Local Search in Combinatorial Optimization
Edited by E. Aarts and J. K. Lenstra

7

Artificial neural networks

Carsten Peterson, Bo Söderberg
University of Lund, Lund

1 INTRODUCTION 173
2 ARTIFICIAL NEURAL NETWORKS 174
2.1 Background 174
2.2 Basic ANN ingredients 175
2.3 Feedforward networks 177
2.4 Feedback networks 178
3 PURE ANN APPROACH TO OPTIMIZATION: BINARY CASE 181
3.1 Basic encoding 181
3.2 Minimizing E 181
3.3 The mean field equations 182
3.4 The mean field neural approach 183
3.5 Analysis of the dynamics 184
3.6 The graph bisection problem 185
4 OPTIMIZATION WITH POTTS NEURAL NETWORKS 186
4.1 Potts spins 187
4.2 Potts mean field equations 188
4.3 Mean field dynamics 189
4.4 A black-box procedure 191
4.5 The graph partitioning problem 191
4.6 The traveling salesman problem 193
4.7 Scheduling problems 194
5 OPTIMIZATION WITH INEQUALITY CONSTRAINTS 198
5.1 Dealing with inequality constraints 199
5.2 The knapsack problem 199
6 DEFORMABLE TEMPLATES 204
6.1 The traveling salesman problem 204
6.2 Track finding 207
7 ROTOR NEURONS 210
7.1 Mean field rotor neurons 210
7.2 Applications 211
8 SUMMARY AND OUTLOOK 211

1 INTRODUCTION

The use of artificial neural networks (ANNs) to find good solutions to combinatorial optimization problems [Hopfield & Tank, 1985; Peterson & Söderberg, 1989; Durbin & Willshaw, 1987; Peterson, 1990a] has recently caught some

attention. Whereas the use of ANNs for pattern recognition and prediction problems constitutes a nonlinear extension of conventional linear interpolation/extrapolation methods, ANNs in the optimization domain really bring something new to the table. In contrast to existing search and heuristics methods, the ANN approach does not fully or partly explore the different possible configurations. Rather it 'feels' its way in a fuzzy manner towards good solutions. This is done in a way that allows for a statistical interpretation of the results. The ANN approach is therefore conceptionally and technically very different from conventional approaches. Two basic steps are involved when using ANNs to find good solutions to combinatorial optimization problems:

- Formulate the problem as the minimization of a feedback ANN energy function $E(s_1, \ldots, s_N)$, where the neurons s_i encode possible solutions.
- Find an approximate solution by iteratively solving the corresponding mean field (MF) equations.

This procedure often produces high-quality solutions, as will be demonstrated below. The neural approach has the additional advantage that the MF equations are isomorphic to VLSI (very large scale integration) RC equations, which makes hardware implementations straightforward. It is rare to have such tight bonds between algorithms and hardware.

This chapter aims to give a self-contained introduction to the use of ANNs for finding good approximate solutions to combinatorial optimization problems. Each theoretical description is illustrated by one or more examples. The chapter is organized as follows. Section 2 gives a general introduction to both feedforward and feedback networks. Section 3 discusses how to map an optimization problems onto a system of binary (Ising) spins, derives the corresponding mean field (MF) equations, and analyzes the resulting neural dynamics. The approach is illustrated with the graph bisection problem. An analogous path is followed in Section 4 for the more general multistate (Potts) neurons. Here the examples are graph partitioning, the traveling salesman, and scheduling problems. Optimization problems with inequality constraints require special care, dealt with in Section 5. An alternative procedure for low-dimensional geometrical problems, the deformable templates approach, is discussed in Section 6. This method is illustrated with the traveling salesman and track finding problems. In Section 7 we discuss the generalization to rotor neurons in order to deal with the optimization of systems with angular variables. Section 8 contains a brief summary and outlook.

2 ARTIFICIAL NEURAL NETWORKS

2.1 Background

The introduction of artificial neural networks was inspired by the structure of biological neural networks and their way of encoding and solving problems. The human brain contains approximately 10^{12} *neurons*. They can be of many different types, but most of them have the same general structure. The *cell body* or *soma*

receives electric input signals to the *dendrites*, signals carried by ions. The interior of the cell body is negatively charged against a surrounding *extracellular fluid*. Signals arriving at the dendrites depolarize the resting potential, enabling Na^+ ions to enter the cell through the membrane, producing an electric discharge from the neuron – the neuron fires. The accumulated effect of several simultaneous signals arriving at the dendrites is usually almost linearly additive, whereas the resulting output is a strongly nonlinear, all-or-none type process. The discharge propagates along the *axon* to a *synaptic junction*, where *neurotransmitters* travel across a *synaptic cleft* and reach the dendrites of the postsynaptic neuron. A synapse that repeatedly triggers the activation of a postsynaptic neuron will grow in strength; others will gradually weaken. This *plasticity*, which is known as the *Hebb rule*, plays a key part in *learning*.

The *connectivity* (number of neurons connected to a neuron) varies from ~ 1 to $\sim 10^5$. For the *cerebral cortex* $\sim 10^3$ is an average. This corresponds to $\sim 10^{15}$ synapses per brain. Synapses can be either *excitatory* or *inhibitory* of varying strength. In the simplified binary case of just two states per synapse the brain thus has $\sim 2^{10^{15}}$ possible configurations! The neural network consequently stands in sharp contrast to a von Neumann computer both with respect to architecture and functionality.

The von Neumann computer was originally developed for 'heavy duty' numerical computing, but has later also turned out to be profitable for data handling, word processing and the like. However, when it comes to matching the vertebrate brain in terms of performing 'human' tasks, it has very strong limitations. There are therefore strong reasons to design an architecture and an algorithm that show more resemblance to the vertebrate brain.

The philosophy of the ANN approach is to abstract some key biological ingredients from which to construct simple mathematical models that exhibit most of the appealing features we have just considered; for a general textbook on ANN see Hertz, Krogh & Palmer [1991]. In physics one has good experience of model building out of major abstractions. For example, details of individual atoms in a solid can be lumped into effective 'spin' variables in such a way that a good description of *collective* phenomena (phase transitions, etc.) is obtained. It is indeed the collective behavior of the neurons that is interesting from the point of view of intelligent data processing.

2.2 Basic ANN ingredients

The basic computational entities of an ANN are the neurons v_i, which normally can take real values within the interval $[0, 1]$ (or $[-1, 1]$); $i = 1, 2, \ldots, N$ is an index labeling the N individual neurons. Sometimes even simpler, discrete neurons s_i are used, with $s_i \in \{0, 1\}$ (or $\{-1, 1\}$), simplifications of the biological neurons described above. Common to most neural models is a *local updating rule*

$$v_i = g\left(\sum_{j=1}^{N} \omega_{ij} v_j - \theta_i\right), \tag{1}$$

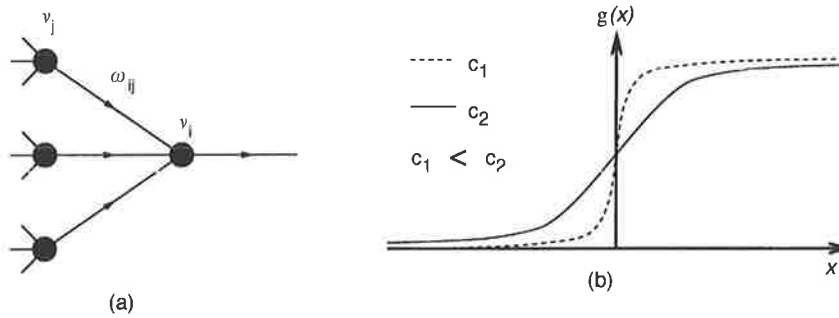

Figure 7.1 (a) Neuron updating and (b) sigmoid response functions of equations (1) and (2) for different temperatures c

where the weights (synapses) $\omega_{ij} \in \mathbb{R}$ are nonzero only for neurons v_j that feed to the neuron v_i. These weights can have both positive (excitatory) and negative (inhibitory) values. The θ_i term is a threshold, corresponding to the membrane potential in a biological neuron. The nonlinear *transfer function* g: $\mathbb{R} \to [0, 1]$ is typically a sigmoid-shaped function like

$$g(x) = \tfrac{1}{2}(1 + \tanh(x/c)), \tag{2}$$

where the 'temperature' $c > 0$ sets the inverse gain: a lower temperature gives a steeper transfer function (Figure 7.1(b)). The limit $c \to 0$ produces a step function corresponding to discrete neurons. This simple artificial neuron mimics the main features of real biological neurons in terms of linear additivity for the inputs and strong nonlinearity for the resulting output. If the integrated input signal is larger than a certain threshold θ_i, the neuron will fire. There are two different kinds of architecture in neural network modeling: feedforward (Figure 7.2(a)) and and feedback (Figure 7.2(b)); we will describe them below.

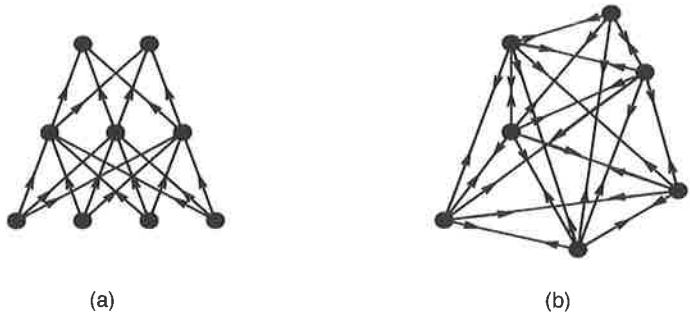

Figure 7.2 (a) Feedforward and (b) feedback architectures

2.3 Feedforward networks

Feedforward networks process signals in a one-way manner, from a set of input units in the bottom to output units in the top, layer by layer using the local updating rule (1). Feedback networks allow information to go both ways – the synapses are bidirectional. How can the power of these networks be exploited? Feedforward networks have two major areas of application, somewhat overlapping; they are feature recognition and function approximation.

Feature recognition

Feature recognition (or pattern classification) is about categorizing input patterns $\mathbf{x} \in \mathbb{R}^{N_i}$ in terms of different features $o_1, \ldots, o_{N_f}$. An input pattern $\mathbf{x} = (x_1, x_2, \ldots, x_{N_i})$ is fed into the input layer (receptors), and the output nodes represent the features. For the architecture in Figure 7.2(a), with one intermediate (or hidden) layer of N_h neurons, o_i depends on $\mathbf{x}$ (cf. (1)) as

$$o_i(\mathbf{x}) = g\left(\sum_{j=1}^{N_h} \omega_{ij}^{(2)} g\left(\sum_{k=1}^{N_i} \omega_{jk}^{(1)} x_k\right)\right), \quad i = 1, \ldots, N_f, \tag{3}$$

where $\omega_{ij}^{(l)}$ are the weight parameters between layers l and $l+1$. Equation (3) may be generalized to any number of layers. (The thresholds, or bias terms, θ_i appearing in (1) have in (3) been transformed into weights by adding to each layer an extra dummy unit, which is constantly equal to 1.)

Fitting $\omega_{ij}^{(l)}$ to a given set of input patterns (learning) takes place with gradient descent on a suitable *error function*. In this process the *training patterns* are presented over and over again with successive adjustments of the weights – *back-propagation* of the error [Rumelhart & McClelland, 1986]. Once this iterative learning has reached an acceptable level, in terms of a low error rate, the weights are frozen and the ANN is ready to be used on patterns it has never seen before. The capability of the network to correctly characterize these *test patterns* is called *generalization* performance. This procedure is somewhat analogous to 'normal' curve fitting, where a smooth parameterization from a training is used to interpolate between the data points (generalization).

Function approximation

Rather than having a 'logical' output unit (o_i) with a threshold behavior described by (1) and (2), one could imagine having an output representing an unrestricted real number, obtained e.g. by replacing the sigmoid in (1) and (2) with a linear transfer function for the output units. In this case one adjusts the weight parameters to parameterize an unknown real-valued function. Such an approach can be useful in *time-series predictions*, where one aims at predicting future values of a series given previous values [Lapedes & Farber, 1987; Weigend, Huberman & Rumelhart, 1990], e.g.

$$x_t = \mathcal{F}(x_{t-1}, x_{t-2}, \ldots),$$

where x_t is the real-valued output node and $x_{t-1}, x_{t-2}, \ldots$ are the values of the series at previous times.

2.4 Feedback networks

In contrast to the feedforward networks, the activation in feedback networks continues until a steady state has been reached. Feedback networks appear in the context of associative memories using the Hopfield model [Hopfield, 1982] and difficult optimization problems [Hopfield & Tank, 1985; Peterson & Söderberg, 1989], which is the focus of this chapter, but also in feature recognition applications using the Boltzmann machine [Ackley, Hinton & Sejnowski, 1985] and its mean field approximation [Peterson & Hartman, 1989]. Simple models for magnetic systems have a lot in common with feedback networks, so they have been the source of much inspiration. We therefore start this section by familiarizing the reader with *Ising models* for magnetic systems.

Magnetic systems

The *Ising model* describes a magnetic system in terms of a set of binary spins $s_i \in \{-1, 1\}$, $i = 1, \ldots, N$, which are effective variables for the individual atoms, assumed to be positioned on a lattice. The two spin states at each site i represent the possible magnetization directions. The model is governed by an energy function $E(s)$ (from here on we use the convention that a symbol stripped of its indices represents the whole collection of variables denoted by the indexed symbol), given by

$$E(s) = -\frac{J}{2} \sum_{\langle i,j \rangle} s_i s_j, \tag{4}$$

where the sum runs over pairs i, j of neighboring sites. Thus the neighboring spins interact via a constant attractive coupling of strength $J > 0$. The lowest energy state is reached by iterative updating of the spins according to

$$s_i = \operatorname{sgn}\left(J \sum_{\langle j \rangle_i} s_j \right), \tag{5}$$

where the sum runs over the neighbors j of i, and sgn is the algebraic sign function. Eventually, a state is reached where all spins point in one of the two possible directions (Figure 7.3(a)). If the system is embedded in a thermal environment (Figure 7.3(a) characterizes $c = 0$), fluctuations will appear subject to the Boltzmann distribution

$$\mathbb{P}(s) \propto \exp(-E(s)/c),$$

which can be simulated by replacing the dynamics of (5) by some stochastic procedure leading to fluctuating configurations (Figure 7.3(b)).

For large N the behavior of the *average magnetization* $M = \langle \bar{s} \rangle$, $\bar{s} = \sum_{i=1}^{N} s_i / N$, depends on c. Above a phase transition point, $\hat{c}$, there is no global alignment, and

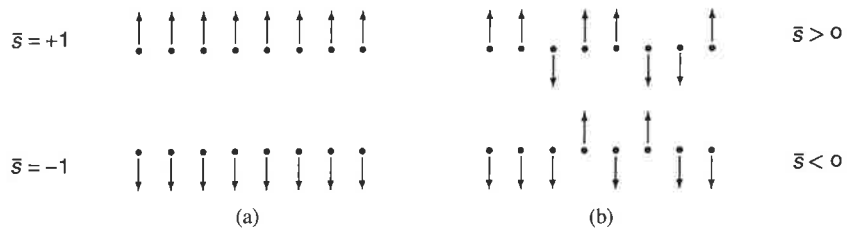

Figure 7.3 (a) The two possible $c = 0$ states for the Ising model and (b) two typical $c > 0$ configurations

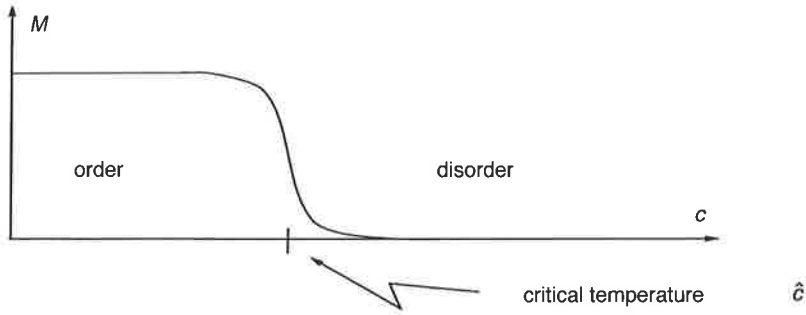

Figure 7.4 Average magnetization M as a function of c, illustrating a phase transition

$M = 0$. At very high temperatures there is no alignment whatsoever – all the spins are completely random. A phase transition between an *ordered* phase and a *disordered* phase is depicted in Figure 7.4. Indicators of phase transitions such as M (order parameters) represent *global* properties of the system. Following the evolution of individual spins does not tell us very much. The transition into an ordered phase will play an important role in the applications of feedback networks described below; this is because information is related to order.

Similar behavior can be found in a more realistic modeling of magnets. It is interesting that the effective description given by the spin models suffices for capturing the global properties of such magnetic systems; all atomic physics details are lumped into *effective* spin variables and their energy function E.

An interesting generalization of the Ising model is a *spin glass* system (which can model certain alloys like AuFe), obtained by allowing for:

- nonlocal interactions, i.e. terms $\propto s_i s_j$ in E for all $j \neq i$;
- pair-dependent (symmetric) couplings $\omega_{ij} = \omega_{ji}$ between s_i and s_j.

Thus (4) is replaced by

$$E(s) = -\frac{1}{2} \sum_{i \neq j = 1}^{N} \omega_{ij} s_i s_j. \tag{6}$$

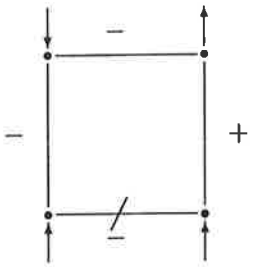

Figure 7.5 Frustration: four spins connected by differently signed couplings

Allowing also for negative couplings produces *frustration*, a situation with conflicting interests (Figure 7.5). Not all the energy terms can be minimized simultaneously, which leads to many almost degenerate 'ground states'. For a Gaussian random distribution of couplings ω_{ij} there exist $\approx e^{0.2N}$ such low energy states for an N spin system. This suggests the possibility of exploiting such a system for information technology – a feedback network.

The Hopfield model

The Hopfield model [Hopfield, 1982] is based on the energy function of (6) with binary neurons $s_i = \pm 1$. By appropriate choice of ω_{ij} the idea is to let the system function as an associative memory. A dynamics that locally minimizes (6) (cf. (5)) is given by

$$s_i = \operatorname{sgn}\left(\sum_{j \neq i} \omega_{ij} s_j\right). \tag{7}$$

Given a set of N_p patterns $\mathbf{x}^{(p)} = (x_1^{(p)}, x_2^{(p)}, \ldots, x_N^{(p)})$, $p = 1, \ldots, N_p$, with $x_i^{(p)} \in \{-1, 1\}$, the *Hebb rule* [Hebb, 1949]

$$\omega_{ij} = \frac{1}{N_p} \sum_{p=1}^{N_p} x_i^{(p)} x_j^{(p)}$$

is used for learning. With this choice of ω_{ij} one can show [Hopfield, 1982] that, under certain conditions and when initiated at some starting value $s_i^{(0)}$, the updating rule of (7) brings the system to the closest stored pattern $x_i^{(p)}$, which is a local minimum of E. One has an associative memory.

Combinatorial optimization

Many combinatorial optimization problems are known to be NP-complete. Exact solutions to such problems require state-space exploration of one kind or another, leading to a growth of the computational effort with the size of the system that is exponential or even factorial. Different kinds of heuristic method are therefore often used to find reasonably good approximate solutions. The ANN approach, which is based on feedback networks, falls within this category.

It has advantages in terms of solution quality and a 'fuzzy' interpretation of the answers through the MF variables. Furthermore, it is inherently parallel, facilitating implementations on concurrent processors, and custom-made hardware is straightforward to design for MF equations. The ANN approach differs from optimization and most other approximation methods in the sense that it has no trial-and-error mechanism; it 'feels' its way to a good solution.

There are two families of ANN algorithms for optimization problems:

- The *'pure' neural approach* is based on a system of either binary [Hopfield & Tank, 1985] or multistate [Peterson & Söderberg, 1989] neurons with mean field equations for the dynamics; it is a very general approach.
- *Hybrid approaches*, such as 'deformable template' algorithms [Durbin & Willshaw, 1987], supplement the neural variables with problem-specific variables, e.g. geometric variables.

Hybrid approaches are appropriate for low-dimensional geometrical problems like the traveling salesman problem (TSP), whereas the more general approach of pure neural networks is suitable for generic multiple-choice problems, like assignment or scheduling problems. The following sections describe both approaches illustrated by relevant applications.

3 PURE ANN APPROACH TO OPTIMIZATION: BINARY CASE

3.1 Basic encoding

In the simplest neural approach to an optimization problem we assume that the problem at hand can be formulated in terms of a system of Ising (binary) spins $s_i = \pm 1$, $i = 1, \ldots, N$, as the minimization of an objective (energy) function $E(s)$. This may take the form of (6),

$$E(s) = -\frac{1}{2} \sum_{i \neq j = 1}^{N} \omega_{ij} s_i s_j, \tag{8}$$

with a suitable choice of couplings ω_{ij} so as to represent the particular problem. Note that in this approach the couplings (or, more generally, the parameters of $E(s)$) are fixed once and for all for each problem instance; they are not adaptive as in some other domains of ANN application.

No approximations have yet been made; everything so far amounts to a particular mathematical encoding of the problem.

3.2 Minimizing E

The next step will be to devise an efficient procedure for minimizing $E(s)$, such that spurious local minima are avoided as much as possible. In the neural approach, this is done by employing the *mean field* (MF) approximation, to be defined below. It will turn out to be very powerful in this respect. Before arriving at the MF equations, we will touch upon some related approaches.

A straightforward method for minimizing E is to update s_i according to a local optimization rule,

$$s_i = \operatorname{sgn}\left(\sum_j \omega_{ij} s_j\right). \tag{9}$$

With this procedure the system typically ends up in a local minimum close to the starting point, which is not desired here.

Instead, a stochastic algorithm might be employed, which allows for uphill moves. One such possibility is *simulated annealing* (SA) [Kirkpatrick, Gelatt & Vecchi, 1983]; see also Chapter 4. This method consists of generating configurations via neighborhood search methods according to the *Boltzmann distribution*

$$\mathbb{P}(s) = \frac{1}{Z} e^{-E(s)/c},$$

where Z is the *partition function*

$$Z = \sum_s e^{-E(s)/c}. \tag{10}$$

Here, $c > 0$ is the temperature of the system, and the sum runs over the allowed values ± 1 of all the spins s_i. When c is small, the distribution should be concentrated around the global energy minimum. By starting at a finite c and generating configurations at successively lower c (annealing), the final configuration is less likely to get stuck in a bad local minimum. But this procedure can be very time-consuming.

3.3 The mean field equations

The MF approach aims at approximating the stochastic SA method with a set of deterministic equations. The derivation has two steps. First, the partition function of (10) is rewritten in terms of an integral over new continuous variables u_i and v_i. Second, Z is approximated by the maximum value of its integrand.

To this end, embed the spins s_i in a linear space $\mathbb{R}$, introduce a new set of variables v_i living in this space, one for each spin, and set them equal to the spins with Dirac delta functions. Then we can express the energy in terms of the v_i, and Z takes the form

$$Z = \sum_s \int dv\, e^{-E(v)/c} \prod_i \delta(s_i - v_i).$$

Next Fourier expand the delta functions, introducing a set of conjugate variables u_i to produce

$$Z \propto \sum_s \int dv \int du\, e^{-E(v)/c} \prod_i e^{u_i(s_i - v_i)}.$$

Finally, carry out the original sum over s:

$$Z \propto \int dv \int du e^{-E(v)/c - \sum_i u_i v_i + \sum_i \log\cosh u_i} \equiv \int dv \int du e^{-E_{\text{eff}}(u,v)/c}. \tag{11}$$

The original partition function is now rewritten entirely in terms of the new variables u_i, v_i, with an effective energy $E_{\text{eff}}(u, v)$ in the exponent. So far no approximation has been made. We next assume that Z in (11) can be approximated by the contribution from the maximal value of the integrand, the saddle-point approximation. For the position of the saddle point we obtain

$$u_i = -\frac{\partial E(v)}{\partial v_i} \Big/ c, \tag{12}$$

$$v_i = \tanh u_i. \tag{13}$$

Combining (12) and (13) we obtain the *mean field equations*

$$v_i = \tanh\left(-\frac{\partial E(v)}{\partial v_i} \Big/ c \right). \tag{14}$$

For the energy of (8) this gives the familiar

$$v_i = \tanh\left(\sum_j \omega_{ij} v_j / c \right).$$

The *mean field variables* v_i can be interpreted as (approximate) thermal averages $\langle s_i \rangle_c$ of the original binary spins. We thus recover the local updating equations (1) and (2). What we have obtained is a set of deterministic equations emulating the stochastic behavior. They correspond to an approximation where each spin feels the others only via their averages.

An alternative derivation of the MF equations can be done in a variational approach, where the free energy is minimized with respect to the coefficients of a linear Ansatz for an approximate energy.

3.4 The mean field neural approach

The MF equations define a feedback neural network, if v_i is interpreted as a neuron with u_i as its input, acted upon by the transfer function $\tanh(\cdot)$. The neural approach for binary systems consists in solving the MF equations (14) *iteratively*, either synchronously (in parallel) or serially (one v_i at a time), starting at a finite c and slowly letting $c \to 0$ (annealing).

For this approach to work well in the low c limit, it is desirable that $E(s)$ contain no self-couplings of the form $s_i s_i$, or more generally, that $\partial E / \partial s_i$ not depend on s_i. This can always be arranged (e.g., $s_i s_i$ can always be replaced by 1). Then in the $c \to 0$ limit, $\tanh(\cdot/c)$, will turn into a step function, and (14) will reduce to local optimization of the original spin system (cf. (9)).

The basic advantage of the MF approach is that, at finite c, the mean field variables can evolve through a continuous space not accessible to the original, discrete spin variables.

3.5 Analysis of the dynamics

The dynamics of this kind of feedback ANN typically exhibits a behavior with two phases (similar to Figure 7.4): at large temperatures the system relaxes into a trivial fixpoint, which is zero for the quadratic objective function (8). As the temperature is lowered, a 'phase transition' occurs at a critical temperature $c = \hat{c}$, where the trivial fixpoint turns unstable, and as $c \to 0$ nontrivial fixpoints $v_i^{(*)} = \pm 1$ emerge, representing a suggestive solution to the optimization problem in question (Figure 7.6).

The position of $\hat{c}$ depends on ω_{ij}, and can be calculated by expanding the sigmoid function $\tanh(\cdot/c)$ in a power series around the origin (Figure 7.6). In this approximation the dynamics is linear, and the v_i evolve according to

$$v_i = \frac{1}{c} \sum_j \omega_{ij} v_j. \tag{15}$$

For *synchronous updating* it is clear that, if the modulus of one of the eigenvalues of the matrix ω/c in (15) is greater than 1, the trivial fixpoint becomes unstable and the system can begin exploring the nonlinear region. This happens when c decreases past the larger of (a) the largest positive eigenvalue of ω, or (b) minus the largest negative eigenvalue of ω. Case (a) is desirable, since case (b) leads to oscillating behavior due to a negative destabilizing eigenvalue. To enforce case (a), a positive self-coupling is sometimes needed to enable synchronous updating; this might overstabilize the dynamics at low c, leading to lower-quality solutions [Peterson & Söderberg, 1989].

In the case of *serial updating* the philosophy is the same but the analysis slightly more complicated; we refer the reader to Peterson & Söderberg [1989] for a more detailed discussion. The result is simple though (and encouraging): $\hat{c}$ is always

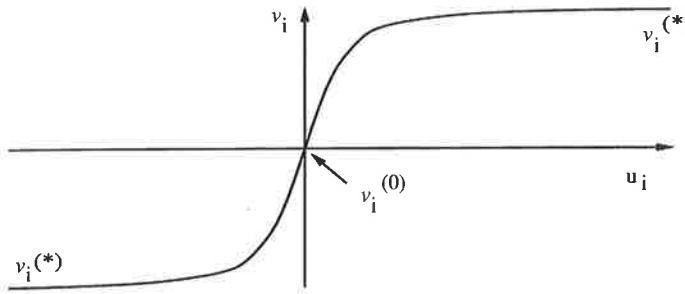

Figure 7.6 Fixpoints and $\tanh u_i$

given by the largest positive eigenvalue of ω, which corresponds to case (a). No self-coupling is needed to stabilize the dynamics.

Finding the largest eigenvalue presents no computational difficulty. For problems where the coupling matrix contains random parts, knowledge of the average coupling and the corresponding standard deviation often suffices to produce a good estimate of $\hat{c}$. Such an analysis is also important in foreseeing and avoiding oscillatory behavior with synchronous updating.

Given a method of estimating $\hat{c}$ in advance, one can devise a reliable, parallelizable 'black-box' algorithm for solving problems of this kind (cf. Section 4.4).

3.6 The graph bisection problem

An application of the approach described above is the graph bisection (GB) problem. The neural approach is very transparent in this case, since the problem has an intrinsic binary structure. The GB problem is defined as follows (Figure 7.7(a)). Partition the N nodes of a given graph into two subsets of equal size ($N/2$) such that the number of connections between the two halves, the cutsize, is minimized. The graph is defined by the connection matrix, J_{ij}, $i,j = 1,\ldots,N$, where J_{ij} is 1 if nodes i and j are connected, and 0 otherwise.

The problem is mapped onto an Ising spin system by the following representation. For each node i, assign a binary neuron $s_i = \pm 1$, signifying whether the node belongs to one half or the other. Then $s_i s_j = 1$ whenever i and j are in the same partition, and -1 otherwise. Neglecting an unimportant additive constant, the cutsize is then proportional to

$$E_{\text{conn}} = -\frac{1}{2}\sum_{i,j=1}^{N} J_{ij} s_i s_j.$$

However, minimization of E_{conn} alone will lead to the result that all nodes are forced into one partition. To enforce the *global constraint* of even partition, we need a *constraint term* that penalizes situations where the nodes are not evenly partitioned. Since $\sum_i s_i = 0$ precisely when the partition is balanced, a term

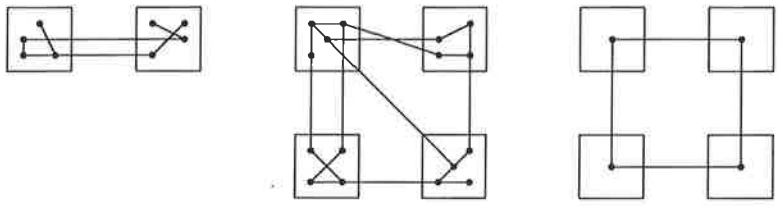

Figure 7.7 (a) A graph bisection problem; (b) a graph partitioning problem with $K = 4$; and (c) a TSP with $N = 4$

proportional to $(\sum_i s_i)^2$ will obviously increase the energy for an unbalanced partition. After a final removal of diagonal terms, a possible energy function for graph bisection takes the form

$$E(s) = -\frac{1}{2}\sum_{i,j=1}^{N} J_{ij} s_i s_j + \frac{\alpha}{2}\left(\left(\sum_{i=1}^{N} s_i\right)^2 - \sum_{i=1}^{N} s_i^2\right), \tag{16}$$

where the *constraint coefficient* $\alpha > 0$ sets the relative strength between the cutsize and constraint terms. Note that the energy in (16) has precisely the form of (6) with $\omega_{ij} = J_{ij} + \alpha(\delta_{ij} - 1)$. The corresponding MF equations are

$$v_i = \tanh\left(\frac{1}{c}\sum_{j=1(j\neq i)}^{N} (J_{ij} - \alpha) v_j\right),$$

which are to be iterated, serially or in parallel, under annealing in c, until convergence.

Note the generic form of (16),

$$E = \text{'cost'} + \text{'global constraint'},$$

which is typical when casting combinatorial optimization problems onto neural networks. The difficulty inherent in this kind of problem is very transparent here: the system is frustrated in the sense that the two terms ('cost' and 'global constraint') are competing with each other. Just as for spin glasses, such frustrations often lead to many local minima.

Our treatment of constraints resembles penalty function methods such as Lagrangian relaxation, but differs from straightforward heuristic approaches. In the case of graph bisection one often starts in a configuration where the nodes are equally partitioned and then proceeds by swapping pairs subject to some acceptance criteria; the constraint of equal partition is respected throughout the updating process. This is in contrast to the neural network technique, where the constraints are implemented in a 'soft' manner by a penalty term. The final MF solutions could therefore sometimes suffer from a minor imbalance. This is easily remedied, either by applying a *greedy heuristic* to the solutions, or by reheating the system and letting it reanneal.

Peterson & Anderson [1988] used serial updating to achieve good numerical results for the graph bisection problem on random graphs of a fixed connectivity and graph sizes N ranging from 20 to 2000. Their quality is comparable to solutions obtained by the time-demanding simulated annealing method. This technique has a time consumption lower than other methods, and it becomes more competitive if the intrinsic parallelism is exploited on dedicated hardware.

4 OPTIMIZATION WITH POTTS NEURAL NETWORKS

For a large group of combinatorial optimization problems, it is natural to choose an encoding in terms of binary elementary variables. However, there are many problems where this is not the case. One kind of complication arises when the

natural elementary decisions are of the type one-of-K with $K > 2$, rather than one-of-two.

Early attempts to approach such problems by neural network methods were in terms of *neuron multiplexing*, where for each elementary K-fold decision, K binary 0/1 neurons were used, with the additional constraint that precisely one of them be on (equal to 1). These *syntax* constraints were implemented in a soft way as penalty terms. As it turned out in the original work on the traveling salesman problem [Hopfield & Tank, 1985], as well as in subsequent investigations for the graph partitioning problem [Peterson & Söderberg, 1989], this approach does not produce high-quality solutions in a parameter-robust way.

As was demonstrated by Peterson & Söderberg [1989], an alternative encoding using *Potts neurons* (see Wu [1983]) is the way to go because the syntax constraint is built-in.

4.1 Potts spins

A K-state Potts spin is a variable that has K possible values (states). Such a variable can be represented in many ways; for our purposes, the best way is as a vector in the Euclidean space $\mathscr{E}_K$. Thus, denoting a spin variable by $\mathbf{s} = (s_1, s_2, \ldots, s_K)$, the ath possible state is given by the ath principal unit vector, defined by $s_a = 1$, $s_b = 0$ for $b \neq a$. These vectors point to the corners of a regular K-simplex (Figure 7.8 shows the case of $K = 3$). They are all normalized and mutually orthogonal, and also fulfill $\sum_a s_a = 1$.

We assume that an optimization problem has been given, and encoded in terms of a set of N distinct K-state Potts spins $\mathbf{s}_i = (s_{i1}, \ldots, s_{iK})$, $i = 1, \ldots, N$, together with an objective function $E(\mathbf{s})$ to be minimized. Often $E(\mathbf{s})$ can be written in the form

$$E(\mathbf{s}) = -\frac{1}{2} \sum_{i,j=1}^{N} \omega_{ij} \mathbf{s}_i \cdot \mathbf{s}_j. \tag{17}$$

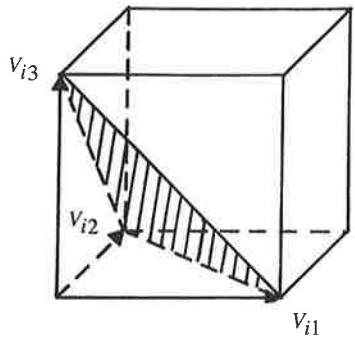

Figure 7.8 The volume of solutions corresponding to the neuron-multiplexing encoding for $K = 3$; the shaded plane corresponds to the solution space of the corresponding Potts encoding

As in the case of Ising spins, it is desirable (and always possible) to choose $E(\mathbf{s})$ not to contain any self-couplings (i.e., $\omega_{ii} = 0$, $i = 1, \ldots, N$).

4.2 Potts mean field equations

We next derive the MF equations corresponding to K-state Potts spins. Again we start off with the partition function

$$Z = \sum_{\mathbf{s}} \mathrm{e}^{-E(\mathbf{s})/c},$$

where the sum runs over the allowed states of the whole set of Potts spins. Proceeding along the same lines as the binary case, the partition function is transformed into

$$Z \propto \int d\mathbf{u} d\mathbf{v} \exp\left(-E(\mathbf{v})/c - \sum_i \mathbf{u}_i \cdot \mathbf{v}_i \right) \sum_{\mathbf{s}} \exp\left(\sum_i \mathbf{u}_i \cdot \mathbf{s}_i \right).$$

Performing the $\mathbf{s}_i$ sums, we are left with

$$Z \propto \int d\mathbf{u} d\mathbf{v} \exp\left(-E(\mathbf{v})/c - \sum_i \mathbf{u}_i \cdot \mathbf{v}_i + \sum_i \log \sum_a \mathrm{e}^{u_{ia}} \right). \tag{18}$$

The MF approximation to $\langle \mathbf{s}_i \rangle$ is again given by the value of $\mathbf{v}_i$ at a saddle point of the effective energy of $(\mathbf{u}, \mathbf{v})$ in the exponent of (18). Differentiating, we obtain the Potts MF equations (cf. (12), (13)):

$$u_{ia} = -\frac{\partial E(\mathbf{v})}{\partial v_{ia}} \Big/ c, \tag{19}$$

$$v_{ia} = \frac{\mathrm{e}^{u_{ia}}}{\sum_b \mathrm{e}^{u_{ib}}}, \tag{20}$$

where indices a and b denote vector components.

The set of *Potts neurons* $\mathbf{v}_i$ defines a Potts neural network, with the dynamics given by iteration of (19) and (20), Equation (20) defines a kind of vector sigmoid function, acting on the input $\mathbf{u}_i$, and it follows trivially that

$$v_{ia} > 0, \sum_a v_{ia} = 1.$$

One can thus think of the neuron component v_{ia} as the probability for the ith Potts spin to be in state a (fuzzy logic). The state space available to $\mathbf{v}_i$ is the interior of the K-simplex spanned by the allowed states for the Potts spin $\mathbf{s}_i$. For $K = 3$ this is a triangular area (Figure 7.8).

In particular, for $K = 2$ we recover the formalism of the Ising case, provided that we make the following identification for the Ising mean field variable v_i:

$$v_i = v_{i1} - v_{i2},$$

equivalent to

$$v_{i1} = (1 + v_i)/2,$$
$$v_{i2} = (1 - v_i/2).$$

The Potts MF formalism thus provides a natural generalization of the MF approach from binary to general multistate systems.

4.3 Mean field dynamics

High temperature

Again one can analyze the linearized dynamics as in Section 3.5 in order to estimate the critical temperature $\hat{c}$. The value of the critical temperature depends on the couplings and on precisely how the updating is done. In *serial mode*, the Potts neurons are updated one by one, using fresh values of previously updated neurons. In *synchronous mode*, all neurons are updated in parallel, using only old values as input.

At large enough temperatures the system relaxes into a trivial fixpoint $v_{ia}^{(0)}$, which is a completely symmetric state, where all v_{ia} are equal:

$$v_{ia}^{(0)} = \frac{1}{K}. \tag{21}$$

As the temperature is lowered, a phase transition is passed at $c = \hat{c}$, and as $c \to 0$ nontrivial fixpoints $v_{ia}^{(*)}$ emerge which are characterized by $\Sigma \to 1$, where

$$\Sigma \equiv \frac{1}{N} \sum_{ia} v_{ia}^2$$

is the *saturation*, which measures the degree to which the neurons are forced into the corners of the accessible state space. Figure 7.9 illustrates this for a graph partitioning problem with $K = 4$, $N = 100$ (see Section 4.5). The trivial fixpoint corresponds to the symmetry point of the nonlinear transfer functions in (20), where the dynamics is almost linear (Figure 7.6). Let us consider fluctuations around the trivial fixpoint

$$v_{ia} = v_{ia}^{(0)} + \varepsilon_{ia}.$$

First, from (20) and (21) it follows that

$$\sum_a \varepsilon_{ia} = 0,$$

so the fluctuations will always be perpendicular to $(1, 1, 1, \ldots)$. In a linear approximation (suppressing the index a) the perpendicular components of ε_{ia} evolve according to

$$\varepsilon_i = \frac{1}{Kc} \sum_j \omega_{ij} \varepsilon_j$$

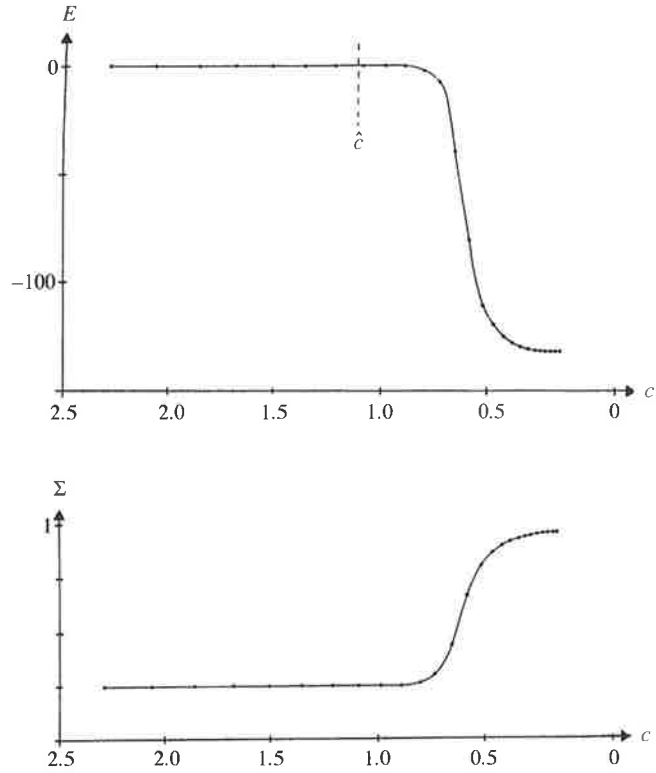

Figure 7.9 (a) Internal energy $E(v_{ia})$ as a function of c for a graph partitioning problem with $K = 4$ and $N = 100$; also shown is an approximation to $\hat{c}$ based on a statistical estimate; (b) the saturation $\sum$ as a function of c for the same problem.

(cf. (15)). This is almost identical to the situation for binary neurons. Thus, provided Kc is substituted for c, the discussion of $\hat{c}$ following (15) in Section 3 applies in every detail. The result of the analysis follows.

For *synchronous updating*, $\hat{c}$ is given by the larger of (a) $1/K$ (largest positive eigenvalue of ω) and (b) $-1/K$ (largest negative eigenvalue of ω). In (a) a positive eigenvalue is responsible for the destabilization of the trivial fixpoint; this is to be preferred. In (b), which pertains for some problems, a negative eigenvalue is involved, leading to oscillatory behavior. Case (b) can be avoided by adding an artificial self-coupling term to the objective function, but this can have the side effect of overstabilizing the dynamics at low c, leading to a lower expected solution quality.

For *serial updating*, we always have case (a), so this mode of updating is safe to use for all kinds of problems, and no self-coupling is needed.

Low temperature

At low temperature, any difference between the components u_{ia} of a neuron's input will be strongly magnified. As a result, we have a *winner-take-all* situation, where one neuron component will be almost one, and the others close to zero. The dynamics turns discrete, and a kind of local optimization results as $c \to 0$. Eventually the network should settle at a fixpoint close to an allowed state of the Potts spin system, representing a suggested solution to the optimization problem.

A self-coupling is sometimes needed with synchronous updating, to stabilize the high-c dynamics. Adding a self-coupling term, $(-\beta/2)\sum_{ia} v_{ia}^2$, has a clear impact at low c; a positive β value tends also to stabilize bad decisions (positive feedback), whereas a negative value destabilizes even a good solution, and can lead to cyclic or even chaotic behavior.

For serial mode, there is never a need for self-coupling, so serial mode will be understood where not otherwise stated. For some problems, implicit self-couplings are difficult to avoid; a general method to deal with such complications will be discussed in Section 5.

4.4 A black-box procedure

For most problems, the proper value of $\hat{c}$ can be calculated or estimated in advance. The complete neural network algorithm for a generic K-state Potts system (the Ising system is just a special case) will then look as follows.

A generic Potts neural network algorithm

1. Choose a problem instance $\Rightarrow \{\omega_{ij}\}$.
2. Calculate (or estimate) the phase transition temperature $\hat{c}$ by linearizing (20). (For synchronous updating: add a self-coupling $\beta\delta_{ij}$ to ω_{ij}, if necessary, and modify $\hat{c}$ accordingly.)
3. Initialize the neurons v_{ia} with $1/K \pm$ random values, and set $c = \hat{c}$.
4. Until $\Sigma \geqslant 0.99$, do:

   update all v_{ia}: $v_{ia} = e^{u_{ia}}/\sum_b e^{u_{ib}}$, with u_{ia} given by $(-\partial E(\mathbf{v})/\partial v_{ia})/c$;
   anneal: $c = 0.9 \times c$.

5. Finally, if needed to correct for violations of softly implemented constraints, perform a greedy heuristic on the obtained solution.

4.5 The graph partitioning problem

A generalization of the graph bisection problem is *graph partitioning* (GP): an N-node graph, defined by a symmetric connection matrix $J_{ij} = 0,1$, $i \neq j = 1, \ldots, N$, is to be partitioned into K subsets of N/K nodes each, while minimizing the cutsize, i.e., the number of connected node pairs winding up in different subsets (see Figure 7b).

Potts representation

This problem is naturally encoded in terms of K-state Potts spins as follows: For each node $i = 1, \ldots, N$, a Potts spin variable $\mathbf{s}_i = (s_{i1}, \ldots, s_{iK})$ is assigned, such that the spin component s_{ia} takes the value 1 or 0 depending on whether node i belongs to subset a or not. A suitable energy function (cf. (16)) is given by

$$E = -\frac{1}{2}\sum_{i,j=1}^{N} J_{ij}\mathbf{s}_i \cdot \mathbf{s}_j + \frac{\alpha}{2}\left(\left(\sum_{i=1}^{N}\mathbf{s}_i\right)^2 - \sum_{i=1}^{N}\mathbf{s}_i^2\right),$$

where the first term is a cost term (cutsize) and the second is a penalty term with a minimum when the nodes are equally partitioned into the K subsets.

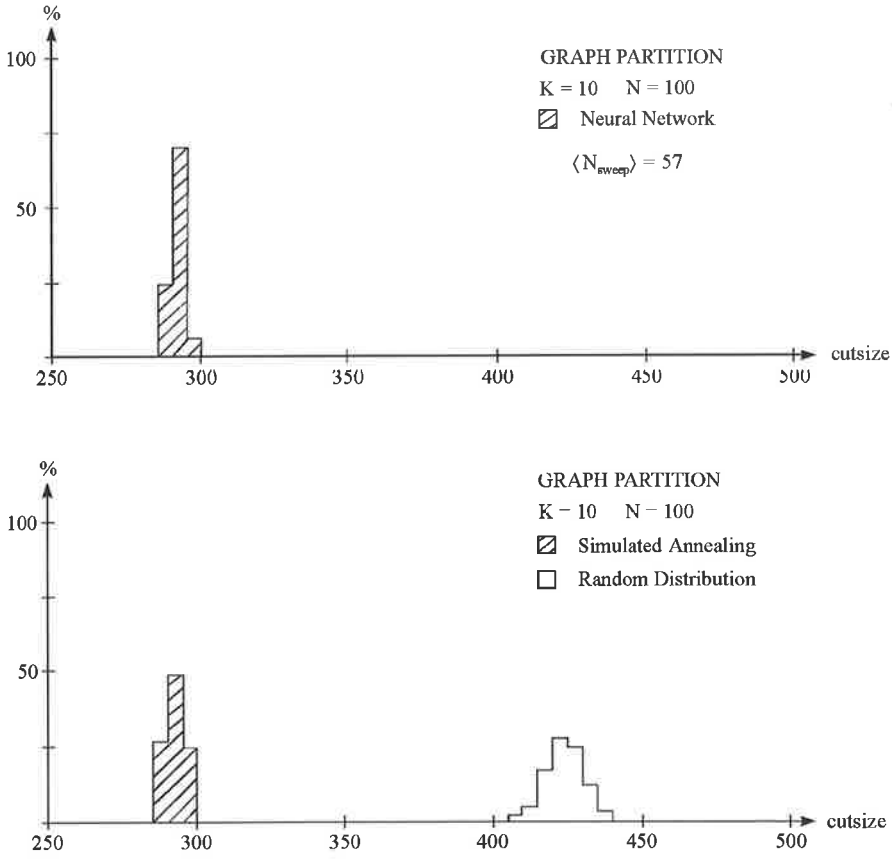

Figure 7.10 Comparison of Potts neural network solutions versus simulated annealing and random partitions for a graph partitioning problem with $K = 10$ and $N = 100$. The histograms are based on 50 problem instances for the neural network and simulated annealing algorithms and 1000 for the random partition.

Results from numerical experiments

Figure 7.10 compares the performance of the Potts neural network with simulated annealing. The graphs were generated by randomly connecting the node pairs with probability $P = 10/N$, and the critical temperature $\hat{c}$ could be estimated by statistical methods. The results are impressive. The neural network algorithm performs as well as, sometimes better than the simulated annealing method with its excessive annealing and sweep schedule [Peterson & Söderberg, 1989]. This is accomplished with between 50 and 100 iterations. In fact, the number of iterations needed was found empirically to be independent of problem size [Peterson & Söderberg, 1989].

4.6 The traveling salesman problem

In the traveling salesman problem (TSP) the coordinates $\mathbf{x}_i \in \mathbb{R}^2$ of a set of N cities are given. A closed tour of minimal total length is to be chosen such that each city is visited exactly once. To encode it we define an N-state Potts neuron $\mathbf{s}_i$ for each city $i = 1, \ldots, N$, such that the component s_{ia} $(a = 1, \ldots, N)$ is 1 if city i has the tour number a and 0 otherwise. Let d_{ij} be the distance between city i and j. Then a suitable energy function is given by

$$E = \sum_{i,j=1}^{N} d_{ij} \sum_{a=1}^{N} s_{ia} s_{j(a+1)} + \frac{\alpha}{2}\left(\left(\sum_{i=1}^{N} \mathbf{s}_i\right)^2 - \sum_{i=1}^{N} \mathbf{s}_i^2\right),$$

where the first term is a cost term, and the second a soft constraint term penalizing configurations where two cities are assigned the same tour number. In the first term, $a+1$ is to be taken mod N. Note that E does not have the form of (17). (An alternative encoding results from interchanging the roles of the labels i and the components a of the Potts spins.)

Problem instances are generated by positioning the cities at random in a square. As in the graph partitioning case, the energy is minimized by iteratively solving the Potts MF equations, (19) and (20), using the generic prescription of Section 4.4. Again the linearized dynamics can be analyzed to estimate $\hat{c}$. Peterson & Söderberg [1989] explore the Potts approach for the TSP numerically for problem sizes up to $N = 200$, again with encouraging results. The average neural network solution has a quality comparable to simulated annealing, and as in the graph partitioning case, there are no really bad solutions.

As for convergence time, we have observed no dependence on N for the problem sizes probed. The comparisons with simulated annealing concern quality only. When discussing the total time consumption one has to distinguish between serial and parallel implementations. With serial execution, the time consumption of the Potts neural algorithm is proportional to N^3 for the TSP. This should be compared with $N^2\sigma(N)$ for simulated annealing, where $\sigma(N) > O(N)$ is the total number of sweeps needed. A further speed increase may be obtained by exploiting the parallelism inherent in the neural approach, which would give constant time on a general-purpose parallel machine or custom-made

hardware. The TSP may also be tackled by *deformable templates*, a hybrid approach discussed in Section 6. See also Chapter 8, Section 7.

4.7 Scheduling problems

A Potts neural network formulation is almost ideal for scheduling problems because they lend themselves to a natural formulation. In its purest form, a scheduling problem consists entirely of fulfilling a set of basic constraints, each of which can be encoded as a *penalty term* that will vanish when the constraint is obeyed. Thus, the minimum energy is known (0), and one can recognize exact (legal) solutions a posteriori by inspection. But in many applications there exist additional preferences within the set of legal schedules, preferences which lead to the appearance of *cost terms*.

First we will discuss a synthetic problem, where the principles of the neural mapping are very transparent. Then we will briefly discuss how to deal with the additional complication in a real-world problem [Gislén, Söderberg & Peterson, 1992b], in this case a Swedish high school.

A synthetic example

Gislén, Söderberg & Peterson [1989] map a simplified scheduling problem, where N_p teachers lecture N_q classes in N_x classrooms at N_t time slots, onto a Potts neural network. In this problem one wants solutions where every teacher p gives a lecture to each of the classes q, using the available rooms x and the available time-slots t, with no conflicts in space (classrooms) or time. This defines the *basic constraints* that have to be satisfied; various *preferences* regarding continuity in classrooms, etc., were also considered.

There is a very transparent way to describe this problem that naturally lends itself to the Potts neural encoding, where *events*, defined by teacher–class pairs (p, q), are mapped onto space–time slots (x, t) (Figure 7.11). The basic constraints in this picture are as follows:

1. An event (p, q) should occupy precisely one space–time slot (x, t).
2. Different events (p_1, q_1) and (p_2, q_2) cannot occupy the same space–time slot (x, t).
3. A teacher p cannot take part in more than one event at a time.
4. A class q cannot take part in more than one event at a time.

A schedule fulfilling all the basic constraints is said to be *legal*.

The obvious encoding is in terms of Potts spins $\mathbf{s}_{pq}$; the component $s_{pq;xt}$ is defined to be 1 if the event (p, q) takes place in the space–time slot (x, t) and 0 otherwise. Thus we need N_pN_q distinct K-state Potts spins, with $K = N_xN_t$. Then the first constraint is trivially satisfied through the usual Potts condition

$$\sum_{x,t} s_{pq;xt} = 1 \tag{22}$$

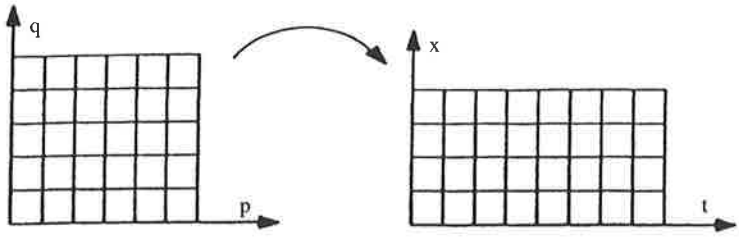

Figure 7.11 Mapping of events (p, q) onto space–time slots (x, t)

for each event (p, q). The other three constraints are implemented using energy penalty terms as follows:

$$E_{XT} = \frac{1}{2}\sum_{x,t}\left(\sum_{p,q} s_{pq;xt}\right)^2,$$

$$E_{PT} = \frac{1}{2}\sum_{p,t}\left(\sum_{q,x} s_{pq;xt}\right)^2,$$

$$E_{QT} = \frac{1}{2}\sum_{q,t}\left(\sum_{p,x} s_{pq;xt}\right)^2.$$

The energy E to be minimized is the sum of these terms, with all diagonal terms subtracted.

Again, mean field variables $v_{pq;xt} \sim \langle s_{pq;xt}\rangle_c$ are introduced, and the corresponding MF equations (cf. (19) and (20)) read

$$u_{pq;xt} = -\frac{1}{c}\cdot\frac{\partial E}{\partial v_{pq;xt}},$$

$$v_{pq;xt} = \frac{e^{v_{pq;xt}}}{\sum_{x't'} e^{u_{pq;x't'}}}.$$

With the usual annealing, a very efficient algorithm results.

However, an important simplification can be made. With the above encoding, it turns out that the MF equations produce two separate phase transitions, one in x and one in t. In other words, the system spontaneously factorizes into two parts. It is therefore economical to implement this factorization at the encoding level. This can be done by replacing each spin $\mathbf{s}_{pq}$ by the direct product of two spins: an N_x-state Potts spin $\mathbf{s}^{(X)}_{pq}$ for assigning classrooms, and an N_t-state Potts spin $\mathbf{s}^{(T)}_{pq}$ for assigning time-slots,

$$s_{pq;xt} \to s^{(X)}_{pq;x}\, s^{(T)}_{pq;t},$$

with separate Potts conditions replacing (22):

$$\sum_x s^{(X)}_{pq;x} = 1$$

and

$$\sum_t s^{(T)}_{pq;t} = 1$$

respectively. This reduces the dimensionality from $N_p N_q N_x N_t$ to $N_p N_q (N_x + N_t)$, so the sequential execution time goes down; the solution quality is not affected. Note that the factorization brings a situation where the Potts neurons will have different numbers of components. Also this situation can be dealt with when analyzing the MF dynamics in terms of fixpoints and $\hat{c}$.

The performance of these algorithms was investigated by Gislén, Söderberg & Peterson [1989] for a variety of problem sizes and for various levels of difficulty, as measured by the ratio between the number of events and the number of available space–time slots. Legal solutions were consistently found with very modest convergence times for problem sizes $(N_p, N_q) = (5,5), \ldots, (12,12)$. By convergence time we mean the total number of sweeps needed to obtain a legal solution, no matter how many trials it takes. And very good solutions were obtained when preferences were introduced, e.g., for having subsequent lessons in the same room.

High-school scheduling

The synthetic scheduling problem contains several simplifications as compared to realistic problems. Gislén, Söderberg & Peterson [1992b] explored real-world problems from the Swedish high-school system, which required an extended formalism. Here is a list of complications that must be handled by such an extended formalism:

1. A week–day partition of the time range, with a one-week periodicity, occasionally extended to two- or four-week periodicity.
2. In the toy example each teacher had each class exactly once. In the real world the teacher has to give lessons in certain subjects to a subset of the classes a few hours a week.
3. In the earlier paper it was assumed that all classrooms were available for every event. This is not true in reality; many subjects require purpose-built spaces.
4. Many subjects are taught for two hours in a row (double hours).
5. For some optional subjects the classes are broken up into option groups, temporarily forming new classes.
6. Lunch events have to appear within three hours around noon.
7. Various preferences have to be considered.

Item 1 presents no problem: the time index t is simply subdivided into weekdays (d) and daily hours (h). For item 2 we cannot use p and q as independent spin labels. Instead we define an independent label i (event index), to which p and q are attributes, $p(i)$ and $q(i)$. Other event attributes are the subject, and whether the hours are double or single; they are stored in a table containing all the a priori information relevant for each event i.

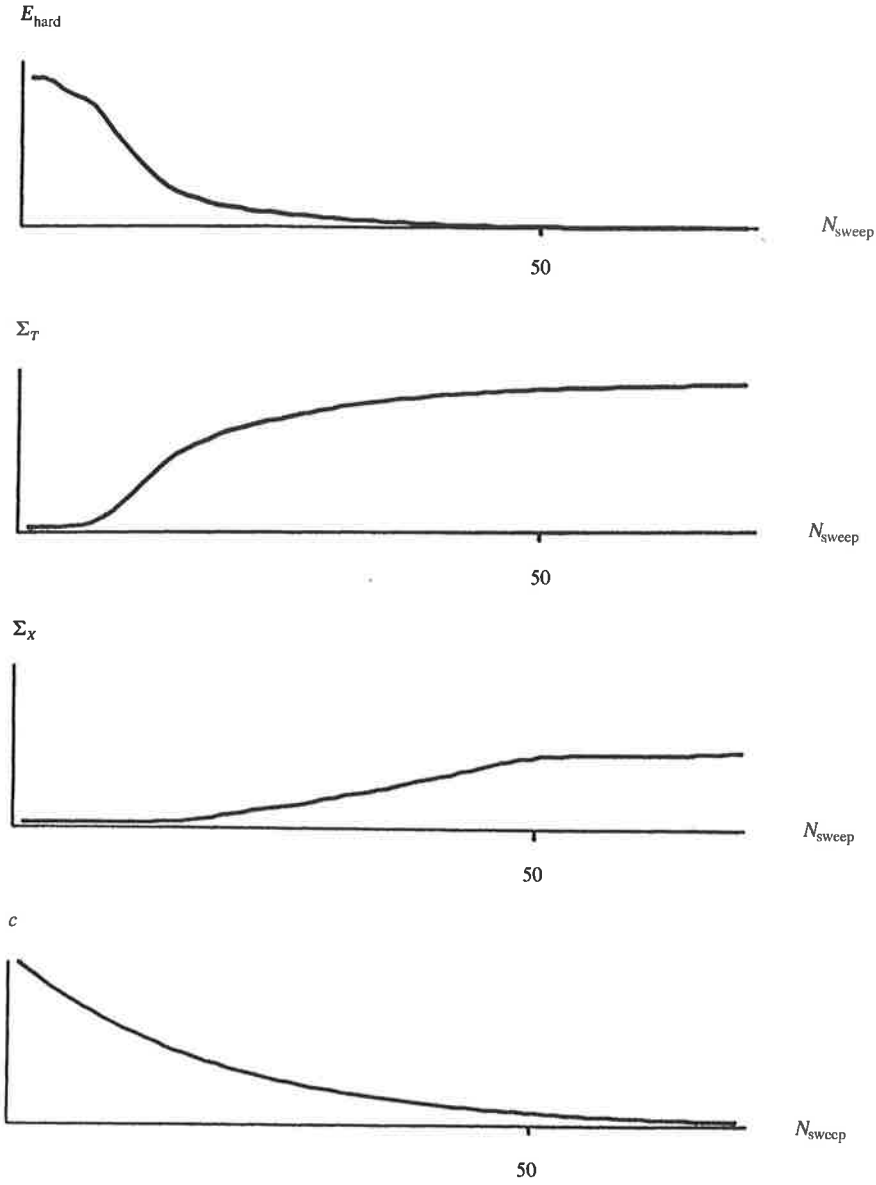

Figure 7.12 Energy (E_{hard}), saturations (Σ_T, Σ_X) and temperature (c) as functions of N_{sweep} for one run with a Swedish high-school problem; the estimated phase transition temperature $\hat{c}$ is indicated

To facilitate the handling of double-hour events, their t-neurons must be substituted in the energy with effective ones, $\tilde{s}_{i;t}^{(T)}$, defined as

$$\tilde{s}_{i;t}^{(T)} \equiv s_{i;t}^{(T)} + s_{i;t-1}^{(T)}.$$

Items 5 and 6 are syntactic constraints, which can be taken into account by restricting the Potts index ranges for the relevant events.

We encounter three kinds of *preferences* when scheduling Swedish high schools:

- The different lessons for a class in a particular subject should be *spread* as much as possible over the weekdays.
- The class schedules should have as few 'holes' as possible; lessons should be *glued* together.
- Teachers may have various individual preferences.

These preferences have to be accommodated by appropriate penalty terms; see Gislén, Söderberg & Peterson [1992b].

High-school scheduling problems typically have ~ 90 teachers, ~ 50 weakly hours, ~ 45 classes, and ~ 60 classrooms, which corresponds to $\sim 10^{4600}$ possible choices. In the factorized Potts formulation these are handled by $\sim 10^5$ neural variables.

In spite of the above complications, an automated implementation of the MF algorithm can be made along the lines of the generic prescription defined in Section 4.4, and high-quality solutions emerge [Gislén, Söderberg & Peterson, 1992b]. Figure 7.12 shows a typical evolution of the basic constraint part of the energy, the separate neuron saturations, and the temperature c.

A revision capability is inherent in the neural formalism, which is useful when encountering unplanned events once a schedule exists. Such rescheduling is performed by *clamping* those events not subject to change and heating up and cooling the remaining dynamical neurons.

One should keep in mind that problems of this kind and this size are so complex that even several staff months of human planning will in general not yield solutions that will meet all the requirements in an optimal way. We have not been able to find any algorithm in the literature that solves a real-world problem of this complexity. Existing commercial software packages do not solve the entire problem. The problem is solved with an interactive user taking stepwise decisions.

5 OPTIMIZATION WITH INEQUALITY CONSTRAINTS

The application areas dealt with earlier (traveling salesman, graph partitioning, and scheduling) are characterized by having low-order polynomial *equality* constraints. Hence they can be implemented by polynomial penalty terms. However, in many other optimization problems, especially resource allocation, one has to deal with *inequalities*. The objective of this section is to develop a method to deal with this kind of problem in the MF approach. We illustrate resource allocation by the knapsack problem, a typical example.

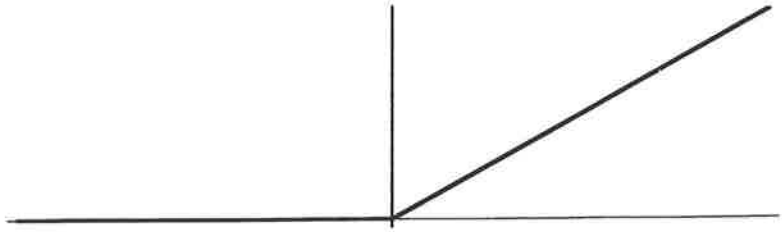

Figure 7.13 The penalty function $C\Phi(C)$ of equation (23)

5.1 Dealing with inequality constraints

In a neural formalism, an equality constraint $C=0$ with C some function of the spins, can be taken into account by adding an appropriate penalty term to the objective function. This can be chosen as a suitable function $\Phi(C)$ of the constraint variable, the obvious choice being $\Phi(C)=C^2$.

With an inequality constraint, $C\leqslant 0$, we need a $\Phi(C)$ that only penalizes configurations for which $C>0$. A possible choice is

$$\Phi(C)=C\Theta(C), \tag{23}$$

where Θ is the standard step function. This penalty function (Figure 7.13), multiplied by a suitable coefficient, gives a penalty in proportion to the degree of violation of the constraint.

Even if C happens to be a polynomial function of the spins, the nature of Φ ensures that the constraint term is not a polynomial function, and special care is needed when implementing the MF approximation. As shown by Ohlsson, Peterson & Söderberg [1993], this can be done by replacing the derivative $\partial E/\partial v_i$ in the MF equation (we assume binary ± 1 neurons):

$$v_i=g\left(-\frac{\partial E}{\partial v_i};c\right)$$

by a difference:

$$\frac{\partial E}{\partial v_i}\rightarrow\frac{1}{2}(E(v)|_{v_i=1}-E(v)|_{v_i=-1}). \tag{24}$$

This is equivalent to assuming that E is a linear function of v_i between the extreme values ± 1. This trick also has the desirable side effect of killing all self-couplings, and can be used for any energy function, not necessarily with inequality constraints. It can easily be generalized to Potts systems, too.

5.2 The knapsack problem

In the knapsack problem one has a set of N *items* i, each with an associated *utility* $u_i>0$ and a set of *loads* $a_{ki}>0$, $k=1,\ldots,M$. The goal is to fill a knapsack with

a subset of the items such that their total utility,

$$U = \sum_{i=1}^{N} u_i s_i, \tag{25}$$

is maximized, subject to a set of M load constraints,

$$\sum_{i=1}^{N} a_{ki} s_i \leqslant b_k, \quad k = 1, \ldots, M, \tag{26}$$

defined by load *capacities* $b_k > 0$. The encoding is in terms of binary decision variables (spins) $s_i \in \{1, 0\}$, representing whether or not item i goes into the knapsack.

We will consider a class of problems, where a_{ki} and u_i are independent uniform random numbers on the unit interval $[0, 1]$, and b_k are fixed to a common value b. With $b > N/2$ the problem becomes trivial; the solution will have almost all $s_i = 1$. Conversely, with $b \ll N/4$ the number of allowed configurations will be small and an optimal solution can easily be found. We pick the most difficult case, defined by $b = N/4$. The expected number of items used in an optimal solution will then be about $N/2$, and an optimal solution becomes inaccessible for large N.

In the optimal solution to such a problem, there will be a strong correlation between the value of u_i and the probability for s_i to be 1. With a simple heuristic based on this observation, one can often obtain near-optimal solutions very fast. We will therefore also consider a class of harder problems with narrower u_i distributions, *homogeneous* problems. The extreme case is when u_i is independent of i, and the utility proportional to the number of items used.

We note in passing that the *set covering problem* is a special case of the general problem, with random $a_{ki} \in \{0, 1\}$, and $b_k = 1$. This defines a comparatively simple problem class, according to the above discussion, and we will stick to the knapsack problem in what follows.

Neural approach

A suitable energy function for the problem defined in (25) and (26) is

$$E(s) = -\sum_{i=1}^{N} u_i s_i + \alpha \sum_{k=1}^{M} \Phi\left(\sum_{i=1}^{N} a_{ki} s_i - b_k \right), \tag{27}$$

where the first term measures the utility, and the rest are constraint terms, with Φ the penalty function of (23) ensuring that the constraints in (26) are fulfilled. The coefficient α governs the relative strengths of the utility and constraint terms.

Minimizing (27) is achieved with the mean field equations,

$$v_i = g\left(-\frac{\partial E}{\partial v_i}; c \right),$$

though with differences substituted for derivatives, as in (24), and modified for

[0, 1] neurons. This leads to

$$\frac{\partial E}{\partial v_i} \to -u_i + \alpha \sum_{k=1}^{M} \left(\Phi\left(\sum_{j=1}^{N} a_{kj} v_j - b_k \right)\Big|_{v_i=1} - \Phi\left(\sum_{j=1}^{N} a_{kj} v_j - b_k \right)\Big|_{v_i=0} \right).$$

The modified MF equations are solved iteratively by annealing in c. Again there exists a more or less automated scheme for solving them (cf. Section 4.4). The high-c fixpoint analysis is somewhat more difficult in this case; we refer the reader to Ohlsson, Peterson & Söderberg [1993] for a discussion of this point.

Performance comparisons

Ohlsson, Peterson & Söderberg [1993] compare the neural network (NN) approach with several other methods. The *branch-and-bound* (BB) method is an optimization algorithm and consists of going down a search tree, checking bounds on constraints or utility for entire subtrees, thereby avoiding exhaustive search. In particular, for nonhomogeneous problems, this method is accelerated by ordering the utilities according to magnitude:

$$u_1 > u_2 > \cdots > u_N. \tag{28}$$

In a naive implementation, this implementation approach is only feasible for small problem instances. The *greedy heuristic* (GH) is a fast and dirty approximation method applicable to nonhomogeneous problems. Proceeding from larger to smaller u_i (cf. (28)), collect every item that does not violate any constraint. *Simulated annealing* (SA) [Kirkpatrick, Gelatt & Vecchi, 1983] is a stochastic method, implemented in terms of attempted single-spin flips, subject to the constraints. *Linear programming* (LP) based on the simplex method [Press et al., 1986] applies only to a modified problem with $s_i \in [0, 1]$. For the ordered (cf. (28)) nonhomogeneous knapsack problem it gives solutions with a set of leading 1's and a set of trailing 0's, with a window in between containing fractional numbers. Fairly good solutions emerge when augmented by greedy heuristics for the elements in this window.

Table 7.1 Comparison of performance and time consumption for the different algorithms on a knapsack problem with $N = M = 30$; the time consumption refers to a DEC3100 workstation

Algorithm	$u_i = \text{rand}[0, 1]$		$u_i = \text{rand}[0.45, 0.55]$		$u_i = 0.5$	
	Perf.	CPU time	Perf.	CPU time	Perf.	CPU time
BB	1	16	1	1500	1	1500
NN	0.98	0.80	0.95	0.70	0.97	0.75
SA	0.98	0.80	0.95	0.80	0.96	0.80
LP	0.98	0.10	0.93	0.25	0.93	0.30
GH	0.97	0.02	0.88	0.02	0.85	0.02

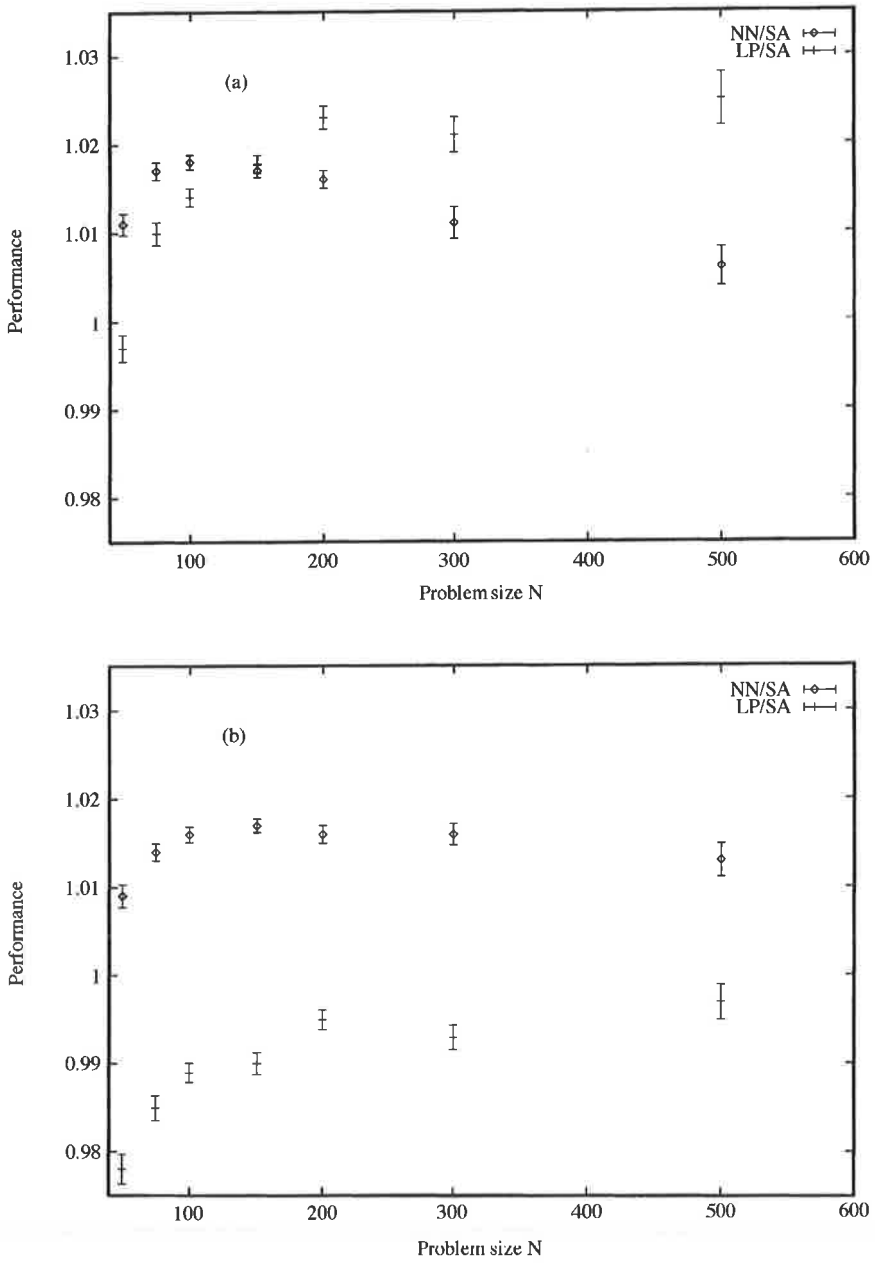

Figure 7.14 Performance of the neural network (NN) and linear programming (LP) approaches normalized to simulated annealing (SA) for problem sizes ranging from 50 to 500 with $M = N$: (a) $u_i = \text{rand}[0.45, 0.55]$ and (b) $u_i = 0.5$

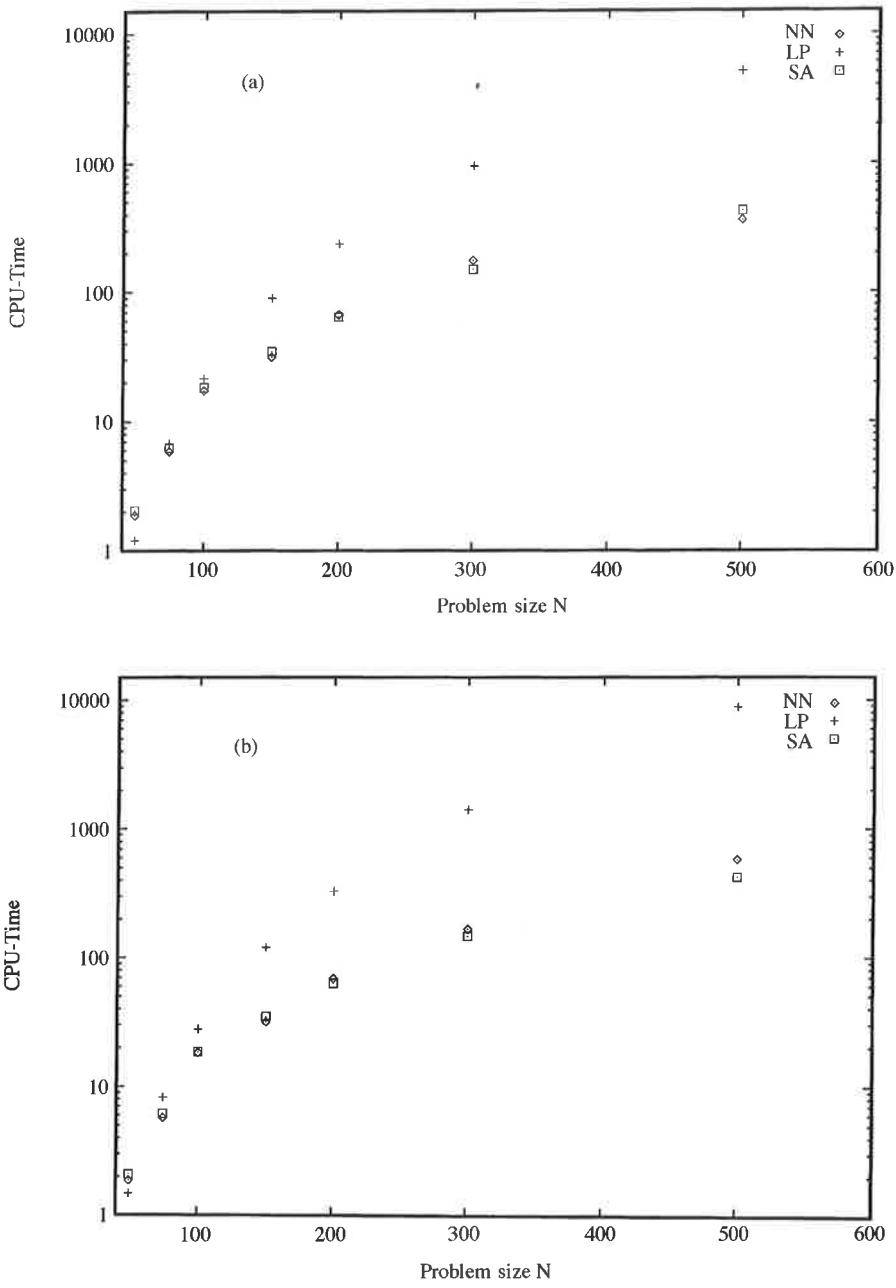

Figure 7.15 Time consumption of the neural network (NN) and linear programming (LP) approaches normalized to simulated annealing (SA) for problem sizes ranging from 50 to 500 with $M = N$: (a) $u_i = \text{rand}[0.45, 0.55]$ and (b) $u_i = 0.5$; the times are for a DEC3100 workstation

First of all, the NN, SA, and LP approaches are compared with the exact BB for $N = M = 30$, on homogeneous and nonhomogeneous problems; the results are shown in Table 7.1. Performance is measured as average utility normalized to BB (for large problems, normalized to SA).

LP and GH obviously benefit from nonhomogeneity in terms of quality and time (BB benefits on time), whereas the NN algorithm wins on homogeneous problems [Ohlsson, Peterson & Söderberg, 1993]. We cannot use BB for larger problem sizes, so only the approximative approaches are compared. The conclusions from problem sizes $N = M = 50$ to 500 are the same as above. The real strength of NN is best exploited for more homogeneous problems. Figures 7.14 and 7.15 show the performance and time consumption for $N \in [50, 500]$ with $M = N$.

In summary, the MF approach is competitive compared to other approximative methods for the hard homogeneous problems, both with respect to solution quality and time consumption. It also compares well with exact solutions, and the accessibility of an exact solution depends on the problem size. The ability to find good approximate solutions to difficult knapsack problems opens up several application areas within the field of resource allocation.

6 DEFORMABLE TEMPLATES

The above optimization problems were all treated as *pure assignment* problems – all variables were logical (neurons). In some areas it is advantageous to take a *parametric assignment* approach, which produces a hybrid picture where MF Potts decision variables are supplemented by geometric *template* variables. The TSP [Durbin & Willshaw, 1987] and particle physics track finding [Yuille, Honda & Peterson, 1991; Ohlsson, Peterson & Yuille, 1992; Gyulassy & Harlander, 1991] are examples where it pays to use the deformable templates approach. In the context of the TSP, this approach is often called an 'elastic net' [Durbin & Willshaw, 1987].

The deformable template formulation is rather application specific. Hence we will illustrate derivations, etc., using the TSP and track finding examples. Our presentation follows a general probabilistic path rather than the more intuitive approach in the original elastic network [Durbin & Willshaw, 1987].

6.1 The traveling salesman problem

Denote the N city positions in a TSP instance by $x_i \in \mathbb{R}^2$, $i = 1, \ldots, N$. The idea is to use a template trajectory, consisting of a closed chain of $M > N$ ordered points $\mathbf{y}_a \in \mathbb{R}^2$, $a = 1, \ldots, M$, in addition to a set of N distinct M-state Potts spins $\mathbf{s}_i$. The role of the Potts spins is to assign a point a in the chain to each city i, by $s_{ia} = 1$. The template coordinates $\mathbf{y}_a$ and the Potts spins $\mathbf{s}_i$ are to be chosen so as to minimize the chain length, while for each city i forcing the assigned $\mathbf{y}_a$ to match $\mathbf{x}_i$. In order to accomplish this the following energy expression is used:

$$E(\mathbf{s}, \mathbf{y}) = \sum_{ia} s_{ia} |\mathbf{x}_i - \mathbf{y}_a|^2 + \gamma \sum_a |\mathbf{y}_a - \mathbf{y}_{a+1}|^2. \tag{29}$$

The first term in (29) enforces matching: it is minimized when each $\mathbf{x}_i$ coincides with that $\mathbf{y}_a$ for which $s_{ia} = 1$. The second term minimizes the tour length while preserving an even spacing between the template points. The parameter γ governs the relative strength between the matching and tour length terms, and should be suitably chosen depending on M, N and the typical distances in the problem. In order to avoid getting trapped in local minima when minimizing E, noise is added by using the Boltzmann distribution for the system

$$\mathbb{P}(\mathbf{s}, \mathbf{y}; c) = \frac{e^{-E(\mathbf{s},\mathbf{y})/c}}{Z},$$

with the partition function Z given by

$$Z = \sum_{\mathbf{s}} \int d\mathbf{y} e^{-E(\mathbf{s},\mathbf{y})/c}.$$

Performing the trivial sums over the allowed states of the Potts spins [Yuille, 1990; Yuille, Honda & Peterson, 1991], we can rewrite Z as

$$Z = \int d\mathbf{y} e^{-E_{\text{eff}}(\mathbf{y})/c},$$

where the *effective energy* E_{eff} of the template is given by

$$E_{\text{eff}}(\mathbf{y}) = -c \sum_i \log \left(\sum_a e^{-|\mathbf{x}_i - \mathbf{y}_a|^2/c} \right) + \gamma \sum_a |\mathbf{y}_a - \mathbf{y}_{a+1}|^2. \tag{30}$$

Next we minimize E_{eff} with respect to $\mathbf{y}_a$ using gradient descent

$$\Delta \mathbf{y}_a = \eta \left(\sum_{ia} v_{ia}(\mathbf{x}_i - \mathbf{y}_a) + \gamma(\mathbf{y}_{a+1} - 2\mathbf{y}_a + \mathbf{y}_{a-1}) \right), \tag{31}$$

where $\eta > 0$ is a suitable step size, and the Potts factor (cf. (20)) v_{ia} is given by

$$v_{ia} = \frac{e^{-|\mathbf{x}_i - \mathbf{y}_a|^2/c}}{\sum_b e^{-|\mathbf{x}_i - \mathbf{y}_b|^2/c}}. \tag{32}$$

This is all done under slow annealing in the temperature c, which enters the dynamics only via the Potts factors. A typical evolution of (31) and (32) is schematically depicted in Figure 7.16. In contrast to the Potts description of Section 4, the logical units are only implicit in the dynamics; the nontrivial dynamics lies with the analog variables $\mathbf{y}_a$. Here is an example of a complete algorithm.

An elastic net algorithm for the TSP

1. Choose problem instance $-\{\mathbf{x}_i, i = 1, N\}$.
2. Choose multiplicity M of template ($M \gg N$).
3. Choose a suitable γ and initial temperature.

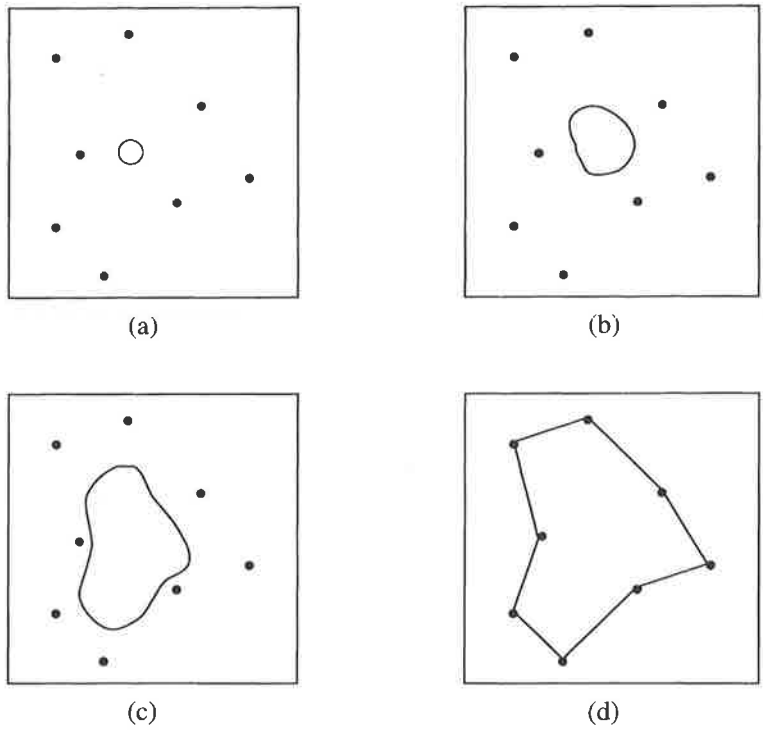

Figure 7.16 Evolution of the template chain from high to low temperatures; the dots denote city coordinates $\mathbf{x}_i$ and the template coordinate $\mathbf{y}_a$ reside on the closed curves

4. Place template points $\mathbf{y}_a$ equally spaced on a small circle with a slight random displacement from the center of gravity of the cities.
5. Until the system has settled, do:

   update the template points: $\mathbf{y}_a = \mathbf{y}_a + \Delta\mathbf{y}_a$, with $\Delta\mathbf{y}_a$ given by (31);
   $c = 0.9 \times c$.

How does this algorithm work? The first term in (30) contains a sum of Gaussians of width $\sqrt{c}$ around the templates. At high c this makes all the $\mathbf{y}_a$ compete to match all the cities; in effect, they will be attracted towards the center of mass. As c decreases, the Potts factors become more selective, and the range of competition for each $\mathbf{x}_i$ is focused on a smaller neighborhood. Finally one $\mathbf{y}_a$ is singled out to match each $\mathbf{x}_i$, and the remaining $\mathbf{y}$'s become arranged equidistantly along straight line segments connecting the cities.

This algorithm produces high-quality solutions [Peterson, 1990a]. And, very important, an N-city problem requires only $O(N)$ variables. It is a good example of a low-dimensional geometric problem where a template method is to be preferred over the pure neural method. See also Chapter 8, Section 7.

6.2 Track finding

Track finding is the problem of fitting smooth tracks to a given set of signal points. In its most general form – as in the problem of reconstructing the trajectories of airborne objects from radar signals – the parametric form of the tracks is unknown. A pure neural approach is appropriate in this case: assign a decision element (neuron) s_{ij} between two signals i and j, which is equal to 1 if i and j are connected and 0 otherwise. An energy function is then constructed in terms of s_{ij} such that smooth tracks will correspond to minima [Denby, 1988; Peterson, 1989]. Another possibility is to have a rotor neuron [Gislén, Söderberg & Peterson, 1992a] (see Section 7) associated with each signal, with an interaction such that smooth tracks emerge [Peterson, 1990b].

In particle physics experiments, a known magnetic field makes charged particles bend in tracks of known parametric form. The values of the track parameters contain information about the momenta of the particles. Even though the pure neural approach [Denby, 1988; Peterson, 1989; Stimpfl-Abele & Garrido, 1991] seems to work reasonably well, it is natural to use a deformable templates approach [Yuille, Honda & Peterson, 1991; Ohlsson, Peterson & Yuille, 1992; Gyulassy & Harlander, 1991] for the following reasons:

1. Not only does it solve the combinatorial optimization part of the problem, assigning signals to tracks, it also delivers the track parameters, which contain the physical quantities of interest.
2. The pure neural approach is more general than this problem requires. One should benefit from the fact that the parametric form of the tracks is known in advance, straight lines or helices.
3. The number of variables needed to solve an N-signal problem is large even with the connectivity restrictions imposed by Peterson [1989] and Stimpfl-Abele & Garrido [1991]. For a problem with N signals and M tracks one should only need $O(M)$ variables.
4. As demonstrated by Gyulassy & Harlander [1991], the neural approach is somewhat sensitive to noise. Again, with prior knowledge of the parametric form, the method should be more robust with respect to noise.

The strategy of the deformable templates approach for tracking [Yuille, Honda & Peterson, 1991; Gyulassy & Harlander, 1991] is to match the observed events to simple parameterized models, *templates*, the form of which reflects the a priori knowledge about the possible track shapes, e.g., helices passing through the origin (the collision point). In addition, the formalism allows for some data points (sensor noise) to be unmatched. The mechanism involved is closely related to redescending M-estimators used in robust statistics [Huber, 1981].

We assume we are given a set of N signal coordinates $\mathbf{x}_i \in \mathbb{R}^3$, $i = 1, \ldots, N$. For the case of a constant magnetic field, we define the templates, labeled by $a = 1, \ldots, M$, to be helices passing through the origin, parameterized each by an elevation angle θ_a, a transverse curvature κ_a, and a longitudinal velocity parameter γ_a. The assignment of templates a to the signals i is handled by an M-state

Potts spin $\mathbf{s}_i$ for each signal i. The algorithm works in two steps. First, for example a Hough transform [Duda & Hart, 1973] is used to determine the number M of templates required, and to initialize the templates (Hough transforms are essentially variants of 'histogramming' or 'binning' techniques, which are commonly applied to particle tracking). The elastic arms method then takes over, resolves ambiguities, and makes detailed fits to the signals, based on the following energy function (cf. (29)):

$$E(\mathbf{s}, \theta, \kappa, \gamma) = \sum_{i=1}^{N} \sum_{a=1}^{M} (M_{ia} - \lambda) s_{ia}, \tag{33}$$

where M_{ia}, short for $M(\theta_a, \kappa_a, \gamma_a, \mathbf{x}_i)$, is a measure of the squared distance between signal i and the helix a, and the Potts spin component s_{ia} is 1 if the ith point is assigned to the ath arm, and zero otherwise, subject to the *modified Potts constraint*,

$$\sum_{a} s_{ia} = 0 \text{ or } 1. \tag{34}$$

Thus, for each i there should be *at most* one a such that $s_{ia} = 1$. This allows for noise signals not being assigned to any track. It is equivalent to having an extra null component

$$s_{i0} = 1 - \sum_{a=1}^{M} s_{ia}$$

in the Potts spin, signifying no assignment. The parameter λ is a penalty for the nonassignment of a signal point. In effect a distance cutoff is introduced such that signals with no templates within the distance $\sqrt{\lambda}$ tend not to be assigned.

We want to minimize $E(\mathbf{s}, \theta, \kappa, \gamma)$ with respect to the Potts spins and the template parameters, subject to the constraint of (34). As in the TSP the problem is encoded in two kinds of variables, assignment variables s_{ia} and template parameters θ_a, κ_a, and γ_a. Following the steps of the TSP application, we define the Boltzmann distribution as

$$P(\mathbf{s}, \theta, \kappa, \gamma) = \frac{\mathrm{e}^{-E(\mathbf{s},\theta,\kappa,\gamma)/c}}{Z}.$$

Summing over the Potts spins [Ohlsson, Peterson & Yuille, 1992], one obtains

$$P_M(\theta, \kappa, \gamma) = \frac{1}{Z} \mathrm{e}^{-E_{\mathrm{eff}}(\theta,\kappa,\gamma)/c},$$

where the effective energy E_{eff} amounts to

$$E_{\mathrm{eff}}(\theta, \kappa, \gamma) = -c \sum_{i=1}^{N} \log\left(\mathrm{e}^{-\lambda/c} + \sum_{a=1}^{M} \mathrm{e}^{-M_{ia}/c} \right).$$

Straightforward gradient descent on E_{eff} with a step size η gives

$$\Delta\theta_a = -\eta \frac{\partial E_{\mathrm{eff}}}{\partial \theta_a} = -\eta \sum_{i=1}^{N} v_{ia} \frac{\partial M_{ia}}{\partial \theta_a}$$

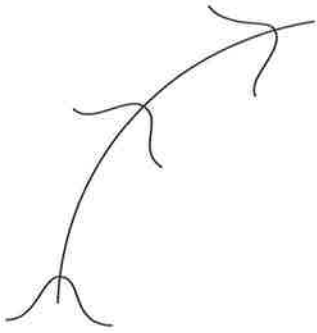

Figure 7.17 An elastic arm at temperature c

with the Potts factor

$$v_{ia} = \frac{e^{-M_{ia}/c}}{e^{-\lambda/c} + \sum_{b=1}^{M} e^{-M_{ib}/c}} \tag{35}$$

and analogous equations for $\Delta\kappa_a$ and $\Delta\gamma_a$. These are iterated under annealing in c. The structure of these equations is reminiscent of the TSP case, with decision elements v_{ia} intermixed with geometric fitting terms proportional to $\partial M_{ia}/\partial\theta_a$, $\partial M_{ia}/\partial\kappa_a$, and $\partial M_{ia}/\partial\gamma_a$. In a more sophisticated treatment, the step size η is replaced by a matrix, allowing for an interaction between the different parameters, so as to take into account an inhomogeneous metric in the space of parameters.

How does this algorithm work? At a high starting temperature a set of initial template arms are placed according to the Hough transform values for the parameters θ_a, κ_a, and γ_a. The templates compete for the signals by means of Gaussian distributions of width $\sqrt{c}$ centered around the arm positions (Figure 7.17). Initially each arm can attract many signals. The relative importance of the different signals is measured by the Potts factor (35). As the temperature is lowered, the different arms are attracted more exclusively to nearby signals only.

As discussed in connection with (33), λ governs the amount of noise points or *outliers* the algorithm allows for. It enters the Potts factor (35) via an extra component contributing to the denominator. For $\lambda \to \infty$ no signals are ignored and the extra component vanishes. For finite values of λ the domain of attraction of the arms is cut off (for small c) at a distance $\sqrt{\lambda}$.

The elastic arms approach is very similar to human processing for this kind of recognition problem. A human looks for helices in a global way and then makes fine-tuning adjustments. Indeed, when confronted with high-multiplicity data, the algorithm performs very well [Ohlsson, Peterson & Yuille, 1992].

7 ROTOR NEURONS

A binary (Ising) neuron can be considered as a vector living on a 'sphere' in one dimension. The MF approach can be generalized to variables defined on spheres in higher dimensions. Such *rotor* neurons may be used in geometrical optimization problems with angular variables.

7.1 Mean field rotor neurons

We consider the general problem of minimizing an energy function $E(\mathbf{s}_1, \ldots, \mathbf{s}_N)$ with respect to a set of N D-dimensional unit vectors (rotors), $\mathbf{s}_i \in \mathbb{R}^D$, $|\mathbf{s}_i| = 1, i = 1, \ldots, N$. To derive the MF equations, consider the partition function

$$Z = \int \mathrm{e}^{-E(\mathbf{s})/c} d\mathbf{s}_1 \cdots d\mathbf{s}_N,$$

where the integrations over $d\mathbf{s}_i$ are performed over the directions only. Along the same lines as in the binary (Ising) and Potts cases previously discussed [Gislén, Söderberg & Peterson, 1992a], rewrite Z as

$$Z \propto \int \exp\left(-E(\mathbf{v})/c - \sum_i \mathbf{v}_i \cdot \mathbf{u}_i + \sum_i F(u_i) \right) d\mathbf{v}_1 d\mathbf{u}_1 \cdots d\mathbf{v}_N d\mathbf{u}_N, \tag{36}$$

where $u_i = |\mathbf{u}_i|$ and

$$F(u) = \log I_{(D-2)/2}(u) - \frac{D-2}{2} \log u + \text{constant}. \tag{37}$$

In (37) I_n is the modified Bessel function of order n. Next we seek a saddle point of the effective potential in the exponent of (36) as in the previous cases, which leads to the *mean field* equations

$$\mathbf{u}_i = -\frac{1}{c} \nabla_i E(\mathbf{v}), \tag{38}$$

$$\mathbf{v}_i = \mathbf{g}(\mathbf{u}_i) \equiv \hat{u}_i g(u_i) \equiv \hat{u}_i F'(u_i), \tag{39}$$

where $\hat{u}_i = \mathbf{u}_i / u_i$. They give $\mathbf{v}_i$ as the average of $\mathbf{s}_i$ in the *local field* $-\nabla_i E(\mathbf{v})$. For $D = 1$ the standard sigmoid (cf. (2))

$$v_i = g(u_i) = \tanh u_i$$

is of course recovered.

The obvious dynamics consists of iterating (38) and (39) by annealing in the temperature c. At high temperature the system is in a symmetric phase, characterized by a stable trivial fixpoint $\mathbf{v}_i \approx 0$. At lower c this becomes unstable, and the mean fields $\mathbf{v}_i$ will be repelled by the origin. For low enough temperature they will stabilize close to the sphere $\mathbf{v}_i^2 = 1$ (Figure 7.18).

The dynamics is thus very different from conventional methods, where moves typically take place on the surface. Although the ability of exploring an 'off-shell'

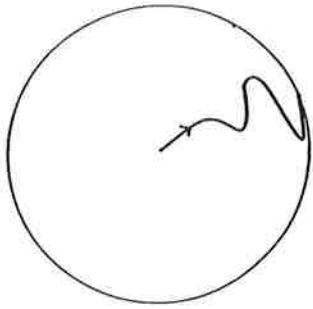

Figure 7.18 Schematic evolution of a $D = 2$ rotor initialized close to the center for $c < \hat{c}$

interpolating state space probably does not give as large an advantage as in the discrete Ising and Potts cases, it does provide an additional dimension through which to escape spurious local minima.

7.2 Applications

Gislén, Söderberg & Peterson [1992a] applied the rotor approach to the specific problem of the equilibrium configuration of N equal charges on a sphere ($D = 3$). Configurations were computed for 3, 5, 10, 20, 30, and 100 charges. The MF rotor model gave the correct solutions where they were explicitly known ($N = 2, 3, 4, 6$). For larger problems the solutions were compared to those from a local optimization (LO) and a simulated annealing (SA) algorithm.

The quality of the MF rotor algorithm turns out to be better or equal to algorithms produced by LO and SA. As for time consumption, the rotor algorithm empirically scales like N^2, whereas LO and SA scale like N^3 or worse.

Even though the rotor formalism applies to optimization problems that are not of a strictly combinatorial type, it is a natural extension of the binary decision neuron formalism that seems to be working well on the test beds explored. Possible realistic applications are molecular foldings and related problems.

8 SUMMARY AND OUTLOOK

Using neural networks to find good solutions to combinatorial optimization problems is an approach whose very nature differs from other approximative methods. The problems are mapped onto energy functions very similar to those of magnetic systems. Hence tools from statistical mechanics can be exploited. In addition to the simulated annealing method, one borrows the mean field approximation; and continuous variables feel their ways in a fuzzy manner towards good solutions. This is in contrast to most other methods, where moves are performed within the discrete solution space.

There are two main alternatives within the ANN paradigm, the pure ANN approach and deformable templates. For generic combinatorial optimization problems, such as scheduling and assignment problems, one uses the pure neural approach in which the basic steps are as follows:

- Map the problem onto a neural network by a suitable encoding of the solution space and an appropriate choice of energy function.
- Utilize prior knowledge about phase transition properties from analyzing the linearized dynamics. Special care is needed when defining the MF equations in cases when the penalty terms are nonpolynomial, as in the case of inequality constraints.
- While annealing solve the corresponding mean field equations iteratively.
- When the MF equations have converged, check the solutions for their legality, whether or not they satisfy the basic constraints. If not, supplement the algorithm with a greedy heuristic, or reanneal the system (possibly with modified constraint coefficients).

The MF equations of this method show an appealing similarity to the Kirchoff equations for analog VLSI circuitry.

For low-dimensional geometrical problems, like the traveling salesman problem and track finding problems, it is often advantageous to use a hybrid procedure, the deformable templates method. In this case, one has the following basic steps:

- Map the problem onto an energy function in terms of *both* template (geometric) and assignment (neural) variables. The dependence on the assignment variables is usually trivial.
- Form the corresponding partition function, and integrate (or sum) out the assignment variables to produce an effective energy in terms of template variables only.
- Minimize the effective energy with a gradient descent method. The templates will then be updated in such a way that the contributions from the data points are weighted with their relevance through Potts factors.

This template method has a lot in common with clustering and robust statistics methodologies as well as with Bayesian method [Ohlsson, Peterson & Yuille, 1992]. Not only is the pure neural approach closely related to analog VLSI, both the pure and hybrid approaches easily lend themselves to parallel execution.

With respect to quality of the solutions, the ANN methods produce very competitive results compared to other approximative schemes. See Peterson [1990a] and Ohlsson, Peterson & Söderberg [1993] for comparisons between methods of solving the traveling salesman and knapsack problems, respectively. No comparisons have been possible in full-size high-school scheduling applications, since no results exist from other approaches.

The neural language is very natural for encoding many combinatorial optimization problems. The approach has the advantage of being easily used for

revision of solutions due to new situations. One simply reanneals the network, clamping any nonrevisable units.

An Ising neuron can be considered as a one-dimensional rotor. The MF approach can also be extended to rotors in high dimensions. This can be fruitful when dealing with configurational problems with angular minimization.

By considering the MF approximation as resulting from a variational principle, the approach might be extended to systems with any type of discrete or continuous variables. Work in that direction has been pursued in molecular conformation problems [Bouchaud et al., 1991].