LU TP 92–07

Combinatorial Optimization with Neural Networks

Carsten Peterson[1] and Bo Söderberg[2]

Department of Theoretical Physics, University of Lund
Sölvegatan 14A, S-22362 Lund, Sweden

in *Modern Heuristic Techniques for Combinatorial Problems*,
ed. C. R. Reeves, 197–242 (Blackwell, London 1993).

Abstract:

A general introduction to the use of feed-back artificial neural networks (ANN) for obtaining good approximate solutions to combinatorial optimization problems is given, assuming no previous knowledge in the field. In particular we emphasize a novel neural mapping technique which efficiently reduces the solution space. This approach maps the problems onto Potts glass rather than spin glass models. The real strength in mapping optimization problems onto neural systems lies in the fact that local minima in the cost functions can be avoided with the use of mean field equations. The system "feels" its way towards good solutions. In the settling process it may encounter phase transitions. A systematic prescription can be given for estimating the phase transition temperatures in advance, which facilitates an automatized choice of optimal parameters. This analysis, which can be performed for both serial and synchronous updating of the mean field theory equations, also makes it possible to consistently avoid chaotic behaviour. These techniques are illustrated for the graph partition and the traveling salesman problems (TSP). Also, a realistic high school scheduling problem is dealt with in some detail. These problems are all characterized by equality constraints. The formalism can also be modified to deal with inequality constraints. This is illustrated with the knapsack problem.

We also discuss the deformable templates approach, which is closely related to the Potts neural encoding. In problems of geometrical nature like TSP and track finding this method is more economical with its fewer degrees of freedom.

[1]carsten@thep.lu.se
[2]bosse@thep.lu.se

Contents

1 Introduction

There has been an upsurge in interest in artificial neural networks (ANN) over the last 5 years, mostly in the area of *feed-forward* architectures for pattern recognition applications. There are several reasons for this. One is that some theoretical obstacles with the so-called perceptron, which was popular in the late sixties, have been overcome. Also the recently available inexpensive access to powerful CPU has facilitated the development of the field both with respect to model building, and very importantly, to deal with real-world problems. Indeed, results from impressive "product" quality application work keeps appearing both from the commercial and academic sector.

Two main kind of architectures exist, feed-forward and feed-back, with major application areas feature recognition and optimization respectively. The feed-forward ANN approaches to classification problems constitute in a sense extensions of "conventional" fitting and principal component analysis tools into the non-linear domain. This is in contrast to using feed-back ANN to find solutions to difficult combinatorial optimization problems. The neural approach to these problems really brings something new to the table. It it not just a question of mapping the problem onto an energy function of discrete neural degrees of freedom. Using mean field (MFT) equations to find the lowest energy is a novel technique for operations research borrowed from physics. Furthermore these MFT equations are isomorphic to VLSI RC-equations, making the hardware implementation straightforward.

This article is intended as a self-contained introduction to using ANN for finding good approximate solutions to difficult optimization problems. It is organized as follows. Section 2 gives a general introduction to neural systems in general, biological neural networks and the abstraction to artificial neural networks. In section 3 an overview is given on the problem of using ANN for combinatorial optimization problems. The graph bisection problem, which is particularly suited for binary neurons is dealt with in section 4. Multi-state or Potts neurons are introduced and illustrated with the graph partition problem in section 5 and the traveling salesman problem in section 6. The Potts formulation is very suitable for scheduling problems. This is discussed in section 7. An alternative procedure for geometrical low dimensional problems, the deformable templates approach, is described in section 8 in the context of the traveling salesman problem. Using ANN to find solutions to resource allocation problems (knapsack) is found in section 9. Section 10 contains a brief summary and outlook.

The most technical parts are found in appendices. The article can nevertheless be read coherently when ignoring these.

2 Neural Networks

Artificial neural networks is a computational paradigm that differs substantially from those based on the "standard" von Neumann architecture. Feature recognition ANN's generally learn from "experience", instead of being expressly "programmed" with rules like in conventional artificial intelligence (AI). Feedback ANN's "feel" their way to good solutions, rather than explicitly exploring possibilities.

2.1 Biological Neural Networks

ANN is inspired from the structure of biological neural networks and their way of encoding and solving problems. We will here very briefly review the basic components and functionality of the vertebrate central nervous system (CNS), whose details were revealed around 1940 with the emergence of the electron microscope. For more extensive literature on this subject we refer the reader to ref. [1].

The human brain contains approximately 10^{12} *neurons*. These can be of many different types, but most of them have the same general structure (see fig. 2.1). The *cell body* or *soma* receives electric input signals to the *dendrites* by means of 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, resulting in an electric discharge from the neuron – the neuron "fires". The accumulated effect of several simultaneous signals arriving at the dendrites is usually approximately linearly additive whereas the resulting output is a strongly non-linear 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 which 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*.

Figure 2.1: Schematic structure of a neuron

This general neuron structure is implemented in many different sizes and forms with different functionalities. Cell bodies have diameters in the range 5-80 μm and dendrite "trees" extend from 10 μm up to 2-3 mm. Axons can be up to 1 m in length, why sizes of entire neurons vary from 0,01 mm to 1 m. *Primary sensory neurons* connect muscles or receptors to neurons, *secondary sensory neurons* and *interneurons* connect neurons with neurons, while *motor neurons* connect neurons with muscle fibres.

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 $\sim 2^{10^{15}} \approx 10^{10^{14}}$ possible configurations! The neural network is consequently in sharp contrast to a von Neumann computer both with respect to architecture and functionality (see table 2.1).

Vertebrate Brain (Parallel Distributed Proc.)	Conventional Computer (von Neumann machine)
(fig)	(fig)
Power dissipation: $\approx$ 100W	Power dissipation: $\approx 10^5$W
Good at: • recognition • adaptation • optimization (rough)	Good at: • $a+b$, $a \cdot b$, ... • if ... then ... else ... • state space exploration
$\tau = \mathcal{O}(\mathrm{ms})$	$\tau = \mathcal{O}(\mathrm{ns})$
Parallel Robust Fault Tolerant	Serial/parallel Fragile

Table 2.1: Comparison of characteristics of neural networks and conventional computers.

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 suchlike. 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 algorithm that shows more resemblance with that of the vertebrate brain.

2.2 Artificial Neural Networks

The philosophy of the ANN approach is to abstract some key ingredients from biology and out of those construct simple mathematical models that exhibit most of the above-mentioned appealing features[3]. 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" degrees of freedom in such a way that good description of *collective* phenomena (phase transitions etc.) are obtained. It is exactly the collective behavior of the neurons that is interesting from the point of view of intelligent data processing.

[3]For a general textbook on ANN see e.g. ref. [2].

2.2.1 Basics

The basic computational entities of an ANN are the neurons v_i, which can take real values within the interval [0,1] (or [-1,1]). Sometimes the even simpler binary neuron s_i is used, where $s_i = \{0, 1\}$ (or $\{-1, 1\}$).

These are of course simplifications of the biological neurons described in the previous section. Common to most neural models is a *local updating rule*

$$v_i = g(\sum_j \omega_{ij} v_j - \theta_i) \tag{2.1}$$

where v_j are all neurons that are feeding to neuron v_i, through weights (synapses) ω_{ij}. 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 non-linear *transfer function* $g()$ is typically a sigmoid-shaped function like

$$g(x) = 0.5[1 + \tanh(x/T)] \tag{2.2}$$

where the "temperature" T sets the inverse gain; a low temperature corresponds to a very steep sigmoid and a high temperature corresponds to an approximately constant $g()$ (see fig. 2.2b). The limit $(T \to 0)$ corresponds to binary neurons s_i .

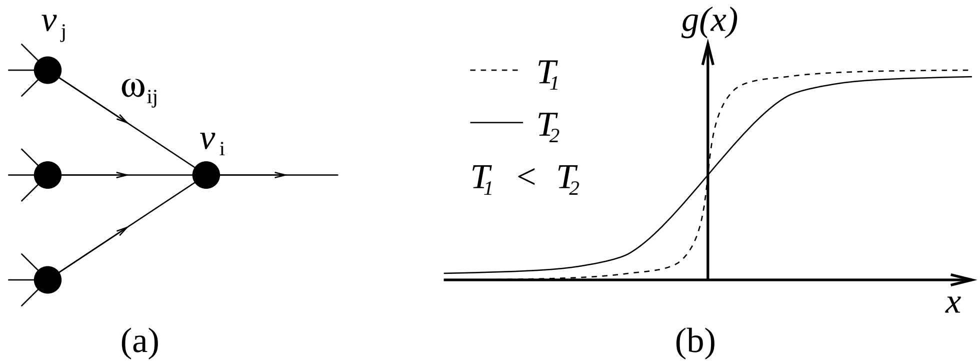


Figure 2.2: **(a)** Neuron updating and **(b)** sigmoid response functions of eqs. (2.1,2.2) for different temperatures T.

This simple artificial neuron mimics the main features of real biological neurons in terms of linear additivity for the inputs and strong non-linearity 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 architectures in neural network modeling; feed-forward (fig. 2.3a) and feed-back (fig. 2.3b). In feed-forward networks signals are processed from a set of input units in the bottom to output units in the top, layer by layer, using the local updating rule of eq. (2.1). In feed-back networks, on the other hand, the synapses are bidirectional. Activation continues until a fixed point has been reached, reminiscent of a statistical mechanics system.

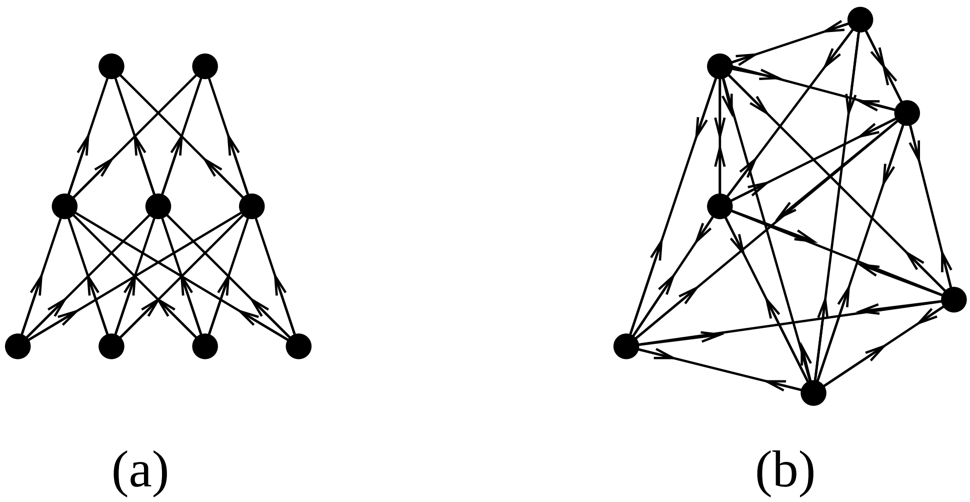


Figure 2.3: **(a)** Feed-forward and **(b)** feed-back architectures

2.2.2 Application Areas

How can the power of these networks be exploited? There are four major application areas with soft borderlines in between – feature recognition, function approximation, associative memories and optimization. Feature recognition is the most exploited application domain to date.

Feature recognition
In feature recognition (or pattern classification) situations one wants to categorize a set of input patterns $\vec{x}^{(p)} = (x_1, x_2, ..., x_N)^{(p)}$ in terms of different features o_i. The input patterns are fed into an input layer (receptors) and the output nodes represent the features. For the architecture depicted in fig. 2.3a, o_i depends on $\vec{x}^{(p)}$ through the functional form (cf. eq. (2.1))

$$o_i(\vec{x}^{(p)}) = g(\sum_{j=0} \omega_{ij} g(\sum_{k=0} \omega_{jk} x_k^{(p)})) \tag{2.3}$$

where ω_{ij} and ω_{jk} are the parameters. Eq. (2.3) can of course be generalized to any number of layers.[4]. Fitting ω_{ij} and ω_{jk} to a given data set (or *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 [3]). Once this iterative learning has reached an acceptable level, in terms of a low error, 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 equivalent to "normal" curve fitting where a smooth parameterization from a training set 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 eqs. (2.1,

[4]The thresholds, or bias terms, θ_i appearing in eq. (2.1) have in eq. (2.3) been transformed into weights ω_{i0} by adding an extra "dummy" unit v_0 to each layer, which is constantly firing.

2.2), one could imagine having an output representing a real number, corresponding to replacing eqs. (2.1, 2.2) with a linear behavior. 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 [4, 5], e.g.

$$x_t = \mathcal{F}(x_{t-1}, x_{t-2}, ...) \tag{2.4}$$

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

Associative memory
In this kind of application there is no input/output distinction. Rather the network learns a set of identity mappings by fitting ω_{ij} to memory patterns or "words" $\vec{s} = (s_1, s_2, s_3, ...s_N)$. If an incomplete or partly erroneous memory is presented to the network, it completes the pattern (see e.g. ref. [6]). In a certain sense feature recognizers are special cases of associative memories – pattern completion.

Optimization
Feed-back networks have shown great promise in finding good solution problems to difficult combinatorial optimization problems. In this case again non-linear updating equations (eq. (2.1)) are allowed to settle. This corresponds to minimizing an energy function, which typically looks like

$$E = -\frac{1}{2}\sum_{ij} \omega_{ij} s_i s_j \tag{2.5}$$

The problem is mapped onto the form of eq. (2.5) by a clever choice of ω_{ij}. In this case ω_{ij} are fixed once and for all for each problem – they are not adaptive as in the other domains of application. The neurons s_i are then allowed to settle into a stable state, where the solution to the problem is given by the configuration $\vec{s} = (s_1, s_2, ...)$ with minimum energy.

3 Combinatorial Optimization Problems

Many combinatorial optimization problems are NP-complete[5]. Exact solutions to these problems require state space exploration of one kind or another leading to $n!$ or a^n computations for a system with n degrees of freedom. Different kind of heuristic methods are therefore often used to find reasonably good solutions. The ANN approach, which is based on feed-back networks, falls within this category. It has advantages in terms of solution quality and a "fuzzy" interpretation of the answers through the MFT variables. Furthermore it is inherently parallel, facilitating implementations on concurrent processors and with MFT equations custom made hardware is straight-forward to design. The ANN approach differs from exact and most other heuristic 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:

- *"Pure" neural approach* based on either binary [7] or multi-state [8] neurons with MFT equations for the dynamics.

[5] A problem is NP if there is no known algorithm that solves it in polynomial time. *NP-complete* is a class of NP problems such that if one finds a polynomial solution to one of the members, all the others are solved as well.

- *Deformable templates* [9], where the neural degrees of freedom have been integrated out and one is left with coordinates for possible solutions.

The latter pathway is a winner for low-dimensional geometrical problems like the traveling salesman problem (TSP), whereas the former is the only possibility for high-dimensional problems like scheduling. The following sections describe both the approaches illustrated with suitable problems.

4 The Graph Bisection Problem

The neural approach is very transparent in this problem, since it is of binary nature. We will here systematically go through the three basic steps which are common to all problems:

- Map the problem onto a neural network.
- Derive the MFT equations.
- Analyze the dynamics of the MFT equations.

In addition we will of course test the approaches numerically.

4.1 Neural Mapping

The graph bisection problem (GP) is defined as follows (see fig. 4.1a): Partition a set of N nodes with given connectivity into two halves such that the connectivity (cutsize) between the two halves is minimized.

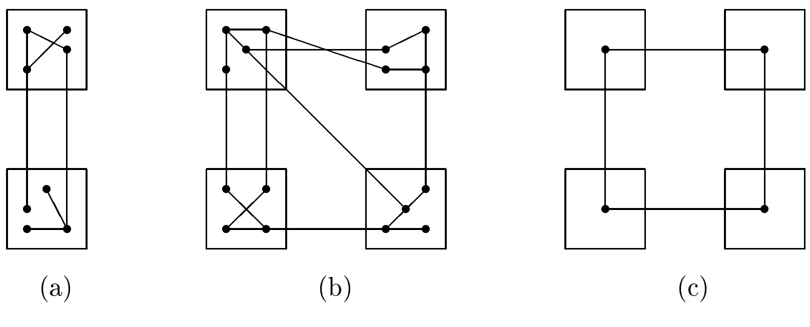


Figure 4.1: **(a)** A graph bisection problem. **(b)** A $K = 4$ graph partition problem. **(c)** A N=4 TSP problem

The problem is mapped onto the Hopfield energy function (cf. eq. (2.5)) by the following representation: For each node, assign a binary neuron $s_i = \pm 1$ and for each pair of vertices $s_i s_j$ $(i \neq j)$ we assign a value $\omega_{ij} = 1$ if they are connected, and $\omega_{ij} = 0$ if they are not. In terms of fig. 4.1a, we let $s_i = \pm 1$ represent whether node i is in the left or in the right position. With this notation,

$$\omega_{ij} s_i s_j \Rightarrow \begin{cases} > 0 & \text{whenever } i \text{ and } j \text{ are in the same partition} \\ = 0 & \text{if } i \text{ and } j \text{ are not connected at all} \\ < 0 & \text{whenever } i \text{ and } j \text{ are in different partitions} \end{cases} \tag{4.1}$$

Minimization of the energy function (2.5) will maximize the connections within a partition while minimizing the connections between partitions, with the result that all nodes are forced into one partition. Hence we must add a "constraint term" to the right hand side of eq. (2.5) that penalizes situations where the nodes are not equally partitioned. We note that $\sum s_i = 0$ when the partitions are balanced and a term proportional to $(\sum s_i)^2$ will subsequently increase the energy whenever the partition is unbalanced. Our energy function for graph bisection thus takes the form:

$$E = -\frac{1}{2}\sum_{ij} \omega_{ij} s_i s_j + \frac{\alpha}{2}(\sum_i s_i)^2 \tag{4.2}$$

where the *constraint coefficient* (imbalance parameter) α sets the relative strength between the cutsize and the balancing term. This balancing term represents a *global constraint.* The generic form of eq. (4.2) is

$$E = \text{“cost”} + \text{“global constraint”} \tag{4.3}$$

which is typical when casting combinatorial optimization problems onto neural networks. The origin of the difficulty inherent in these kind of problems is very transparent here; the problem is frustrated in the sense that the two constraints ("cost" and "global constraint") are competing with each other. This leads to the appearance of many local minima. Similar situations occur in spin-glass systems in physics, which are very closely related to artificial neural networks.

The next step will be to find an efficient procedure for minimizing eq. (4.2), such that local minima are avoided as much as possible. The MFT equations will turn out to be very powerful in this respect.

4.2 The Mean Field Equations

A straight-forward method for minimizing eq. (4.2) would be to update s_i according to a local optimization rule,

$$s_i = sgn\left(\sum_j (\omega_{ij} - \alpha)\right) \tag{4.4}$$

With this procedure the system typically ends up in the local minimum closest to the starting point, which is not desired. Thus, a stochastic algorithm is needed that allows for uphill moves. One such procedure is *simulated annealing* (SA) [10, 11]. This method consists of generating, via neighbourhood search methods, configurations obeying the Boltzmann distribution

$$P[s] = \frac{1}{Z} e^{-E[s]/T} \tag{4.5}$$

where Z is the *partition function*

$$Z = \sum_{[s]} e^{-E[s]/T} \tag{4.6}$$

and T is the temperature of the system. By generating configurations at successively lower T (annealing) they are less likely to get stuck in a bad local minimum. Needless to say such a procedure can be very CPU consuming.

The MFT approach aims at approximating the stochastic SA method with a set of deterministic equations. The derivation has two steps. First the partition function of eq. (4.6) 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 (in this case $\mathcal{R}$), introduce a new set of variables v_i living in this space, one for each spin, and set them equal to the spins with a Dirac delta-function. Then we can express the energy in terms of the v_i and Z takes the form

$$Z = \sum_{[s]} \int d[v] e^{-E[v]/T} \prod_i \delta(s_i - v_i) \tag{4.7}$$

Now, Fourier-expand the delta-functions, introducing a set of conjugate variables u_i:

$$Z = \sum_{[s]} \int d[v] \int d[u] e^{-E[v]/T} \prod_i e^{u_i(s_i - v_i)} \tag{4.8}$$

Then carry out the original sum over $[s]$:

$$Z \propto \int d[v] \int d[u] e^{-E[v]/T - \sum_i u_i v_i + \sum_i \log\cosh u_i}. \tag{4.9}$$

The original partition function is now rewritten entirely in terms of the new variables $[u, v]$, with an effective energy in the exponent. So far no approximation has been made. We next assume that Z in eq. (4.8) can be approximated by the maximal value of the integrand – the saddlepoint approximation. For the position of the saddlepoint we obtain

$$u_i = -\frac{\partial E[v]}{\partial v_i}/T\ , \tag{4.10}$$

$$v_i = \tanh u_i\ . \tag{4.11}$$

Combining eqs. (4.10, 4.11) one obtains

$$v_i = \tanh\left(-\frac{\partial E[v]}{\partial v_i}/T\right)\ . \tag{4.12}$$

which for the energy of eq. (4.2) reads

$$v_i = \tanh\left(\sum_j (\omega_{ij} - \alpha) v_j / T\right)\ . \tag{4.13}$$

The mean field variables v_i can be interpreted as $\langle s_i \rangle_T$, i.e. as thermal averages of the original binary spins. We thus recover the local updating equation (eqs. (2.1, 2.2)). What we have obtained is a set of deterministic equations emulating the stochastic behaviour. High temperatures correspond to very smooth sigmoids and the low temperature limit is given by a step function. The MFT equations are solved iteratively, either synchronously or asynchronously, under annealing in T.

The generic treatment of constraints here is very different from that in a more conventional heuristic treatment of optimization problems. For example in the case of graph bisection one typically 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 sharp contrast to neural network technique, where the constraints are implemented in a "soft" manner by a penalty term. The final MFT solutions could therefore sometimes suffer from a minor imbalance which is easily remedied, either by applying a *greedy heuristic* to the solutions, or by reheating the system and letting it anneal down again.

Very good numerical results were obtained for the graph bisection problem in ref. [12] for problem sizes ranging from 20 to 2000. The quality of the solutions are comparable to those of the CPU demanding simulated annealing method. The time consumption is lower than any other known method. The method of course becomes even more competitive with respect to time consumption if the intrinsic parallelism is exploited on dedicated hardware.

4.3 Mean Field Dynamics

The dynamics of feed-back ANN typically exhibit a behaviour with two phases (cf. fig. 4.2): at large enough temperatures ($T \rightarrow \infty$) the system relaxes into the trivial fixed point $v_i^{(0)} = 0$. As the

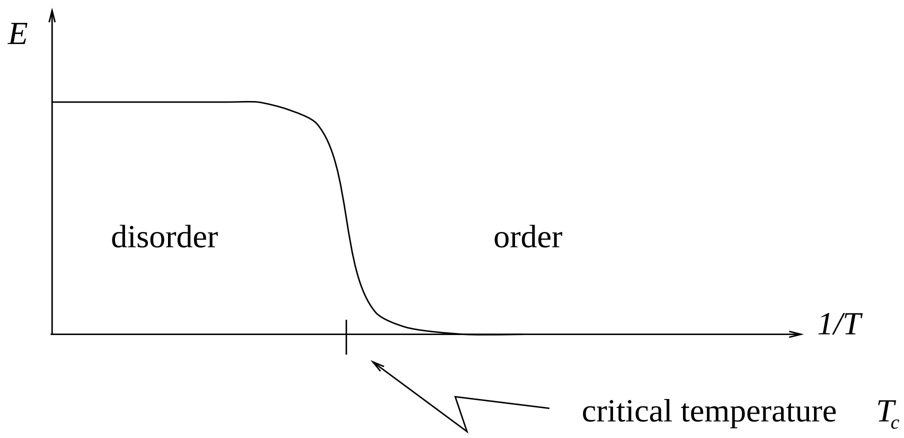


Figure 4.2: E as a function of T illustrating a phase transition.

temperature is lowered a phase transition is passed at $T = T_c$ and as $T \rightarrow 0$ fixed points $v_i^{(*)} = \pm 1$ emerge representing a specific decision made as to the solution to the optimization problems in question (see fig. 4.3). The position of T_c, which here depends on ω_{ij} and α, can be estimated by expanding the sigmoid function (tanh) in a power series around $v_i^{(0)} = 0$ (see fig. 4.3). The fluctuations around $v_i^{(0)}$

$$v_i = v_i^{(0)} + \epsilon_i \tag{4.14}$$

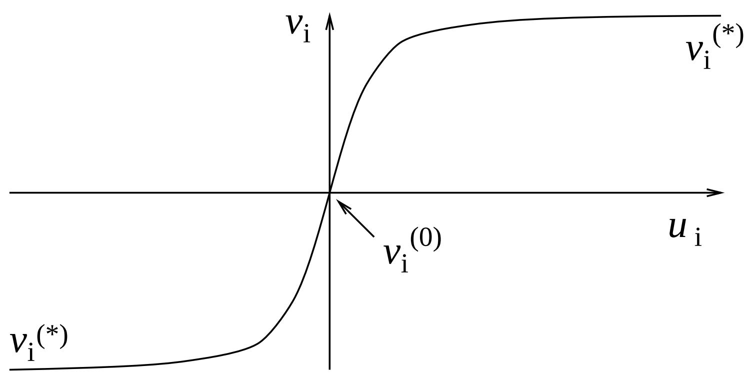


Figure 4.3: Fixed points in $\tanh(u_i)$.

satisfy

$$\epsilon_i = \frac{1}{T} \sum_j m_{ij} \epsilon_j \tag{4.15}$$

where $m_{ij} = \omega_{ij} - \alpha$. For synchronous updating it is clear that if one of the eigenvalues of the matrix $\mathbf{m}/T$ in eq. (4.15) is > 1 in absolute value, the fixpoint becomes unstable and the solutions will wander away into the nonlinear region. In the case of serial updating the philosophy is the same but the analysis slightly more complicated. We refer the reader to ref. [8] for a more detailed discussion. Finding the largest eigenvalue of $\mathbf{m}$ could be computationally explosive by itself. Is there an approximate way of doing it? Yes, providing $\langle \omega_{ij} \rangle$ and the corresponding standard deviation σ turns out to be sufficient for obtaining estimates within 10% of T_c. Also, this analysis is important for avoiding oscillatory behavior [14], which appears for eigenvalues < -1. We return to a more general and detailed treatment of this phase structure analysis in section 5.4.

With this method of estimating T_c in advance we can construct a reliable, parallelizable "black box" algorithm for solving problems of this kind. A prescription is given in section 5.5.

5 The Graph Partition Problem

A generalization of the graph bisection problem is *graph partitioning* (GP), where the N nodes are to be partitioned into K subsets of N/K nodes, again with minimal cutsize (see fig. 4.1b). This requires the introduction of a second index for the neurons. This can be done in two different ways, *neuron multiplexing* and *K-state neurons*. It turns out that the latter have several advantages.

5.1 Neuron Multiplexing – Ising Representation

We define the neurons in this case as follows

$$s_{ia} = 0, 1 \tag{5.1}$$

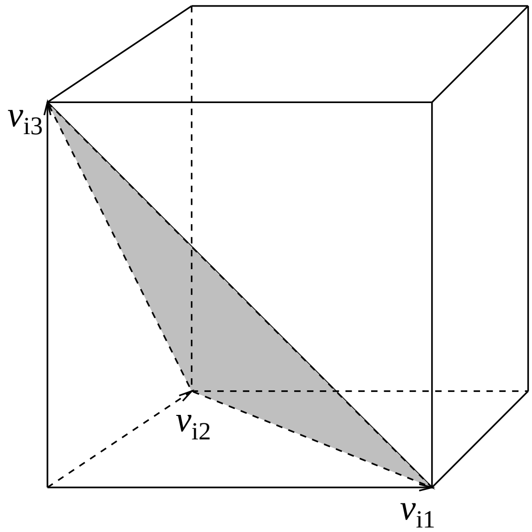


Figure 5.1: The volume of solutions corresponding to the neuron multiplexing encoding for K=3. The shaded plane corresponds to the solution space of the corresponding Potts encoding.

where the index i denotes the node ($i = 1, ..., N$) and a the subset ($a = 1, ..., K$). The neuron s_{ia} takes the value 1 or 0 depending on whether node i belongs to set a or not[6]. In analogy with eq. (4.2) we choose the energy

$$E = \frac{1}{2}\sum_{ij}\sum_{a\neq b}\omega_{ij}s_{ia}s_{jb} + \frac{\beta}{2}\sum_{i}\sum_{a\neq b}s_{ia}s_{ib} + \frac{\alpha}{2}\sum_{a}(\sum_{i}s_{ia} - \frac{N}{K})^2 \tag{5.2}$$

The first term minimizes the cutsize as in the bisection case and the last term represents the *global constraint* of equipartition; it is zero only if each of the K sets contains N/K nodes. The second term represents an additional *syntax constraint* that penalizes situations where a node ends up in more than one partition – only one a allowed for each i. Again we define mean field variables, $v_{ia} = \langle s_{ia}\rangle_T$ and the corresponding MFT equations are given by

$$v_{ia} = \frac{1}{2}\left[1 + \tanh\left(-\frac{\partial E[\vec{v}]}{\partial v_{ia}}/T\right)\right] \tag{5.3}$$

The solution space of these MFT equations consists of the interior of the direct product of N K-dimensional hypercubes. In fig. 5.1 we show the cube corresponding to $K = 3$. Let us next turn to an alternative encoding which compactifies the solution space by one dimension.

5.2 K-state Neurons – Potts Representation

The syntax term will be redundant, if the state space of the neurons is restricted such that exactly one neuron at every site is allowed to be on. This restriction can be compactly written as

$$\sum_{a} s_{ia} = 1 \tag{5.4}$$

[6]We here use $\{0, 1\}$ notation in order to get a more convenient form of the energy function.

Thus for every i, s_{ia} should be 1 for only one value of a, and 0 for the rest. To explain how this is done, we choose a K-dimensional vector representation of the neuron states: The a:th possible state of the neuron vector $\vec{s}_i$ associated with the node i is represented by the unit vector in the a-direction. In this way, the K different states, $(1, 0 \ldots 0)$, $(0, 1 \ldots 0)$ $\ldots$ $(0, 0 \ldots 1)$, are all normalized and mutually orthogonal. The number of states available at every node is thereby reduced from 2^K to K. Technically, we have a K-state Potts model [15] at our hands. In fig. 5.1 is shown the space of states at one node for the case K=3.

The energy function of eq. (5.2) can now be rewritten, using the constraint of eq. (5.4), as

$$E = -\frac{1}{2}\sum_{ij}\sum_{a}\omega_{ij}s_{ia}s_{ja} - \frac{\beta}{2}\sum_{i}\sum_{a}s_{ia}^2 + \frac{\alpha}{2}\sum_{ij}\sum_{a}s_{ia}s_{ja} \tag{5.5}$$

or, in vector notation,

$$E = -\frac{1}{2}\sum_{ij}\omega_{ij}\vec{s}_i\vec{s}_j - \frac{\beta}{2}\sum_{i}\vec{s}_i^2 + \frac{\alpha}{2}(\sum_{i}\vec{s}_i)^2 \tag{5.6}$$

disregarding an unimportant constant term. Note that the second term can now be dropped, since it is anyway a constant. We will however keep it, since it will for some applications improve the solution quality. It also turns out to be a convenient regulator for avoiding chaotic bahaviour in synchronous updating (see below). With $\beta = 0$, this expression has exactly the same structure as the energy (eq. 4.2) for the graph bisection problem. Indeed, for $K = 2$, they are completely equivalent (apart from a factor 2).

5.3 Mean Field Potts Equations

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

$$Z = \sum_{[\vec{s}]} e^{-E[\vec{s}]/T} \tag{5.7}$$

Proceeding in analogy to what was done in the binary case in the previous section, the partition function is transformed into

$$Z = \int d[\vec{u}]d[\vec{v}]\exp(-E[\vec{v}]/T - \sum_{i}\vec{u}_i\cdot\vec{v}_i)\sum_{[\vec{s}]}e^{\sum_i \vec{u}_i\cdot\vec{s}_i}. \tag{5.8}$$

Performing the original sum, we are left with

$$Z = \int d[\vec{u}]d[\vec{v}]\exp(-E[\vec{v}]/T - \sum_{i}\vec{u}_i\cdot\vec{v}_i + \sum_{i}\log\sum_{a}e^{u_{ia}}). \tag{5.9}$$

The MFT approximation to $\langle\vec{s}_i\rangle$ is again given by the value of $\vec{v}_i$ at a saddle-point of the effective energy of $[\vec{u}, \vec{v}]$ in the exponent of eq. (5.9). Differentiating, we obtain the Potts MFT equations (cf. eqs. (4.10, 4.11)):

$$u_{ia} = -\frac{\partial E[\vec{v}]}{\partial v_{ia}}/T, \tag{5.10}$$

$$v_{ia} = \frac{e^{u_{ia}}}{\sum_b e^{u_{ib}}}, \tag{5.11}$$

where a denotes the vector components.

From the latter follows, that

$$\sum_a v_{ia} = 1. \tag{5.12}$$

One can thus think of the mean field v_{ia} as the probability for the Potts neuron i to be in state a (*fuzzy logic*). The set of values available to $\vec{v}_i$ is the inside of a simplex in a $(K-1)$-dimensional hyperplane, with its corners on the allowed states for the Potts neuron $\vec{s}_i$. For $K = 3$ this is a triangle (c.f. Fig. 5.1). For $K = 2$, we recover the formalism of the Ising case, provided that we define v_i as $v_{i1} - v_{i2}$, etc. Thus, we have a natural generalization of the MFT approach from $K = 2$ to general multi-state systems.

5.4 Mean Field Dynamics

5.4.1 High Temperature

Again we can analyze the linearized equations as in section 4.3 in order to estimate the critical temperature T_c. The value of the critical temperature depends on the selfcoupling β, and on precisely how the updating is done. In *serial mode*, 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 the trivial fixed point $v_{ia}^{(o)}$, which is a completely symmetrical state, where all v_{ia} are equal

$$v_{ia}^{(0)} = \frac{1}{K} \tag{5.13}$$

As the temperature is lowered a phase transition is passed at $T = T_c$ and as $T \to 0$ fixed points $v_{ia}^{(*)}$ emerge which are characterized by

$$\Sigma \equiv \frac{1}{N} \sum_{ia} v_{ia}^{(*)2} \to 1 \tag{5.14}$$

where we have introduced an order parameter, the **saturation** Σ. In fig. 5.2 this is illustrated for a $K = 4, N = 100$ graph partition problem. The position of T_c depends on ω_{ij}, α and β. As mentioned above the fixed point corresponds to the symmetry point of the nonlinear gain functions in the MFT equations (eqs. (5.10, 5.11)), where these are almost linear (see fig. 4.3). Let us consider fluctuations around the trivial fixed point (cf. eq. (4.14))

$$v_{ia} = v_{ia}^{(0)} + \epsilon_{ia} \tag{5.15}$$

First, from eqs. (5.5, 5.10, 5.11) it follows that

$$\sum_a \epsilon_{ia} = 0 \tag{5.16}$$

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

$$\epsilon_i = \frac{1}{KT} \sum_j M_{ij} \epsilon_j \tag{5.17}$$

Figure 5.2: (a). Internal energy $E(v_{ia})$ as a function of T for a $K = 4, N = 100$ graph partition problem. Also shown is the prediction from the linearized analysis below for T_c with $\xi = 1.5$. (b). The saturation Σ as a function of T for the same problem

(cf. eq. (4.15)) where

$$M_{ij} = \omega_{ij} - \alpha + \beta\delta_{ij} \tag{5.18}$$

in the case of GP. For synchronous updating it is clear that if one of the eigenvalues of M_{ij}/KT in eq. (5.17) is > 1 in absolute value the solutions will wander away into the nonlinear region. Hence T_c will be determined by the eigenvalue distribution of M_{ij}, and will obviously scale like $1/K$ for a fixed graph. In the case of serial updating the philosophy is the same but the analysis slightly more complicated. We will therefore treat the two cases separately. A more detailed treatment of the contents in this section can be found in [8].

Synchronous Updating

In this case T_c is given by

$$T_c = \frac{1}{K}(largest\ \ absolute\ \ eigenvalue\ \ of\ \ \mathbf{M}) \tag{5.19}$$

It turns out to be convenient to divide $\mathbf{M}$ up into its diagonal and off-diagonal parts

$$M_{ij} = (\beta - \alpha)\delta_{ij} + A_{ij} \tag{5.20}$$

where the off-diagonal part $\mathbf{A}$ is given by

$$A_{ij} = \omega_{ij} - \alpha(1 - \delta_{ij}) \tag{5.21}$$

In terms of the extreme eigenvalues λ_{max} and λ_{min} of $\mathbf{A}$ (which depend on α but <u>not</u> β) we thus have

$$T_c = \frac{1}{K}max(\alpha - \beta - \lambda_{min}, \beta - \alpha + \lambda_{max}) \tag{5.22}$$

Figure 5.3: **(a).** The $(KT, \beta\ )$-plane for synchronous updating. The two lines correspond to eigenvalues -1 and $+1$ respectively for $t = 0$. **(b).** The $(KT, \beta\ )$-plane for serial updating. The two lines correspond to unitary eigenvalues $\neq 1$ and $= 1$ respectively for $t = 0$.

We stress that this is an exact equation for T_c. Also note the simple β-dependence; for a fixed α, $T_c(\beta)$ can be obtained by probing it for two values of β (cf. fig. 5.3a). In ref. [8] it is shown, using the average $t = \omega_{ij}$ and the standard deviation σ_t of the off-diagonal elements of ω_{ij}, that for reasonable values of α, λ_{min} is well approximated by

$$\lambda_{min} = -(N-1)(\alpha - t) \tag{5.23}$$

and that the other eigenvalues are comparatively closely distributed around a mean value $\alpha - t$ with the standard deviation $\sqrt{N}\sigma_t$. We can thus write λ_{max} as

$$\lambda_{max} = \alpha - t + \xi\sqrt{N}\sigma_t \tag{5.24}$$

where $\xi > 0$ is a numerical factor that gives the deviation of λ_{max} from the average in units of the standard deviation. Empirically we find ξ to lie in the region 1.5 to 2.0 for the graph partition problem.

From eq. (5.22) we can identify two distinct regions in the (KT, β)-plane, with different behaviours (see fig. 5.3a).

Region (1). In this region one has

$$\beta \leq \frac{N}{2}\alpha - \frac{N-2}{2}t - \frac{1}{2}\xi\sqrt{N}\sigma_t \tag{5.25}$$

and

$$T_c = \frac{1}{K}[\alpha - \beta + (N-1)(\alpha - t)] \tag{5.26}$$

corresponding to a non-degenerate eigenvalue -1 of the evolution matrix in eq. (5.20). This is a single eigenvalue. Due to its sign the trivial fixed point solution will bifurcate into an alternating phase when the temperature is lowered through T_c; all the $\vec{v}_i$ will jump more or less uniformly back and forth. Thus when executing the algorithm synchronously one should avoid this region. Such oscillating behaviour has been observed in spin-glass systems when updating MFT equations synchronously [14].

Region (2). In this region one has

$$\beta \geq \frac{N}{2}\alpha - \frac{N-2}{2}t - \frac{1}{2}\xi\sqrt{N}\sigma_t \tag{5.27}$$

and

$$T_c = \frac{1}{K}[\beta - t + \xi\sqrt{N}\sigma_t] \tag{5.28}$$

corresponding to the eigenvalue +1, which in the limit of small fluctuations (σ_t) is ($N-1$)-fold degenerate. Hence the situation is very different. The dynamics will be smooth with no oscillations. Due to the high approximate degeneracy of the relevant eigenvalue, the time evolution of the system will have a chance to "feel" its way to a good solution.

Serial Updating

The analysis for this case is somewhat more elaborate. Again the relevant matrix is $\mathbf{M}/KT$, but in this case the updating proceeds row by row, and the linearized updating equation for the perpendicular fluctuations ϵ_{ia} (cf. eq.(5.17)) now reads

$$\epsilon'_i = \frac{1}{KT}(\sum_{j=1}^{i-1} M_{ij}\epsilon'_j + \sum_{j=i}^{N} M_{ij}\epsilon_j) \tag{5.29}$$

where new values ϵ'_j are used for the previously updated components. Dividing $\mathbf{M}$ up into an upper ($\mathbf{U}$), a diagonal ($\mathbf{D}$) and a lower ($\mathbf{L}$) part, eq.(5.29) can be rewritten in matrix notation

$$\epsilon' = \frac{1}{KT}[\mathbf{L}\epsilon' + (\mathbf{D}+\mathbf{U})\epsilon]\ . \tag{5.30}$$

Rearranging, we obtain

$$\epsilon' = (KT - \mathbf{L})^{-1}(\mathbf{U}+\mathbf{D})\epsilon \tag{5.31}$$

where the LHS is the effective synchronous updating matrix that corresponds to serial updating. In other words we have "synchronized" the analysis and again T_c is characterized by a unit modulus for the largest eigenvalue μ. In the case of $\mu = 1$, the result is the same as that of synchronous updating. For $|\mu| = 1, \mu \neq 1$, it follows from a Hermiticity argument that this is possible only for $T = (1/K)(\alpha - \beta)$. Thus one obtains for the critical temperature

$$T_c = \frac{1}{K}max[\alpha - \beta, \beta - \alpha + \lambda_{max}] \rightarrow T_c = \frac{1}{K}max[\alpha - \beta, \beta - t + \xi\sqrt{N}\sigma_t] \tag{5.32}$$

with ξ as in eq. (5.24), and again we find two regimes (see fig. 5.3b).

5.4.2 Low Temperature

At low temperature, any difference between the components 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 $T \rightarrow 0$. Eventually the network should settle at a fixpoint close to an allowed state of the Potts spin system, representing a (possibly approximate) solution to the optimization problem.

Also here, the β value has an impact: A higher value tends to stabilize also bad decisions (positive feed-back), while a lower value destabilizes even a good solution, and can lead to cyclic or even chaotic behaviour.

For serial mode, there is never a problem to choose a β that is satisfactory in both temperature limits [8]. This is often not the case for synchronous mode, so serial mode will be understood where not otherwise stated.

5.5 A Generic Algorithm

For most problems, the proper values of β and T_0 can be calculated or estimated in advance. The complete neural network algorithm for a generic K-state Potts system will then look as follows:

A generic Potts neural network algorithm.

1. Choose problem - ω_{ij}
2. Find the approximate phase transition temperature by linearizing eq. (5.11).
   (for details see ref. [8])
3. Add a β-term if necessary.
4. Initialize the neurons v_{ia} with 1/K $\pm$ random values, and set $T_0 = T_c$.
5. Until $\Sigma \equiv \frac{1}{N}\sum_{ia} v_{ia} \geq 0.99$, do:
   5.1 At each T_n update all v_{ia};
   $v_{ia} = \frac{e^{u_{ia}}}{\sum_b e^{u_{ib}}}$.
   5.2 $T_{n+1} = 0.9 \times T_n$.
6. After $\Sigma \geq 0.99$ is reached perform a greedy heuristics if needed to account for possible imbalances or rule violations.

5.6 Numerical Results

In ref. [8] the neural approach to graph partition was compared with that of simulated annealing. Not unexpectedly it turns out that the neuron multiplexing method requires "custom-made" tailoring of parameters for each application. This is not acceptable. We interpret this weakness as being due to the redundancy in the encoding – the dynamics is not explicitly confined to the hyperplane ($\sum_a v_{ia} = 1$). The Potts encoding, on the other hand, turns out to be much more stable. We therefore stick to this approach in what follows. In fig. 5.4 we compare the performance of the Potts neural network with simulated annealing. The graphs were generated by randomly connecting the nodes with probability $P = 10/N$. The results are impressive! The neural network

Figure 5.4: Comparison of Potts neural network solutions versus simulated annealing and random distributions for a 10×100 graph partition problem. The histograms are based on 50 experiments for the neural network and simulated annealing algorithms and 1000 for the random distributions (from ref. [8]).

algorithm performs as well (in some cases even better) than the simulated annealing method with its excessive annealing and sweep schedule (see [8]). This is accomplished with a very modest number of iterations, $O(50-100)$. In fact the number of iterations is empirically independent of problem size [8].

6 The Traveling Salesman Problem

In the traveling salesman problem (TSP) a closed tour with minimal distance is to be chosen such that each city is visited exactly once. To encode this we define neurons s_{ia} to be 1 if city i has the a:th tour number and 0 otherwise and let d_{ij} be the distance between city i and j. This problem (see fig. 4.1c) is somewhat related to graph partition and based on our experience above we stick to the Potts representation, eq. (5.4), in which the energy reads

$$E = \sum_{ij=1}^{N} d_{ij} \sum_{a=1}^{K} s_{ia} s_{j(a+1)} - \frac{\beta}{2} \sum_{i=1}^{N} \sum_{a=1}^{K} s_{ia}^2 + \frac{\alpha}{2} \sum_{a} (\sum_{i} s_{ia})^2 \tag{6.1}$$

where we again have included an auxiliary β-term for dynamical reasons[7]. As in the graph partition case the energy is minimized by iteratively solving the Potts MFT equations (eqs. (5.10, 5.11)) using the generic prescription of sec. 5.5. Again an analysis of the linearized dynamics along the lines of the previous sections can be done to estimate T_c. In ref. [8] the Potts approach for TSP

[7]This is of course not the only way to code the problem. One could e.g. interchange the roles of the labels i and a.

was explored numerically for problem sizes up to N_{cities}=200, again with encouraging results. The neural network solution qualities fall on the tail of the simulated annealing ones. Defining the relative quality q as the ratio

$$q = \frac{\langle E(\text{random config.})\rangle - \langle E(\text{neur. netw.})\rangle}{\langle E(\text{random config.})\rangle - \langle E(\text{sim. ann.})\rangle} \tag{6.2}$$

we find that q never falls below 92%. Furthermore, as in the graph partition case, there are no real bad solutions.

As far as convergence time goes we have observed no K or N dependence 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 τ of the Potts neuron algorithm is $\propto N^3$ for TSP, as in the neuron multiplexing case [7]. This should be compared with N^2 for simulated annealing. The real advantage lies in the inherent parallelism, which when exploited gives $\tau \propto$ *constant* for a general purpose parallel machine or custom made hardware.

For TSP there is also an alternative neural approach, the deformable templates, to which we will return in section 8.

7 Scheduling Problems

A Potts neural network formulation is almost ideal for scheduling problems – these have a natural formulation in terms of Potts systems. In its purest form, a scheduling problem consists entirely in 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 identify exact (= legal) solutions by inspection. In many applications, there exist additional preferences within the set of legal schedules.

First we will discuss a synthetic problem [16], 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 [17], in this case a Swedish high school.

7.1 A Synthetic Example

In ref. [16] a simplified scheduling problem, where N_p teachers give N_q classes in N_x class rooms at N_t time slots, was mapped onto a Potts neural network. In this problem one wants solutions where all N_p teachers give a lecture to each of the N_q classes, using the available space-time slots with no conflicts in space (class rooms) or time. These are the **basic constraints** that have to be satisfied. In addition various **preferences** regarding continuity in class-rooms etc. were considered. The basic entities of this problem can be represented by four sets consisting of N_p,N_q,N_x and N_t elements respectively.

There is a very transparent way to describe this problem that naturally lends itself to the Potts

Figure 7.1: Mapping of events (p,q) onto space-time slots (x,t).

neural encoding, where **events**, defined by teacher-class pairs (p,q), are mapped onto **space-time slots** (x,t) (see fig. 7.1). Potts neurons $s_{pq;xt}$ are defined to be 1 if the event (p,q) takes place in the space-time slot (x,t) and 0 otherwise. 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) should not occupy the same space-time slot (x,t).
3. A teacher p should have at most one class at a time.
4. A class q should have at most one teacher at a time.

A schedule fulfilling all the basic constraints is said to be **legal**. The first constraint can be embedded in the neural network in terms of the Potts normalization condition

$$\sum_{x,t} s_{pq;xt} = 1 \tag{7.1}$$

for each event (p,q). In other words we have N_pN_q neurons, each of which has N_xN_t possible states. The other three constraints are implemented using energy penalty terms as follows:

$$E_{XT} = \frac{1}{2}\sum_{x,t}\sum_{p_1,q_1}\sum_{p_2,q_2} s_{p_1q_1;xt}s_{p_2q_2;xt} = \frac{1}{2}\sum_{x,t}[\sum_{pq} s_{pq;xt}]^2 \tag{7.2}$$

$$E_{PT} = \frac{1}{2}\sum_{p,t}\sum_{q_1,x_1}\sum_{q_2,x_2} s_{pq_1;x_1t}s_{pq_2;x_2t} = \frac{1}{2}\sum_{pt}[\sum_{qx} s_{pq;xt}]^2 \tag{7.3}$$

$$E_{QT} = \frac{1}{2}\sum_{q,t}\sum_{p_1,x_1}\sum_{p_2,x_2} s_{p_1q;x_1t}s_{p_2q;x_2t} = \frac{1}{2}\sum_{qt}[\sum_{px} s_{pq;xt}]^2 \tag{7.4}$$

This way of implementing the constraints is by no means unique. In particular one can add various diagonal terms that merely have the effect of adding a fixed constant to the energy. Again mean field variables are introduced, $v_{pq;xt} \sim \langle s_{pq;xt}\rangle_T$ and the corresponding MFT equations read (cf. eqs. (5.10, 5.11))

$$u_{pq;xt} = -\frac{1}{T}\cdot\frac{\partial E}{\partial v_{pq;xt}} \tag{7.5}$$

$$v_{pq;xt} = \frac{e^{u_{pq;xt}}}{\sum_{x't'} e^{u_{pq;x't'}}}. \tag{7.6}$$

This is a straightforward approach that works well. However, an important simplification can be made. It turns out that with the encoding $v_{pq;xt}$ the MFT-equations give rise to two phase transitions; one in x and one in t. In other words the system naturally factorizes into two parts. It is therefore economical to implement this factorization already at the encoding level. This is done by replacing $s_{pq;xt}$ by x-neurons $s^{(X)}_{pq;x}$ and t-neurons $s^{(T)}_{pq;t}$

$$s_{pq;xt} = s^{(X)}_{pq;x} s^{(T)}_{pq;t} \tag{7.7}$$

with separate Potts conditions replacing eq. (7.1)

$$\sum_x s^{(X)}_{pq;x} = 1 \tag{7.8}$$

and

$$\sum_t s^{(T)}_{pq;t} = 1 \tag{7.9}$$

respectively. This means that the number of degrees of freedom reduces from $N_p N_q N_x N_t$ to $N_p N_q (N_x + N_t)$. As a consequence the sequential execution time goes down.

The performance of this algorithm was investigated in ref. [16, 17] for a variety of problem sizes and for different levels of difficulty measured by the ratio between the number of events and the number of available space-time slots. It was found that the algorithm consistently found legal solutions with very modest convergence times for problem sizes $(N_p, N_q) = (5,5)$,...,$(12,12)$. With convergence time we here mean the total number of sweeps needed to obtain a legal solution no matter how many trials it takes. Also when introducing preferences, e.g. for having subsequent lessons in the same room, very good solutions were obtained.

7.2 A Realistic Example

The synthetic scheduling problem of ref. [16] contains several simplifications as compared to realistic problems. In ref. [17] more realistic problems from the Swedish high school system were explored, which required an extended formalism. Here we give a list of items that an extended formalism will need to handle.

1. One week periodicity (occasionally extended to two- or four-week periodicity).
2. In ref. [16] each teacher has a class exactly once. In this case he has to give lessons in certain subjects a few hours a week, appropriately spread out.
3. In ref. [16] it was assumed that all class rooms were available for all subjects. This is not the case in reality. Many subjects require special purpose rooms.
4. Many subjects are taught for two hours in a row (double hours).
5. Group formation. For some optional subjects the classes are broken up into option groups temporarily forming new classes.

6. Various preferences have to be considered.

When taking these points into account in our formalism we cannot keep p and q as the independent quantities. Rather we define an independent variable i (event index) to which p and q are attributes, $p(i)$ and $q(i)$. The values of these exist in a look-up table. This table contains all the necessary information to process each event i. The time index t we subdivide into weekdays (d) and daily hours (h). The Potts conditions (eqs. (7.8, 7.9)) now read

$$\sum_x s_{i;x}^{(X)} = 1 \tag{7.10}$$

and

$$\sum_t s_{i;t}^{(T)} = 1 \tag{7.11}$$

This also implies that the Potts neurons will have different number of components. This situation can also be handled when analyzing the MFT dynamics in terms of fixed points and T_c [17].

To facilitate the handling of double hours we introduce effective t-neurons $\tilde{s}_{i;t}^{(T)}$ defined as

$$\tilde{s}_{i;t}^{(T)} = \sum_{k=0}^{g_i-1} s_{i;t-k}^{(T)} \tag{7.12}$$

where the multiplicity g_i is *1* for single hours and *2* for double hours.

There are basically four kinds of *preferences* we encounter when scheduling Swedish high schools. (i). The lunch "lesson" has to appear within 3 hours around noon. (ii). The different lessons for a class in a particular subject should be *spread* as much as possible over the week days. (iii). The schedules should have as few "holes" as possible: lessons should be *glued* together. (iv). Teachers may have various individual preferences.

Point i) is really a syntactic constraint rather than a preference. It is easily taken into account by restricting the Potts index range for the relevant events. Spreading, gluing and teacher preferences can be accommodated by an appropriate choice of penalty functions (see ref. [17]).

As in the case of graph problems and TSP above an automated implementation of the algorithm exists along the lines of the generic prescription defined in sec. 5.5. In fig. 7.2 we show a typical evolution of E_{hard} (the basic constraint part), Σ_T, Σ_X and T as a function of N_{sweep} for a Swedish high school problem. These problems typically have ~ 90 teachers, ~ 50 weakly hours, ~ 45 classes and ~ 60 class-rooms, which corresponds to $\sim 10^{4600}$ possible choices. In the factorized Potts formulation these are handled by $\sim 10^5$ neural degrees of freedom. High quality solutions emerge.

A revision capability is inherent in our 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 size are so complex that even several man-months of human planning will in general not yield solutions that will meet all the requirements

Figure 7.2: Energy (E_{hard}), saturations (Σ_T, Σ_X) and temperature (T) as functions of N_{sweep} for one run with a Swedish high school problem.

in an optimal way. We have not been able to find any algorithm in the literature, that solves a real-world problem with this complexity. Existing commercial software packages do not solve the entire problem. Rather the problem is solved with an interactive user taking step-wise decisions.

The scheduling problems we considered here were all based upon equality constraints. There are many resource allocation problems, which contain inequalities. In section 9 we will show how do deal with these within the ANN framework.

8 Deformable Templates

For low-dimensional geometric optimization problems it is sometimes advantageous to abandon the pure Potts probabilistic description in favour of an expressly geometric template formulation, thereby reducing the number of degrees of freedom. For the traveling salesman problem, such a formulation is given by the *elastic net* approach [9], to be described below.

Denote the N city positions in the TSP by $\vec{x}_i$ (see fig. 8.1). We are going to match these cities with $M > N$ template coordinates $\vec{y}_a$ such that $\sum_a |\vec{y}_a - \vec{y}_{a+1}|$ is minimized and each $\vec{x}_i$ is matched by precisely one $\vec{y}_a$. For each city i define a Potts spin s_{ia} to be 1 if a is matched to i and 0 otherwise.

Consider the following energy expression, to be minimized w.r.t. s_{ia} and $\vec{y}_a$:

$$E(s_{ia}, \vec{y}_a) = \frac{1}{2T} \sum_{ia} s_{ia} |\vec{x}_i - \vec{y}_a|^2 + \frac{\gamma}{2} \sum_a |\vec{y}_a - \vec{y}_{a+1}|^2 \tag{8.1}$$

The first term enforces matching: it is minimized when each $\vec{x}_i$ coincides with that $\vec{y}_a$ for which

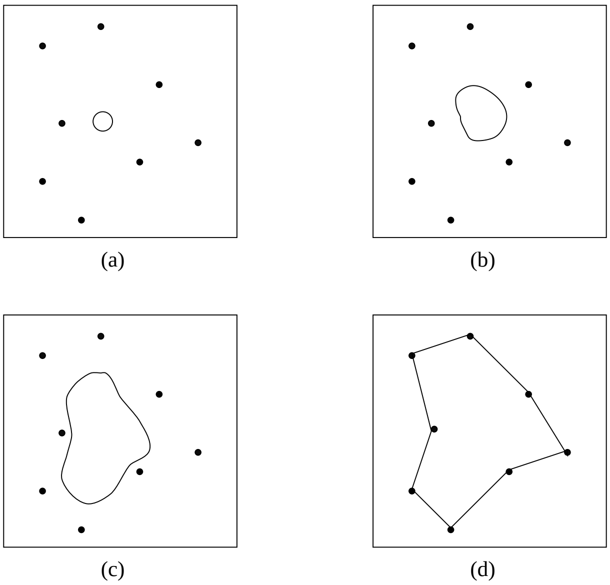


Figure 8.1: Evolution of the template "snake" from high to low temperatures.

$s_{ia} = 1$. The second term is a kind of tour length term. As $T \to 0$, the full E will be minimized when the $\vec{y}_a$ equidistantly trace out the shortest closed tour visiting all cities.

The parameter γ governs the relative strength between matching and tour length (cf. α and β in the neural descriptions above), and should be suitably chosen depending of M, N and the typical distances in the problem.

The problem is thus encoded in terms of two distinct types of variables: the Potts spins s_{ia} and the template coordinates $\vec{y}_a$. The Boltzmann distribution for the energy in eq. (8.1) reads

$$P(s_{ia}, \vec{y}_a; T) = \frac{e^{-E(s_{ia}, \vec{y}_a)/T}}{Z} \tag{8.2}$$

with the partition function Z given by

$$Z = \sum_{s_{ia}} \int d\vec{y}_a e^{-E(s_{ia}, \vec{y}_a)/T} \tag{8.3}$$

We can now define so-called *marginal distributions* by integrating out either the Potts spin or the template coordinate degrees of freedom. If we choose the latter alternative we end up with a pure neural description of the problem similar to the one in the previous section. We will here choose the former alternative. Performing the sum over s_{ia} [18, 19] we obtain

$$Z = \int d\vec{y}_a e^{-\gamma \sum_a |\vec{y}_a - \vec{y}_{a+1}|^2/2T} \prod_i \left(\sum_a e^{-|\vec{x}_i - \vec{y}_a|^2/2T^2} \right) \tag{8.4}$$

Only those configurations where s_{ia} is 1 for only one a for each i have been summed over. We rewrite eq. (8.4) as

$$Z = \int d\vec{y}_a e^{-E_{\text{eff}}(\vec{y}_a)/T} \tag{8.5}$$

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

$$E_{\text{eff}}(\vec{y}_a) = -T \sum_i \log(\sum_a e^{-|\vec{x}_i - \vec{y}_a|^2/2T^2}) + \gamma \sum_a |\vec{y}_a - \vec{y}_{a+1}|^2/2 \tag{8.6}$$

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

$$\Delta \vec{y}_a = \eta \left[\sum_{ia} \tilde{v}_{ia}(\vec{x}_i - \vec{y}_a)/T + \gamma(\vec{y}_{a+1} - 2\vec{y}_a + \vec{y}_{a-1}) \right] \tag{8.7}$$

where η is a time-step, and the Potts factor (cf. eq. (5.11)) $\tilde{v}_{ia}$ is given by

$$\tilde{v}_{ia} = \frac{e^{-|\vec{x}_i - \vec{y}_a|^2/2T^2}}{\sum_b e^{-|\vec{x}_i - \vec{y}_b|^2/2T^2}} \tag{8.8}$$

A typical snake evolution is schematically depicted in fig. 8.1. Here, in contrast to the Potts description in the previous section, the logical units are only implicit in the dynamics; the primary dynamical degrees of freedom are instead the analog variables $\vec{y}_a$. The full algorithm will look as follows.

An elastic net algorithm for the TSP.

1. Choose problem – $\{\vec{x}_i,\ i = 1, N\}$.
2. Choose number of templates M, $M \gg N$.
3. Choose suitable γ and initial temperature T_0.
4. Compute the center of gravity of the cities $\vec{x}_i$ and displace it slightly with a random seed. Place templates $\vec{y}_a$ with equal spacing on a circle around this center.
5. Until the system has settled, do:

   5.1 Update templates $\vec{y}_a$;
   $\vec{y}_a(T_n) = \vec{y}_a(T_{n-1}) + \Delta\vec{y}_a$
   where $\Delta\vec{y}_a$ is given by eq. (8.7).

   5.2 $T_n = 0.9 \times T_{n-1}$.

How does this algorithm work? The first term in eq. (8.6) contains a sum of Gaussians around the templates with width T. At high T this makes all the $\vec{y}_a$ compete to match all cities; in effect they will be attracted towards the center of mass. As T decreases, the Potts factors become more selective, and the range of competition for each $\vec{x}_i$ is focused on a smaller neighbourhood. Finally one $\vec{y}_a$ is singled out to match each $\vec{x}_i$, while the remaining $\vec{y}$'s become arranged equidistantly along straight lines connecting the cities.

This algorithm produces very high quality solutions [13]. And, very importantly, an N-city problem requires only $2N$ degrees of freedom. For problems like TSP, embedded in a low dimensional space, a template method is to be preferred to the pure neural (Potts) one. However, for high-dimensional or non-geometric problems, like scheduling [16, 17], the Potts neural network approach is ideal.

The structure of the first term in eq. (8.7) is similar to the one for collective updating of self-organizing networks [20]. The templates $\vec{y}_a$ play the role of Gaussian centres that adapt to the distribution of cities. The main differences between this approach and the self-organizing networks are **i**) the neighborhood function, here given by $\Lambda = \tilde{v}_{ia}$, i.e. each "weight" (template) is updated in some sense through the Potts updating equation (eq. (8.8)), and **ii**) the number of templates, which is here $\gg N$.

This deformable templates approach has also been successfully applied to track finding [19, 21, 22]

9 Optimization Problems with Inequality Constraints – the Knapsack Problem

The application areas dealt with above (traveling salesman, graph partition 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, in particular those of resource allocation type, one has to deal with *inequalities*. The objective of this section is to develop a mapping and MFT method to deal with this kind of problems. As a typical resource allocation problem we choose the knapsack problem for our studies.

In the knapsack problem one has a set of N *items* i with associated *utilities* c_i and *loads* a_{ki}. The goal is to fill a "knapsack" with a subset of the items such that their total utility,

$$U = \sum_{i=1}^{N} c_i s_i \ , \tag{9.1}$$

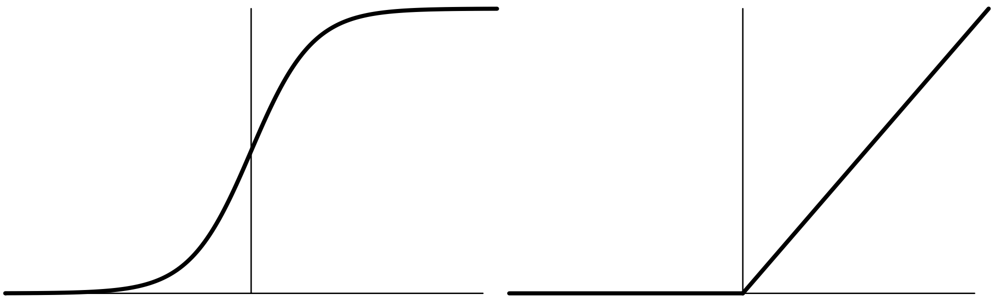
is maximized, subject to a set of M load constraints,

$$\sum_{i=1}^{N} a_{ki} s_i \leq b_k \ , \ k = 1, M \ , \tag{9.2}$$

defined by load *capacities* b_k.

In eqs. (9.1, 9.2) s_i are binary $\{0, 1\}$ decision variables, representing whether item i goes into the knapsack. The variables (c_i, a_{ki} and b_k) that define the problem are all real numbers.

We will consider a class of problems, where a_{ki} and c_i are independent uniform random numbers on the unit interval, while b_k are fixed to a common value b. With a b above $N/2$, the problem becomes trivial – the solution will have almost all $s_i = 1$. Conversely, with $b \ll N/4$, the number



(a) (b)

Figure 9.1: (a) A sigmoid $g(x;T)$ (cf. eq. 2.2). (b) The penalty function $x\theta(x)$ of eq. (9.4).

of allowed configurations will be small and an exact solution can easily be found. We pick the most difficult case, defined by $b = N/2$. The expected number of used items in an optimal solution will then be about $N/2$, and an exact solution becomes inaccessible for large N.

In the optimal solution to such a problem, there will be a strong correlation between the value of c_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 more homogeneous c_i distributions – *homogeneous* problems. The extreme case is when c_i are constant, 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.

We start by mapping the problem defined in eqs. (9.1, 9.2) onto a generic neural network energy function E,

$$E = -\sum_{i=1}^{N} c_i s_i + \alpha \sum_{k=1}^{M} \Phi \left(\sum_{i=1}^{N} a_{ki} s_i - b_k \right) , \tag{9.3}$$

where Φ is a penalty function to ensure that the constraint in eq. (9.2) is fulfilled. The coefficient α governs the relative strength between the utility and constraint terms. For equality constraints an appropriate choice of $\Phi(x)$ would be $\Phi(x) = x^2$. Having inequalities we need a $\Phi(x)$ that only penalizes configurations for which $x \geq 0$. One possibility is to use a sigmoid, $\Phi(x) = g(x;T)$ (see fig. 9.1a). This option has the potential disadvantage that the penalty becomes constant ($= \alpha$) for large constraint violations. An alternative that gives penalty in proportion to the degree of violation is

$$\Phi(x) = x\Theta(x) \tag{9.4}$$

This function (see fig. 9.1b) has the additional advantage that no extra parameter like the temperature T in the sigmoid is needed. The slope of Φ is implicitly given by α in eq. (9.3). The $x\Theta(x)$ alternative consistently gives better performance and is used in what follows.

Minimizing eq. (9.3) is again done with the mean field approximation. Due to the non-polynomial form of the constraint (eq. (9.4)) special care is needed when implementing the MFT approximation.

As is shown in ref. [23] this can be done by replacing the derivative $\frac{\partial E}{\partial v_i}$ in

$$v_i = g\left(-\frac{\partial E}{\partial v_i}; T\right) \tag{9.5}$$

by a difference,

$$\frac{\partial E}{\partial v_i} \rightarrow -c_i + \alpha \sum_{k=1}^{M} \left[\Phi(\sum_{j=1}^{N} a_{kj} v_j - b_k)|_{v_i=1} - \Phi(\sum_{j=1}^{N} a_{kj} v_j - b_k)|_{v_i=0} \right] . \tag{9.6}$$

Eqs. (9.5, 9.6) are solved iteratively by annealing in T. Again, a more or less automatized scheme for doing this exists (cf. sec. 5.5). The high-T_c fixpoint analysis is somewhat more difficult in this case. We refer the reader to ref. [23] for a discussion of this point.

In ref. [23] this ANN approach is compared with other approaches. For reasonably small problem sizes it is feasible to use an exact algorithm, branch and bound, for comparison. For larger problem sizes, one is confined to other approximate methods, such as simulated annealing [10], greedy heuristics and linear programming based on the simplex method [24].

With the **branch and bound (BB)** tree search technique one can reduce the number of computational steps. This method consists in going down a search tree, checking bounds on constraints or utility for entire subtrees, thereby avoiding unnecessary searching. In particular for non-homogeneous problems, this method is accelerated by ordering the c_i's according to magnitude:

$$c_1 > c_2 > .... > c_N . \tag{9.7}$$

For problems where the constraints are "narrow" (b not too large) this method can require substantially lower computation needs. However, it is still based on exploration and it is only feasible for problem sizes less than $M = N \approx 30 - 40$.

Greedy heuristics (GH) is a simple and fast approximate method for a non-homogeneous problem. Proceeding from larger to smaller c_i (cf. eq. (9.7)), collect every item that does not violate any constraint. This method scales like NM.

Simulated annealing (SA) [10] is easily implemented in terms of attempted single-spin flips, subject to the constraints. Suitable annealing rates and other parameters are given below. This method also scales like NM times the number of iterations needed for thermalization and cooling.

Linear programming (LP based on the simplex method [24] is not designed to solve discrete problems like the knapsack one. It does apply, however, to a modified problem with $s_i \in [0,1]$. For the ordered (eq. (9.7)) non-homogeneous knapsack problem this gives solutions with a set of leading 1's and a set of trailing 0's, with a window in between containing real numbers. Augmented by greedy heuristics for the elements in this window, fairly good solutions emerge. The simplex method scales like N^2M^2.

First we compare the NN, SA and LP approaches with the exact BB for an $N = M = 30$ problem. This is done both for non-homogeneous and homogeneous problems. The results are shown in table 9.1. As expected LP and in particular GH benefit from non-homogeneity both quality- and CPU-wise (the latter is also very true for BB), while for homogeneous problems the NN algorithm is the

Algorithm	c_i=rand[0,1]		c_i=rand[0.45,0.55]		c_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

Table 9.1: Comparison of performance and CPU time consumption for the different algorithms on a N=M=30 problem. The CPU consumption refers to a DEC3100 workstation.

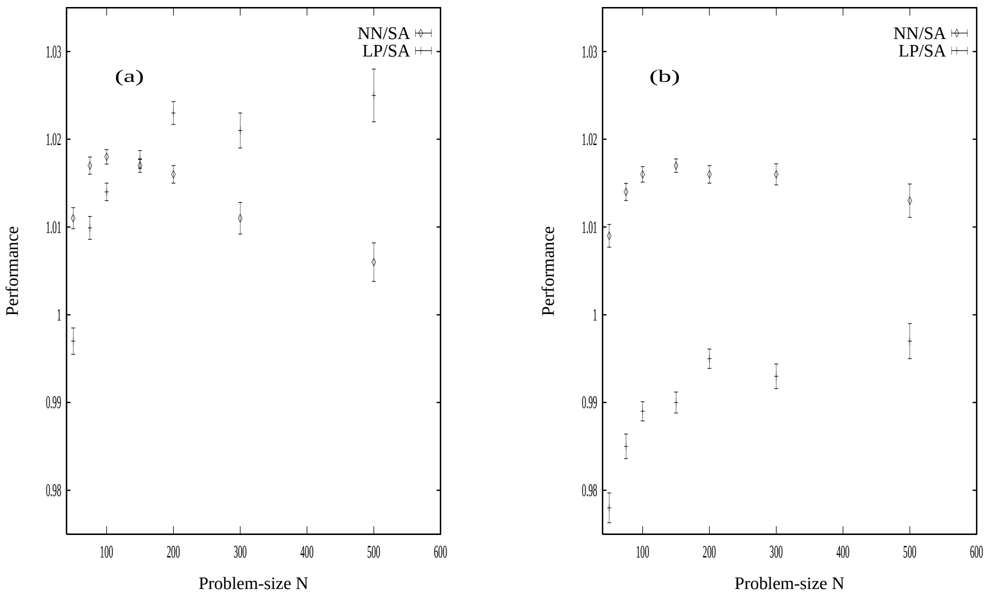


Figure 9.2: Performance of the neural network (NN) and linear programming approaches (LP) normalized to simulated annealing (SA) for problem sizes ranging from 50 to 500 with $M = N$. (a) c_i=rand[0.45,0.55] and (b) c_i=0.5.

winner. For parameter choices, annealing rates etc. we refer the reader to ref. [23]. For larger problem sizes it is not feasible to use the exact BB algorithm. The best we can do is to compare the different approximate approaches, NN, SA and LP. The conclusions from problem sizes ranging from 50 to 500 are the same as above. The real strength in the NN approach is best exploited for more homogeneous problems. In figs. 9.2 and 9.3 we show the performance and CPU consumption for $N \in [50, 500]$ with $M = N$.

In summary MFT approach is very competitive as compared to other approximate methods for the hard homogeneous problems, both with respect to solution quality and time consumption. It also compares very well with exact solutions for problem sizes where these are accessible.

Being able to find good approximate solutions to difficult knapsack problems opens up several

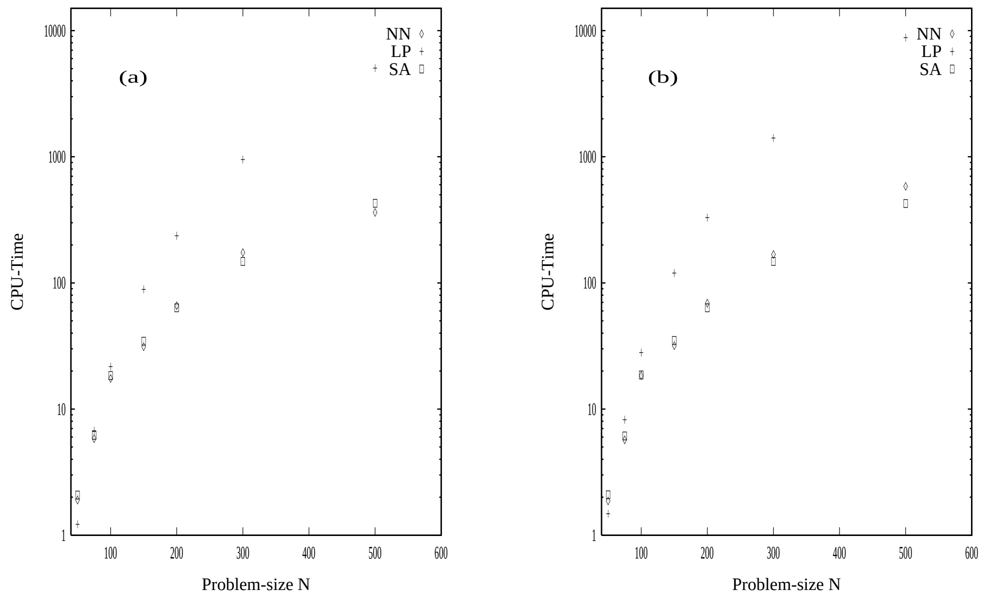


Figure 9.3: CPU consumption of the neural network (NN) and linear programming approaches (LP) normalized to simulated annealing (SA) for problem sizes ranging from 50 to 500 with $M = N$. (a) c_i=rand[0.45,0.55] and (b) c_i=0.5. The numbers refer to DEC3100 workstations.

application areas within the field of resource allocation. Hence the achievements in this section could be very important.

10 Summary

The neural approach is in its nature very different from other approximate methods. It has its roots in statistical mechanics treatment of systems with many degrees of freedom as has simulated annealing in general. With the MFT approximation it feels its way in a "fuzzy" manner towards good solutions. This is in contrast to most other methods, where discrete moves take place in solution space. The basic steps for solving combinatorial optimization problems with an ANN are:

- Map the problem onto a neural network by a suitable coding of the possible solutions and an appropriate choice of energy function. Where applicable, use Potts encoding rather than Ising neuron multiplexing.
- Utilize as much as possible prior knowledge about phase transition properties from analyzing the linearized dynamics. Special care is needed when defining the MFT equations in cases when the penalty terms are non-polynomial, as in the case of inequality constraints.
- Solve the corresponding mean field equations iteratively, under annealing in T.
- When the MFT equations have converged the solutions are checked with respect to "legality"– if they satisfy the basic constraints. If not one either supplements the algorithm with a greedy

heuristics, or re-anneals the system (possibly with modified constraint coefficients).

For low-dimensional geometrical problems like the traveling salesman problem, it is often advantageous to use an alternative procedure, the deformable templates method. This reduces the number of degrees of freedom and hence the computing time.

With respect to quality the ANN method in general gives results comparable to those of other state-of-the-art approximative approaches, like simulated annealing. In full-sized high school scheduling applications comparisons have not been possible since no results exist from other approaches.

The neural language is very natural for encoding many optimization problems like scheduling. The approach has the advantage of being easily used for revision of solutions due to new situations. One simply re-anneals the network with non-revisable units clamped.

As with all other ANN applications the neural optimization approach has the advantage of being fully parallelizable.

References

[1] E.R. Kandel and J.H. Schwartz, *Principles of Neural Science*, 3^{rd} ed., Elsevier Publ. :New York (1991).

[2] J. Hertz, A. Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, Ca. (1991).

[3] D. E. Rumelhart and J. L. McClelland (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Vol. 1)*, MIT Press (1986).

[4] A. Lapedes and R. Farber, "Nonlinear Signal Processing using Neural Networks: Prediction and System Modeling", Los Alamos Report LA-UR 87-2662 (1987).

[5] A. Weigend, B.A. Huberman and D. Rumelhart, "Predicting the Future: A Connectionist Approach", *International Journal of Neural systems* **3**, 193 (1990).

[6] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Science, USA*, **79** 2554 (1982).

[7] J.J. Hopfield and D.W.Tank, "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics* **52**, 141 (1985).

[8] C. Peterson and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems* **1**, 3 (1989).

[9] R. Durbin and D. Willshaw, "An Analog Approach to the Traveling Salesman Problem using an Elastic Net Method". *Nature* **326**, 689 (1987).

[10] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing", *Science* **220**, 671 (1983).

[11] K. Downsland, this volume.

[12] C. Peterson and J.R. Anderson, "Neural Networks and NP-complete Optimization Problems; A Performance Study on the Graph Bisection Problem", *Complex Systems* **2**, 59 (1988).

[13] C. Peterson, "Parallel Distributed Approaches to Combinatorial Optimization Problems – Benchmark Studies on TSP", *Neural Computation* **2**, 261 (1990).

[14] M.Y. Choi and B.A. Huberman, "Digital Dynamics and the Simulation of Magnetic Systems", *Physical Review* **B 28**, 2547 (1983).

[15] F.Y. Wu, "The Potts Model", *Review of Modern Physics*, **54**, 235 (1983).

[16] L. Gislén, B. Söderberg and C. Peterson, "Teachers and Classes with Neural Networks", *International Journal of Neural Systems* **1**, 167 (1989).

[17] L. Gislén, B. Söderberg and C. Peterson, "Scheduling High Schools with Neural Networks", Lund Preprint *LU TP 91-9* (submitted to *Neural Computation*) (1991).

[18] A.L. Yuille, "Generalized Deformable Models, Statistical Physics, and Matching Problems", *Neural Computation.* **2**, 1 (1990).

[19] A. Yuille, K. Honda and C. Peterson, "Particle Tracking by Deformable Templates", *Proceedings of 1991 IEEE INNS International Joint Conference on Neural Networks*, Vol. 1, pp 7-12, Seattle, WA (July 1991)

[20] T. Kohonen, "Self Organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics* **43**, 59 (1982);
T. Kohonen, *Self-organization and Associative Memory*, 3^{rd} ed., Springer-Verlag, Berlin, Heidelberg (1990).

[21] M. Ohlsson, C. Peterson and A. Yuille, "Track Finding with Deformable Templates – The Elastic Arms Approach", Lund Preprint *LU TP 91-27* (to appear in *Computer Physics Communications*) (1991).

[22] M. Gyulassy and H. Harlander, "Elastic Tracking and Neural Network Algorithms for Complex Pattern Recognition", *Computer Physics Communications* **66**, 31 (1991).

[23] M. Ohlsson, C. Peterson and B. Söderberg, "Neural Networks for Optimization Problems with Inequality Constraints – the Knapsack Problem", Lund Preprint *LU TP 92-11* (submitted to *Neural Computation*) (1991).

[24] See e.g. W.P. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vettering, *Numerical Recipes, The Art of Scientific Computing*, Cambridge University Press, Cambridge, (1986).