilities of the form $\mathcal{P}\{X \mid Y, Z\}$ seems to be relatively easy, especially if there are direct causal relationships between the variables. To show how to formally utilize this intuitive argument, let us start by proving the following lemma:

Lemma 3.1 Given an ordering of the random variables $U_1, \ldots, U_N$, the joint probability of a variable assignment can be represented as a product of conditional probabilities $\mathcal{P}\{U_i \mid \boldsymbol{F_{U_i}}\}$, where $\boldsymbol{F_{U_i}}$ denotes the predecessors of variable U_i:

$$\mathcal{P}\{U_1 = u_1, U_2 = u_2, \ldots, U_N = u_N\} =$$
$$\mathcal{P}\{U_1 = u_1\}\mathcal{P}\{U_2 = u_2 \mid U_1 = u_1\} \cdots \mathcal{P}\{U_N = u_N \mid U_{N-1} = u_{N-1}, \ldots, U_1 = u_1\}.$$

Proof: According to Bayes' theorem,

$$\mathcal{P}\{U_1 = u_1, U_2 = u_2, \ldots, U_N = u_N\} =$$
$$\mathcal{P}\{U_N = u_N \mid U_{N-1} = u_{N-1}, \ldots, U_1 = u_1\}\mathcal{P}\{U_{N-1} = u_{N-1}, \ldots, U_1 = u_1\}.$$

The lemma is now proved by applying Bayes' theorem to the second term, and repeating this procedure recursively N times. □

In many natural domains, each variable U_i may in fact depend, or depend "directly", only on a small subset of variables in $\boldsymbol{F}_{\boldsymbol{U}_i}$, and not on all the preceding variables. For example, to be able to determine whether the variable corresponding to the fact "can fly" is true or not, it seems to be useful to know whether the object in question is a bird or not, and moreover, knowing this fact, the values of other variables representing the shape, color, size etc. of the object seem to be more or less irrelevant. This kind of relationships between the variables can be expressed formally as follows:

Definition 3.1 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 $\mathcal{P}\{\vec{Y} = \vec{y}, \vec{Z} = \vec{z}\} > 0$.

Intuitively, 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 3.2 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 $(\mathcal{B}_\mathcal{S}, \mathcal{B}_\mathcal{P})$, where $\mathcal{B}_\mathcal{S}$ is a directed acyclic graph whose nodes correspond to the variables in $\boldsymbol{U}$, and whose topology satisfies the following: each variable $X \in \boldsymbol{U}$ is conditionally independent of all its non-descendants in $\mathcal{B}_\mathcal{S}$, given its set of parents $\boldsymbol{F}_{\boldsymbol{X}}$, and no proper subset of $\boldsymbol{F}_{\boldsymbol{X}}$ satisfies this condition. The second component $\mathcal{B}_\mathcal{P}$ is a set consisting of the corresponding conditional probabilities of the form $\mathcal{P}\{X \mid \boldsymbol{F}_{\boldsymbol{X}}\}$.

For simplicity, we shall henceforth forget about the nodes, and treat the random variables U_i as if they were actually nodes of a graph. An example of a simple Bayesian network is given in Figure 3.1.

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* [60]. Thorough introductions to the Bayesian network theory can be found in [96, 92].

The importance of a Bayesian network structure lies in the way the network facilitates computing conditional probabilities. To make this precise, let

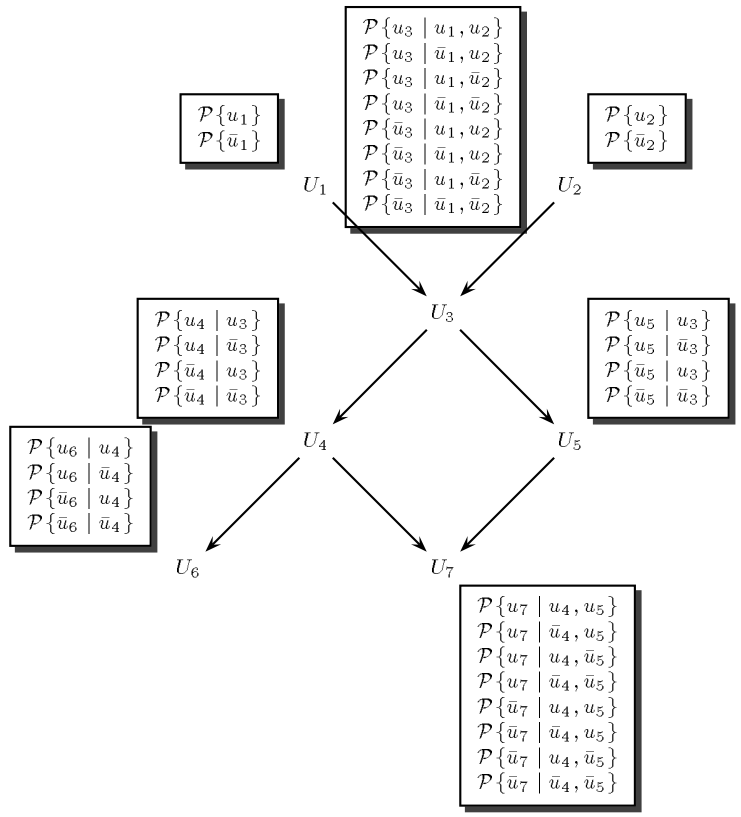


Figure 3.1: A simple Bayesian network structure $\mathcal{B}_{\mathcal{S}}$ with 7 binary variables $U_1, \ldots, U_7$, and the corresponding conditional probabilities that form the set $\mathcal{B}_{\mathcal{P}}$. By "$u_i$" we mean here the value assignment $\langle U_i = 1 \rangle$, and by "$\bar{u}_i$" the value assignment $\langle U_i = 0 \rangle$.

us use the following notations: given a variable X in a Bayesian network $\mathcal{B}_{\mathcal{S}}$, let $\boldsymbol{F_X}$ denote the set of parents (predecessors) of X, $\boldsymbol{S_X}$ the set of children (successors) of X, and $\boldsymbol{U_X}$ the set of all variables except X. The following result is then an immediate consequence of Definition 3.2:

Theorem 3.2 Given a variable X in a Bayesian network $\mathcal{B}$, define

$$\boldsymbol{B_X} = \boldsymbol{F_X} \cup \boldsymbol{S_X} \cup \bigcup_{Y \in \boldsymbol{S_X}} \boldsymbol{F_Y}.$$

Then X is conditionally independent of $\boldsymbol{U_X} - \boldsymbol{B_X}$, given $\boldsymbol{B_X}$.

Proof: See [96, p. 121]. $\Box$

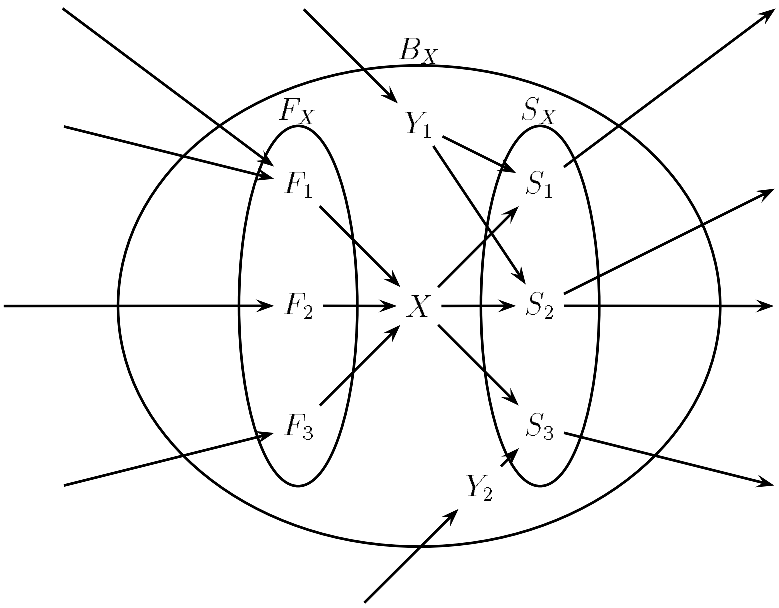


Figure 3.2: Markov blanket B_X of a variable X.

Thus, to determine the distribution of a variable X, given the values of all the other variables, it suffices to consider only the values of the variables in $\boldsymbol{B_X}$. A set of variables covering X in this sense is called a *Markov blanket* of X [96, p. 97].

Even an explicit formula for computing the distribution of X from its Markov blanket can be given as follows:

Theorem 3.3 Let $\mathcal{B}$ be a Bayesian network over a variable set $\{U_1, \ldots, U_N\}$. Given any configuration vector $\vec{u} = (u_1, \ldots, u_N)$, the probability distribution of each variable U_i in the network, conditioned on the values of all the other variables, may then be expressed as:

$$\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{U_X}} U_j = u_j\} =$$
$$c\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F_X}} U_j = u_j\} \prod_{U_j \in \boldsymbol{S_X}} \mathcal{P}\{U_j = u_j \mid \bigwedge_{U_k \in \boldsymbol{F_{U_j}}} U_k = u_k\}, \quad (3.1)$$

where c is a constant independent of u_i.

Proof: See [96, p. 218]. $\square$

In [95], this formula was used for finding an approximate solution to the EVE task, and in principle the same method could be used for the MAP task as well. In this study, however, we use another approach which exploits the following theorem:

Theorem 3.4 Let $\mathcal{B}=(\mathcal{B}_\mathcal{S},\mathcal{B}_\mathcal{P})$ be a Bayesian network over a variable set $\boldsymbol{U} = \{U_1,\ldots,U_N\}$. The probability for any variable value configuration vector $\vec{u} = (u_1,\ldots,u_N)$, can be expressed as

$$\mathcal{P}\{\vec{u}\} = \prod_{i=1}^{N} \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{\boldsymbol{U}_i}} U_j = u_j\},$$

where the conditional probabilities $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{\boldsymbol{U}_i}} U_j = u_j\}$ can be found in the set $\mathcal{B}_\mathcal{P}$, and are defined as the apriori probability $\mathcal{P}\{U_i = u_i\}$, if $\boldsymbol{F}_{\boldsymbol{U}_i} = \emptyset$.

Proof: Follows directly from Lemma 3.1 and Definition 3.2. $\square$

Consequently, having defined a set of conditional independencies in a graphical form as a Bayesian network structure $\mathcal{B}_\mathcal{S}$, we can use the conditional probabilities $\mathcal{B}_\mathcal{P}$ to fully determine the underlying joint probability distribution. The number of parameters needed, $|\mathcal{B}_\mathcal{P}|$, depends on the density of the Bayesian network structure:

$$|\mathcal{B}_\mathcal{P}| = \sum_{i=1}^{N}(|U_i| \prod_{U_j \in \boldsymbol{F}_{\boldsymbol{U}_i}} |U_j|).$$

In many natural situations, this number is much smaller than the size of the full configuration space, M. Consider for instance the simple example shown in Figure 3.1: the size of the configuration space is $2^7 = 128$, but the Bayesian network representation of the same probability distribution uses only 32 (structurally simple) conditional probabilities.

3.2 Complexity of the MAP problem

The nodes of a Bayesian network are usually considered to have a state which represents the value of the corresponding random variable. As the nodes represent all the random variables of the problem domain, a vector of the states of the nodes of the network, an *instantiation* of the network, corresponds to one possible configuration vector of the configuration space $\boldsymbol{\Omega}$. An evidence value assignment $\langle \vec{E} = \vec{e} \rangle$ is represented by "clamping" (permanently setting) nodes corresponding to variables $E_i \in \boldsymbol{E}$ to a state corresponding to the value e_i, respectively. Consequently, in the Bayesian network domain the MAP problem can be formulated as a problem of finding a maximum probability instantiation of a given network, while keeping the states of the clamped nodes unchanged.

It has been shown that if the probability distribution is represented as a Bayesian network, both the EVE task [19] and the MAP problem [112] belong to the class of NP-hard problems with respect to the size of the corresponding network. These results mean that it is very unlikely that a polynomial-time algorithm for solving the MAP problem for Bayesian networks could exist.[1]. Even finding an approximation of the EVE solution (in any constant degree of accuracy) is an NP-hard problem [23, 102]. As the answer to a MAP problem is a configuration vector, and not a probability, it is not obvious what an approximative solution in this case means. If we define the approximative MAP solution (with an accuracy of ϵ) to be any vector $\vec{u}$ with the property

$$\frac{P_{max} - \mathcal{P}\{\vec{u}\}}{\mathcal{P}\{\vec{u}\}} \leq \epsilon,$$

it is easy to see that (in the worst case) this problem is not easier than finding the exact solution: let us imagine a solution space where there is one MAP solution vector having probability one, and other solutions have probability zero. Now the problem of finding a solution which approximates the MAP solution within any accuracy of $\epsilon < 1$ is identical to the problem of finding the actual maximum probability solution.

Although the MAP problem seems to be intractable (in the worst case sense) for Bayesian networks in general, for *singly-connected* Bayesian network structures (networks with at most one path between any two variables, disregarding the directions of the connecting arcs), Bayesian reas-

[1] Strictly speaking, the NP-hardness results show that it is an NP-complete problem to decide whether there are any configurations $\vec{u}$ with the probability $\mathcal{P}\{\vec{u}\}$ higher than some given value p. As finding an MAP configuration gives directly the answer to this decision problem, the MAP problem is said to be NP-hard. (For a good introduction to the theory of complexity classes, see [38].)

oning (meaning here both the EVE and MAP tasks) can be done in polynomial time [96, 92]. Consequently, most existing systems for Bayesian reasoning first transform a given multi-connected BN structure to a singly-connected network, and then use the existing polynomial-time algorithms (see e.g. [4, 13]). This transformation can be done either explicitly by clustering several nodes together as in [76, 106] or [96, Ch.4.4.1], or implicitly by blocking multiply connected paths by conditioning a set of variables as in [96, Ch.4.4.2] (for a discussion of different transformation techniques, see [107]). Alternatively, clustering can also be done by introducing new *latent variables* which represent separate clusters of original variables [17, 88]. The latent variable model offers also 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 exponentially large conditional probability tables $\mathcal{B}_{\mathcal{P}}$, which causes exponential execution times with respect to the size of the original network. Nevertheless, it should be also noted that Bayesian reasoning process on a singly connected network can be realized on a massively parallel neural network structure [86, 87], which allows efficient implementations of Bayesian reasoning even for large networks, provided that suitable hardware is available.

In another approach to Bayesian reasoning the structure of the network is not changed, but the quantitative conditional probabilities are replaced by a finite set of qualitative measures of uncertainty [117]. It seems that Bayesian reasoning can in this case be done in polynomial time [29], but it is not yet clear what is the expressive power of this kind of qualitative Bayesian networks. For networks with less restricted conditional probabilities — networks with probabilities bounded by their *dependence value*, which is, intuitively speaking, a measure of how much the probabilities differ from uniform probabilities — the Bayesian reasoning tasks stay NP-hard [24].

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 is small with high probability). In *Monte Carlo methods* the goal is to find a stochastic random process 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. However, very little is known about the quality of the solutions produced by the Markov chain models, if a limited amount of time is available, but some analyses of the convergence speed can be found in [32, 1, 113, 101, 98, 22]. In particular, for Bayesian networks restricted by their dependence value (see above), there exists a stochastic polynomial-time algorithm which solves the EVE problem in the PAC sense [24]. Unfortunately, as the above mentioned convergence results deal mainly with the EVE problem, the analysis of the probabilistic solution for the MAP task is still very much an open problem. In Section 3.3, we go briefly through the basic characteristics of Markov chains and the corresponding Monte Carlo methods, and show how they can be used for constructing a simulated annealing process which solves the MAP problem approximately.

3.3 Simulated annealing

3.3.1 Markov chains

Let us consider a stochastic process $S(0), S(1), \ldots$ in our configuration space $\mathbf{\Omega}$, $S(t) \in \mathbf{\Omega}$. In Markov chains the probability of choosing the next state $S(t+1)$ depends only on the current state $S(t)$, and not (directly) on the previous states $S(t-1), S(t-2), \ldots, S(0)$:

Definition 3.3 (Markov chain)
A *Markov chain* is a sequence $\{S(t) \mid t = 0, 1, \ldots\}$, where

$$\begin{aligned} &P(S(t+1) = s \mid S(t) = s_t, S(t-1) = s_{t-1}, \ldots, S(0) = s_0) \\ &= P(S(t+1) = s \mid S(t) = s_t). \end{aligned}$$

A Markov chain is *finite* if it is defined on a finite set. In our model, this is clearly the case, as our configuration space $\mathbf{\Omega}$ is finite.

Let i be the state after t time steps, $S(t) = i$. A *transition probability* p_{ij} is the probability that the value of the next state $S(t+1)$ is j:

$$S(t+1) = \begin{cases} j & \text{with probability } p_{ij}. \\ i & \text{with probability } 1 - \sum_{j \neq i} p_{ij}. \end{cases}$$

We say that our stochastic process *moves from state i to state j with probability p_{ij}*. In the sequel, we are mainly concerned with *homogeneous* Markov chains, where the transition probabilities do not depend on time t, and consequently they can be written as a big transition probability matrix P:

$$P = \begin{bmatrix} P_{11} & \ldots & P_{1j} & \ldots & P_{1M} \\ \vdots & & \vdots & & \vdots \\ P_{i1} & \ldots & P_{ij} & \ldots & P_{iM} \\ \vdots & & \vdots & & \vdots \\ P_{M1} & \ldots & P_{Mj} & \ldots & P_{MM} \end{bmatrix},$$

where M is the size of the configuration space $\mathbf{\Omega}$. The transition probability matrix is a *stochastic* matrix, i.e. the sum of all the probabilities on each row is one.

So far we have only been concerned with the one-step transition probabilities. The following theorem states how we can derive the multi-step transition probabilities from the one-step probabilities using a single matrix operation:

Theorem 3.5 The probabilities of moving from state i to state j in n steps, denoted by $P_{ij}^{(n)}$, are the elements of the matrix P^n:

$$P_{ij}^{(n)} = (P^n)_{ij}.$$

Proof: See [69, p. 58]. □

Definition 3.4 A Markov chain with transition matrix P is *irreducible*, if

$$\forall i, j \; \exists n \geq 1 : P_{ij}^{(n)} > 0.$$

Definition 3.5 The *period* of a state i is the greatest common divisor of all integers $n \geq 1$ for which $P_{ii}^{(n)} > 0$. If the period is 1, then the state is said to be *aperiodic*. A Markov chain is aperiodic, if all of its states are aperiodic.

From the theory of Markov chains, it is known that under certain conditions, the probability of finding the system in state j after n iterations convergences to a certain probability π_j as n approaches infinity:

$$\pi_j = \lim_{n\to\infty} P_{ij}^{(n)} = \lim_{t\to\infty} P(S(t) = j | S(0) = i), \text{ for all } i, j = 1, \ldots, M.$$

The convergence is independent of the initial state $S(0)$. The probabilities $\pi_1, \ldots, \pi_M$ form a limiting probability distribution π called the *stationary probability distribution*. The following theorem shows how the stationary probability distribution can be calculated:

Theorem 3.6 Let P be the transition matrix of a finite, irreducible and aperiodic homogeneous Markov chain. Then there exists a stationary distribution π which is uniquely determined by the set of equations

$$\pi_j = \sum_i \pi_i P_{ij}, \quad j = 1, \ldots, M,$$

under the constraints $\pi_i \geq 0, \sum_i \pi_i = 1$.

Proof: See [69, p. 85]. □

3.3.2 Markov chain Monte Carlo methods

Markov chain Monte Carlo (MCMC) methods are stochastic algorithms that are based on Markov chain processes (for a survey of using MCMC methods in the Bayesian reasoning framework, see [91]). As pointed out in the previous section, under certain conditions, Markov chains will converge to a unique stationary distribution. In the MCMC framework, this kind of a converging Markov chain process is usually produced by forming transition probabilities P_{ij} consisting of two parts: a *generation probability* G_{ij} and an *acceptance probability* A_{ij}. The generation probability G_{ij} represents the probability that we consider moving from state i to state j, and the acceptance probability A_{ij} expresses the probability that we accept this move. Hence the transition probability is the product of these two probabilities:

$$P_{ij} = \begin{cases} G_{ij}A_{ij} & \text{,if } i \neq j, \\ 1 - \sum_{j \neq i} G_{ij}A_{ij} & \text{,if } i = j. \end{cases}$$

A generic MCMC algorithm is defined as follows:

Algorithm 3.1
Markov chain Monte Carlo (MCMC)

- ▷ $S(0) :=$ RandomState($\vec{u}_1,\ldots,\vec{u}_M$);
- ▷ for t := 1 to ∞ do
  - ▷ $\vec{u}_i := S(t-1)$;
  - ▷ /* generate a candidate for the next state */
    $\hat{S}(t) := \vec{u}_j$ with probability G_{ij};
  - ▷ if (RANDOM(0,1) $< A_{ij}$)
    then $S(t) := \hat{S}(t)$; /* accept */
    else $S(t) := S(t-1)$; /* reject */

Under certain conditions, the probability of finding the MCMC process (Algorithm 3.1) in state $\vec{u}_i$ converges to $\mathcal{P}\{\vec{u}_i\}$. Sufficient conditions for this kind of behavior are usually given as the following requirements for the matrices G and A :

Theorem 3.7 Let us define a MCMC process with the generation and acceptance matrices fulfilling the following requirements:

$$\begin{array}{lll}
(G1) & \forall i,j \in \{1,\ldots,M\}: & \exists p \geq 1, \exists l_0, l_1, \ldots, l_p \in \{1,\ldots,M\} \\
 & & (\text{where } l_0 = i, l_p = j) \\
 & & \text{and } G_{l_k l_{k+1}} > 0, k = 0,1,\ldots,p-1. \\
(G2) & \forall i,j \in \{1,\ldots,M\}: & G_{ij} = G_{ji} \\
(A1) & \forall i,j \in \{1,\ldots,M\}: & A_{ij} = 1, \text{if } \mathcal{P}\{\vec{u}_j\} \geq \mathcal{P}\{\vec{u}_i\} \\
 & & A_{ij} \in (0,1), \text{if } \mathcal{P}\{\vec{u}_j\} < \mathcal{P}\{\vec{u}_i\} \\
(A2) & \forall i,j,k \in \{1,\ldots,M\} & \text{with } \mathcal{P}\{\vec{u}_i\} \geq \mathcal{P}\{\vec{u}_j\} \geq \mathcal{P}\{\vec{u}_k\} : A_{ik} = A_{ij}A_{jk}.
\end{array}$$

The limiting distribution of the resulting stochastic process is $\mathcal{P}$.

Proof: See [1, p.42]. □

For the acceptance probability matrix A, the original choice suggested by Metropolis et al. in [80] was the following:

$$A_{ij} = \begin{cases} 1 & , \text{ if } \frac{\mathcal{P}\{\vec{u}_j\}}{\mathcal{P}\{\vec{u}_i\}} \geq 1, \\ \frac{\mathcal{P}\{\vec{u}_j\}}{\mathcal{P}\{\vec{u}_i\}} & , \text{ if } \frac{\mathcal{P}\{\vec{u}_j\}}{\mathcal{P}\{\vec{u}_i\}} < 1. \end{cases} \tag{3.2}$$

This is still perhaps the most commonly used method in MCMC applications.

The Metropolis method clearly fulfills the conditions (A1) and (A2) in Theorem 3.7. In an alternative model, *Barker's method* [8], these requirements are not met:

$$A_{ij} = \frac{\mathcal{P}\{\vec{u}_j\}}{\mathcal{P}\{\vec{u}_j\} + \mathcal{P}\{\vec{u}_i\}} = \frac{1}{1 + \frac{\mathcal{P}\{\vec{u}_i\}}{\mathcal{P}\{\vec{u}_j\}}}. \tag{3.3}$$

Nevertheless, existence of a unique limiting distribution can still be proved:

Theorem 3.8 Let $\Phi = \{S(t); t = 0,1,\ldots\}$ be a stochastic simulation process with the generation probabilities fulfilling the conditions (G1) and (G2) of Theorem 3.7, and let the acceptance probabilities be computed according to the formula (3.3), with all the state probabilities $\mathcal{P}\{\vec{u}_i\}$ assumed to be positive. Then the unique limiting distribution of Φ is $\mathcal{P}$.

Proof: As $\mathcal{P}\{\vec{u}_i\} > 0$ for all i (and assuming that condition (G1) is met), the resulting Markov chain is irreducible and aperiodic (see [1, p.39]). Moreover, using assumption (G2) we get

$$\begin{aligned}
\sum_i \mathcal{P}\{\vec{u}_i\}P_{ij} &= \sum_i \mathcal{P}\{\vec{u}_i\}G_{ij}\frac{\mathcal{P}\{\vec{u}_j\}}{\mathcal{P}\{\vec{u}_j\}+\mathcal{P}\{\vec{u}_i\}} \\
&= \sum_i \mathcal{P}\{\vec{u}_j\}G_{ij}\frac{\mathcal{P}\{\vec{u}_i\}}{\mathcal{P}\{\vec{u}_j\}+\mathcal{P}\{\vec{u}_i\}} \\
&= \sum_i \mathcal{P}\{\vec{u}_j\}G_{ji}\frac{\mathcal{P}\{\vec{u}_i\}}{\mathcal{P}\{\vec{u}_j\}+\mathcal{P}\{\vec{u}_i\}} \\
&= \sum_i \mathcal{P}\{\vec{u}_j\}P_{ji} \\
&= \mathcal{P}\{\vec{u}_j\}\sum_i P_{ji} \\
&= \mathcal{P}\{\vec{u}_j\}.
\end{aligned}$$

According to Theorem 3.6, $\mathcal{P}$ is now the stationary distribution of the Markov chain behind the Barker's method. □

In practice, the Metropolis algorithm is sometimes preferred over Barker's method, as in the case of equally probable states, Barker's method changes the state with probability 1/2, whereas Metropolis' method changes the state with probability 1, thus offering possibly a better sampling of the states. However, it should be noted that there is no theoretical preference for either of these models, but they both lead to the same stationary distribution.

3.3.3 Gibbs sampling

The simplest alternative for the generation probability matrix G fulfilling the requirements (G1) and (G2) of Theorem 3.7 is to use the uniform distribution,

$$G_{ij} = \frac{1}{M}, \text{ for all } i, j \in \{1, \dots, M\}. \tag{3.4}$$

However, when the uniform generation distribution is used, all the variables have equal probability of changing their value, which means that when a good solution is found, it can easily be lost during the simulation process[2]. In practice, on the other hand, it has been empirically observed that updating only one variable or a small group of variables at a time gives better results, as the

[2]What is more, it is easy to see that this kind of an MCMC sampling can never work better than simple uniform random sampling, regardless of the acceptance probability matrix A.

resulting stochastic process is then a kind of a local search algorithm, preserving (probably) most of the good solutions found. This type of a MCMC process is usually called *Gibbs sampling.*

When only one randomly chosen variable is updated at a time, the generation probabilities of the corresponding Gibbs sampler can be written as

$$G_{ij} = \begin{cases} \frac{1}{|\mathbf{\Omega}(i)|} & \text{, if } \vec{u}_j \in \mathbf{\Omega}(i), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.5}$$

where $\mathbf{\Omega}(i) \subset \mathbf{\Omega}$ denotes the set of states that can be obtained from state $\vec{u}_i$ by changing the value of one variable, and $|\mathbf{\Omega}(i)|$ denotes the number of states in $\mathbf{\Omega}(i)$. This is the most usual form of Gibbs sampling. In the sequel, we are mainly concerned with the Gibbs sampling type of MCMC processes, and their variations.

Let us consider a Gibbs sampler based on acceptance probabilities of the form (3.2) or (3.3), generation probabilities of the form (3.5), and let U_k be the variable to be updated at some state $S(t) = \vec{u}_i$ of the stochastic process by changing its value from u_k to u'_k. The actual decision of whether to move to a new state $S(t+1) = \vec{u}_j$ or not is based on the ratio $\mathcal{P}\{\vec{u}_i\}/\mathcal{P}\{\vec{u}_j\}$. As this ratio can also be represented as

$$\begin{aligned}
\frac{\mathcal{P}\{\vec{u}_i\}}{\mathcal{P}\{\vec{u}_j\}} &= \frac{\mathcal{P}\{U_1 = u_1, \ldots, U_k = u_k, \ldots, U_N = u_N\}}{\mathcal{P}\{U_1 = u_1, \ldots, U_k = u'_k, \ldots, U_N = u_N\}} \\
&= \frac{\mathcal{P}\{U_k = u_k, \bigwedge_{i \neq k} U_i = u_i\}}{\mathcal{P}\{U_k = u'_k, \bigwedge_{i \neq k} U_i = u_i\}} \\
&= \frac{\mathcal{P}\{U_k = u_k \mid \bigwedge_{i \neq k} U_i = u_i\}\mathcal{P}\{\bigwedge_{i \neq k} U_i = u_i\}}{\mathcal{P}\{U_k = u'_k \mid \bigwedge_{i \neq k} U_i = u_i\}\mathcal{P}\{\bigwedge_{i \neq k} U_i = u_i\}} \\
&= \frac{\mathcal{P}\{U_k = u_k \mid \bigwedge_{i \neq k} U_i = u_i\}}{\mathcal{P}\{U_k = u'_k \mid \bigwedge_{i \neq k} U_i = u_i\}},
\end{aligned}$$

Gibbs sampling can also be regarded as a stochastic local search which samples from the "local distribution" $\mathcal{P}\{U_k = u \mid \bigwedge_{i \neq k} U_i = u_i\}$. In this respect, Gibbs sampling can be regarded as a stochastic version of a local search algorithm belonging to the very general family of *expectation maximization (EM)* [26] (or, as they are also called, *alternating minimization* [21]) algorithms.

A Gibbs sampling process can be made much faster by parallelizing the state generation-acceptance process. This can be done easily if some variables are found to be independent of each other: independent variables can then be updated at the same time without losing the "local search"–property of the process. A certain class of neural net algorithms can be regarded as massively parallel implementations of Barker's method, as we shall later see in Chapter 4.

It is easy to see that a Gibbs sampling generation probability matrix of the form (3.5) fulfills the conditions (G1) and (G2) of theorem (3.7), and hence the limiting distribution of a Gibbs sampler is $\mathcal{P}$ ($\mathcal{P}$ being the probability measure used for computing the acceptance probabilities). Consequently, the frequency of a variable U_i being in state u_i converges (in the limit) to the probability $\mathcal{P}\{U_i = u_i \mid \vec{E} = \vec{e}\}$, and hence the Gibbs sampler can be used for approximating the EVE solution. However, to be able to solve the MAP problem, we need to change the Gibbs sampling process so that it converges to a stable state, and what is more, we need to show that this stable state is the desired MAP state. In the next section, we present a technique for this purpose.

3.3.4 Gibbs distributions and simulated annealing

The stochastic simulation algorithm presented in the previous section was first introduced by Metropolis et al. [80] in the context of condensed matter physics. It can be shown that the probability of a heated solid being in state $\vec{u}$ at a given temperature $\tilde{T}$ obeys the following *Gibbs distribution*:

Definition 3.6 A *Gibbs distribution* on a configuration space $\mathbf{\Omega}$ is a probability measure of the form

$$P_{\tilde{T}}\{\vec{u}\} = \frac{1}{Z_{\tilde{T}}} e^{(-E(\vec{u})/k_B\tilde{T})},$$

where $Z_{\tilde{T}}$ is a normalizing constant called the partition function,

$$Z_{\tilde{T}} = \sum_{\vec{u}\in\mathbf{\Omega}} e^{(-E(\vec{u})/k_B\tilde{T})},$$

$E(\vec{u})$ is the energy of the system, and k_B is the Boltzmann constant.

As k_B is a constant, it is usually omitted from the formulas, and a single term $T = k_B\tilde{T}$ is used instead to denote the scaled temperature.

As mentioned earlier, originally the acceptance probabilities were based on the Metropolis formula (3.2), resulting in the following acceptance probabilities:

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

where E is the energy function of the Gibbs distribution, and T is the temperature. When applying Barker's method (3.3) with the Gibbs distribution, the resulting acceptance probabilities become of the form

$$\begin{aligned} A_{ij}(T) &= \frac{1}{1 + \frac{\exp(-E(\vec{u}_i)/T)}{\exp(-E(\vec{u}_j)/T)}} \\ &= \frac{1}{1 + \exp((E(\vec{u}_j) - E(\vec{u}_i))/T)} \end{aligned} \tag{3.6}$$

As we shall see in Section 4, this latter form of Gibbs sampling is equivalent to the updating rule of certain neural network models. It is also important to notice that Gibbs sampling uses only differences of state energies, not the actual probabilities. In particular, the method does not need the value of the partition function Z_T of the Gibbs distribution, which is usually not feasibly computable.

The limiting distribution of a Gibbs sampling process is in the context of physics often called the *thermal equilibrium. Annealing* is a physical process where a solid is first heated up, and the temperature is then slowly decreased to zero. Empirically it has been known for a long time that if the temperature is decreased slowly enough, the particles of the matter are arranged in a highly structured manner, producing a stable low-energy *ground state* of the matter. The following theorem proves that Gibbs sampling combined with annealing will find one of the minimal energy states of the physical system:

Theorem 3.9 Let $\mathbf{\Omega}_{opt}$ denote the set of maximal probability (minimal energy) configurations with respect to the Gibbs distribution in Definition 3.6, and let $S(0), S(1), \ldots$ be a stochastic sampling process fulfilling the requirements in Theorem 3.7, with the additional temperature parameter added to the acceptance probabilities as shown in (3.6) above. As the temperature of the Gibbs sampler converges to zero, the probability of finding the annealing process in state $\vec{u}_i$ converges to the uniform distribution in the set $\mathbf{\Omega}_{opt}$:

$$\lim_{T\downarrow 0} \lim_{t\to\infty} P_T(S(t) = \vec{u}_i) = \begin{cases} 1/|\mathbf{\Omega}_{opt}| & , \text{ if } \vec{u}_i \in \mathbf{\Omega}_{opt} \\ 0 & \text{otherwise,} \end{cases}$$

Proof: See [1, p.18]. □

The sampling/annealing procedure presented above applies naturally also outside the actual statistical physics environment, and hence it is usually referred to as *simulated annealing (SA)*. As SA in principle finds the global minimum of any energy function, the algorithm is applicable to combinatorial optimization problems in general. This possibility was brought to general attention of the optimization theory community by Kirkpatrick et al. in [71], and SA has since been widely applied for many different optimization problems (see the references in [1]). Simulated annealing is also widely used in image processing, in particular due to the pioneering theoretical work done by Geman & Geman (see [40]). In Section 3.4 we show how to represent a Bayesian network probability distribution in the Gibbs distribution form, which allows us to use simulated annealing for finding MAP configurations on Bayesian networks.

A generic simulated annealing algorithm can be defined as follows:

Algorithm 3.2
Simulated annealing (SA)

- ▷ $S(0) := \text{RandomState}(\vec{u}_1, \ldots, \vec{u}_M)$;
- ▷ $T := \text{InitTemp}()$;
- ▷ Repeat until convergence criterion is satisfied
  - ▷ Repeat until equilibrium criterion is satisfied
    - ▷ $i := S(t-1)$;
    - ▷ /* generate a candidate for the next state */
      $\hat{S}(t) := j$ with probability G_{ij};
    - ▷ if (Random(0,1) $< A_{ij}(T)$)
      then $S(t) := \hat{S}(t)$; /* accept */
      else $S(t) := S(t-1)$; /* reject */
  - ▷ $T := \text{NewTemperature}()$;

Theorem 3.9 states that the annealing process will, with probability one, find one of the MAP configurations of the given Gibbs distribution *in the limit*:

$$\lim_{T\downarrow 0} \lim_{t\to\infty} P_T(S(t) \in \mathbf{\Omega}_{opt}) = 1.$$

However, in practice we are not able to run the algorithm infinitely long at infinitely many temperatures, as the theorem assumes. We define a *cooling schedule* as a set of rules that determine how the temperature is decreased during a finite-time simulation:

Definition 3.7 A *cooling schedule* specifies

- the initial value of the temperature,

- an *equilibrium criterion*, which defines the number of iterations to be performed at each temperature,
- a decrement function, which defines how fast the temperature decreases, and
- a *convergence criterion*, which defines the final value of the temperature.

The following theorem of Geman & Geman shows that if the temperature is lowered sufficiently slowly, then the convergence of the simulated annealing is guaranteed in a finite number of steps.

Theorem 3.10 Let Γ denote the product $N\Delta$, where

$$\Delta = \max_{\vec{u}\in\mathbf{\Omega}} E(\vec{u}) - \min_{\vec{u}\in\mathbf{\Omega}} E(\vec{u}),$$

and N is the number of random variables. At time t, let a new state $S(t)$ be generated by considering changing the value of only one variable, and let $I(t)$ denote the index of the variable under consideration. Assume that there exists an integer $\tau \geq N$ such that for every $t = 0, 1, \ldots$ we have

$$\{1, \ldots, N\} \subseteq \{I(t+1), \ldots, I(t+\tau)\},$$

and let $T(t)$ be a decreasing sequence of temperatures for which

1. $T(t) \rightarrow 0$ as $t \rightarrow \infty$.
2. $T(t) \geq \Gamma / log(t+2)$, for all $t = 0, 1, \ldots$

Now the convergence result of Theorem 3.9 applies.
Proof: See [40].

From the implementational point of view, Theorem 3.10 has two major drawbacks. First of all, although the number of steps required in the theorem is not infinite, it is still exponential with respect to time t and hence impractical for most applications. Secondly, in many cases good estimates for the value of the parameter Δ are not available, making the determination of a suitable initial temperature very difficult. It is also important to notice that this theorem states nothing about the convergence of simulated annealing if the cooling schedule is faster than what is required in Theorem (3.10). However, with certain additional constraints to the problem it can be shown that exponential time is not only sufficient, but also necessary [44]. In the light of the complexity of the MAP problem (see Section 3.2), it seems probable that a similar kind of result could be proven for the general case also, although this has not been done. In contrast to the negative theoretical results, it has been empirically observed that in many cases good quality solutions can be found quite reliably with polynomial time cooling schedules [1]. Nevertheless, the task of finding a suitable cooling schedule seems to be a very difficult problem.

The condition concerning the constant τ in Theorem 3.10 is used only to ensure that the Gibbs sampling process is fair, i.e. no variables are going to be ignored during the sequential variable updating process. Actually, as noted later by several researches (see the list of references in [1, Ch. 3.4.]), the assumption of changing only one variable at a time is not necessary at all. This fact is very important when considering parallel implementations of simulated annealing, as we shall see in Chapter 4.

3.4 Simulated annealing for Bayesian networks

Gibbs sampling and simulated annealing are an appealing method for solving MAP problems approximatively, offering a possibility to avoid the exponential time required for solving the problems exactly. To be able to use SA in our Bayesian network framework, we need a method for representing the Bayesian network probability distribution $\mathcal{P}$ in the Gibbs distribution form (Definition 3.6). In the following, we show how this can be done by exploiting the equivalence between the graphical *Markov Random Field* models and Gibbs distributions.

3.4.1 Markov random fields

As before, let $\boldsymbol{U}$ denote a variable base of N variables $U_1, \ldots, U_N$, and let $\mathcal{G}$ denote a binary neighborhood relation on $\boldsymbol{U}$, consisting of pairs (U_i, U_j). As with Bayesian networks, we can view this relation graphically by drawing a network where for each variable U_i there is a corresponding node S_i in the network, and two nodes S_i and S_j are connected if and only if $(U_i, U_j) \in \mathcal{G}$. In this case the neighborhood relation $\mathcal{G}$ is symmetric, resulting in an undirected graph. As with Bayesian networks, we shall henceforth forget about the nodes S_i, and treat the random variables U_i as if they were actually nodes of a graph.

For each variable U_i, we define a set of neighbors $\mathcal{G}_i$:

$$\mathcal{G}_i = \{U_j \mid (U_i, U_j) \in \mathcal{G}\}.$$

Using the concept of neighborhood we can now give the definition for a *Markov Random Field (MRF)* [27, 11, 70, 40]:

Definition 3.8 (Markov Random Field) A family of random variables $\boldsymbol{U}$ is a Markov Random Field with respect to a relation $\mathcal{G}$ and a probability distribution $\mathcal{P}$, if

1. $\mathcal{P}\{\vec{U} = \vec{u}\} > 0$ for all $\vec{u} \in \boldsymbol{\Omega}$.
2. $\mathcal{P}\{U_i = u_i \mid \bigwedge_{j \neq i} U_j = u_j\} = \mathcal{P}\{U_i = u_i \mid \bigwedge_{j \in \mathcal{G}_i} U_j = u_j\}$ for all the variables $U_i \in \boldsymbol{U}$, and for all the configurations $\vec{u} \in \boldsymbol{\Omega}$.

A subset $\boldsymbol{C} \subseteq \boldsymbol{U}$ is called a *clique*, if all the variables in $\boldsymbol{C}$ are neighbors to each other. Let $\boldsymbol{\mathcal{C}}$ denote a set of cliques in $\boldsymbol{U}$, and let $V_{\boldsymbol{C}}$ denote a function on $\boldsymbol{\Omega}$ that depends only on the values of the variables in clique $\boldsymbol{C} \in \boldsymbol{\mathcal{C}}$, and maps each state vector to a real number, according to the value configuration on the clique $\boldsymbol{C}$. The value of the function $V_{\boldsymbol{C}}(\vec{u})$ is called the *clique potential*

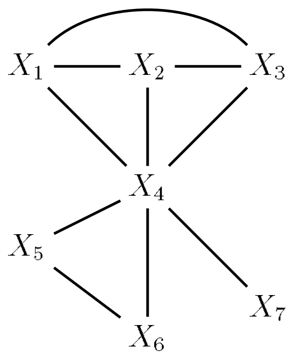


Figure 3.3: A simple Markov random field with three maximal cliques, $\{X_1, X_2, X_3, X_4\}$, $\{X_4, X_5, X_6\}$ and $\{X_4, X_7\}$.

of a clique $\boldsymbol{C}$. The following theorem proves an important relation between Markov random fields and Gibbs distributions:

Theorem 3.11 Let $\mathcal{G}$ denote a neighborhood relation on $\boldsymbol{U}$. Then $\boldsymbol{U}$ is a MRF with respect to $\mathcal{G}$ and a probability distribution $\mathcal{P}$ if and only if $\mathcal{P}$ can be represented as a Gibbs distribution of the form

$$\mathcal{P}\{\vec{U} = \vec{u}\} = \frac{1}{Z} e^{V(\vec{u})/T},$$

where the *potential function* V is the sum of the values of all the clique potentials in the network:

$$V(\vec{u}) = \sum_{C} V_C(\vec{u}),$$

and Z_T is a normalizing constant.
Proof: See [40]. □

Consequently, any Gibbs distribution can be defined by using a graphical Markov random field representation, and correspondingly, for each MRF there exists a corresponding Gibbs distribution (and a corresponding Gibbs sampler).

In practice, the set of the cliques of a graph is usually taken to be the set of the the maximal cliques of the graph. However, sometimes it can be more convenient to define a Gibbs distribution by using a slightly different set of cliques, as we shall see in the next section.

For notational convenience, we prefer to use in the sequel the Gibbs distribution formulation given above with a negative potential function function

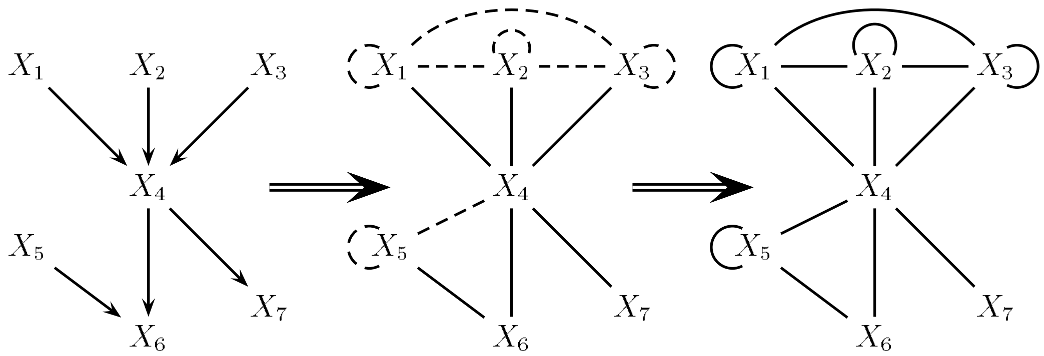


Figure 3.4: Moralization of a Bayesian network.

V, instead of the positive energy function E used in Definition 3.6 (naturally, $V(\vec{u}) = -E(\vec{u})$, so the difference is only syntactical). In this case, Barker's acceptance function (3.3) becomes

$$\begin{aligned} A_{ij}(T) &= \frac{1}{1 + \frac{\exp(V(\vec{u}_i)/T)}{\exp(V(\vec{u}_j)/T)}} \\ &= \frac{1}{1 + \exp((V(\vec{u}_i) - V(\vec{u}_j))/T)} \\ &= \frac{1}{1 + \exp(-(V(\vec{u}_j) - V(\vec{u}_i))/T)} \qquad (3.7) \end{aligned}$$

In Chapter 5 we see that this probability is equal to the node updating probability of certain stochastic neural network models.

3.4.2 Mapping Bayesian networks to Markov random fields

Let $\mathcal{B}$ be a Bayesian network representing a probability distribution $\mathcal{P}$. As noted earlier by several authors [61, 75, 76, 115], we can construct for any given Bayesian network a corresponding MRF having the same probability distribution, by using a transformation called *moralization* of a Bayesian network. In the moralizing process, a given Bayesian network is transformed to a MRF by first making all the existing arcs undirected, and then adding new undirected arcs between any two variables that have a common child in the Bayesian network ("moralizing" the relationship). For each variable U_i without any predecessors in the Bayesian network, there is a one node clique $\{U_i\}$ (see Figure 3.4).

In the resulting MRF, we distinguish N cliques, one for each of the variables $U_1, \ldots, U_N$, so the corresponding probability distribution is a Gibbs distribution of the form

$$\mathcal{P}_T\{\vec{U} = \vec{u}\} = \frac{1}{Z_T} e^{\sum_{i=1}^{N} V_i(\vec{u})/T}. \tag{3.8}$$

The following theorem shows how to choose the clique potentials of the moralized MRF so that the corresponding probability distribution is equal to the original Bayesian network distribution $\mathcal{P}$:

Theorem 3.12 Let $\mathcal{B}$ be a Bayesian network with a probability distribution $\mathcal{P}$, and let $\mathcal{M}$ denote the corresponding MRF with a Gibbs probability distribution $\mathcal{Q}$ of the form (3.8). If we set the potential of a clique C_i, corresponding to a variable U_i, to

$$V_i(\vec{u}) = \ln \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{\boldsymbol{U_i}}} U_j = u_j\} + K_i, \tag{3.9}$$

where the constants K_i fulfill the condition

$$\prod_i e^{K_i} = Z_T, \tag{3.10}$$

Z_T being the partition function of the Gibbs distribution $\mathcal{Q}$, then, at temperature 1, $\mathcal{Q} = \mathcal{P}$.

Proof: According to Theorem 3.4, the probability distribution $\mathcal{P}$ can be represented as a product of conditional probabilities, one for each variable U_i:

$$\mathcal{P}\{\vec{u}\} = \prod_i \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{\boldsymbol{U_i}}} U_j = u_j\}.$$

On the other hand, from (3.9) and (3.10) it follows that, at temperature 1,

$$\mathcal{Q}\{\vec{u}\} = \frac{1}{Z} e^{\sum_i V_i(\vec{u})} = \frac{1}{Z} \prod_i \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{\boldsymbol{U_i}}} U_j = u_j\} \prod_i e^{K_i} = \mathcal{P}\{\vec{u}\}.\Box$$

It should be noted, that as in Gibbs sampling we are only concerned with the proportions $\mathcal{P}\{\vec{u}_i\}/\mathcal{P}\{\vec{u}_j\} = \exp(V(\vec{u}_i) - V(\vec{u}_j))$, the values of constants K_i are actually irrelevant.

If we wish to sample a Bayesian network probability distribution $\mathcal{P}$ for EVE problem tasks, we can map the Bayesian network to a MRF and use the corresponding Gibbs distribution for constructing a Gibbs sampling process at temperature 1. On the other hand, if we wish to solve a given MAP problem, we need to run the Gibbs sampler at different temperatures T, with T decreasing towards zero. In Section 6.1 we present one possible realization of this type of a simulated annealing process. In Chapter 5, we present how this type of a sequential process can be replaced by a massively parallel neural network updating process, in the sense that the neural network updating process provably converges to the same final state as the sequential SA process.

Chapter 4

Solving the MAP problem by stochastic neural networks

> "*A neural network is a parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and carry out localized information processing operations) interconnected via unidirectional signal channels called connections. Each processing element has a single output connection that branches ("fans out") into as many collateral connections as desired; each carries the same signal — the processing element output signal. The processing element output signal can be of any mathematical type desired. The information that goes on within each processing element can be defined arbitrarily with the restriction that it must be completely local; that is, it must depend only on the current values of the input signals arriving at the processing element via impinging connections and on values stored in the processing element's local memory.*" – Robert Hecht-Nielsen [47].

The processing elements (units, nodes or "neurons") of a neural network are usually based on a simple artificial neuron model, inspired originally [79] by the structure of a real biological neuron. In this model, the local memory of a node S_i consists of two real-valued parameters: a constant parameter θ_i called the *threshold* or *bias*, and a variable parameter s_i, the *activity value* or the *state* of the node. In addition, each connection (i, j) between two nodes S_i and S_j is provided with a real-valued parameter w_{ij} called the *weight* of the connection.

As the output signal, each node sends its state to other nodes through the connecting arcs. The total net input to a node is the weighted sum of the

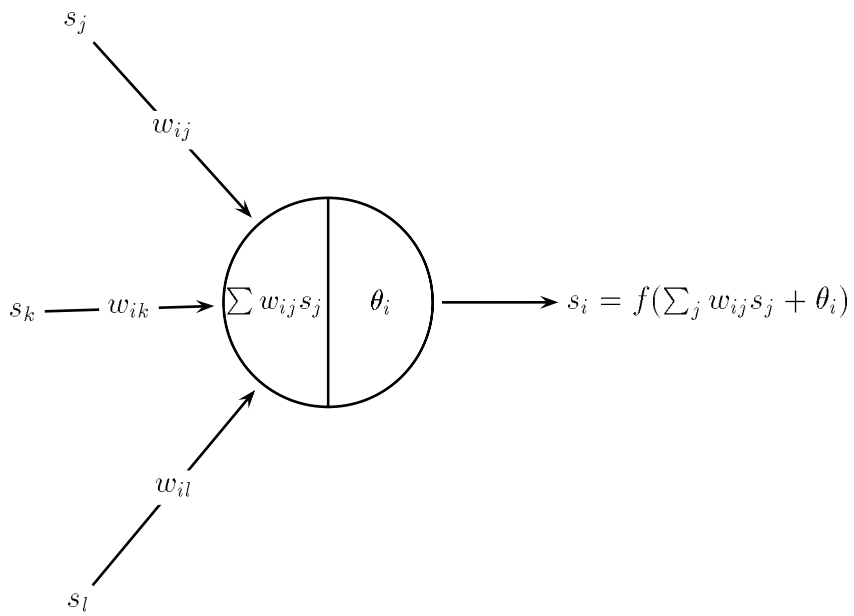


Figure 4.1: An artificial neuron.

incoming output signals of other nodes, plus the threshold. The new state of a node is computed by using an *activation function* or *transfer function* f, with the net input as the parameter (see Figure 4.1).

The most commonly used form for an activation function is the *sigmoid function*,

$$f(x) = \frac{1}{1 + e^{-\beta x}},$$

where β is some positive constant (see Figure 4.2). Note that the sigmoid function becomes in the limit a two-value threshold function as β approaches infinity. One of the main reasons for using the sigmoid function is the fact that it has a very simple derivative, $f'(x) = f(x)(1 - f(x))$, and besides, efficient implementations of sigmoid function elements have been developed for analogical and digital circuits [47, Ch. 8],[51, Ch. 3].

The interconnected nodes form a network structure, a neural network. Neural network architectures can be divided into two main categories: feedforward models and feedback models. In feedforward neural networks the network can be partitioned into hierarchical layers of nodes, where each node on one layer is connected only to nodes on layers situated higher in the layer hierarchy. The computational model for feedforward neural networks is a one-pass feedforward propagation of signals, starting from the first layer, and ending at the last layer. In feedback neural networks, on the other hand, the connections between the nodes can form loops, and the computational mechanism is an iterative relaxation process. For a good introduction to neural

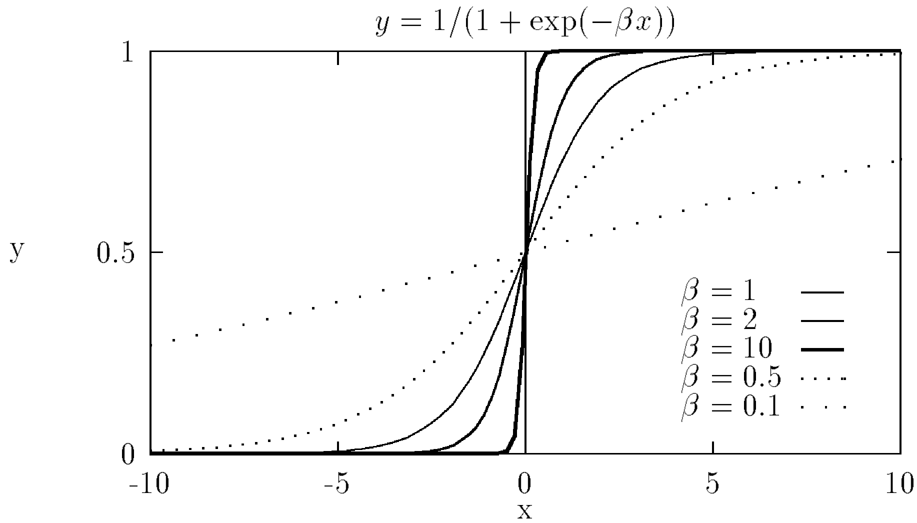


Figure 4.2: The sigmoid function.

computing, see e.g. the collections [5, 6, 103, 78], or the books [46, 54, 47]. In the following, we are mainly concerned with one of the most common of the feedback models, the *Boltzmann machine* [57, 58] neural network architecture, and its variants.

4.1 Boltzmann machines

A *Boltzmann machine (BM)* [57, 58] is a neural network consisting of a set of binary nodes $\{S_1, \ldots, S_n\}$, where state s_i of node S_i is either 1 ("on"), or 0 ("off"). Let C denote the set of all connections (i,j) in a network. In BM models, all the connections are symmetric, i.e. if $(i,j) \in C$ then also $(j,i) \in C$, and $w_{ij} = w_{ji}$.

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

$$C(\vec{s}) = \sum_{j=1}^{n} \sum_{i=j}^{n} w_{ij} s_i s_j, \tag{4.1}$$

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.

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

$$P(s_i = 1) = \frac{1}{1 + e^{-\sum_j w_{ij} s_j / T}}, \tag{4.2}$$

where T is a real-valued parameter called temperature, which decreases with time 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.

Theorem 4.1 If the nodes of a Boltzmann machine are updated by changing one randomly chosen unit at a time, then the network will converge to a maximal consensus state almost surely.

Proof: Let $\vec{s}$ denote the current state vector, and let S_k be the node to be updated. The new state vector $\vec{s}'$ is obtained by flipping the state s_k of node S_k while the other nodes remain unchanged:

$$s'_i = \begin{cases} s_i & , \text{if } i \neq k, \\ 1 - s_i & , \text{if } i = k. \end{cases}$$

The difference in consensus between the states $\vec{s'}$ and $\vec{s}$ is now

$$C(\vec{s'}) - C(\vec{s}) = (\sum_{j=1}^{n}\sum_{i=j}^{n} w_{ij}s'_i s'_j) - (\sum_{j=1}^{n}\sum_{i=j}^{n} w_{ij}s_i s_j)$$

$$= (\sum_{j=1}^{n}\sum_{i=j,i\neq k}^{n} w_{ij}s'_i s'_j + \sum_{j=1}^{n} w_{kj}s'_k s'_j) - (\sum_{j=1}^{n}\sum_{i=j,i\neq k}^{n} w_{ij}s_i s_j + \sum_{j=1}^{n} w_{kj}s_k s_j)$$

$$= \sum_{j=1}^{n} w_{kj}s'_k s_j - \sum_{j=1}^{n} w_{kj}s_k s_j$$

$$= (s'_k - s_k)\sum_{j=1}^{n} w_{kj}s_j$$

In the BM updating scheme, if $s_k = 0$ (and hence $s'_k = 1$), then it follows that $C(\vec{s'}) - C(\vec{s}) = \sum_{j=1}^{n} w_{kj}s_j$, and hence the probability of accepting the new state $\vec{s'}$ is

$$\begin{aligned} P(\vec{s'}) = P(s_k = 1) &= \frac{1}{1 + e^{-\sum_{j=1}^{n} w_{kj}s_j/T}} \\ &= \frac{1}{1 + e^{-(C(\vec{s'}) - C(\vec{s}))/T}} \end{aligned}$$

On the other hand, if $s_k = 1$ (and hence $s'_k = 0$), then it follows that $C(\vec{s'}) - C(\vec{s}) = -\sum_{j=1}^{n} w_{kj}s_j$, and the probability of accepting the new state $\vec{s'}$ is

$$\begin{aligned} P(\vec{s'}) = 1 - P(s_k = 1) &= 1 - \frac{1}{1 + e^{-\sum_{j=1}^{n} w_{kj}s_j/T}} \\ &= 1 - \frac{1}{1 + e^{(C(\vec{s'}) - C(\vec{s}))/T}} \\ &= \frac{1}{1 + e^{-(C(\vec{s'}) - C(\vec{s}))/T}} \end{aligned}$$

Consequently, in both cases the updating probability is identical to that proposed by Barker (3.7), and hence the BM updating process can be regarded as a Gibbs sampling-simulated annealing process, which according to Theorem 3.9 converges almost surely to a state which maximizes the following Gibbs distribution:

$$P\{\vec{s}\} = \frac{1}{Z}e^{C(\vec{s})},$$

where Z is a normalizing constant. As this distribution has the same maximum points as the consensus function C, the BM updating process converges to a maximal consensus state. □

It follows that in principle any BM updating scheme could be seen as a massively parallel implementation of the general Gibbs sampling algorithm with respect to the consensus function (4.1) and acceptance probability (4.2), provided that the generating probability matrix used fulfills the requirements (G1) and (G2) of Theorem 3.7. However, the acceptance probability (4.2) sets here implicitly some additional requirements to the generation probabilities, since the difference in consensus is calculated by keeping all 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 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, we can apply Theorem 3.7, but then the parallel nature of the algorithm is lost. To solve this dilemma, it has been suggested [25] 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 a *clustered BM* we can maintain some parallelism and update all the nodes in one cluster at the same time, while a convergence theorem similar to 4.1 can be proved:

Theorem 4.2 Let the nodes of a Boltzmann machine be divided into separate clusters (independent sets), where no two nodes in the same cluster are connected to each other. At time t, update simultaneously all the nodes in a randomly chosen cluster using the formula (4.2). Let $S(0), S(1), \ldots$ denote the resulting states of the network and let C_{opt} denote the set of states with globally maximal consensus. Now it follows that

$$\lim_{T\downarrow 0}\lim_{t\to\infty} P_T(S(t)\in C_{\text{opt}}) = 1.$$

Proof: As Theorem 4.1, but instead of Theorem 3.7, apply Theorem 8.2 in [1]. □

Obviously, as all the nodes in a cluster can be updated simultaneously, 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 identical to the graph coloring problem and is hence NP-complete. However, in the sequel we deal with a special class of two-layer BM architectures which have by definition only two clusters, being in this sense optimal BM architectures.

4.2 Harmony networks

A basic difficulty with understanding Smolensky's theory of harmony networks [114] lies in the fact that Smolensky uses two disturbingly similar, structurally identical models on different levels of abstraction. The first model, the *conceptual harmony network* can be used for defining a function (the *harmony function*) on a set of binary variables, conceptually in same way as the MRF model is used for defining a Gibbs distribution. There are some interesting cognitive questions related to harmony network representations, but they are not to be addressed in this study. The second model, the *computational harmony network*, is a two-layer neural network which can be used for finding the maximum of the harmony function, and it is functionally very close to the Boltzmann machine model. We shall first introduce the conceptual model and the definition of the harmony function in Section 4.2.1, and the computational model is then presented in Section 4.2.2.

4.2.1 The harmony function

Let our problem domain be defined on a set of N *binary* random variables $U_1, \ldots, U_N$. For notational convenience, the variables U_i are here assumed to be bipolar, i.e. they have either value -1 or 1, instead of 0 and 1. A harmony network consists of two layers of nodes: a set of N feature nodes $X_1, \ldots, X_N$ corresponding to the variables $U_1, \ldots, U_N$, and a set of R *pattern* nodes $\{Y_1, \ldots, Y_R\}$ (Smolensky's "knowledge" nodes). Each node in a harmony network has an *activity level*, which can have two values: the possible activity values for the feature nodes are $\{-1, +1\}$, and the values for the pattern nodes are $\{0, 1\}$. The activity values of the feature nodes form a *feature vector* $\vec{x}$, and the activity values of the pattern nodes, denoted by $y_1, \ldots, y_R$, form a *pattern activation vector* $\vec{y}$. These two vectors combined form the global state of the harmony network.

To each pattern node Y_j, we attach a *pattern potential* λ_j, which is a real number assumed to be computed from a feature vector $\vec{x} \in \{-1, +1\}^N$ using a *potential function* V: $\lambda_j = V(\vec{x})$. We express this in a graphical form by drawing a positive arc between the pattern node Y_j and a feature node U_i if the value of the variable U_i in the feature vector $\vec{x}$ is 1; a negative arc if the value is -1; and we make no connection between the nodes if the value of the variable is irrelevant for determining the value of the potential λ_j. The incoming arcs to a pattern node Y_j can be represented as a *pattern vector* $\vec{\omega}_j \in \{-1, 0, +1\}^N$, where +1 denotes a positive connection, -1 a negative connection, and 0 stands for a missing connection (see Figure 4.3).

For each state of the harmony network, we attach a global measure called

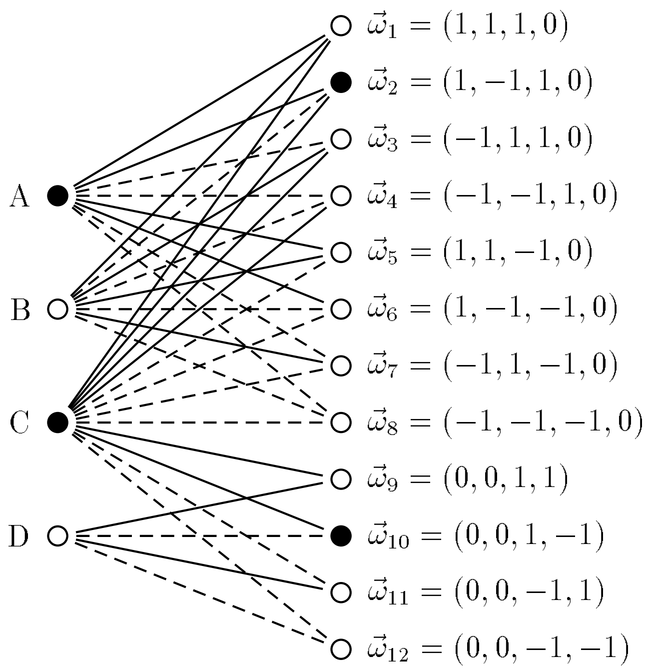


Figure 4.3: A simple harmony network with the corresponding pattern vectors. Nodes with activity level 1 are shown in black: the current feature vector $\vec{x}$ is (1,-1,1,-1), and the current pattern activation vector $\vec{y}$ is $(0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)$ Solid lines represent positive arcs, negative arcs are printed with dashed lines.

the harmony function. Given a feature vector $\vec{x}$, we first define the *local harmony* h for a pattern node Y_i as

$$h(\vec{x}, \vec{\omega}_i) = \frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} - \kappa, \tag{4.3}$$

where $\vec{\omega}_i \cdot \vec{x}$ is the normal vector inner product, $\vec{\omega}_i$ is the pattern vector corresponding to node Y_i, $|\vec{\omega}_i|$ is the size of the pattern vector $\vec{\omega}_i$ (the number of nonzero connections at node Y_i),

$$|\vec{\omega}_i| = \sum_{j=1}^{N} |\omega_{ij}|,$$

denoting $\vec{\omega}_i = (\omega_{i1}, \ldots, \omega_{iN})$, and $\kappa < 1$ is a constant fulfilling the condition

$$\frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} > \kappa \Leftrightarrow \frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} = 1. \tag{4.4}$$

It is easy to see, that if we, for instance, choose

$$\kappa = 1 - \frac{2}{\max_i |\vec{\omega}_i|},$$

condition (4.4) is always met: Let $|\vec{\omega}_i^+|$ denote the number of consistent elements between $\vec{x}$ and $\vec{\omega}_i$ (number of vector components ω_{ij} with the property $\omega_{ij} x_j = 1$), and let $|\vec{\omega}_i^-|$ denote the number of inconsistent elements, $|\vec{\omega}_i^-| = |\vec{\omega}_i| - |\vec{\omega}_i^+|$. Now we can write

$$\begin{aligned}
\frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} &= \frac{|\vec{\omega}_i^+|}{|\vec{\omega}_i|} - \frac{|\vec{\omega}_i^-|}{|\vec{\omega}_i|} \\
&= \frac{|\vec{\omega}_i^+| - (|\vec{\omega}_i| - |\vec{\omega}_i^+|)}{|\vec{\omega}_i|} \\
&= \frac{2|\vec{\omega}_i^+| - |\vec{\omega}_i|}{|\vec{\omega}_i|} = \begin{cases} 1, & \text{if } |\vec{\omega}_i^+| = |\vec{\omega}_i|, \\ r, & \text{otherwise } (-1 \leq r < 1). \end{cases}
\end{aligned}$$

We wish to have $\frac{2(|\vec{\omega}_i|-1)-|\vec{\omega}_i|}{|\vec{\omega}_i|} \leq \kappa < 1$, which is guaranteed by

$$1 > \kappa \geq \frac{|\vec{\omega}_i| - 2}{|\vec{\omega}_i|} = 1 - \frac{2}{|\vec{\omega}_i|}. \tag{4.5}$$

We now define the *harmony function* H for a state $(\vec{x}, \vec{y})$ as

$$H(\vec{x}, \vec{y}) = \sum_i \sigma_i y_i h(\vec{x}, \vec{\omega}_i), \tag{4.6}$$

where y_i is the activation value of pattern node Y_i in pattern activation vector $\vec{y}$, $h(\vec{x}, \vec{\omega}_i)$ is the local harmony as defined in (4.3), and σ_i, the *strength* of a pattern node Y_i, is obtained by scaling the potential by a constant:

$$\sigma_i = \frac{\lambda_i}{1 - \kappa}. \tag{4.7}$$

Intuitively speaking, the harmony function is a measure of how consistent the feature vector $\vec{x}$ and the pattern activation vector $\vec{y}$ are with each other, weighted by the strengths of the pattern nodes, which can be seen as some kind of a priori measures.

4.2.2 Maximizing the harmony function

For finding the maximum of the harmony function H, Smolensky proposed the following Boltzmann machine type architecture. Consider a network consisting of two layers of stochastic binary-valued units, where the structure of the network and the activation levels of the nodes are identical to the conceptual harmony network structure presented in the previous section. The weight of the connection between pattern node Y_i and feature node X_j is given by

$$w_{ij} = \omega_{ij}\sigma_i/|\vec{\omega}_i|, \tag{4.8}$$

where ω_{ij} is the value (-1 or +1) of the arc in the corresponding harmony network; $|\vec{\omega}_i|$ is the size of the pattern vector $\vec{\omega}_i$; σ_i is the *strength* of the node Y_i, as defined in (4.7), and κ is a parameter that can be given any value satisfying condition (4.5). As the undirected arcs are symmetric, $w_{ji} = w_{ij}$.

The nodes in the network are updated according to the following rules: if the node selected for updating is a pattern node Y_i, and the current states of the feature nodes are given by $\vec{x} = (x_1, \ldots, x_N)$, then a net input value of

$$I_i = \sum_{j=1}^{N} w_{ij}x_j - \sigma_i\kappa$$

is computed, and the node obtains value 1 with probability

$$P(y_i = 1) = \frac{1}{(1 + e^{-I_i/T})}, \tag{4.9}$$

where T denotes the value of a temperature parameter, which decreases towards zero as the temperature in simulated annealing. For a feature node X_i the net input is computed as

$$I_i = 2\sum_{j=1}^{R} w_{ij}y_j,$$

where $\vec{y} = (y_1, \ldots, y_R)$ are the current states of the pattern nodes; the node then obtains value 1 with probability

$$P(x_i = 1) = \frac{1}{(1 + e^{-I_i/T})}. \tag{4.10}$$

We use the term *harmonium Boltzmann machine (HBM)* to denote a neural network architecture corresponding to the definitions (4.8), (4.9), and (4.10). The following proposition shows that the HBM model provably maximizes the harmony function with high probability:

Proposition 4.3 Given a harmony network, let us construct a corresponding HBM model with weights defined as in (4.8). Assuming that the nodes are updated as defined in (4.9) and (4.10), and assuming that all the nodes at the same level are updated simultaneously and the two levels are updated alternately, the HBM model will converge to a maximal harmony state with probability one as T approaches zero.

Proof: Let $\vec{y}$ denote an arbitrary activation vector of the pattern nodes with $y_i = 0$, and $\vec{x}$ denote an arbitrary activation vector of the feature nodes. We now change the activation value of the node Y_i of $\vec{y}$ to 1, and denote the new vector by $\vec{y}'$. The difference in harmony between these two situations is

$$\begin{aligned}
&H(\vec{x},\vec{y}') - H(\vec{x},\vec{y}) \\
&= (1 \cdot \sigma_i h(\vec{x},\vec{\omega}_i) + \sum_{j\neq i} y_j \sigma_j h(\vec{x},\vec{\omega}_j)) - (0 \cdot \sigma_i h(\vec{x},\vec{\omega}_i) + \sum_{j\neq i} y_j \sigma_j h(\vec{x},\vec{\omega}_j)) \\
&= \sigma_i h(\vec{x},\vec{\omega}_i) = \sigma_i (\sum_{j=1}^{N} \frac{\omega_{ij} u_j}{|\vec{\omega}_i|} - \kappa) \\
&= \sum_{j=1}^{N} \frac{\omega_{ij}\sigma_i u_j}{|\vec{\omega}_i|} - \sigma_i \kappa = \sum_{j=1}^{N} w_{ij} u_j - \sigma_i \kappa = I_i,
\end{aligned}$$

which is the net input to pattern node Y_i. Similarly, let $\vec{x}$ denote the activation vector for the feature nodes with the node X_i set to -1, and $\vec{x}'$ the vector with the node set to 1. Now the difference in harmony between these two situations is

$$\begin{aligned}
&H(\vec{x}',\vec{y}) - H(\vec{x},\vec{y}) \\
&= \sum_{j=1}^{R} \sigma_j y_j (h(\vec{x}',\vec{\omega}_j) - h(\vec{x},\vec{\omega}_j)) \\
&= \sum_{j=1}^{R} \sigma_j y_j [(\frac{\sum_{k\neq i} \omega_{jk} u_k + \omega_{ji} \cdot 1}{|\vec{\omega}_j|} - \kappa) - (\frac{\sum_{k\neq i} \omega_{jk} u_k + \omega_{ji} \cdot -1}{|\vec{\omega}_j|} - \kappa)] \\
&= \sum_{j=1}^{R} \sigma_j y_j \frac{2\omega_{ij}}{|\vec{\omega}_j|} = 2\sum_{j=1}^{R} \sigma_j y_j \frac{\omega_{ij}}{|\vec{\omega}_j|} = 2\sum_{j=1}^{R} w_{ij} y_j = I_i.
\end{aligned}$$

As in the case of Boltzmann machines, we can now regard the HBM model updating scheme as a stochastic simulation process. Since the net input to a pattern node Y_i is $I_i = H(\vec{x},\vec{y}') - H(\vec{x},\vec{y})$, and the net input to a feature node X_i is $I_i = H(\vec{x}',\vec{y}) - H(\vec{x},\vec{y})$, the updating probabilities (4.9) and (4.10) can be regarded as acceptance probabilities identical to those proposed by Barker (3.7), with respect to a Gibbs distribution

$$P_T\{\vec{x}\} = \frac{1}{Z_T} e^{H(\vec{x})/T}. \tag{4.11}$$

Moreover, as the net inputs to the feature nodes are independent of the feature vector $\vec{x}$, and similarly, as the net inputs to the pattern nodes are independent of the pattern activation vector $\vec{y}$, all the nodes on one layer can be updated simultaneously without affecting the convergence process. According to Theorem 4.2, the resulting HBM updating process will find a maximum of the Gibbs distribution (4.11), which has the same maximum points as the harmony function H. $\square$

In [114], Smolensky gives no formal method for determining the number of harmony network pattern nodes, or for choosing the network structure. In Section 5.1, we show how Bayesian networks can used as a tool for constructing harmony networks from a priori knowledge: given a Bayesian network representation for a probability distribution $\mathcal{P}$, we show how to construct an equivalent HBM representation, in the sense that the harmony function of the resulting HBM has the same maximum points as $\mathcal{P}$.

Chapter 5

Mapping Bayesian networks to stochastic neural networks

As noted in [86, 87], singly connected Bayesian networks offer a possibility for implementing Bayesian reasoning on massively parallel architectures. In this study, however, we are concerned with general BN structures, and we wish to apply the stochastic simulated annealing method for solving the MAP task. Given a Bayesian network, our goal is to construct a massively parallel simulated annealing process for solving MAP problems by using a stochastic neural network which has the same maximum points as the probability distribution corresponding to the given Bayesian network. To prevent theoretically valid, but practically unrealistic models, we restrict ourselves to standard neural network structures with structurally simple and homogeneous processing elements. It should also be noted that the methods developed here apply only for massively parallel neural network implementations — parallelization of simulated annealing using more conventional computing architectures is discussed in [42].

In Chapter 4 we saw how the two stochastic neural network models, BM and HBM, can be regarded as massively parallel implementations of simulated annealing, being capable of finding the optimum of the given objective function. On the other hand, in Section 3.4.2 it was shown how any Bayesian network probability distribution can be expressed as a Gibbs distribution, utilizing the concept of Markov random fields. Consequently, if we find a way to represent a given Gibbs distribution as an objective function of a stochastic neural network, we have accomplished our goal: massively parallel architecture for solving the the MAP task.

Let $\mathcal{B}=(\mathcal{B}_{\mathcal{S}}, \mathcal{B}_{\mathcal{P}})$ be a Bayesian network corresponding to a probability distribution $\mathcal{P}$. In Section 3.4.2 we showed how $\mathcal{P}$ can be expressed as a Gibbs distribution of the form (3.8). For finding the maximum of $\mathcal{P}$, it is now sufficient to maximize the potential function V,

$$V(\vec{u}) = \sum_{i}^{N} V_i(\vec{u}), \tag{5.1}$$

where the clique potentials V_i are of the form (3.9). Let us now consider a clique $\boldsymbol{C}_i = \{U_i\} \cup \{U_j \mid U_j \in \boldsymbol{F}_{\boldsymbol{U_i}}\}$ corresponding to a variable U_i, let the vectors $\vec{\omega}_{i1}, \ldots, \vec{\omega}_{im_i}$ denote the m_i possible value combinations on the set $\boldsymbol{C}_i$,

$$m_i = |U_i| \prod_{U_j \in \boldsymbol{F}_{\boldsymbol{U_i}}} |U_j|,$$

and let $\lambda_{i1}, \ldots, \lambda_{im_i}$ denote all the possible values of the clique potential V_i, $\lambda_{ij} = V_i(\vec{\omega}_{ij})$. We can now express the clique potential V_i as a sum

$$V_i(\vec{u}) = \sum_{j=1}^{m_i} \lambda_{ij} \chi(\vec{u}, \vec{\omega}_{ij}),$$

where the characteristic function χ expresses whether two vectors $\vec{\omega}_{ij}$ and $\vec{u}$ are consistent or not:

$$\chi(\vec{u}, \vec{\omega}_{ij}) = \begin{cases} 1, & \text{if } \vec{u} \in \{\vec{\omega}_{ij}\}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

The potential function (5.1) now becomes

$$V(\vec{u}) = \sum_{i=1}^{N} \sum_{j=1}^{m_i} \lambda_{ij} \chi(\vec{u}, \vec{\omega}_{ij}).$$

What is more, by re-indexing all the $m = |\mathcal{B}_{\mathcal{P}}| = \sum_{i=1}^{N} m_i$ possible clique assignments $\{\vec{\omega}_{11}, \ldots, \vec{\omega}_{1m_1}, \ldots, \vec{\omega}_{N1}, \ldots, \vec{\omega}_{Nm_N}\}$ as $\{\vec{\omega}_1, \ldots, \vec{\omega}_m\}$, we can forget about the cliques $\boldsymbol{C}_i$ altogether, and write the potential function V as

$$V(\vec{u}) = \sum_{i=1}^{m} \lambda_i \chi(\vec{u}, \vec{\omega}_i). \tag{5.3}$$

To solve the MAP problem by neural networks, it is now sufficient to find a neural network with an objective function which has the same maximum points as the potential function (5.3).

Let us first consider an MRF with a maximal clique size of 2. As the consensus function (4.1) of the basic BM model is defined by using a sum of parameters depending on only two variables, a BM network with a consensus function equal to the potential function of a given MRF can be easily constructed by assigning one node in the BM network to each random variable in the MRF [57, 39]. However, in this case the parallelism of the resulting BM is lost, as no two adjacent nodes (variables) can be updated at the same time. Moreover, the Bayesian network structure corresponding to this kind of a simple MRF is a tree, which means that there is no need for approximate stochastic methods since polynomial time exact algorithms for solving the MAP problem exist in the first place [96, 92].

Let us then consider Markov random fields with an unrestricted clique size, corresponding to general BN structures. In the noisy-OR approximation model, only a linear number of simple two-member conditional probabilities (one for each incoming arc) are stored for each variable in the BN structure, and the missing parameters are approximated as a function of the combination of the stored parameters (see e.g. [96, 92]). In this case, a BM structure corresponding to a given MRF can be constructed in a similar way as in the 2-variable clique MRF case [90]. However, using only basic two-member conditional probabilities, it is generally not possible to find any function capable of approximating the missing probabilities accurately, or even to obtain informative upper or lower bounds for the missing values [92, p.138]. As a result, the noisy-OR model is usually not accurate enough for practical applications [50].

For an accurate representation for general BN structures, it seems at a first glance that binary connections are not sufficient, but higher-order connections (arcs connecting three or more nodes to each other) are needed. As a matter of fact, this kind of a generalization of the basic BM model, containing also higher-order hyper-arcs, has been suggested earlier [39]. However, as in this study we do not allow such extensions of the basic neural network models, this approach is not examined here. Nevertheless, in Section 5.1 we show how to map a given Bayesian network $\mathcal{B}=(\mathcal{B}_\mathcal{S},\mathcal{B}_\mathcal{P})$ to a two-layer HBM network structure, in the sense that the resulting harmony function has the same maximum points as the potential function of the MRF corresponding to the Bayesian network (and hence has the same maximum points as the probability distribution on $\mathcal{B}$). The nodes in the first layer of the harmony network ("visible" nodes) correspond to the nodes in the given Bayesian network structure $\mathcal{B}_\mathcal{S}$, while the nodes in the second layer ("hidden" nodes) correspond to the conditional probabilities $\mathcal{B}_\mathcal{P}$ (in a sense, the hidden nodes can be regarded as a type of higher-order hyper-arcs suggested above). This means that the harmony network updating process provably converges to a state where the activity levels of the visible nodes can be projected to a MAP solution on the original BN structure. Consequently, any given MAP problem, given as a partial instantiation on $\mathcal{B}$, can be solved by permanently fixing the activity levels of the corresponding visible nodes in the HBM structure, and by letting the HBM run until converged. In Section 5.2 we present a similar two-layer construction using the basic BM model. Of these two solutions, the BM model can be considered more preferable as it is more homogeneous, containing only one single type of processing elements, thus being easier to implement in hardware. Moreover, as the BM model is more widely known than the harmony network model, it may be easier to find suitable hardware or software for practical implementations of standard Boltzmann machines than of harmony networks. Both of these constructions allow only binary variables in the original Bayesian network, but in Section 5.3 we discuss different ways of handling multi-valued variables, and show a straightforward extension in which the number of variable values does not have to be restricted.

5.1 Mapping Bayesian networks to harmony networks

Let $\mathcal{B}=(\mathcal{B}_{\mathcal{S}}, \mathcal{B}_{\mathcal{P}})$ be a Bayesian network with N binary variables $\{U_1, \ldots, U_N\}$, let $\{p_1, \ldots, p_m\}$ denote the conditional probabilities forming the set $\mathcal{B}_{\mathcal{P}}$, and let the corresponding probability distribution $\mathcal{P}$ be of the form (3.8). As noted in the previous section, the potential function V can be represented in the form (5.3), where the parameters $\lambda_1, \ldots, \lambda_m$ are of the form (3.9),

$$\lambda_j = \ln \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in \boldsymbol{F}_{U_i}} U_k = u_k\} + K_i = \ln p_j + K_i. \tag{5.4}$$

Corresponding to this Bayesian network, we construct a harmony network with N feature nodes $X_1, \ldots, X_N$, one for each variable U_i, and m pattern nodes $Y_1, \ldots, Y_m$, one for each parameter λ_i. The pattern potential of a node Y_i is set to λ_i, and hence according to formula (4.6), the harmony function of the resulting harmony network is now

$$H(\vec{x}, \vec{y}) = \sum_{i=1}^{m} \sigma_i y_i h(\vec{x}, \vec{\omega}_i),$$

where $\sigma_i = \lambda_i/(1-\kappa)$, and κ is a constant fulfilling the condition (4.4). In the sequel, we use HBM_2 to denote this type of a two-layer HBM network.

The bipolar feature units of an HBM_2 network are associated with the corresponding binary Bayesian network variables in the obvious way: an HBM_2 state with $X_i = 1$ corresponds to a configuration vector with $U_i = 1$, and a state with $X_i = -1$ to vector with $U_i = 0$. Consequently, each feature vector $\vec{x} = \{x_1, \ldots, x_N\}$ represents an instantiation of the corresponding Bayesian network. Using this binding, we can now redefine the potential function (5.3) as

$$V(\vec{u}) = V(\vec{x}) = \sum_{i=1}^{m} \lambda_i \chi(\vec{x}, \vec{\omega}_i), \tag{5.5}$$

where

$$\chi(\vec{x}, \vec{\omega}_i) = \begin{cases} 1, & \text{if } \vec{\omega}_i \cdot \vec{x}/|\vec{\omega}_i| = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

Proposition 5.1 The HBM_2 network updating process maximizes the potential function (5.3), provided that all the parameters λ_i are nonnegative.

Proof: Using a clever trick presented by Smolensky in [114], we can express the function (5.6) as

$$\chi(\vec{x},\vec{\omega}_i) = \max_{y_i \in \{0,1\}} [y_i \frac{\frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} - \kappa}{1-\kappa}] = \begin{cases} 1, & \text{if } \frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} > \kappa, \\ 0, & \text{if } \frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} \leq \kappa. \end{cases}$$

Now the potential function (5.5) can be expressed as

$$\begin{aligned} V(\vec{x}) &= \sum_i \lambda_i \max_{y_i \in \{0,1\}} [y_i(\frac{\frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} - \kappa}{1-\kappa})] \\ &= \sum_i \frac{\lambda_i}{1-\kappa} \max_{y_i \in \{0,1\}} [y_i(\frac{\vec{\omega}_i \cdot \vec{x}}{|\vec{\omega}_i|} - \kappa)] \\ &= \sum_i \sigma_i \max_{y_i \in \{0,1\}} [y_i h(\vec{x},\vec{\omega}_i)] \\ &= \max_{\vec{y}} \sum_i \sigma_i y_i h(\vec{x},\vec{\omega}_i) \\ &= \max_{\vec{y}} H(\vec{x},\vec{y}), \end{aligned}$$

provided that all the parameters σ_i are non-negative. Consequently, we can find the maximum of the potential function V by maximizing the harmony function H,

$$\max_{\vec{x}} V(\vec{x}) = \max_{\vec{x}} \max_{\vec{y}} H(\vec{x},\vec{y}).$$

As the HBM updating process provably maximizes the harmony function H (see Proposition 4.3), it also maximizes the potential function (5.5), which is equal to the potential function (5.3). □

The potential function (5.3) has the same maximum points as the original Bayesian network probability distribution $\mathcal{P}$, so a given MAP instantiation problem can be solved by mapping the BN to the corresponding HBM_2 structure, permanently fixing the activity levels of the feature nodes corresponding to the instantiated variables of the BN to the given values, and running the HBM_2 network with decreasing temperature until converged. The final feature vector, when mapped back to the Bayesian network, is the MAP solution. A simple example of the BN→MRF→HBM transformation is given in Figure 5.1.

If we set in (5.4) all the constants $K_i, i = 1,\ldots,N$ to zero, the resulting harmony function is equal to the potential function corresponding to the probability distribution $\mathcal{P}$. However, it is important to notice that Proposition 5.1 applies only if all the parameters λ_j are nonnegative, so the constants

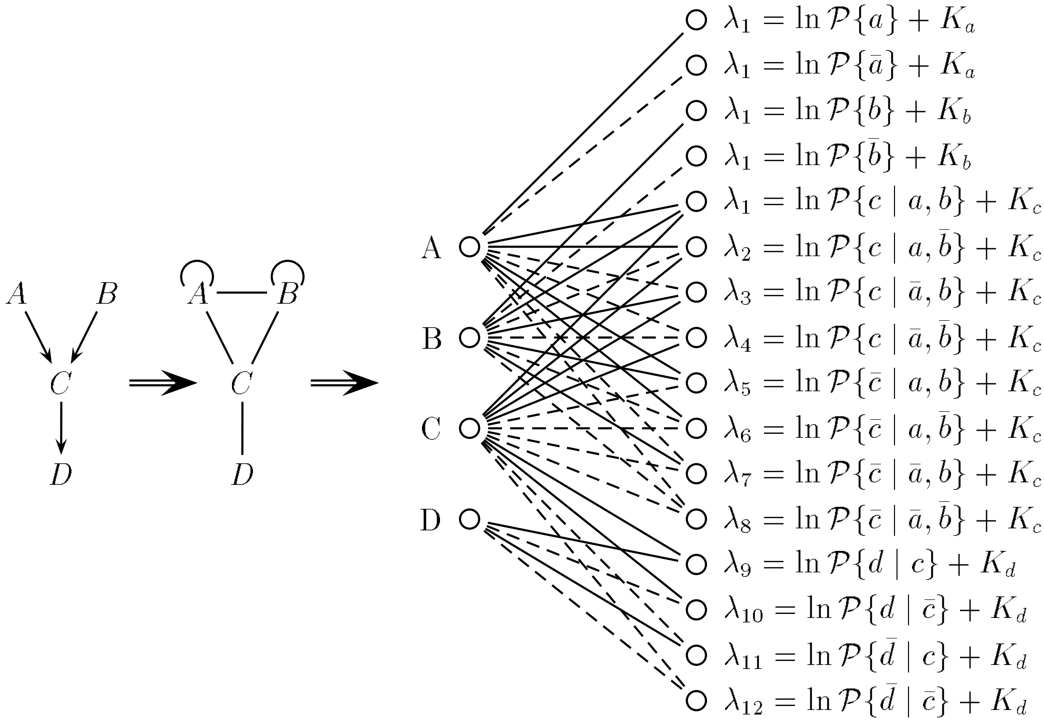


Figure 5.1: A simple Bayesian network with four binary variables, the corresponding MRF, and the resulting HBM network with the corresponding pattern node parameters λ_i.

K_i have to be chosen appropriately.[1] We suggest the following simple method for determining the parameters:

$$\lambda_j = \ln \frac{p_j}{k\hat{p}(C(j))} = \ln p_j - \ln k\hat{p}(C(j)), \tag{5.7}$$

where $C(j)$ is the index of the clique corresponding to the parameter λ_j, $\hat{p}(i)$ denotes the minimal probability within clique i, and $k \leq 1$ is some constant. After this scaling all the parameters λ_j are nonnegative, and moreover, as noted in Section 3.4.2, the scaling does not affect the convergence of the sampling process.

[1]This means that we cannot construct a harmony network with a harmony function exactly identical to the given potential function — we are only able to construct a harmony function with the same maximum points as the potential function.

To verify this fact in the harmony network framework, let us consider the behavior of the network at temperature T with T decreasing towards zero. As T approaches zero, the probability of exactly one pattern node for each of the cliques to be "on" approaches one, and at the final (ideal) zero temperature, the number of pattern nodes to be "on" becomes constant. Let H be a harmony function corresponding to the situation where all the constants K_i are set to zero (in which case $\lambda_j = \ln p_j$), and let H^* denote a harmony function with the parameters λ_j set as in formula (5.7). Assuming that the number of active pattern nodes is constant, the scaled harmony function H^* can be expressed as

$$\begin{aligned}
H^*(\vec{x},\vec{y}) &= \sum_{j=1}^{m} \frac{\ln p_j - \ln k\hat{p}(C(j))}{1-\kappa} a_j h(\vec{x},\vec{\omega}_j) \\
&= \sum_{j=1}^{m} \frac{\ln p_j}{1-\kappa} a_j h(\vec{x},\vec{\omega}_j) - \sum_{j=1}^{m} \frac{\ln k\hat{p}(C(j))}{1-\kappa} a_j h(\vec{x},\vec{\omega}_j) \\
&= \sum_{j=1}^{m} \frac{\ln p_j}{1-\kappa} a_j h(\vec{x},\vec{\omega}_j) - \sum_{i=1}^{N} \frac{\ln k\hat{p}(i)}{1-\kappa},
\end{aligned}$$

where N is the number cliques (the number of variables). This is the original harmony function plus a constant, $H^*(\vec{x},\vec{y}) = H(\vec{x},\vec{y}) + K$, where

$$K = -\sum_{i=1}^{N} \frac{\ln k\hat{p}(i)}{1-\kappa}.$$

This new function has the same maximum points as the original one, so (in the limit) the scaling does not affect the convergence of the simulation process. In Chapter 6, we also verify this fact empirically.

5.2 Mapping Bayesian networks to two-layer Boltzmann machines

As in the previous section, let $\mathcal{B}=(\mathcal{B}_\mathcal{S}, \mathcal{B}_\mathcal{P})$ be a Bayesian network with N binary variables, and V the corresponding potential function of the form (5.3). Let us now consider a two-layer Boltzmann machine network with a structure identical to the HBM$_2$ network presented in the previous section (with N feature units and m pattern units). It is easy to see that the consensus of such a network can be written as

$$C(\vec{s}) = \sum_{j=1}^{m} s_j \sum_{i=1}^{N} w_{ji} s_i = \sum_{j=1}^{m} s_j I_j, \tag{5.8}$$

where $I_j = \sum_i w_{ji} s_i$ is the *net input* to pattern node Y_j. The main idea here is to choose the weights in the network in such a way that the net input I_j to a pattern node Y_j is positive, and, what is more, exactly λ_j only when the corresponding value assignment $\vec{\omega}_j$ is consistent with the given feature vector $\vec{x}$. This ensures that the updating process converges to a state where only one pattern node for each clique is on, and thus the consensus of such a final state is equal to potential V. In [82] we suggested one possible way of choosing the weights; here we present an alternative solution, which seems to work better in practice.

Let us consider a pattern node Y_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,

$$\lambda_j = \ln \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_k \in \boldsymbol{F_{U_i}}} U_k = u_k\} + K. \tag{5.9}$$

We now set the weights of the arcs between the pattern node Y_j and the features nodes $X_1, \ldots, X_n$ in the following way:

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

2. The weight from the pattern node Y_j to a feature node corresponding to a variable $U_k \in \{U_i, \boldsymbol{F_{U_i}}\}$ is set to λ_j if $u_k = 1$, and to $-\lambda_j$ if $u_k = 0$

3. The bias θ_j of the pattern node Y_j is set to $-(n_j^+ - 1)\lambda_j$, where n_j^+ is the number of positive arcs leaving from node Y_j. However, if $n_j^+ = 0$, we set $\theta_j = 0$.

Following this construction for each of the pattern nodes Y_j, we get a structure which we call a *two-layer Boltzmann machine (BM$_2$)*.

Lemma 5.2 Provided that all the parameters λ_j are nonnegative, the net input I_j to a pattern node Y_j of a BM_2 network is positive only if the feature vector $\vec{x}$ is consistent with the value assignment $\vec{\omega}_j$. Moreover, in this case $I_j = \lambda_j$.

Proof: Let n_j^{++} denote the number of incoming signals on positive arcs (arcs with a weight λ_j and let n_j^{-+} denote the number of incoming signals on negative arcs (arcs with a weight $-\lambda_j$). As defined above, the bias θ_j is $-(n_j^+ - 1)\lambda_j$, where n_j^+ is the number of positive arcs coming to node Y_j. A feature vector $\vec{x}$ is consistent with the assignment $\vec{\omega}_j$ if and only if $n_j^{++} = n^+$ and $n_j^{-+} = 0$. On the other hand, the net input I_j can be written as

$$I_j = n_j^{++}\lambda_j - n_j^{-+}\lambda_j + \theta_j = n_j^{++}\lambda_j - n_j^{-+}\lambda_j - n_j^{+}\lambda_j + \lambda_j = \lambda_j(n_j^{++} - n_j^{+} - n_j^{-+} + 1).$$

As $n_j^{++} \leq n_j^+$ and $n_j^{-+} \geq 0$, this can be made positive only by setting $n_j^{++} = n_j^+$ and $n_j^{-+} = 0$, and in this case $I_j = \lambda_j$. $\square$

Now we can prove the following result:

Proposition 5.3 The BM_2 network updating process maximizes the potential function V, provided that all the parameters λ_i are nonnegative.

Proof: By exploiting Lemma 5.2, we can write the potential function (5.3) as

$$V(\vec{u}) = V(\vec{x}) = \sum_{j=1}^{m} \max_{y_j \in \{0,1\}} (y_j I_j).$$

Now it follows that

$$\max_{\vec{u}} V(\vec{u}) = \max_{\vec{x}} \sum_{j=1}^{m} \max_{y_j \in \{0,1\}} (y_j I_j) = \max_{\vec{x}} \max_{\vec{y}} \sum_{j=1}^{m} y_j I_j = \max_{\vec{s}} C(\vec{s}).$$

$\square$

Consequently, the BM_2 updating process converges (with high probability) to a state, where the feature vector represents a MAP state on the original Bayesian network. As with the HBM_2 model, the nodes on one layer are not connected to each other, and therefore do not affect the net input of each other, so they can all be updated at the same time (see Theorem 4.2).

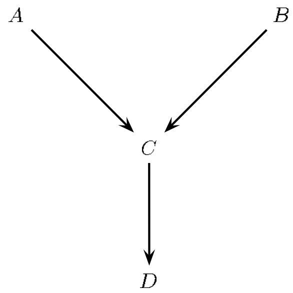
5.3 Coping with multi-valued variables

As the neural networks models used in this study have binary processing elements, the BN-MRF-NN transformation scheme described in the previous sections applies directly only to Bayesian networks with binary variables. However, in many problem domains the probability distribution is represented in the most natural way by using networks with multi-valued variables. There are now at least two alternative paths we might consider when extending our mapping scheme to multi-valued variables. Firstly, we could work on the Bayesian network level, and modify the Bayesian network structure in such a way that the mappings developed earlier can be applied. Alternatively, we could work on the neural network level, and construct mappings which are not restricted to binary variables. In the following, we first discuss briefly some obvious drawbacks of the first approach, and present then a relatively straightforward extension to the mappings presented in the previous sections, which allows also multivalued variables.

In principle, a Bayesian network with multi-valued variables can be transformed to a binary-variable network as follows: each non-binary variable X with k values $x_1, \ldots, x_k$ is replaced by k new binary variables $X_1, \ldots, X_k$, and each of these new variables is connected to all the successors of X, and similarly all the predecessors of X are connected to all the new k binary variables. In addition, each new binary variable X_i (for $i < k$) is connected to variables $X_{i+1}, \ldots, X_k$ (see Figure 5.2).

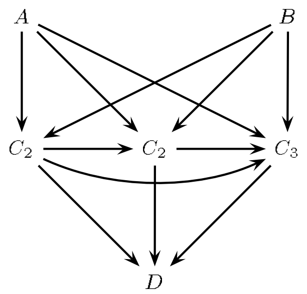
In practice, however, there are some technical difficulties with the BN level transformation. First of all, as can be seen in Figure 5.2, many of the resulting conditional probabilities, corresponding to "impossible" combinations of values (for example the probability $\mathcal{P}\{b_1|a, b_2, b_3\}$), are zero. This is unacceptable, as the configuration space $\mathbf{\Omega}$ would in this case have zero-probability states, which violates the basic assumptions behind our transformation scheme (see for example Definition 3.8). In particular, as the parameters λ_i are computed using logarithms of conditional probabilities (formula (3.9)), they become undefined if any of the probabilities are zero. A standard trick to overcome this difficulty is to replace all the zero-valued probabilities by a small constant ϵ (and, correspondingly, replace all the probabilities equal to one by a probability $1 - \epsilon$). As ϵ approaches zero, the binary-variable Bayesian network approximates the probability distribution of the original network more and more accurately. However, it has been suggested [18] that Bayesian networks containing this kind of extreme probabilities are the most difficult ones to approach by stochastic simulation methods. On the other hand, MAP problems with extreme probabilities cannot be made easier by changing the extreme probabilities of the Bayesian network further from 1 or

(a)



$\mathcal{P}\{a_1\}$	$\mathcal{P}\{c_1 \mid b_1\}$
$\mathcal{P}\{a_2\}$	$\mathcal{P}\{c_1 \mid b_2\}$
$\mathcal{P}\{b_1\}$	$\mathcal{P}\{c_2 \mid b_1\}$
$\mathcal{P}\{b_1\}$	$\mathcal{P}\{c_2 \mid b_2\}$
$\mathcal{P}\{c_1 \mid a_1\}$	$\mathcal{P}\{c_3 \mid b_1\}$
$\mathcal{P}\{c_1 \mid a_2\}$	$\mathcal{P}\{c_3 \mid b_2\}$
$\mathcal{P}\{c_2 \mid a_1\}$	$\mathcal{P}\{d_1 \mid c_1\}$
$\mathcal{P}\{c_2 \mid a_2\}$	$\mathcal{P}\{d_1 \mid c_2\}$
$\mathcal{P}\{c_3 \mid a_1\}$	$\mathcal{P}\{d_1 \mid c_3\}$
$\mathcal{P}\{c_3 \mid a_2\}$	$\mathcal{P}\{d_1 \mid c_1\}$
	$\mathcal{P}\{d_2 \mid c_2\}$
	$\mathcal{P}\{d_2 \mid c_3\}$

(b)



$\mathcal{P}\{a_1\}$	$\mathcal{P}\{c_{21} \mid a, b_2, c_{11}\}$	$\mathcal{P}\{c_{31} \mid a, b, c_{11}, c_{21}\}$	$\mathcal{P}\{c_{31} \mid a, b_2, c_{11}, c_{21}\}$	$\mathcal{P}\{d_1 \mid c_{11}, c_{21}, c_{31}\}$
$\mathcal{P}\{a_2\}$	$\mathcal{P}\{c_{21} \mid a, b, c_{11}\}$	$\mathcal{P}\{c_{31} \mid a_2, b, c_{11}, c_{21}\}$	$\mathcal{P}\{c_{31} \mid a_2, b_2, c_{11}, c_{21}\}$	$\mathcal{P}\{d_1 \mid c_{12}, c_{21}, c_{31}\}$
$\mathcal{P}\{b_1\}$	$\mathcal{P}\{c_{21} \mid a_2, b, c_{11}\}$	$\mathcal{P}\{c_{31} \mid a, b, c_{12}, c_{21}\}$	$\mathcal{P}\{c_{31} \mid a, b_2, c_{12}, c_{21}\}$	$\mathcal{P}\{d_1 \mid c_{11}, c_{22}, c_{31}\}$
$\mathcal{P}\{b_2\}$	$\mathcal{P}\{c_{21} \mid a, b, c_{12}\}$	$\mathcal{P}\{c_{31} \mid a_2, b, c_{12}, c_{21}\}$	$\mathcal{P}\{c_{31} \mid a_2, b_2, c_{12}, c_{21}\}$	$\mathcal{P}\{d_1 \mid c_{12}, c_{22}, c_{31}\}$
$\mathcal{P}\{c_{11} \mid a_1\}$	$\mathcal{P}\{c_{21} \mid a_2, b, c_{12}\}$	$\mathcal{P}\{c_{31} \mid a, b, c_{11}, c_{22}\}$	$\mathcal{P}\{c_{31} \mid a, b_2, c_{11}, c_{22}\}$	$\mathcal{P}\{d_1 \mid c_{11}, c_{21}, c_{32}\}$
$\mathcal{P}\{c_{11} \mid a_2\}$	$\mathcal{P}\{c_{22} \mid a, b, c_{11}\}$	$\mathcal{P}\{c_{31} \mid a_2, b, c_{11}, c_{22}\}$	$\mathcal{P}\{c_{31} \mid a_2, b_2, c_{11}, c_{22}\}$	$\mathcal{P}\{d_1 \mid c_{12}, c_{21}, c_{32}\}$
$\mathcal{P}\{c_{12} \mid a_1\}$	$\mathcal{P}\{c_{22} \mid a_2, b, c_{11}\}$	$\mathcal{P}\{c_{31} \mid a, b, c_{12}, c_{22}\}$	$\mathcal{P}\{c_{31} \mid a, b_2, c_{12}, c_{22}\}$	$\mathcal{P}\{d_1 \mid c_{11}, c_{22}, c_{32}\}$
$\mathcal{P}\{c_{12} \mid a_2\}$	$\mathcal{P}\{c_{22} \mid a, b, c_{12}\}$	$\mathcal{P}\{c_{31} \mid a_2, b, c_{12}, c_{22}\}$	$\mathcal{P}\{c_{31} \mid a_2, b_2, c_{12}, c_{22}\}$	$\mathcal{P}\{d_1 \mid c_{12}, c_{22}, c_{32}\}$
$\mathcal{P}\{c_{11} \mid b_1\}$	$\mathcal{P}\{c_{22} \mid a_2, b, c_{12}\}$	$\mathcal{P}\{c_{32} \mid a, b, c_{11}, c_{21}\}$	$\mathcal{P}\{c_{32} \mid a, b_2, c_{11}, c_{21}\}$	$\mathcal{P}\{d_2 \mid c_{11}, c_{21}, c_{31}\}$
$\mathcal{P}\{c_{11} \mid b_2\}$	$\mathcal{P}\{c_{21} \mid a_2, b_2, c_{11}\}$	$\mathcal{P}\{c_{32} \mid a_2, b, c_{11}, c_{21}\}$	$\mathcal{P}\{c_{32} \mid a_2, b_2, c_{11}, c_{21}\}$	$\mathcal{P}\{d_2 \mid c_{12}, c_{21}, c_{31}\}$
$\mathcal{P}\{c_{12} \mid b_1\}$	$\mathcal{P}\{c_{21} \mid a, b_2, c_{12}\}$	$\mathcal{P}\{c_{32} \mid a, b, c_{12}, c_{21}\}$	$\mathcal{P}\{c_{32} \mid a, b_2, c_{12}, c_{21}\}$	$\mathcal{P}\{d_2 \mid c_{11}, c_{22}, c_{31}\}$
$\mathcal{P}\{c_{12} \mid b_2\}$	$\mathcal{P}\{c_{21} \mid a_2, b_2, c_{12}\}$	$\mathcal{P}\{c_{32} \mid a_2, b, c_{12}, c_{21}\}$	$\mathcal{P}\{c_{32} \mid a_2, b_2, c_{12}, c_{21}\}$	$\mathcal{P}\{d_2 \mid c_{12}, c_{22}, c_{31}\}$
	$\mathcal{P}\{c_{22} \mid a, b_2, c_{11}\}$	$\mathcal{P}\{c_{32} \mid a, b, c_{11}, c_{22}\}$	$\mathcal{P}\{c_{32} \mid a, b_2, c_{11}, c_{22}\}$	$\mathcal{P}\{d_2 \mid c_{11}, c_{21}, c_{32}\}$
	$\mathcal{P}\{c_{22} \mid a_2, b_2, c_{11}\}$	$\mathcal{P}\{c_{32} \mid a_2, b, c_{11}, c_{22}\}$	$\mathcal{P}\{c_{32} \mid a_2, b_2, c_{11}, c_{22}\}$	$\mathcal{P}\{d_2 \mid c_{12}, c_{21}, c_{32}\}$
	$\mathcal{P}\{c_{22} \mid a, b_2, c_{12}\}$	$\mathcal{P}\{c_{32} \mid a, b, c_{12}, c_{22}\}$	$\mathcal{P}\{c_{32} \mid a, b_2, c_{12}, c_{22}\}$	$\mathcal{P}\{d_2 \mid c_{11}, c_{22}, c_{32}\}$
	$\mathcal{P}\{c_{22} \mid a_2, b_2, c_{12}\}$	$\mathcal{P}\{c_{32} \mid a_2, b, c_{12}, c_{22}\}$	$\mathcal{P}\{c_{32} \mid a_2, b_2, c_{12}, c_{22}\}$	$\mathcal{P}\{d_2 \mid c_{12}, c_{22}, c_{32}\}$

Figure 5.2: (a) A simple Bayesian network with three binary variables A, B and D, with possible values $\{a_1, a_2\}$,$\{b_1, b_2\}$ and $\{d_1, d_2\}$, and one multivalued variable C with three possible values $\{c_1, c_2, c_3\}$. (b) The corresponding binary-variable Bayesian network representation. The values of the binary variables C_i are denoted by c_{i1} and c_{i2}, $i = 1, 2, 3$.

0, since in this case the resulting probability distribution may not approximate the original distribution very well [31]. Nevertheless, in [81] we experimented with harmony networks using the transformation scheme suggested above, and got relatively good results even with extreme probabilities.

Another problem with the Bayesian network level transformation is the large number of resulting conditional probabilities, as can be noted in Figure 5.2. However, it is easy to see that there is a large number of conditional probabilities that are actually irrelevant for the functionality of the network. This fact was exploited in [81], where the size of the neural network structure was decreased by pruning the irrelevant pattern nodes.

In the neural network level approach for coping with multivalued attributes, we might consider modifying our neural network models in such a way that mappings from multivalued Bayesian networks become straightforward. Such modifications could include, for example, using winner-take-all subnetworks as modules of the network, or more complex, multi-state processing elements, such as the nodes in the generalized Boltzmann machine model in [110]. However, as we restrict ourselves in this study to standard neural network processing elements and structures, we do not consider such modifications. Instead, in the following we describe a relatively straightforward extension to the mapping presented earlier, introduced in [83], which uses the same simple neural network processing elements as the BM_2 network in the previous section.

Let $\mathcal{B}=(\mathcal{B}_{\mathcal{S}},\mathcal{B}_{\mathcal{P}})$ be a Bayesian network with N (not necessarily binary) variables, and let V be the corresponding potential function of the form (5.3). The suggested Boltzmann machine has two layers, where the first layer consists of $n=\sum_i |U_i|$ *feature nodes* $X_1,\ldots,X_n$, one for each value for each of the variables of the problem domain. The second layer has altogether $m=|\mathcal{B}_{\mathcal{P}}|$ *pattern nodes* $Y_1,\ldots,Y_m$, one node for each of the parameters λ_j used for defining the potential function V. 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. Let Y_j be the pattern node corresponding to a parameter λ_j, where λ_j is of the form (5.9). We now set the weights of the arcs between pattern node Y_j and feature nodes $X_1,\ldots,X_n$ in the following way:

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

2. The weight from node Y_j to a feature node corresponding to 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 Y_j to feature nodes corresponding to the values u_k appearing on the right hand side in (5.9) 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 node Y_j is set to $(n_j^+ - 1)\lambda_j$, where n_j^+ is the number of positive arcs leaving from node Y_j. However, if $n_j^+ = 0$, we set $\theta_j = 0$.

Following this construction for each of the pattern nodes Y_j, we get a structure which we call a *general two-layer Boltzmann machine*. In the sequel, we use the notation BM_2 for this kind of a general BM structure. An example of a BM_2 structure in a case of a simple Bayesian network is shown in Figure 5.3. As arcs with a zero-valued weight are irrelevant to the computations, they are excluded from the network.

Let us now consider a Bayesian network $\mathcal{B}$, and the corresponding BM_2 network $\mathcal{N}$. We say that a feature vector of $\mathcal{N}$ is *consistent* (with $\mathcal{B}$), if there is exactly one feature node active for each of variables in $\mathcal{B}$, representing one possible value of that variable. Consequently, a consistent feature vector can be mapped to an instantiation of $\mathcal{B}$. As before, let Y_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 Y_j is consistent with a feature vector $\vec{x}$ if $\vec{x}$ is consistent with the assignment $\langle U_i = u_i, \bigwedge_{U_k \in \boldsymbol{F_{U_i}}} U_k = u_k \rangle$. Moreover, a pattern activation vector $\vec{y}$ is said to be consistent, if all the consistent pattern nodes are on, and all the inconsistent pattern nodes are off. Finally, a state $\vec{s} = (\vec{x}, \vec{y})$ of $\mathcal{N}$ is called consistent if both $\vec{x}$ and $\vec{y}$ are consistent. It is now easy to prove the following simple lemma:

Lemma 5.4 A BM_2 network converges to a consistent state.

Proof: Let $\vec{s} = (\vec{x}, \vec{y})$ be the final state of a BM_2 updating process. We know that $\vec{s}$ is a state which maximizes the consensus $C(\vec{s}) = \sum_{j=1}^{m} y_j I_j$. It is now easy too see that if $\vec{x}$ is a consistent vector, $\vec{y}$ 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, $\vec{x}$ cannot 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 $\vec{s} = (\vec{x}, \vec{y})$ must be a consistent state. □

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

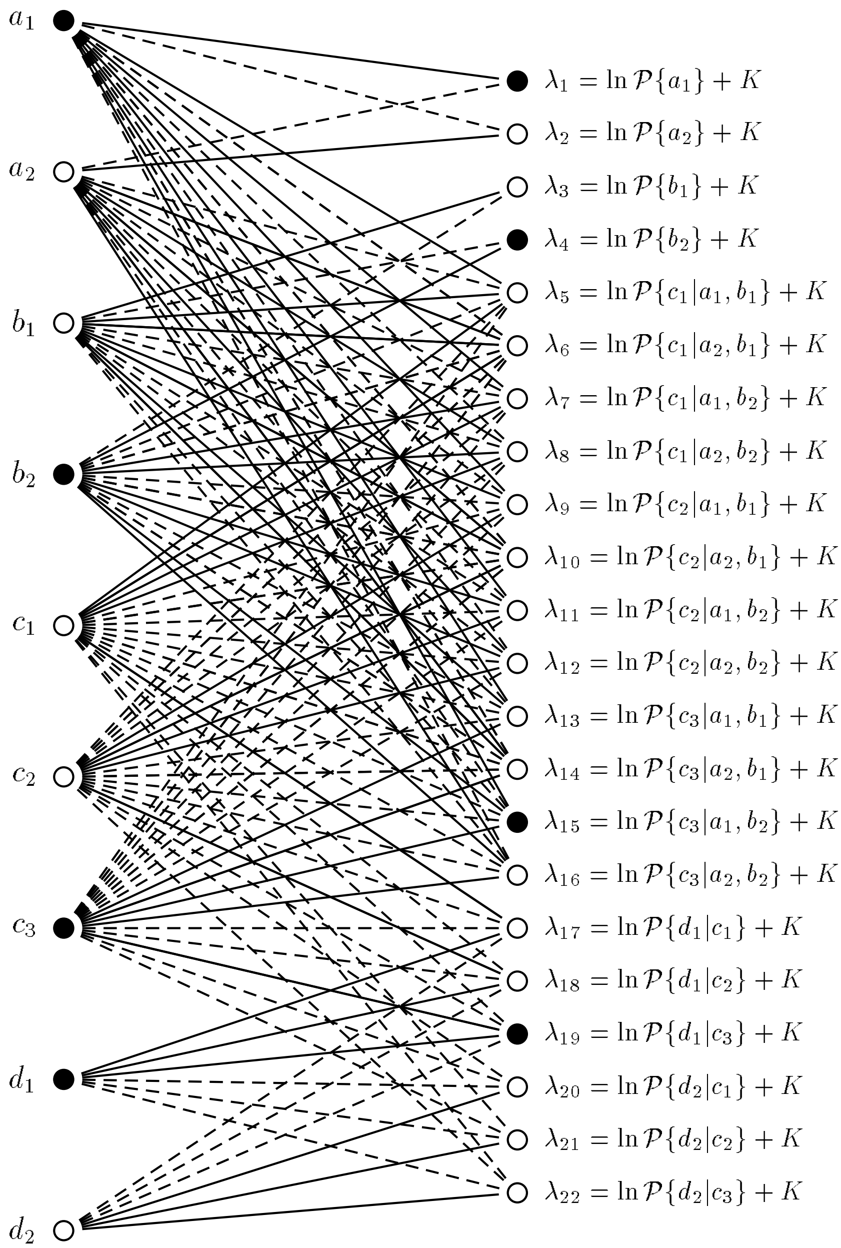


Figure 5.3: The BM_2 structure (with the parameters λ_j) corresponding to the simple Bayesian network in Figure 5.2(a). Solid lines represent arcs with positive weight, negative arcs are printed with dashed lines. Active nodes are shown in black, so the particular state in this figure is a (consistent) state corresponding to an instantiation $\langle A = a_1, B = b_2, C = c_3, D = d_1 \rangle$.

Proposition 5.5 The updating process of the general BM_2 network maximizes the potential function V of the corresponding Bayesian network $\mathcal{B}$, provided that all the parameters λ_j are nonnegative.

Proof: According to Lemma 5.4 the network converges to a stable state, where all the active pattern nodes are consistent with the consistent feature vector, so each final feature vector $\vec{x}$ corresponds to an instantiation $\vec{u}$. As in Lemma 5.2, it is easy to see that the net input to a consistent pattern node Y_j is λ_j, and as in Proposition 5.3, it now follows that

$$\max_{\vec{u}} V(\vec{u}) = \max_{\vec{x}} \sum_{j=1}^{m} \max_{y_j \in \{0,1\}} (y_j I_j) = \max_{\vec{x}} \max_{\vec{y}} \sum_{j=1}^{m} y_j I_j = \max_{\vec{s}} C(\vec{s}).$$

□

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.

Chapter 6

Empirical results

To evaluate the feasibility of the BM_2 construction described in Chapter 5, we experimented with artificial MAP problems created by generating Bayesian networks with a randomly chosen structure $\mathcal{B}_{\mathcal{S}}$ and randomly chosen conditional probabilities $\mathcal{B}_{\mathcal{P}}$. Each MAP problem, corresponding to a random initial assignment on a Bayesian network, was processed by using a version of the general sequential SA algorithm (Algorithm 3.2), and by using the massively parallel updating process of the BM_2 network corresponding to the given Bayesian network. As a reference measure, we used results of a realization of the exhaustive brute force algorithm (Algorithm 2.1). A more detailed description of the algorithms used is given in Section 6.1.

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, if suitable hardware was available. Unfortunately we did not have access to real neural hardware, so the results concerning the BM_2 implementation are based on simulations on conventional Unix workstations. Illustrative examples of the results of the simulations are shown in Section 6.3. In Section 6.2, we discuss the cooling schedule used in the experiments.

6.1 Algorithms

Let us consider a Bayesian network $(\mathcal{B}_{\mathcal{S}}, \mathcal{B}_{\mathcal{P}})$, where the graph $\mathcal{B}_{\mathcal{S}}$ consists of N variables and the set $\mathcal{B}_{\mathcal{P}} = \{p_1, \ldots, p_m\}$ contains m probabilities of the form $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F_{U_i}}} U_j = u_j\}$, and let $\vec{\omega}_1, \ldots, \vec{\omega}_m$ denote the corresponding value assignments of the form $(u_i, \bigwedge_{U_j \in \boldsymbol{F_{U_i}}} u_j)$. Recall from Chapter 2 that the state space $\mathbf{\Omega}$ has altogether M configuration vectors, and the number of vectors in the set $\mathbf{\Omega}_{\mathrm{E}}$, consisting of all the vectors consistent with the given evidence E, is denoted by $|\mathbf{\Omega}_{\mathrm{E}}|$. As before, by $\vec{u} \in \{\vec{\omega}_j\}$ we mean that the vector $\vec{u}$ is consistent with the value assignment $\vec{\omega}_j$.

A straightforward realization of the brute force algorithm 2.1 for finding the MAP solution in $\mathbf{\Omega}_{\mathrm{E}}$ can be given as follows:

Algorithm 6.1
Brute force algorithm (BF)

Input: Bayesian network $(\mathcal{B}_{\mathcal{S}}, \mathcal{B}_{\mathcal{P}})$, partial assignment E.

- $\triangleright$ $\mathcal{P}_{\max} := 0$;
- $\triangleright$ For i:= 1 to M do
  - $\triangleright$ $\vec{u} :=$ GenNextState(); $P := 1$;
  - $\triangleright$ If $\vec{u} \notin \mathbf{\Omega}_{\mathrm{E}}$, then continue;
  - $\triangleright$ for j := 1 to m do /* compute the state probability */
    - $\triangleright$ if $\vec{u} \in \{\vec{\omega}_j\}$, then $P := P * p_j$;
  - $\triangleright$ If $P > \mathcal{P}_{\max}$, then $\mathcal{P}_{\max} := P$ and I := i;

Output: MAP state $\vec{u}_I$, and the corresponding probability $\mathcal{P}_{\max}$.

Function GenNextState() returns an unevaluated configuration vector $\vec{u}$. After running the BF algorithm, I contains the index of a MAP state, and $\mathcal{P}_{\max}$ the corresponding probability (this actually does not solve the MAP problem as formulated in Section 2, but for simplicity we assume here that there is only one MAP state). Considering the time requirements of the BF algorithm, we make the following assumptions:

1. Function GenNextState() requires $O(1)$ time units.
2. Determining whether the current state $\vec{u}$ is consistent with a given partial assignment or not (checking if $\vec{u} \notin \mathbf{\Omega}_{\mathrm{E}}$ or if $\vec{u} \in \{\vec{\omega}_j\}$) requires $O(1)$ time units.

It follows that the total running time of the BF algorithm is $O(M) + |\mathbf{\Omega}_{\mathrm{E}}| * O(m)$. In the sequel, we assume (somewhat unrealistically) to have an efficient realization of the algorithm, where the inconsistent states can be bypassed very quickly, and approximate the time requirement for the BF algorithm simply by $|\mathbf{\Omega}_{\mathrm{E}}| * m$.

For simulated annealing, we used in our tests the following simple algorithm:

Algorithm 6.2
Sequential Simulated Annealing (SSA)

Input: Bayesian network $(\mathcal{B}_{\mathcal{S}}, \mathcal{B}_{\mathcal{P}})$, partial assignment E.

- ▷ $\vec{u}$:= RandomState($\mathcal{B}_{\mathcal{S}}, E$);
- ▷ $P := \mathcal{P}\{\vec{u}\}$; T := InitTemp(); $r := 0$;
- ▷ while not Converged() do
  - ▷ For i := 1 to L do
    - ▷ k := RandomFreeVarIdx($\mathcal{B}_{\mathcal{S}}, E$);
    - ▷ u := RandomValue(U_k);
    - ▷ Generate a new candidate state $\vec{u}'$ by setting $U_k := u$;
    - ▷ $P' := 1$;
    - ▷ for j := 1 to m do /* compute the state probability */
      - ▷ if $\vec{u}' \in \{\vec{\omega}_j\}$, then $P' := P' * p_j$;
    - ▷ Compute the acceptance probability A by using the formula (3.3) with probabilities P and P'.
    - ▷ If RandomNumber()<A, then $\vec{u} := \vec{u}'$ and $P := P'$;
  - ▷ $T := F * T$; r := r+1;

Output: MAP state $\vec{u}$, and the corresponding probability P.

Function RandomFreeVarIdx() returns the index of a randomly chosen variable, which was not instantiated in E. Function Converged() determines whether the simulated annealing process is converged or not. As in [71, 65], we consider a SA process converged when the last c generated states have the same probability. In our tests, we used $c = L$, where L is the *sweepsize* parameter, i.e. the number of iterations performed at each temperature.

In the following, we use the term *run* for denoting one simulation process, where the SSA algorithm is completed once, with the temperature going from its initial value to zero in r iteration steps. As with the BF algorithm, we assume that one iteration step of the SA algorithm can be completed in $O(m)$ time units, i.e. in time proportional to the size of the set of probabilities $\mathcal{B}_{\mathcal{P}}$. It now follows that the total number of time units required for one run is approximately $O(Lmr)$. Naturally, the number of iterations completed, r, is determined by the cooling schedule used, which is in this case determined by the functions InitTemp() and Converged(), and by the two parameters F and L. The *annealing factor* F determines how much the temperature is decreased after processing L iterations at a constant temperature. Although simple, this type of cooling schedule is very common, and has proven successful in many applications [1]. It is also empirically observed that more sophisticated annealing methods do not necessarily produce any better results than this simple method [65]. The cooling schedule used in our simulations is discussed in more detail in the next section.

As a comparison to the SSA algorithm, we experimented with the following massively parallel algorithm, where the structure of the BM_2 network used is determined by using the given BN architecture, as described in Section 5.3. In these experiments, the parameters λ_j were determined by using the formula (5.7). As with the harmony network experiments in [81], we noted that the constant k is not very relevant for the results, as long as it stays relatively close to 1. In the sequel, the results are obtained with models constructed by using $k = 1 - 10^{-12}$.

Algorithm 6.3
Boltzmann Machine Simulated Annealing (BMSA)

Input: BM_2 network corresponding to the BN in question, partial assignment E.

- ▷ Permanently fix the states of the feature units X_i whose value is determined in E.
- ▷ $\vec{x}$:= RandomState(BM_2,E);
- ▷ T := InitTemp(); $r := 0$;
- ▷ while not Converged() do
  - ▷ For i := 1 to L do
    - ▷ $\vec{y}$:= UpdatePatternNodes(BM_2);
    - ▷ $\vec{x}$:= UpdateFeatureNodes(BM_2);
  - ▷ $T := F * T$; $r := r + 1$;

Output: MAP state $\vec{x}$.

The probability of the resulting MAP state can be computed by multiplying the conditional probabilities p_i corresponding to active pattern units.

The functions UpdatePatternNodes() and UpdateFeatureNodes() correspond to processes where all the nodes on a layer are allowed to update their state simultaneously. In our simulations, we assumed that all the nodes in one layer can be updated in parallel, and each node can update its state in $O(1)$ time, and hence the (simulated) total running time for one run of the BMSA algorithm was assumed to be approximately $O(Lr)$.

6.2 Cooling schedule

When considering the performance of the SSA and BMSA algorithms, it is clear that the most critical issue is finding a suitable cooling scheme. Unfortunately, the theoretically correct cooling scheme of Theorem 3.10 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 results are unreliable. In statistical mechanics it has been observed that during the annealing process there is a certain temperature at which the rate of change of the energy is exceptionally high [71]. This *critical temperature* can be seen as a phase transition point which starts the freezing process. It appears that for successful annealing, the cooling should be done very slowly around the critical temperature. Therefore, knowing the critical temperature would be useful for speeding up the convergence as the initial temperature could be chosen just above this point, and the cooling could be done first slowly near this point, and later faster with the temperature approaching zero. Some attempts towards analyzing the critical temperature of harmony networks can be found in [104]. In our tests, we adopted the approach proposed in [71] and selected the initial temperature by an iterative search process: we start with temperature 1, compute the acceptance probabilities, and double the temperature until all the acceptance probabilities are within the range $[0.5-\epsilon, 0.5+\epsilon]$. The final temperature of this "simulated heating" process is then used as the initial temperature for a simulated annealing process. Following the suggestions in [65], we typically set $\epsilon = 0.1$.

The cooling schedule of our version of SA can be made slower by increasing the cooling factor F or by increasing the sweepsize L. In Figure 6.1, we plot the average relative error between the probability of the final state of the BMSA algorithm and the true MAP probability as a function of these two parameters. Each data point corresponds to 1000 annealing runs on 100 different MAP problems (10 runs/problem). The 100 problems were generated by constructing 10 random Bayesian networks with 16 binary variables, and by generating 10 random assignments on each network by randomly clamping half of the variables to randomly chosen values.

In Figure 6.1, the distance between two tickmarks on the F- or L-axis corresponds to doubling the time used for SA. It appears that increasing F to its square root or doubling the sweepsize L will produce an approximately

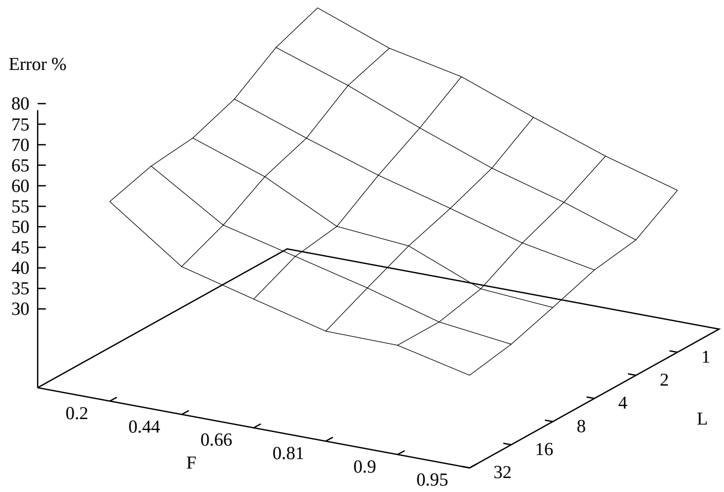


Figure 6.1: Relative error of a single BMSA run as a function of the cooling factor F and sweepsize L.

equal improvement in the results.

It should be noted that the results in Figure 6.1 were obtained by computing the error after each individual annealing run. In the stochastic simulation community, it is generally assumed that one long simulation run produces better results than several short runs, since the shorter runs never quite reach the equilibrium state of the stochastic process [118], [1, p. 94]. However, although this assumption may be true in the EVE problem framework, in our MAP framework the situation is quite different: as our intention is just to find the MAP state (or a good approximation of it), it is very probably more sensible in practice to run several shorter runs, and report the best of the final states of the completed runs, than to perform only one long run, and report its final state. In Figure 6.2, we plot the average relative error as a function of the number of runs allowed. The test set consisted of the same 100 MAP problems as above. In these tests, the cooling factor F was set to 0.66, and sweepsize L was 16.

In the sequel, we use the BMSA and SSA algorithms for solving a given MAP problem by starting new annealing runs (with a randomly chosen initial state) until the relative error between the probability of the final state and the MAP probability is found to be less than 1 %. The time reported for solving

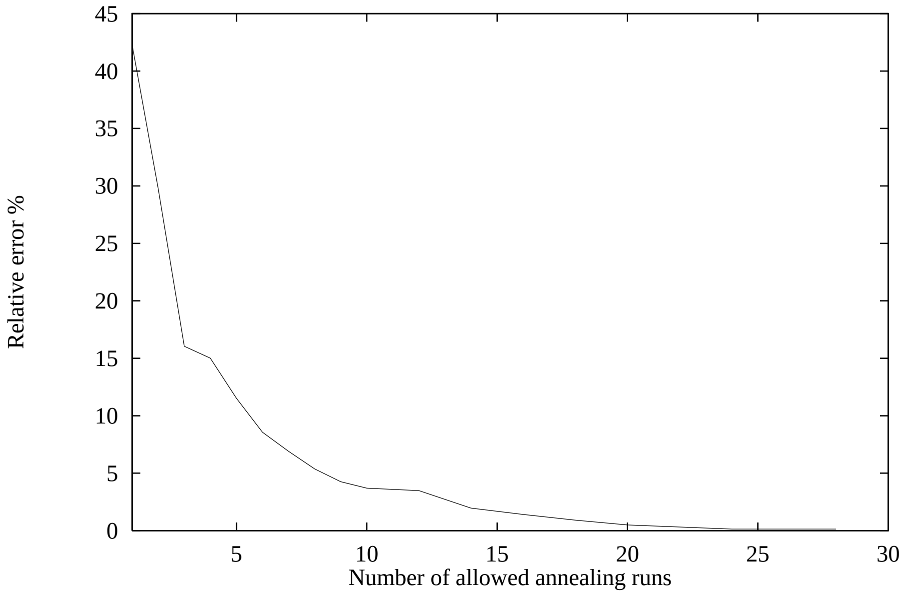


Figure 6.2: BMSA error as a function of the number of annealing runs used.

the problem, *time before success*, is the sum of the times used for completing each separate annealing run. In Figure 6.3, we plot the time before success as a function of the parameters F and L. The test set consists here of the same 100 MAP problems as before. These results suggest that with the MAP problem the best technique is to use a relatively fast annealing schedule, and to repeat the convergence process for several times. What is also important to notice is that this kind of a repetitive computation is parallelized quite naturally even using conventional parallel platforms (for example, a network of workstations), as each of the independent runs can be performed on a different processor at the same time.

As our intention in this study was not to study the efficiency of the simulated annealing algorithm per se, we did not spend a lot of time tuning the parameters of the algorithms, and hence the cooling schedules used here are by no means optimal. In the empirical tests of Section 6.3 for comparing the performance of SSA and BMSA, we used the same cooling schedule with $F = 0.66$ and $L = N$ for both of these methods. As the optimal cooling schedules for these two methods are probably different, this type of comparison may not be quite fair for one or the other. However, tuning the cooling schedule to optimum is a tedious task, and moreover, the optimum is dependent on the complexity of the problem in question. Besides, as our intention

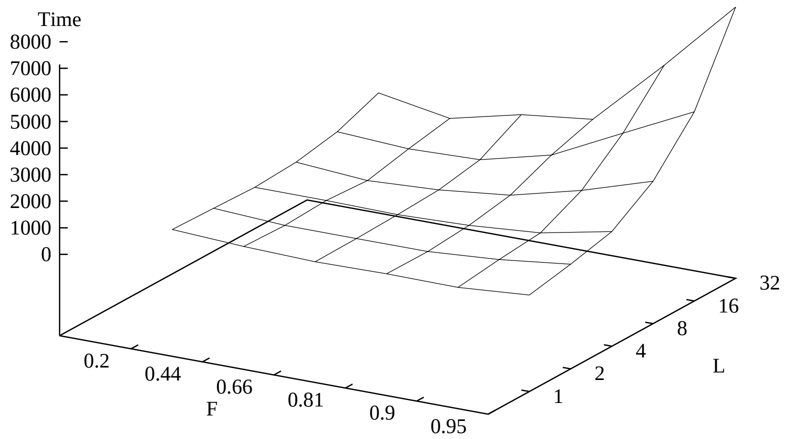


Figure 6.3: The behavior of the BMSA time with different cooling factors and sweepsizes.

was only to examine the general tendency of the SSA and BMSA algorithms with increasing problem complexity, we believe that using the same (more or less arbitrarily chosen) cooling schedule for both of these methods gives us enough information for this purpose. For the same reason, we did not experiment with the more complex cooling schedules listed in [73, 1], or with any of the elaborate techniques for speeding up the convergence speed of the simulated annealing process [97, 12, 3, 7, 100, 63], but assumed that the proportional speedup gained from using any of these methods would be roughly equal with both SSA and BMSA.

6.3 Results

As noted earlier, the primary objective 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 shape 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 by allowing more variables in the Bayesian networks, and allowing the variables to have more values.

It has been noted [18] that solving the EVE problem 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 [81], in our MAP problem framework this does not seem to be true. We experimented by restricting the randomly generated conditional probabilities $\mathcal{B}_{\mathcal{P}}$ 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 is not to be 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 in the set $\mathcal{B}_{\mathcal{P}}$. Since the BMSA algorithm (or actually its imaginary 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 6.4). 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 6.4 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

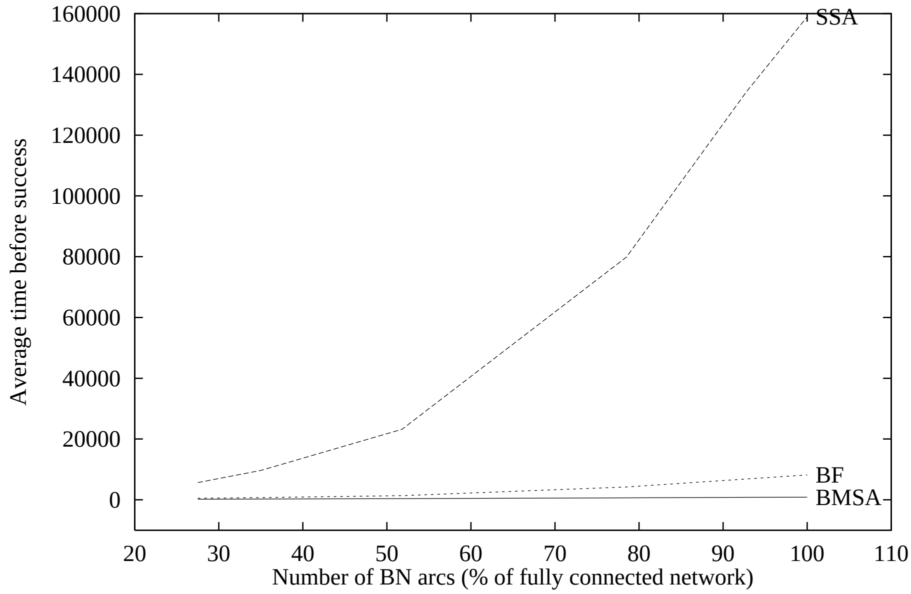


Figure 6.4: The behavior of the BMSA and SSA algorithms as a function of the density of Bayesian network.

increasing size of the networks (as we shall see in Figure 6.7). 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 as the number of unclamped variables increases, the time required for running BF grows rapidly (see Figure 6.5). 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 6.6, 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 6.7). The SSA and BMSA algorithms, on the other hand, seem to scale up very well. In Figure 6.7, 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

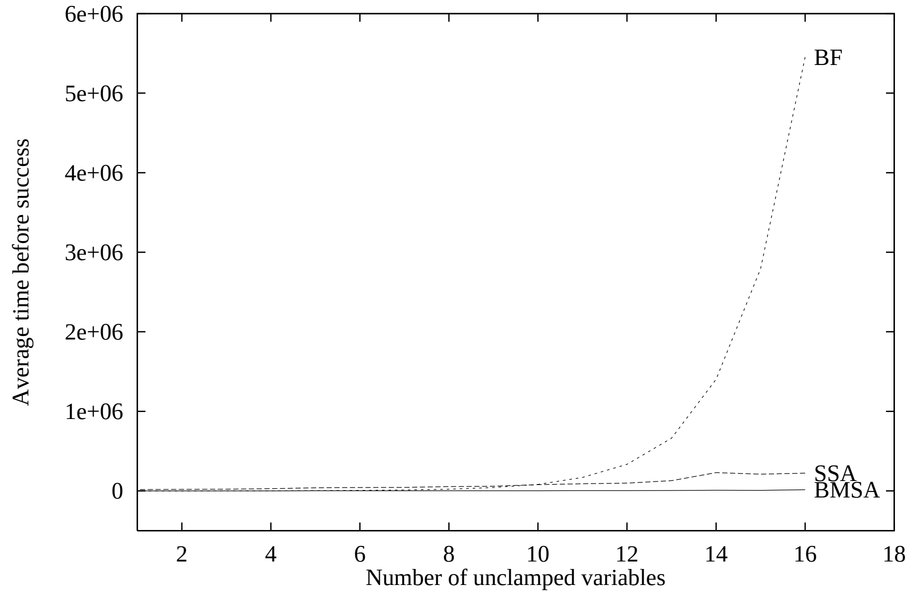


Figure 6.5: The behavior of the algorithms as a function of the number of the unclamped variables.

in advance.

These results strongly suggest that the massively parallel BMSA algorithm outperforms the 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. However, it must be emphasized again that our SSA realization of the Gibbs sampling/annealing method is by no means an optimal one, but there exist several ways to make the SA much more efficient than in the experiments here, either on just a standard personal computer or workstation, on a network of computers, or on other conventional parallel platforms. Nevertheless, 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 MAP problem solver.

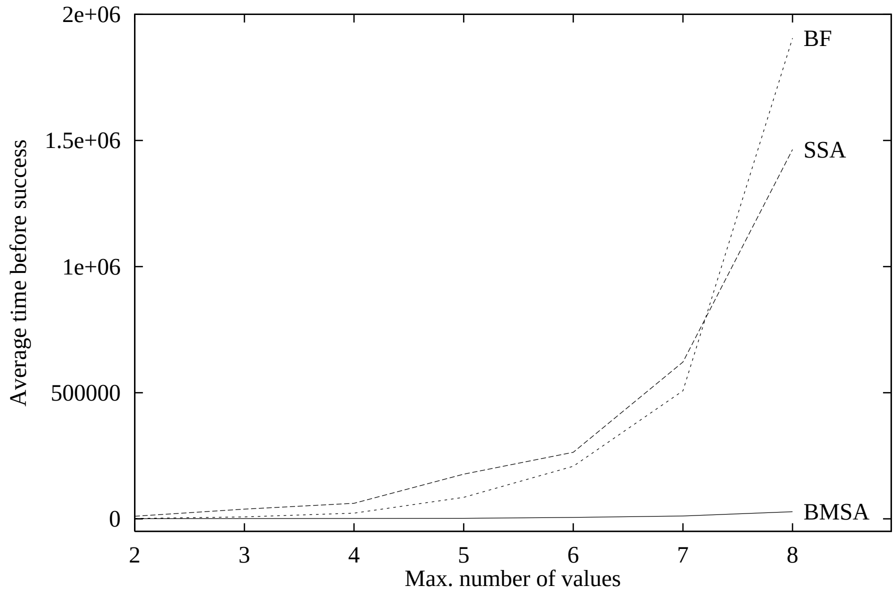


Figure 6.6: The behavior of the algorithms as a function of the number of the values of the variables.

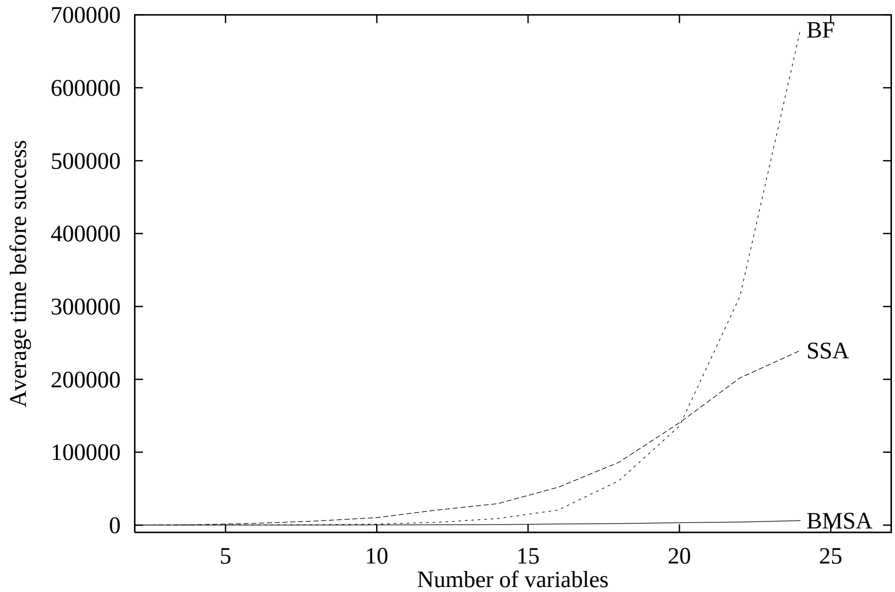


Figure 6.7: The behavior of the algorithms as a function of the number of the variables.

Chapter 7

Conclusion

We considered here two different formalisms for approaching the problem of finding MAP configurations of discrete random variables. The first of these approaches, Bayesian networks, provides us with an elegant method for constructing probabilistic models from domain expert knowledge. Although the MAP problem can be shown to be NP-hard within this framework, we showed how a stochastic simulated annealing method can be used for solving MAP problems approximatively, in the sense that a MAP state can be found with high probability. However, the standard version of simulated annealing uses Gibbs sampling for generating new candidate states, which means that only one of the variables is updated at a time. Consequently, this kind of a sequential simulated annealing process becomes easily impractically slow as the number of variables increases. To overcome this drawback, we presented three mappings from a given Bayesian network to a stochastic neural network architecture, in the sense that the updating process of the resulting neural network provably converges to a state which can be projected to a MAP solution on the original Bayesian network. The first of these mappings used as the neural network platform a harmony network, which can be seen as a special case of the Boltzmann machine architecture. The second mapping used a more standard two-layer Boltzmann machine structure, with homogeneous processing units. As both of these mappings assumed the Bayesian network to be consisted of binary random variables, we showed in our third mapping how to extend the suggested method to Bayesian networks with multi-valued variables.

The BN–NN mappings presented here can be used for constructing a hybrid BN–NN system, where the NN component provides a massively parallel search algorithm (BMSA) for finding MAP configurations, corresponding (in a sense) to a simulated annealing process where all the variables can be updated at the same time. However, although the BMSA algorithm provably converges to the same final state as the normal sequential simulated annealing (SSA) would do, the two stochastic processes follow quite different routes on their way to the same final destination. In particularly, the BMSA process works in a state space much larger than the state space for SSA, and hence some of the states during the BMSA process (the inconsistent NN states) cannot be mapped to a BN instantiation. This means that, first of all, the hybrid scheme presented here can be used for solving MAP problems only — the scheme does not apply directly to the EVE problem framework[1]. Secondly, it should be noted that as the BMSA algorithm does not actually sample the BN probability distribution, but another probability distribution on a much larger state space with equal maximum points with the BN distribution, on a conventional serial computer, SSA can probably always be made more efficient than (a simulation of) BMSA (although it should also be kept in mind that in some cases adding more parameters may make a problem actually easier to solve in practice). Nevertheless, our empirical results very strongly suggest that the speedup gained from parallelization is sufficient to compensate for the loss of accuracy in the stochastic process, provided that suitable massively parallel hardware is available. However, as we did not have access to real neural network hardware, the results are based on relatively small-scale simulations performed on conventional workstations, not actual neural implementations.

As noted above, without proper hardware, the hybrid scheme presented here is mainly of theoretical interest, as on a conventional serial computer, a sequential SA process can probably be made faster than a simulation of the corresponding neural network updating process. Nevertheless, several massively parallel platforms for neural computing already exists: a recent review [51, Ch. 3] lists 9 neural network accelerator boards, 13 neurocomputers with general purpose processors, and 20 neurocomputers built from neurochips. Although not all of these are commercially available at the moment, it is already possible to buy hardware with quite an impressive performance for NN applications. What is more, in the next few years the availability of neural hardware is likely to improve drastically, as Japan started in 1992 a new 10-year programme under the name "Real World Computing (RWC)", which aims at "developing computational bases incorporating massively parallel, neural and optical techniques" [67, 99].

From the Bayesian network point of view, the mappings presented here can be seen as providing an efficient implementational platform for simu-

[1]Nevertheless, it could be possible to obtain good estimates of the EVE probabilities by counting the occurrences of variable-value combinations in the final states during a long series of BMSA runs. This idea, however, is not pursued here further.

lated annealing. From the neural network point of view, on the other hand, the mappings provide a way to incorporate high-level, a priori information directly into neural networks, without recourse to a time-consuming and unreliable learning process. The resulting neural network could also be used as a (cleverly chosen) initial starting point to some of the existing learning algorithms [58, 35, 36, 55] 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 the mappings 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 with the underlying Bayesian network representation.

Finally, we would like to point out that as any function on a finite, discrete space can be represented in the form (5.3), the hybrid scheme 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.

Bibliography

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

[2] Abbott, K. Robust operative diagnosis as problem solving in a hypothesis space. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (Saint Paul, Minnesota, August 1988), Morgan Kaufmann Publishers, San Mateo, CA, pp. 369–374.

[3] Alspector, J., Zeppenfeld, T., and Luna, S. A volatility measure for annealing in feedback neural networks. *Neural Computation 4* (1992), 191–195.

[4] Andersen, S., Olesen, K., Jensen, F., and Jensen, F. Hugin – a shell for building belief universes for expert systems. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Detroit, Michigan, August 1989), Morgan Kaufmann Publishers, San Mateo, CA, pp. 1080–1085.

[5] Anderson, J., and Rosenfeld, E., Eds. *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, MA, 1988.

[6] Anderson, J., and Rosenfeld, E., Eds. *Neurocomputing 2: Directions for Research.* MIT Press, Cambridge, MA, 1991.

[7] Ansari, N., S., R., and Wang, G. An efficient annealing algorithm for global optimization in Boltzmann machines. *Journal of Applied Intelligence 3*, 3 (1993), 177–192.

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

[9] Barnden, J., and Pollack, J., Eds. *Advances in Connectionist and Neural Computation Theory, Vol. I: High Level Connectionist Models.* Ablex Publishing Company, Norwood, NJ, 1991.

[10] Baum, E. Towards practical 'neural' computation for combinatorial optimization problems. In *Proceedings of the AIP Conference 151: Neural Networks for Computing* (Snowbird, UT, 1986), Denker, J., Ed., American Institute of Physics, New York, NY, pp. 53–58.

[11] Besag, J. Spatial interaction and the statistical analysis of lattice systems (with discussion). *J. Royal Statist. Soc., series B 34* (1972), 75–83.

[12] Bilbro, G., Mann, R., and Miller, T. Optimization by mean field annealing. In *Advances in Neural Information Processing Systems I*, D. Touretzky, Ed. Morgan Kaufmann Publishers, San Mateo, CA, 1989, pp. 91–98.

[13] Booker, L., Hota, N., and Ramsey, C. Bart: A Bayesian reasoning tool for knowledge based systems. In Henrion et al. [53], pp. 271–282.

[14] Brown, D., and Huntley, C. A practical application of simulated annealing to clustering. *Pattern Recognition 25*, 4 (1992), 401–412.

[15] Buchanan, B., and Shortliffe, E. *Rule-Based Expert Systems.* Addison-Wesley Publishing Company, Reading, MA, 1984.

[16] Cheeseman, P. Probabilistic versus fuzzy reasoning. In Kanal and Lemmer [68], pp. 85–102.

[17] Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., and Freeman, D. Autoclass: A Bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning* (Ann Arbor, June 1988), pp. 54–64.

[18] Chin, H., and Cooper, G. Bayesian belief network inference using simulation. In *Uncertainty in Artificial Intelligence 3*, L. Kanal and J. Lemmer, Eds. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1989, pp. 129–147.

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

[20] Cox, R. Of inference and inquiry — an essay in inductive logic. In *Maximum Entropy Formalism*, Levine and Tribus, Eds. MIT Press, 1979.

[21] Csiszar, I., and Tusnady, G. Information geometry and alternating minimization procedures. *Statistics & Decisions Supplement Issue No. 1* (1984), 205–237.

[22] Dagum, P., and Horvitz, E. A Bayesian analysis of simulation algorithms for inference in belief networks. *Networks 23* (1993), 499–516.

[23] Dagum, P., and Luby, M. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence 60* (1993), 141–153.

[24] Dagum, P., and Shavez, M. Approximating probabilistic inference in Bayesian networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 3 (March 1993), 246–255.

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

[26] Dempster, A., Laird, N., and Rubin, D. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B 39*, 1 (1977), 1–38.

[27] Dobrushin, R. The description of a random field by means of conditional probabilities and conditions of its regularity. *Theory Prob. Appl. 13* (1968), 197–224.

[28] Doorenbos, R. Matching 100,000 learned rules. In *Proceedings of the Eleventh National Conference on Artificial Intelligence* (Washington, DC, July 1993), AAAI Press/MIT, Menlo Park, CA, pp. 290–296.

[29] Druzdzel, M., and Henrion, M. Efficient reasoning in qualitative probabilistic networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence* (Washington, DC, July 1993), AAAI Press/MIT, Menlo Park, CA, pp. 548–553.

[30] Duda, R., Hart, P., and Nilsson, N. Subjective Bayesian methods for rule-based inference systems. Tech. Rep. TR 124, Stanford Research Institute, Menlo Park, CA, 1976.

[31] Eizirik, L., Barbosa, V., and Mendes, S. A Bayesian-network approach to lexical disambiguation. *Cognitive Science 17*, 2 (April-June 1993), 257–283.

[32] Faigle, U., and Schrader, R. On the convergence of stationary distributions in simulated annealing algorithms. *Information Processing Letters 27* (1988), 189–194.

[33] Floréen, P., Myllymäki, P., Orponen, P., and Tirri, H. Neural representation of concepts for robust inference. In *Proceedings of the International Symposium Computational Intelligence II* (Milano, Italy, September 1989), F. Gardin and G. Mauri, Eds., Elsevier Science Publishers B.V. (North-Holland), pp. 89–98.

[34] Floréen, P., Myllymäki, P., Orponen, P., and Tirri, H. Compiling object declarations into connectionist networks. *AI Communications 3*, 4 (December 1990), 172–183.

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

[36] Galland, C. *Learning in Deterministic Boltzmann Machine Networks*. PhD thesis, Department of Physics, University of Toronto, 1992.

[37] Gärdenfors, P., and Sahlin, N.-E., Eds. *Decision, Probability, and Utility*. Cambridge University Press, New York, 1988.

[38] Garey, M., and Johnson, D. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman, New York, NY, 1979.

[39] Geffner, H., and Pearl, J. On the probabilistic semantics of connectionist networks. Tech. Rep. R-84, UCLA Computer Science Department, Los Angeles, CA, 1987.

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

[41] Goonatilake, S., and Khebbal, S., Eds. *Intelligent Hybrid Systems*. John Wiley & Sons, Chichester, 1995.

[42] Greening, D. Parallel simulated annealing techniques. *Physica D 42* (1990), 293–306.

[43] Haario, H., and Saksman, E. Simulated annealing process in general state space. *Adv. Appl. Prob. 23* (1991), 866–893.

[44] Hajek, B. Cooling schedules for optimal annealing. *Mathematics of Operations Research 13* (1988), 311–329.

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

[46] Haykin, S. *Neural Networks: A Comprehensive Foundation.* IEEE Press/Macmillan College Publishing Company, New York, 1994.

[47] Hecht-Nielsen, R. *Neurocomputing.* Addison-Wesley Publishing Company, Reading, MA, 1990.

[48] Heckerman, D. Probabilistic interpretation for MYCIN's certainty factors. In Kanal and Lemmer [68], pp. 167–196.

[49] Heckerman, D., Geiger, D., and Chickering, D. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning 20*, 3 (September 1995), 197–243.

[50] Heckerman, D., and Shwe, M. Diagnosis of multiple faults: A sensitivity analysis. In *Uncertainty in Artificial Intelligence 9*, D. Heckerman and A. Mamdani, Eds. Morgan Kaufmann Publishers, San Mateo, CA, 1993, pp. 80–87.

[51] Heem, J. *Neurocomputers for Brain-Style Processing. Design, Implementation and Application.* PhD thesis, Leiden University, Unit of Experimental and Theoretical Psychology, 1995. Updated version of Chapter 3 can be found in "ftp.mrc-apu.cam.ac.uk/pub/nn/neurhard.ps".

[52] Henrion, M. An introduction to algorithms for inference in belief nets. In Henrion et al. [53], pp. 129–138.

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

[54] Hertz, J., Krogh, A., and Palmer, R. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[55] Hinton, G. Connectionist learning procedures. *Artificial Intelligence 40*, 1–3 (September 1989).

[56] Hinton, G. Special issue on connectionist symbol processing. *Artificial Intelligence 46*, 1–2 (1990).

[57] Hinton, G., and Sejnowski, T. Optimal perceptual inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Washington DC, June 1983), IEEE, New York, NY, pp. 448–453.

[58] Hinton, G., and Sejnowski, T. Learning and relearning in Boltzmann machines. In Rumelhart and McClelland [103], pp. 282–317.

[59] Hopfield, J., and Tank, D. Neural computation of decisions in optimization problems. *Biological Cybernetics 52* (1985), 141–152.

[60] Howard, R., and Matheson, J. Influence diagrams. In *Readings in Decision Analysis*, R.A.Howard and J.E.Matheson, Eds. Strategic Decisions Group, Menlo Park, CA, 1984, pp. 763–771.

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

[62] Hrycej, T. Common features of neural-network models of high and low level human information processing. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN-91)* (Espoo, Finland, June 1991), T. Kohonen, K. Mäkisara, O. Simula, , and J. Kangas, Eds., Elsevier Science Publishers B.V. (North-Holland), pp. 861–866.

[63] Ingber, L. Very fast simulated re-annealing. *Mathematical and Computer Modelling 8*, 12 (1989), 967–973.

[64] Jeffreys, R. *Probability and the Art of Judgement.* Cambridge University Press, New York, 1992.

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

[66] Johnson, D., Aragon, C., McGeoch, L., and Schevon, C. Optimization by simulated annealing: an experimental evaluation; Part II, graph coloring and number partitioning. *Operations Research 39*, 3 (May-June 1991), 378–406.

[67] Kahaner, D. Japan moves towards information society. *IEEE Expert* (April 1992), 54–58.

[68] Kanal, L., and Lemmer, J., Eds. *Uncertainty in Artificial Intelligence 1*. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1986.

[69] Karlin, S., and Taylor, H. *A First Course in Stochastic Processes*. Academic Press, San Diego, CA, 1975.

[70] Kinderman, R., and Snell, J. *Markov Random Fields and their Applications*. American Mathematical Society, Providence, RI, 1980.

[71] Kirkpatrick, S., Gelatt, D., and Vecchi, M. Optimization by simulated annealing. *Science 220*, 4598 (May 1983), 671–680.

[72] Kosko, B., and Isaka, S. Fuzzy logic. *Scientific American 269*, 1 (July 1993), 62–69.

[73] Laarhoven, P., and Aarts, E. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1987.

[74] Lam, F., and Yeap, W. Bayesian updating: on the interpretation of exhaustive and mutually exclusive assumptions. *Artificial Intelligence 53*, 2–3 (February 1992), 245–254.

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

[76] Lauritzen, S., and Spiegelhalter, D. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Stat. Soc., Ser. B 50*, 2 (1988), 157–224. Reprinted as pp. 415–448 in [108].

[77] Lindley, D. *Making Decisions*, second ed. John Wiley & Sons, London, 1992.

[78] McClelland, J., and Rumelhart, D., Eds. *Parallel Distributed Processing*, vol. 2. MIT Press, Cambridge, MA, 1986.

[79] McCulloch, W., and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Math. Bio. 5* (1943), 115–133. Reprinted as pp. 18–27 in [5].

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

[81] Myllymäki, P. *Bayesian Reasoning by Stochastic Neural Networks.* Ph.Lic. Thesis, Tech. Rep. C-1993-67, Department of Computer Science, University of Helsinki, 1993.

[82] Myllymäki, P. Using Bayesian networks for incorporating probabilistic a priori knowledge into Boltzmann machines. In *Proceedings of SOUTHCON'94* (Orlando, March 1994), IEEE, Piscataway, NJ, pp. 97–102.

[83] Myllymäki, P. Mapping Bayesian networks to Boltzmann machines. In *Proceedings of Applied Decision Technologies 1995* (London, April 1995), A. Gammerman, Ed., Unicom Seminars, London, pp. 269–280.

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

[85] Myllymäki, P., Orponen, P., and Silander, T. Integrating symbolic reasoning with neurally represented background knowledge. In *Proceedings of STeP-92, the Finnish Artificial Intelligence Conference* (Otaniemi, Finland, June 1992), E.Hyvönen, J.Seppänen, and M.Syrjänen, Eds., vol. 2, Finnish Artificial Intelligence Society, Helsinki, pp. 231–240. Also pp. 168–172 in *Workshop Notes of the AAAI-92 Workshop on Integrating Neural and Symbolic Processes.*

[86] Myllymäki, P., and Tirri, H. Bayesian case-based reasoning with neural networks. In *Proceedings of the IEEE International Conference on Neural Networks* (San Francisco, March 1993), vol. 1, IEEE, Piscataway, NJ, pp. 422–427.

[87] Myllymäki, P., and Tirri, H. Massively parallel case-based reasoning with probabilistic similarity metrics. In *Topics in Case-Based Reasoning*, S. Wess, K.-D. Althoff, and M. Richter, Eds., vol. 837 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1994, pp. 144–154.

[88] Myllymäki, P., and Tirri, H. Constructing computationally efficient Bayesian models via unsupervised clustering. In *Probabilistic*

Reasoning and Bayesian Belief Networks, A.Gammerman, Ed. Alfred Waller Publishers, Suffolk, 1995, pp. 237–248.

[89] MYLLYMÄKI, P., TIRRI, H., FLORÉEN, P., AND ORPONEN, P. Compiling high-level specifications into neural networks. In *Proceedings of the International Joint Conference on Neural Networks* (Washington D.C., January 1990), vol. 2, IEEE, New York, NY, pp. 475–478.

[90] NEAL, R. Connectionist learning of belief networks. *Artificial Intelligence 56* (1992), 71–113.

[91] NEAL, R. Probabilistic inference using Markov chain Monte Carlo methods. Tech. Rep. CRG-TR-93-1, University of Toronto, September 1993.

[92] NEAPOLITAN, R. *Probabilistic Reasoning in Expert Systems.* John Wiley & Sons, New York, NY, 1990.

[93] NEWQUIST, H. Struggling to maintain. *AI Expert 8*, 3 (1988), 69–71.

[94] NISHIKAWA, T. Fuzzy theory – the science of human intuition. *Japan Computer Quarterly 79* (1989).

[95] ORPONEN, P., FLORÉEN, P., MYLLYMÄKI, P., AND TIRRI, H. A neural implementation of conceptual hierarchies with Bayesian reasoning. In *Proceedings of the International Joint Conference on Neural Networks* (San Diego, CA, June 1990), vol. 1, IEEE, New York, NY, pp. 297–303.

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

[97] PETERSON, C., AND ANDERSON, J. A mean field theory learning algorithm for neural networks. *Complex Systems 1* (1987), 995–1019.

[98] PILARSKI, S., AND KAMEDA, T. Simple bounds on the convergence rate of an ergodic Markov chain. *Information Processing Letters 45* (1993), 81–87.

[99] POLLITZER, E. Engineering cognitive systems: Japan's real-world computing programme. *AI Communications 5*, 2 (June 1992), 56–61.

[100] Rajasekaran, S., and Reif, J. H. Nested annealing: A provable improvement to simulated annealing. *Theoretical Computer Science 99* (1992), 157–176.

[101] Rego, V. Naive asymptotics for hitting time bounds in Markov chains. *Acta Informatica 29* (1992), 579–594.

[102] Roth, D. On the hardness of approximate reasoning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (1993), vol. 1, pp. 613–618.

[103] Rumelhart, D., and McClelland, J., Eds. *Parallel Distributed Processing*, vol. 1. MIT Press, Cambridge, MA, 1986.

[104] Santos, S. Phase transitions in sparsely connected Boltzmann machines. Tech. Rep. C-1994-15, University of Helsinki, Department of Computer Science, 1994.

[105] Selim, S., and Alsultan, K. A simulated annealing algorithm for the clustering problem. *Pattern Recognition 24*, 10 (1991), 1003–1008.

[106] Shachter, R. Probabilistic inference and influence diagrams. *Operations Research 36*, 4 (July-August 1988), 589–604.

[107] Shachter, R. Evidence absorption and propagation through evidence reversals. In Henrion et al. [53], pp. 173–190.

[108] Shafer, G., and Pearl, J., Eds. *Readings in Uncertain Reasoning.* Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[109] Shastri, L. *Semantic Networks: An Evidential Formalization and Its Connectionist Realization.* Pitman, London, 1988.

[110] Shawe-taylor, J., and Zerovnik, J. Generalized Boltzmann machines. Tech. Rep. CSD-TR-92-29, Department of Computer Science, Royal Holloway, University of London, 1992.

[111] Shimony, S. Cost-based abduction and MAP explanation. *Artificial Intelligence 66* (1994), 345–374.

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

[113] Sinclair, A., and Jerrum, M. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation 82* (1989), 93–133.

[114] Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart and McClelland [103], pp. 194–281.

[115] Spiegelhalter, D. Probabilistic reasoning in predictive expert systems. In Kanal and Lemmer [68], pp. 47–67.

[116] Sun, R. *Integrating Rules and Connectionism for Robust Commonsense Reasoning.* John Wiley & Sons, Chichester, 1994.

[117] Wellman, M. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence 44*, 3 (August 1990), 257–304.

[118] York, J. Use of Gibbs sampler in expert systems. *Artificial Intelligence 56* (1992), 115–130. Addendum in pp. 397–398 of the same volume.

[119] Zadeh, L. Fuzzy logic and approximate reasoning. *Synthese 30* (1975), 407–425.

ISSN 1238-8645
ISBN 951-45-7211-4
Helsinki 1995
Yliopistopaino