# Article

# Multiscaling in binary Hopfield-type neural networks: application to the Set Partitioning Problem †


## A. Likas and A. Stafylopatis

*Computer Science Division, Department of Electrical and Computer Engineering, National Technical University of Athens, 157 73 Zographou, Athens, Greece*



† *Part of this work was supported by the Commission of the European Communities under ESPRIT Project 2092 (ANNIE)*



*Abstract: Convergence in large optimisation problems can be accelerated through multiscaling, which consists of defining a smaller coarse-scale version of the problem and alternating between the fine-scale and coarse-scale instances during solution. Multiscaling techniques for neural networks with continuous valued units have been reported in the literature, but these techniques cannot be applied to binary networks. An original multiscale method for solving optimisation problems is presented in this paper, suitable for Hopfield-type neural networks with binary units. The method has been tested in the context of the Set Partitioning Problem by constructing a Boltzmann Machine Optimizer operating at various levels of coarseness. The approach appears very promising, as results indicate that a significant execution speed-up can be achieved while at the same time the quality of the solution is preserved.*


## 1. Introduction

The neural network approach has been shown to deal effectively with hard optimisation problems commonly arising in real world applications, for which no algorithm is known to provide an exact solution in polynomial computation time (Garey & Johnson 1979; Papadimitriou & Steiglitz 1981).

In many cases, neural networks yield near-optimal solutions in polynomial time. More specifically, the Hopfield neural network model (Hopfield 1982; Hopfield & Tank 1985), and closely related models like the Boltzmann Machine (Aarts & Korst 1987; Ackley *et al.* 1985), are widely used in the context of combinatorial optimisation (Dahl 1987; Gutzmann 1987; Herault & Niez 1991; Zissimopoulos *et al.* 1991). The basic idea is the encoding of the objective function and the problem constraints in terms of an appropriate *energy function* or *consensus function* (Aarts & Korst 1989). This encoding must have the property that the local maxima of the consensus function (local minima of the energy function) correspond to acceptable solutions of the original problem. The objective of the network's operation is the minimisation of the energy function or the maximisation of the consensus function. In the following, our presentation will be done in terms of the consensus function of the neural network.

It is well known (Hopfield 1982) that a symmetric Hopfield network operating in sequential mode will always settle into a stable (equilibrium) state corresponding to a local maximum of the consensus function. This happens because, at each step during operation, the consensus of the network either increases or remains the same. In general, there are many states corresponding to local maxima of the consensus function. As a consequence, it is very likely that the network will settle to a local maximum while our objective is to reach the global consensus maximum, i.e. the local maximum with the most positive consensus.

An efficient approach for finding high consensus states is to use the Simulated Annealing methodology (Kirkpatrick *et al.* 1983; Kirkpatrick 1984; van Laarhoven 1987), which helps the network escape from stable states of low consensus and settle into states of better quality (higher consensus). The resulting Boltzmann Machine Optimizer (Aarts & Korst 1989) is the main neural network technique used to solve optimisation problems. The main idea is that we introduce an annealing schedule from an initial high temperature down to a temperature value near zero. Associated with each temperature is a probability that the network can move towards states of lower consensus during opera-

tion. As temperature decreases the probability also decreases, becoming zero when the temperature reaches near zero values. In this case the operation of the network is the same as in the pure Hopfield case.

In this paper we develop a multiscale method for binary Hopfield-type networks. According to this method, a smaller network is constructed by grouping the nodes of the original network. The maximisation of the objective consensus function is carried out by performing iterations in either the original or the coarse-scale network. An application of the multiscale method is presented for the solution of the Set Partitioning Problem (SPP) using a Boltzmann Machine Optimizer. First we present the architecture of this Optimizer.

## 2. A neural network architecture for solving the SPP

A set-theoretic interpretation of the SPP has the following (Garey & Johnson 1979). Given a set $M = \{m_1, m_2, \ldots, m_l\}$, and a family $S = \{S_1, S_2, \ldots, S_n\}$ of sets $S_j \subseteq M$, any subfamily $S' = \{S_{j_1}, S_{j_2}, \ldots, S_{j_k}\}$ such that $\cup_{i=1}^{k} S_{j_i} = M$ and $S_{j_h} \cap S_{j_p} = \emptyset$ for each $h$, $p \in \{1, \ldots, k\}$ with $h \neq p$ is called a Set-Partitioning of $M$.

Moreover, if there is a cost $c_j \geq 0$ associated with each $S_j \in S$ then the SPP takes the following definition: find that set-partitioning $S'$ of the set $M$ which has the minimum cost, where the cost of $S' = \{S_{j_1}, \ldots, S_{j_k}\}$ is $\sum_{i=1}^{k} c_{j_i}$. In terms of a 0-1 linear program this can be formulated as: minimise

$$f(x) = \sum_{j=1}^{n} c_j x_j$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = 1$$

for each $i = 1, \ldots, l$. The value of $x_j$ is 1 (0) depending on $S_j \in S'$, $(S_j \notin S')$ and the value of $a_{ij}$ is 1 (0) depending on $m_i \in S_j$, $(m_i \notin S_j)$.

From the above formulation it is obvious that a valid SPP solution should satisfy two kinds of constraints: mutual exclusion and covering constraints. In Zissimopoulos *et al.* (1991) the solutions that satisfy the first constraint without satisfying the second are referred to as valid partial solutions, while the solutions satisfying both constraints are referred to as feasible solutions.

A neural network architecture suitable for the SPP can be defined as follows. The number of computing elements in the network is equal to the number of subsets included in the family $S$, so that each computing element $i$ represents a subset $S_i$. Also the computing elements are assumed to be binary.

As far as the connections between elements are concerned, the basic idea is to impose an inhibitory connection (with negative value of strength) between units that correspond to non-disjoint subsets and to impose no connection between units that correspond to disjoint subsets. Moreover the weights are chosen to be symmetrical in order to ensure convergence. It has been found (Zissimopoulos *et al.* 1991) that a connection matrix $W = (w_{ij})$ suitable for the SPP has the following form:

$$W_{ij} = \begin{cases} -\{\max\{\theta_i, \theta_j\} + \epsilon\}\, \delta_{ij} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \qquad (1)$$

where $i = 1, \ldots, n$, $j = 1, \ldots, n$ and the thresholds $\theta_i$ $(i = 1, \ldots, n)$ take the value

$$\theta_i = \Theta |S_i| - c_i \qquad (2)$$

$|S_i|$ is the cardinality of the subset represented by the unit $i$, $c_i$ is the cost of the subset $S_i$, $\Theta = \sum_{i=1}^{n} c_i$, $\epsilon$ is a very small positive value, and $\delta_{ij} = 1(0)$ if subsets $S_i$, $S_j$ are not disjoint (otherwise).

The consensus function corresponding to this neural network has the form

$$C(\vec{y}) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j w_{ij} + \sum_{i=1}^{n} \theta_i y_i \qquad (3)$$

where $\vec{y} = (y_1, \ldots, y_n)$ is the state of the network. At each time instant a unit $i$ is selected randomly and the following quantity is calculated:

$$\delta C_i(\vec{y}) = (1 - 2y_i)(\sum_{j=1}^{n} w_{ji} y_j + \theta_i) \qquad (4)$$

The update rule for each node is as follows. If $\delta C_i(\vec{y}) > 0$ then the state of node $i$ changes. In case $\delta C_i(\vec{y}) < 0$ the state changes according to some probability function which in general depends on the quantity $\exp(-\delta C_i(\vec{y})/T)$, where $T$ is the temperature parameter described in the introduction.

The network will finally settle into a state that corresponds to a local maximum of the consensus function (Aarts & Korst 1989; Hopfield 1982). In this local maximum state only units corresponding to disjoint subsets could be 'on'. If a unit is 'on', the corresponding subset must be incorporated in the final solution. Thus, we see that the mutual exclusion constraint is satisfied. Moreover, it has been proved that a local maximum state corresponding to a feasible solution has greater consensus than a local maximum state corresponding to a valid partial solution (Zissimopoulos *et al.* 1991). In addition, the higher the consensus of the local maximum state, the lower the total cost of the final solution. Since the Boltzmann Machine Optimizer generally leads to final states of high consensus, it is expected that a feasible solution of low cost will finally be obtained.

In the next section we present a general multiscale method suitable for Hopfield-type neural networks with binary units and quadratic consensus function. The efficiency of the method will be investigated in the context of the Set Partitioning Problem.

## 3. The multiscale method

### 3.1. Principles of multiscaling

The idea of *multiscaling* has its origin in the field of numerical analysis (Chatelin & Miranker 1982; Miranker 1981). In particular, this is the idea behind the *multigrid methods* (Huckbusch 1978), which are widely used to solve partial differential equations by optimising quadratic objective functions defined on geometric domains.

The basic function is that we can speed up convergence in a large optimisation problem by introducing a smaller approximate version at a coarser scale and alternating the relaxation steps between the fine-scale and coarse-scale instances of the problem.

Mjolsness *et al.* (1991) use the above idea to optimise a general Hopfield objective function of the form

$$E[\vec{v}] = -\sum_{ijk} w_{ijk} v_i v_j v_k - \sum_{ij} w_{ij} v_i v_j - \sum_i h_i v_i - \sum_i \varphi_i(v_i) \qquad (5)$$

where $v_i$ are *continuous* neural variables. They suggest a quite general methodology, according to which the following additional components must be defined:

- a mapping from the original variables $v_i$ to fewer coarse-scale variables $V_\alpha$, called the *restriction* or *aggregation* map $\vec{V} = R[\vec{v}]$;
- a mapping back from the coarse-scale variables $V_\alpha$ to the original variables $v_i$, called the *prolongation* or *disaggregation* map and given by $\vec{v} = P[\vec{V}]$;
- a coarse-scale objective function $\hat{E}[\vec{V}]$, which is intended to approximate the original objective function $E[\vec{v}]$ and is cheaper to evaluate and differentiate;
- a suitable algorithm (strategy) for alternating between the fine- and coarse-scale domains;
- a modification of the basic cycle in order to be able to operate on several scales concurrently.

The coarse-scale objective function $\hat{E}$ is chosen to be the restriction of $E$ to a subspace parameterised by $\vec{V}$. This subspace is given by the prolongation map $P$:

$$\hat{E}[\vec{V}] = E[P[\vec{V}]] \qquad (6)$$

Thus the choice of the prolongation map is of particular importance. A simple special case is when the objective function is quadratic and the maps $P$ and $R$ are matrix–vector multiplications, with $R = P^\top$. This means that the coarse-scale variables are obtained as a linear combination of the original fine-scale variables. In that case, the coarse-scale objective function $\hat{E}$ has the following form:

$$\hat{E} = \sum_{\alpha\beta} (RWP)_{\alpha\beta} V_\alpha V_\beta \qquad (7)$$

Hence, there is a simple form for the coarse objective function which is easy to evaluate.

The above described procedure is not easy to apply in the case where both vectors $\vec{v}$ and $\vec{V}$ should have binary elements. This happens because the prolongation map and the restriction map could not have the simple form ($\vec{v} = P\vec{V}$, $\vec{V} = R\vec{v}$) defined above, since the linear combination of binary variables does not yield a binary variable. Indeed, the fact that both vectors $\vec{v}$ and $\vec{V}$ contain binary valued elements makes the dependence of the maps on the current state of the fine-scale network necessary. The multiscale method discussed next uses a transformation (or map) of special nature and is applied to neural network optimisers with binary units.

### 3.2. A multiscale approach for binary networks

Consider a Hopfield-type neural network with $n$ binary units ($n$-net). The connection strengths of the network are $w_{ij}$, with $w_{ij} = w_{ji}$ and $w_{ii} = 0$ ($i = 1, \ldots, n$, $j = 1, \ldots, n$), and the threshold values are $\theta_i$ ($i = 1, \ldots, n$). Suppose now that we wish to switch to a smaller network with $p$ nodes that we shall call the $p$-net. The $p$-net is also assumed to be a Hopfield-type neural network with binary units. The mapping from the $n$-net to the $p$-net is based on partitioning the nodes of the $n$-net into $p$ disjoint groups, such that each group corresponds to a node of the $p$-net.

We shall use the notation $s(k)$ to specify the number of nodes corresponding to group $k$ ($k = 1, \ldots, p$). The correspondence between groups of nodes in the $n$-net and nodes in the $p$-net can then be expressed by means of the mapping function $v(k, k')$ ($k = 1, \ldots, p$, $k' = 1, \ldots, s(k)$) as follows. The equation $v(k, k') = i$ ($i = 1, \ldots, n$) expresses the fact that node $i$ of the original network corresponds to the $k'$-th element of group $k$. Thus, the mapping function represents each node $i$ of the $n$-net in terms of the index $k$ of the group to which it belongs and the respective index $k'$ within that group.

If the state of the $n$-net is $\vec{y^\star} = (y_1^\star, \ldots, y_n^\star)$, then the corresponding consensus function $C(\vec{y^\star})$ can be written:

$$C(\vec{y^\star}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i^\star y_j^\star w_{ij} + \sum_{i=1}^{n} \theta_i y_i^\star = \frac{1}{2}\sum_{k=1}^{p}\sum_{k'=1}^{s(k)}\sum_{l=1}^{p}\sum_{l'=1}^{s(l)} y^\star_{v(k,k')} y^\star_{v(l,l')} w_{v(k,k')v(l,l')}$$

$$+ \sum_{k=1}^{p} \sum_{k'=1}^{s(k)} \theta_{v(k,k')} y^*_{v(k,k')} \qquad (8)$$

Suppose that while being in state $\vec{y^*}$ we decide to switch to the small network. Suppose also that we have a way of appropriately constructing the $p$-net. At some later time, after operation of the small network, we decide to switch back to the $n$-net. Then, if the final state of the $p$-net is $\vec{\xi} = (\xi_1, \ldots, \xi_p)$, the transformation for obtaining the new state $\vec{y} = (y_1, \ldots, y_n)$ of the original network has the following form:

$$y_{v(k,k')} = y^*_{v(k,k')} + u_{v(k,k')} \xi_k \qquad (9)$$

The quantities $u_{v(k,k')}$ (or $u_i$) are defined as

$$u_{v(k,k')} = 1 - 2y^*_{v(k,k')} \qquad (10)$$

$k = 1, \ldots, p$, $k' = 1, \ldots, s(k)$. This means that if $y^*_i = 0$ then $u_i = 1$, and if $y^*_i = 1$ then $u_i = -1$. The advantage of the above transformation (9) is that it can ensure that the new state vector $\vec{y}$ of the $n$-net contains binary components. There is an interesting physical interpretation of the transformation: for each group $k$, $k = 1, \ldots, p$, if the final state of unit $k$ in the $p$-net is 1, then all nodes belonging to group $k$ change their state (with respect to the state vector $\vec{y^*}$). Inversely, if the resulting value $\xi_k$ is 0, the nodes belonging to group $k$ retain the state they had before switching to the $p$-net.

The connection weights $x_{kl}$, ($k = 1, \ldots, p$, $l = 1, \ldots, p$) of the $p$-net can be computed by means of the following argument. The consensus of the new state $\vec{y}$ of the $n$-net is:

$$C(\vec{y}) = \frac{1}{2} \sum_{k=1}^{p} \sum_{k'=1}^{s(k)} \sum_{l=1}^{p} \sum_{l'=1}^{s(l)} y_{v(k,k')} y_{v(l,l')} w_{v(k,k')v(l,l')} + \sum_{k=1}^{p} \sum_{k'=1}^{s(k)} \theta_{v(k,k')} y_{v(k,k')} \qquad (11)$$

Using the transformation equation (9) the above equation can be rewritten:

$$C(\vec{y}) = \frac{1}{2} \sum_{k=1}^{p} \sum_{k'=1}^{s(k)} \sum_{l=1}^{p} \sum_{l'=1}^{s(l)} \left[ \begin{array}{l} (y^*_{v(k,k')} + u_{v(k,k')} \xi_k) \\ \times (y^*_{v(l,l')} + u_{v(l,l')} \xi_l) \\ \times w_{v(k,k')v(l,l')} \end{array} \right] + \sum_{k=1}^{p} \sum_{k'=1}^{s(k)} \left[ \begin{array}{l} \theta_{v(k,k')} \\ \times (y^*_{v(k,k')} + u_{v(k,k')} \xi_k) \end{array} \right] \qquad (12)$$

After performing some algebra, equation (12) takes the form

$$C(\vec{y}) = C(\vec{y^*}) + \hat{C}(\vec{\xi}) \qquad (13)$$

where $C(\vec{y^*})$ is the consensus function given by (8) and the quantity $\hat{C}(\vec{\xi})$ can be written as:

$$\hat{C}(\vec{\xi}) = \frac{1}{2} \sum_{k=1}^{p} \sum_{l=1}^{p} \xi_k \xi_l \left[ \sum_{k'=1}^{s(k)} \sum_{l'=1}^{s(l)} \left( \begin{array}{l} u_{v(k,k')} u_{v(l,l')} \\ \times w_{v(k,k')v(l,l')} \end{array} \right) \right] + \sum_{k=1}^{p} \xi_k \left[ \begin{array}{l} \sum_{k'=1}^{s(k)} \theta_{v(k,k')} u_{v(k,k')} \\ + \sum_{k'=1}^{s(k)} \sum_{l=1}^{p} \sum_{l'=1}^{s(l)} \left( \begin{array}{l} u_{v(k,k')} y^*_{v(l,l')} \\ \times w_{v(k,k')v(l,l')} \end{array} \right) \end{array} \right] \qquad (14)$$

We can easily observe that the above equation corresponds to the consensus of a network of size $p$ whose state is given by the vector $\vec{\xi} = (\xi_1, \ldots, \xi_p)$ and whose connection weights $x_{kl}$, $k = 1, \ldots, p$, $l = 1, \ldots, p$, take the form

$$x_{kl} = \sum_{k'=1}^{s(k)} \sum_{l'=1}^{s(l)} u_{v(k,k')} u_{v(l,l')} w_{v(k,k')v(l,l')} \qquad (15)$$

Note that to be consistent with the consensus definition the quantities $x_{kk}$, $k = 1, \ldots, p$, should be equal to zero. We can achieve this by making a slight change to equation (14). We add the terms

$$x_{kk} = \sum_{k'=1}^{s(k)} \sum_{k''=1}^{s(k)} u_{v(k,k')} u_{v(k,k'')} w_{v(k,k')v(k,k'')}$$

$k = 1, \ldots, p$, to the threshold values $\omega_k$ and set $x_{kk} = 0$, $k = 1, \ldots, p$. Thus, taking advantage of the fact that the $\xi_k$ values are binary and hence $\xi_k^2 = \xi_k$, for the consensus of the small network we can write

$$\hat{C}(\vec{\xi}) = \frac{1}{2} \sum_{k=1}^{p} \sum_{l=1}^{p} x_{kl} \xi_k \xi_l + \sum_{k=1}^{p} \omega_k \xi_k \qquad (16)$$

where

$$x_{kl} = \begin{cases} \sum_{k'=1}^{s(k)} \sum_{l'=1}^{s(l)} \left( \begin{array}{l} u_{v(k,k')} u_{v(l,l')} \\ \times w_{v(k,k')v(l,l')} \end{array} \right) & \text{if } k \neq l \\ 0 & \text{if } k = l \end{cases} \qquad (17)$$

and the threshold $\omega_k$ of each node $k$ ($k = 1, \ldots, p$) of the small network is

$$\omega_k = \sum_{k'=1}^{s(k)} \theta_{v(k,k')} u_{v(k,k')} + \sum_{k'=1}^{s(k)} \sum_{l=1}^{p} \sum_{l'=1}^{s(l)} u_{v(k,k')} y^*_{v(l,l')} w_{v(k,k')v(l,l')}$$

$$+ \frac{1}{2} \sum_{k'=1}^{s(k)} \sum_{k''=1}^{s(k)} u_{v(k,k')} u_{v(k,k'')} w_{v(k,k')v(k,k'')} \tag{18}$$

Since the change in the network's consensus from state $\vec{y^\star}$ due to update of node $v(k,k')$ is

$$\delta C_{v(k,k')}(\vec{y^\star}) = u_{v(k,k')} \left[ \sum_{l=1}^{p} \sum_{l'=1}^{s(l)} \left( \begin{array}{c} y^\star_{v(l,l')} \\ \times\, w_{v(k,k')v(l,l')} \end{array} \right) + \theta_{v(k,k')} \right] \tag{19}$$

the threshold value $\omega_k$ can also be expressed as

$$\omega_k = \sum_{k'=1}^{s(k)} \delta C_{v(k,k')}(\vec{y^\star}) + \frac{1}{2} \sum_{k'=1}^{s(k)} \sum_{k''=1}^{s(k)} \left( \begin{array}{c} u_{v(k,k')} u_{v(k,k'')} \\ \times\, w_{v(k,k')v(k,k'')} \end{array} \right) \tag{20}$$

It is important to note that, since the original network has symmetrical weights (i.e. $w_{ij} = w_{ji}$ for all $i = 1,\ldots,n$, $j = 1,\ldots,n$), the $p$-net also has symmetrical weights (i.e. $x_{kl} = x_{lk}$ for all $k = 1,\ldots,p$, $l = 1,\ldots,p$), as we can easily observe from equation (17). This symmetric behaviour ensures convergence of the $p$-net when used in Hopfield-type operations. Another important observation concerns the computation of the values $x_{kl}$, using equation (17). The importance lies in the fact that in order to compute the weight $x_{kl}$ only the weights $w_{ij}$ connecting the nodes belonging to groups $k$ and $l$ are involved. This fact implies on the one hand a reduced complexity and on the other hand that there is an inherent parallelism in the transformation task, i.e. the weights of the small network can be computed in parallel and each computation requires a different portion of the connection matrix $W$.

As a conclusion we can state that, once in a state $\vec{y^\star}$ of the original network, we can create a small $p$-net as described above. We can then perform iterations on this network until it reaches a large positive consensus value. Using equation (9) we can return to the original network whose consensus is now given by equation (13), i.e. it is increased by an amount equal to the value of the small network's consensus. It should be noted that, in case the consensus of the $p$-net does not manage to reach a positive value, the transformation may not be accepted (i.e. we can continue to work with the original network without affecting its state, which remains $\vec{y^\star}$). An alternative is to accept the transformation according to some probability function. The main drawback of this method is that the computation of the weights $x_{kl}$ and the values $\omega_k$ depends strongly on the state $\vec{y^\star}$ of the $n$-net at the time of transformation. Thus, the computation must be repeated each time we wish to switch to a $p$-net. This imposes a considerable overhead that may restrict the number of transitions between the networks. It should be stressed, however, that the cost of switching back to the $n$-net is negligible.

Another issue that must be stated is that it is not necessary for *all* nodes of the fine-scale network to participate in a group, i.e. the grouping procedure may ignore some nodes and not incorporate them in any group. In fact we may consider that all ignored nodes constitute a group and that the node of the coarse-scale network which corresponds to that group is set initially to 0 and is never selected for update during the operation of the $p$-net; the state of the corresponding fine-scale nodes thus remains unchanged. Since the state of the node is 0, it has no effect on the input of the other $p$-net nodes and its connection weights need not be computed.

As a final remark it must be noted that there is no hint of the initial state of the $p$-net. The above algorithm does not provide any information concerning the calculation of a suitable initial value of the vector $\vec{\xi}$. Thus, the initial state could be assigned at random. In the next section we shall provide an effective algorithm for grouping the nodes of the $n$-net and constructing the $p$-net.

### *3.3. An efficient transformation scheme*

As stated previously, once in a state $\vec{y^\star}$ of the original fine-scale network, we can construct a coarse-scale network by computing the connection weights $x_{kl}$ and the threshold values $\omega_k$ using equations (17) and (18) (or (20)). It has also been stated that due to the dependence of these equations on the state $\vec{y^\star}$, the computations must be repeated every time we wish to switch to a coarse-scale network.

It can be verified that the computational cost of the $p$-net construction is equivalent to performing $1.5n$ iterations in the original fine-scale network. (This figure corresponds to the case where every node of the $n$-net participates in one of the groups.) This cost is not excessively high with respect to total execution time. Moreover, it is possible to take advantage of the computations performed during the creation of the $p$-net so that an improved performance can be ensured for the small network without any additional cost. The idea is to determine a reasonable grouping of the nodes of the $n$-net and at the same time to create the corresponding $p$-net.

Suppose that starting from the state $\vec{y^\star}$ of the fine-scale network, we perform $n$ iterations in the $n$-net, concerning all $n$ nodes of the network. This means that all $n$ nodes are sequentially considered for update according to an arbitrary order. (To facilitate the presentation we assume that nodes are selected following an increas-

ing index order.) In reality, we simply simulate these updates as follows.

We define two sets $P$ and $N$. The set $P$ contains the indexes of the nodes for which the trial was successful (i.e. the state of these nodes should change if updates were performed), while the set $N$ contains the indexes of the nodes for which the trial was unsuccessful. Initially the two sets are empty. After each iteration, if the resulting $\delta C_i$ is positive then the corresponding node $i$ is added to the set $P$, otherwise it is added to the set $N$. The state of the node is not affected, i.e. the network remains in state $\vec{y^\star}$. The consensus difference computed during the $i$-th iteration (concerning node $i$) can be expressed as:

$$\delta C_i = \delta C_i(\vec{y^\star}) + u_i\Big(\sum_{j<i,\, j\in P} u_j w_{ij}\Big) \qquad (21)$$

From equations (17) and (20) we can observe that during the $p$-net's construction the quantities $\delta C_i(\vec{y^\star})$ and $u_i u_j w_{ij}$ are computed. Thus, the main idea is to calculate the $x_{kl}$ and $\omega_k$ while constructing the sets $P$ and $N$ and to create groups dynamically with the property that *all* their nodes belong either to $P$ or to $N$. The implementation is easy once the group size has been specified in advance.

Consider now a state $\vec{\xi^\star}$ of the $p$-net defined as

$$\xi_k^\star = \begin{cases} 1 & \text{if the nodes of group } k \text{ belong to } P \\ 0 & \text{if the nodes of group } k \text{ belong to } N \end{cases} \qquad (22)$$

The state $\vec{\xi^\star}$ of the $p$-net corresponds to a state $\vec{y}$ of the $n$-net, which is derived from state $\vec{y^\star}$ by changing the state of the nodes belonging to the set $P$. It is clear that, if $P$ is not empty, the consensus $\hat{C}(\vec{\xi^\star}) = C(\vec{y}) - C(\vec{y^\star})$ has a positive value. Hence, the above construction scheme ensures the existence of at least one state of the $p$-net that increases the consensus of the $n$-net. Moreover, a considerable increase can be expected because moving to state $\vec{\xi^\star}$ is equivalent to performing $n$ iterations in the $n$-net.

A first idea could be to choose $\vec{\xi^\star}$ as the initial state for the operation of the $p$-net. However, since the consensus $\hat{C}(\vec{\xi^\star})$ is expected to have a large positive value, a steep ascent could take place and hence the search area of the network in the consensus space would be restricted. So, it seems more reasonable to start the operation of the $p$-net from a state of low consensus and to let it evolve to a state whose consensus will probably be much higher than that of $\vec{\xi^\star}$.

This construction scheme ensures that the $p$-net obtained can provide a considerable improvement in the network's consensus. Moreover, an efficient exploitation of the computations is achieved since the construction of the $p$-net is now effectively used to simulate $n$ iterations in the $n$-net. Since these iterations are not performed in reality, the benefit thereof lies in the quality of the constructed $p$-net. Hence, we could claim that the above procedure essentially reduces the cost of the construction algorithm from $1.5n$ to $0.5n$ iterations.

## 4. Implementation and results

We have already shown that every state of the $p$-net with consensus greater than zero corresponds to an increase in the consensus of the original network. Actually, if the state of the $p$-net is of positive consensus then every iteration in this network that increases its consensus also increases the consensus of the original network. This notion of group updates is of particular importance, since it exhibits several useful characteristics. Besides reducing the computational cost of the update it can also be seen as introducing a kind of 'thermal noise' in the operation of the original network. Grouping nodes together and 'simulating' the interaction among the groups can help us avoid local minima. As a matter of fact, the problem of local minima arising in Hopfield-type neural networks is a direct consequence of the locality of control existing in those networks. Each node operates asynchronously and independently and the network is unable to avoid the local minima 'traps'. In the multiscale approach described here, a small network is constructed by grouping nodes together. Each node of the $p$-net also operates asynchronously and independently but each iteration in the coarse-scale network simulates the interaction among a group of nodes in the fine-scale network. Thus, using the multiscale technique, on the one hand locality of control is maintained while on the other hand multiple interaction is achieved.

It is apparent that the multiscale method is of great potential, but a lot of appropriate decisions have to be made in order to exploit that potential. We use the term *switching strategy* to denote the control scheme which determines the sequence of alternations between the original network and the smaller size network. Several strategies could be developed. For example, instead of alternating between the original and the small network, we could start from the original network and create a smaller network. Then we could apply the multiscale technique to this smaller network and proceed by repeatedly applying multiscaling to smaller networks. Thus we could construct a 'nested' multiscaling approach consisting of a scale hierarchy (Mjolsness *et al.* 1991), such that multiscale operation at level $l$ of the scale hierarchy consists of *ordinary operation* at level $l$ alternated with *multiscale operation* at level $l+1$, which is supposed to be coarser than level $l$.

After specifying the sequence of switches between the networks, as well as the sizes of the constructed coarse-scale networks, we have to determine the *grouping strategy*. This term characterises the partitioning of nodes of the original network into groups in order to create the small network, i.e. the way in which the function $v(k,l)$ is determined. For example one could group nodes that

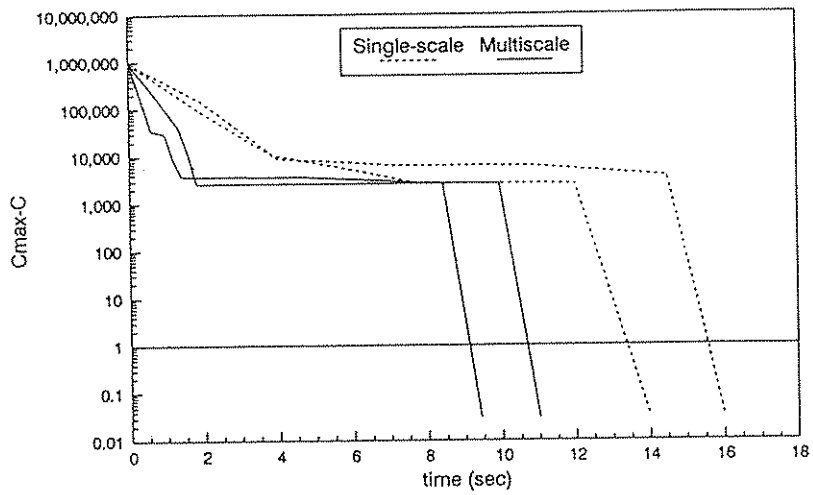


Figure 1. *Multiscale versus single-scale:* $n = 180$, $p = 60$.

are strongly connected together, as suggested in Mjolsness *et al.* (1991), but this option requires a preprocessing stage to be included in the multiscale procedure, which may impose unaffordable overhead. In the case where this preprocessing stage is not acceptable, one could choose the groups of nodes to be almost equal (the size of some groups is equal to $n/p$, while the size of the remaining groups is equal to $n/p + 1$) and the nodes of the original network participating in each group to be selected at random. Also, another grouping scheme is suggested by the algorithm presented in the previous section.

Another important point in developing the overall strategy is to specify the conditions for switching. This must be considered in conjunction with the specification of the annealing schedule, and there are several choices. For example, each network could have its own annealing schedule, i.e. we could start the original network from an initial temperature and perform some iterations (temperature decrements) in that network, and then switch to a small network where we perform the annealing procedure from a suitable initial temperature until we reach a low temperature. We could then return to the original network where we would continue the annealing schedule from the point where it was previously interrupted. Another possible strategy considers a unified annealing schedule for the entire system. This means that we keep a global temperature variable which is successively decremented, either in the $n$-net or in the $p$-net. We have elected to experiment with the second approach, which seems more elegant, and to keep one annealing schedule independent of the network alternations.

The basic element that should be kept in mind when designing the switching strategy is the overall execution time. Operation on the coarse-scale network implies the following advantages:

- Since the size of the $p$-net is considerably smaller compared to the size of the original network, the computation time required to perform an iteration is considerably reduced.
- The state-space of the $p$-net is logarithmically reduced with respect to the state-space of the $n$-net and the number of iterations required to perform a search of the state-space of the $p$-net is much lower. Thus, the number of iterations per temperature that should be performed on the $p$-net is significantly lower.
- As already stated, the notion of group update that is incorporated in the multiscale approach assists the original network in escaping from consensus traps and can lead to higher consensus maxima.

The previous considerations indicate that the execution time associated with performing iterations on the $p$-net is a small fraction of the overall execution time. On the other hand, there is an overhead associated with the cost of creating the $p$-net and computing its weights. Since this computation depends on the current state of the original network, it must be performed every time we wish to switch to the coarse-scale network. This is a fact that imposes a restriction on the number of switches between the networks. Actually, as stated in Mjolsness *et al.* (1991) and also indicated from our experiments, there is a limit in the number of switches, over which no significant benefit is achieved. It should be recalled that the computational cost of the switch from the coarse-scale to the fine-scale network is negligible. Therefore, in order to achieve acceleration it is important that the benefit from increasing the consensus using the $p$-net exceeds the transformation overhead.

Figures 1, 2 and 3 present experimental results of the

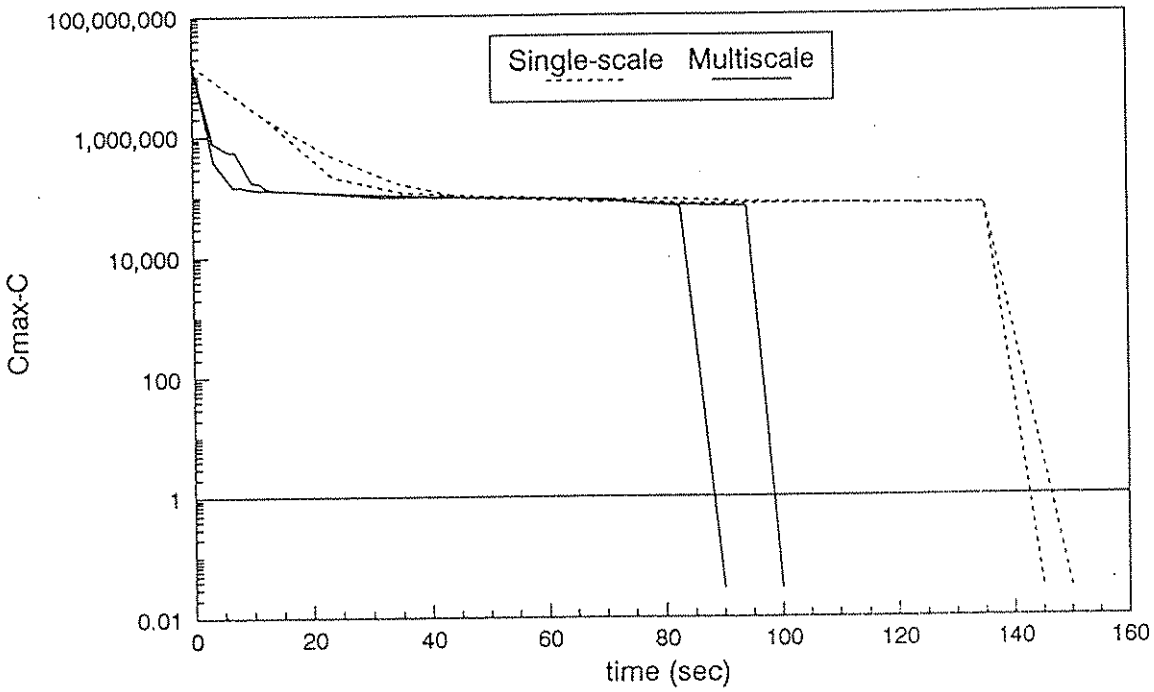


**Figure 2.** *Multiscale versus single-scale:* $n = 450$, $p = 150$.

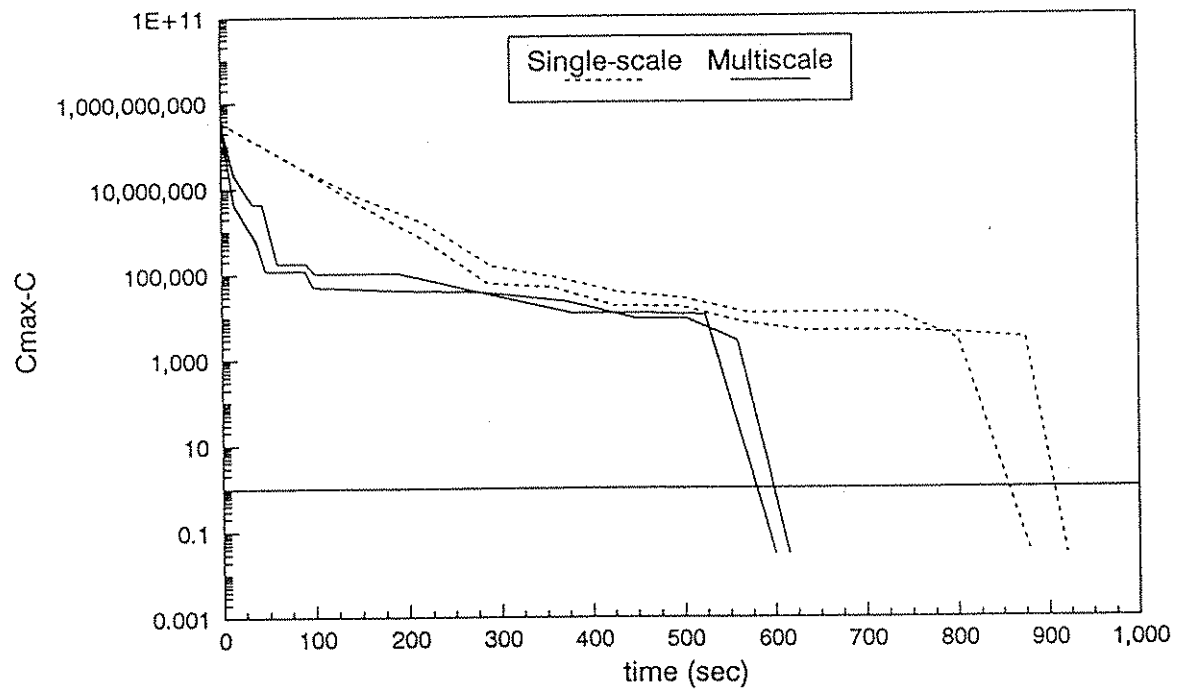


**Figure 3.** *Multiscale versus single-scale:* $n = 1300$, $p = 130$.

Table 1. *Consensus values:* $n = 180$, $p = 60$.

| Initial | Multiscale | | Single-scale | Single-scale |
|---|---|---|---|---|
| | Switch | Final | Final (High $T_0$) | Final (Low $T_0$) |
| − 4 122 017 | 2867 | 11 477 | 11 911 | 8321 |
| − 4 237 126 | 2915 | 12 193 | 10 759 | 8324 |

Table 2. *Consensus values:* $n = 450$, $p = 150$.

| Initial | Multiscale | | Single-scale | Single-scale |
|---|---|---|---|---|
| | Switch | Final | Final (High $T_0$) | Final (Low $T_0$) |
| − 93 734 912 | 28 841 | 94 699 | 89 987 | 51 286 |
| − 88 616 672 | 35 696 | 101 478 | 101 497 | 64 450 |

Table 3. *Consensus values:* $n = 1300$, $p = 130$.

| Initial | Multiscale | | Single-scale | Single-scale |
|---|---|---|---|---|
| | Switch | Final | Final (High $T_0$) | Final (Low $T_0$) |
| − 1 258 327 112 | 610 054 | 811 471 | 811 410 | 657 194 |
| − 1 271 694 848 | 687 760 | 908 400 | 899 212 | 690 910 |

use of the multiscaling technique for solving the Set Partitioning Problem, compared against the single network case. The curves indicate the evolution of the quantity $C_{\max} - C$ versus elapsed time, where $C$ is the current value of the network consensus and $C_{\max}$ corresponds to the final maximum consensus state that is attained in each run. The method has been tested on problems of several network sizes on a Sun 3/60 workstation with 16 Mb of main memory. The results displayed concern three sizes—$n = 180$, $n = 450$ and $n = 1300$—and two runs are shown for each case. The group size was selected to be 3 for the first two cases and 10 for the last case, resulting in coarse-scale networks of size $p = 60$, $p = 150$ and $p = 130$ respectively. As already stated, the strategy that we have selected considers a unified annealing schedule, i.e. there is a global temperature which is updated in turn by the networks. In these experiments we consider only one transition to the $p$-net and this transition takes place at the beginning of each experiment. Figures 1, 2 and 3 do not show the construction phase. An initial temperature value $T_0 = 2.0$ was used in all the experiments, whereas switching from the coarse scale to the fine-scale took place at a temperature equal to 1.0.

The choice to perform only one switch to the $p$-net is due to two reasons. The first concerns the minimisation of the overhead associated with the construction of the $p$-net. The second reason is that, as the figures indicate, a single run in the coarse-scale network significantly increases the consensus in a very short time interval. Thus, starting from an arbitrary initial state of the $n$-net, the switch to the coarse-scale network causes a fast transition to a near maximum state. Afterwards, we return to the fine-scale network where a more localised search is performed to find the best possible consensus maximum.

Tables 1–3 display characteristic values of the consensus function related to the experiments corresponding to the figures. The first column contains the consensus of the initial state, which is the same for the runs represented in each row. For each run of the multiscale method both the consensus at the moment of switch and the final consensus value ($C_{\max}$) are reported. It is apparent from the table values that the largest part of the consensus ascending process is carried out by the coarse-scale network. The last two columns of the tables display the final values ($C_{\max}$) of the consensus function when a single-scale network is used starting from two different initial temperature values $T_0$. The high temperature is the one used as initial temperature in the multiscale annealing schedule, while the low temperature corresponds to the point of switch from the coarse-scale to the fine-scale network. Note that the runs corresponding to the last column are not represented in the figures.

It must be emphasised that whether we perform only one alternation between the $n$-net and the $p$-net depends on the specific problem we are dealing with and perhaps on the size of the problem. For sizes of tens or hundreds of thousands of nodes we do not expect that a single transformation could be adequate.

Experiments indicate that both the single-scale and the multiscale methods converge to a solution of acceptable quality (near optimal). As indicated by the tables, the application of multiscaling does not result in any loss of quality in the solution; on the contrary, in many cases it provides a slightly higher consensus value. But in terms of the computational cost and the elapsed time for convergence the multiscale algorithm is strongly superior. A comparison of the two methods can be obtained in terms of the $p$ speed-up achieved through multiscaling until a state close to convergence is attained. In general the improvement lies in the range between 4 and 8, without taking into account the cost of the $p$-net construction which is a constant term and represents a small percentage of the execution cost.

The values of the last column in each table are intended to illustrate the impact of the coarse-scale phase on the quality of the solution. As a matter of fact, using a single-scale network starting from a lower temperature provides faster execution but yields solutions of clearly inferior quality, as indicated by the tables. However, it should be noted that the above considerations are closely related to the role of initial temperature, which is a critical issue for the performance of Boltzmann Machines in general.

Further experiments are needed to investigate the capabilities of the method. Since there is a significant number of parameters that have to be adjusted (determining the size $p$, the switching strategy, the grouping strategy, etc.), the multiscale method must be extensively studied in the context of several problems in order for its full potential to be explored.

## 5. Conclusions

We have developed an original multiscale approach suitable for Hopfield-type neural networks with binary units. The method has been tested in the context of the Set Partitioning Problem, where a Boltzmann Machine Optimizer operating at various levels of coarseness has been constructed. Results indicate that significant improvements can be achieved and the method appears very promising.

An important aspect of multiscaling concerns the specification of appropriate grouping strategies. Hence, further research should be directed towards specifying general or problem-dependent criteria for determining suitable groups and techniques should be developed for the efficient partitioning of nodes. Also, attempts should be made to assign a meaningful physical interpretation to the consensus function of the coarse-scale network, as well as to the states where it finally settles.

Finally, it is interesting to examine the multiscale method in the context of Hopfield-type neural networks with bipolar computing elements. We have recently obtained results indicating that the cost of the construction of the coarse-scale network can be significantly reduced if bipolar units are used, but we have not yet experimented extensively with this case.

## References


AARTS, E. AND J. KORST (1987) *Boltzmann Machines and their Applications*, Lecture Notes in Computer Science 258, Springer Verlag, pp. 34–50.

AARTS, E. AND J. KORST (1989) *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley & Sons, Chichester.

ACKLEY, D., G. HINTON AND T. SEJNOWSKI (1985) A learning algorithm for Boltzmann Machines, *Cognitive Science*, **9**, 147–169.

CHATELIN, F. AND W. L. MIRANKER (1982) Acceleration by aggregation of successive approximation methods, *Linear Algebra and its Applications*, **43**, 17–47.

DAHL, E. D. (1987) Neural network algorithm for an NP-complete problem: Map and graph coloring, *Proc. International Conference on Neural Networks*, San Diego, Vol. 3, pp. 113–120.

GAREY, M. AND D. JOHNSON (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., San Francisco.

GUTZMANN, K. M. (1987) Combinatorial optimization using a continuous state Boltzmann Machine, *Proc. International Conference on Neural Networks*, San Diego, Vol. 3, pp. 745–752.

HERAULT, L. AND J. NIEZ (1991) Neural networks and combinatorial optimization: A study of NP-complete graph problems, in E. Gelenbe (Ed.), *Neural Networks: Advances and Applications*, North-Holland, Amsterdam.

HOPFIELD, J. (1982) Neural networks and physical systems with emergent collective computational abilities, *Proc. National Academy of Sciences, USA*, Vol. 79, pp. 2554–2558.

HOPFIELD, J. J. AND D. W. TANK (1985) Neural computation of decisions in optimization problems, *Biological Cybernetics*, **52**, 141–152.

HUCKBUSCH, W. (1978) On the multigrid method applied to difference equations, *Computing*, **20**, 291–306.

KIRKPATRICK, S., C. D. GELLAT JR. AND M. P. VECCHI (1983) Optimization by simulated annealing, *Science*, **220**, 671–689.

KIRKPATRICK, S. (1984) Optimization by simulated annealing: Quantitative studies, *J. Stat. Phys.*, **34**, 975–986.

VAN LAARHOVEN, P. AND E. AARTS (1987) *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht.

MIRANKER, W. L. (1981) *Numerical Methods for Stiff Equations*, Reidel, Dordrecht.

MJOLSNESS, E., C. GARRET AND W. L. MIRANKER (1991) Multiscale optimization in neural nets, *IEEE Trans. on Neural Networks*, **2**(2), 263–274.

PAPADIMITRIOU, C. H. AND K. STEIGLITZ (1981) *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall.

ZISSIMOPOULOS, V., V. PASCHOS AND F. PEKERGIN (1991) On the approximation of NP-complete problems by using Boltzmann Machine method: The cases of some covering and packing problems, *IEEE Trans. on Computers*, **40**(12), 1413–1419.

# The authors

## Aristidis Likas

Aristidis Likas was born in Athens, Greece in 1968. He received a Diploma in Electrical and Electronic Engineering in 1990 from the National Technical University of Athens, where he is currently a PhD student in computer science. His research interests include parallel processing and neural networks. He is a student member of the IEEE Computer Society and the Association for Computing Machinery.

## Andreas Stafylopatis

Andreas Stafylopatis was born in Athens, Greece in 1956. He received a Diploma in Electrical and Electronic Engineering in 1979 from the National Technical University of Athens and the Doctor Ingenieur degree in Computer Science in 1982 from the University of Paris-Sud, Orsay, France. Since 1984 he has been with the Division of Computer Science at the National Technical University of Athens, where he is currently an Assistant Professor. His research interests include performance evaluation, information systems, parallel processing and neural networks. He is a member of the IEEE Computer Society, the Association for Computing Machinery, the European Neural Network Society and the International Neural Network Society.