Multiscale Optimization in Neural Nets

Eric Mjolsness, Charles Garrett, and Willard L. Miranker

Research Report YALEU/DCS/RR-797
June 1990

Multiscale Optimization in Neural Nets

Eric Mjolsness and Charles Garrett
Department of Computer Science, Yale University
P.O. Box 2158 Yale Station, New Haven CT 06520-2158

Willard L. Miranker
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
and Department of Computer Science, Yale University

June 6, 1990

Abstract

One way to speed up convergence in a large optimization problem is to introduce a smaller, approximate version of the problem at a coarser scale and to alternate between relaxation steps for the fine-scale and the coarse-scale problems. Done recursively, this is the idea behind the Multigrid methods which are widely used in the solution of partial differential equations, usually by optimizing quadratic objective functions defined on geometric domains.

We exhibit a similar optimization method for neural networks governed by quite general objective functions. At the coarse scale there is a smaller, approximating neural net. Like the original net it is nonlinear and has a nonquadratic objective function, so our coarse-scale net is a more accurate approximation than a quadratic objective would be. The transitions and information flow from fine to coarse scale and back do not disrupt the optimization. The problem need not involve any geometric domain; all that is required is a partition of the original fine-scale variables. Given this partition the rest of the multiscale optimization method requires no problem-specific design effort on the part of the user, because the mapping between coarse and fine scales is determined. Thus the method can be applied easily to many problems and networks. We show positive experimental results including cost comparisons.

1 Introduction

A rather general neural net objective function for continuous neural variables v_i is (Hopfield, 1984)

$$E[\vec{v}] = -\sum_{ijk} T_{ijk} v_i v_j v_k - \sum_{ij} T_{ij} v_i v_j - \sum_i h_i v_i + \sum_i \phi_i(v_i), \tag{1}$$

although many networks are designed without the cubic term. The quadratic and cubic terms can involve quite general patterns of connectivity between neurons. All higher-order polynomial objectives can be reduced to this form (Mjolsness and Garrett, 1990).

If T_{ijk} and ϕ_i are absent from $E[\vec{v}]$ then the objective is quadratic and analogous to many numerical problems, defined on geometric domains, for which multigrid methods are successfully

[0]This work was supported in part by AFOSR grant 88-0240.

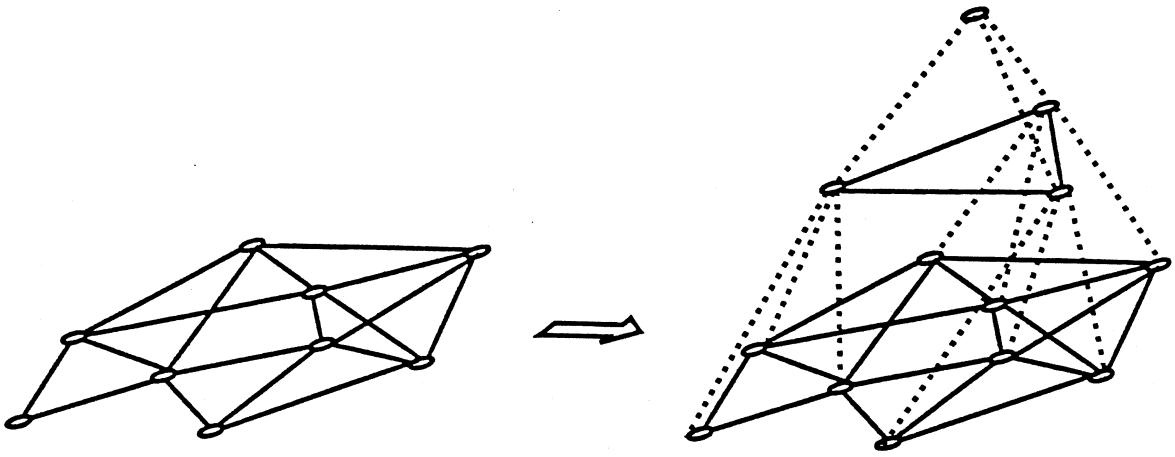

Figure 1: Multiscale network. An ordinary neural network, depicted on the left in this figure, may be governed by an objective function $E[v]$ with numerous simple interactions (links) between many nonlinear neurons (nodes). Such a network may be tranformed into a multiscale network, shown here on the right, by the addition of smaller and cheaper approximating networks at successive scales, with associated objective functions.

used as fast relaxation algorithms (Hackbusch, 1978; Miranker, 1981). Such algorithms proceed by introducing a smaller, approximate version of the problem at a coarser scale (i.e. using a coarser mesh) and alternating between relaxation steps for the fine-scale and the coarse-scale problems. This is done recursively, at many scales, and it is the two-way passage of information between scales which is responsible for the unusual effectiveness of the technique. In this paper we generalize the multigrid approach and propose a multiscale relaxation method which does not require an underlying geometric domain, and which incorporates the T_{ijk} and ϕ_i nonlinearities of equation (1) at all scales if those terms are present in the original fine-scale problem. Thus, we define and explore a multiscale optimization method appropriate for neural nets.

2 Theory

The desired multiscale neural network is illustrated schematically in Figure 1. To obtain such a design, let us consider the usual multigrid method for optimization.

The constituents of the multigrid method are: (1) a map from the original variables v_i to fewer coarse-scale variables V_a, called the "restriction" or "aggregation" map $\vec{V} = R[\vec{v}]$; (2) a map from coarse scale to fine scale called the "prolongation" or "disaggregation" map $\vec{v} = P[\vec{V}]$; (3) a coarse-scale objective function $\hat{E}[\vec{V}]$ which is intended to approximate $E[\vec{v}]$ although $\hat{E}[\vec{V}]$ is cheaper to evaluate and differentiate; (4) an algorithmic cycle by which $E[\vec{v}]$ is partially relaxed to produce a point $\vec{v}$, then $\vec{v}$ is restricted to produce $\vec{V}$, then $\hat{E}[\vec{V}]$ is partially relaxed by updating $\vec{V}$, then $\vec{V}$ is prolonged to produce a new value of $\vec{v}$ for relaxation under E again; and (5) modification of the basic cycle to handle many scales recursively. We must supply versions of these constituents suitable for the neural nets. The prolongation map and the coarse-scale objective are particularly

important.

The choice of $\hat{E}$ will be very simple for us: it is the restriction of E to a subspace parameterized by $\vec{V}$. The subspace is given by the prolongation map P:

$$\hat{E}[\vec{V}] = E[P[\vec{V}]]. \tag{2}$$

Thus, relaxation of $\hat{E}[\vec{V}]$ is equivalent to relaxation of $E[\vec{v}]$ in a subspace, namely the range of $P[\vec{V}]$. This form of $\hat{E}$ will be modified slightly in section 2.2, equation (13).

One special case of equation (2) covers much of what is done under the name of "multigrid" methods. If $E[\vec{v}]$ is purely quadratic and the maps P and R are matrix-vector multiplications, with $R = P^T$, we get a form of $\hat{E}$ which occurs often in multigrid problems:

$$\hat{E}[\vec{V}] = E_{\text{quadratic}}[P[\vec{V}]] = \sum_{ij} T_{ij} \Big(\sum_a P_{ia} V_a \Big) \Big(\sum_b P_{jb} V_b \Big) = \sum_{ab} (RTP)_{ab} V_a V_b. \tag{3}$$

Thus, there is a corresponding quadratic form on the coarse scale with matrix $\hat{T} = RTP$. If P and R are fixed, the reduced-dimension connection matrix $\hat{T}$ can be computed once and then used throughout the multigrid procedure. Of course if T is sparse, then the cost advantage of relaxing at the coarse scale requires that R and P be sparse as well, so that $\hat{T}$ will not suffer much fill-in. Since $\hat{T}$ is smaller than T, some fill-in can be tolerated.

2.1 The Prolongation Map

Our problem now is to design the prolongation map $P[\vec{V}]$. One might design it separately for each objective function E, so that the prolongation allows $\hat{E}$ to approximate E. For example, prolongations for canonical problems on two-dimensional grids have been extensively studied. This is very expensive in the designer's time. Or one might automatically learn an effective P for a particular E by optimizing some measurement of the degree to which $\hat{E}$ approximates E over a training set (after all we are studying neural nets), but that involves quite an escalation of computational cost and must happen on a slow time scale: P would be nearly constant during one multiscale optimization. We look for a less complex and less general choice of P. Our choice of P is intended to require relatively little effort on the part of the user of the multiscale procedure.

At the other extreme in complexity for P, one could just take $P[\vec{V}] = B\vec{V}$,where B is a 0-1 matrix representing the partition of the v_i variables into blocks each of which is summarized by V_a. (Note that $\sum_a B_{ia} = 1$ since, in our treatment, we require that each fine-scale neuron is a member of exactly one block of the partition. We will exploit this fact in equation (10) of section 2.2.) This scheme would be very inexpensive since B is very sparse, and it seems reasonable to request the user of the multiscale algorithm to guess a relevant partition of the variables. The problem with this simple scheme is that B is a 0-1 partition matrix, so when a particular coarse-scale variable V_a is updated there is a corresponding motion of $\vec{v}$ induced by P. This motion of $\vec{v}$ is along the $(1, 1, \ldots 1)$ direction within block a of the partition of the variables $\{v_i\}$. So, within each block the $(1, 1, \ldots 1)$ direction is always favored as the relaxation direction during coarse-scale relaxation, though for most objective functions E that particular direction is without special merit.

We therefore propose a compromise between generality and cost in the choice of P. Letting the matrix B correspond to a user-supplied partition of the original variables, take

$$\begin{aligned} v_i &= v_i^0 + \textstyle\sum_a P_{ia} V_a \\ P_{ia} &= B_{ia} z_i(\vec{v}^0) \\ \vec{z}(\vec{v}) &= -\nabla_{\vec{v}} E \end{aligned} \tag{4}$$

i.e.

$$P[\vec{V}]_i = v_i^0 - \sum_a B_{ia} \left(\frac{\partial E}{\partial v_i} \right)_{\vec{v}^0} V_a. \tag{5}$$

(C.f. the form of the prolongation matrix P_{ia} in (Chatelin and Miranker, 1982).) Notice that the prolongation (5), while linear, is not homogeneous. Here $\vec{v}^0$ is the value of $\vec{v}$ obtained by the last fine-scale relaxation of E, just before the coarse-scale relaxation of $\hat{E}$. Note that $\vec{v}^0$ is the image of $\vec{V} = \vec{0}$ under $P[\vec{V}]$. That is, we center our coarse-scale coordinate system so that the origin corresponds to the most recent fine-scale state vector, $\vec{v}^0$. Consequently there is no change in the value of E in the actual restriction step of the multiscale method. The restriction step may be written out as

$$\begin{aligned} v_i^0 &= v_i \\ V_a &= 0. \end{aligned} \tag{6}$$

Remark 1. Not only is there no change in E during the restriction step; after coarse-scale relaxation occurs, equation (2) guarantees that there is no change in E in the prolongation step of the multiscale method either. So E changes only during the fine-scale and coarse-scale relaxations. The conventional multigrid method, by contrast, may suffer an increase of E during the restriction or prolongation steps.

A second property of equation (5) is that the coarse-scale relaxation may be understood in terms of gradient descent on the fine scale. Each block of the partition relaxes in the direction specified by the projection of the gradient $\vec{z}$ onto that block's subspace in $\vec{v}$; in other words each block a undergoes its particular gradient descent, with V_a as the parameter describing the distance moved along the projection of the descent direction within that block. Initially $V_a = 0$ for all blocks. Notice that the prolongation map $P[\vec{V}]$ adapts dynamically as the optimization of E proceeds.

Some analytical results concerning the reduction of error available at the coarse scale under equations (2) and (5) are presented in the Appendix. Experimental results on the behavior of the entire multiscale network will be presented in section 4. In the remainder of this section we consider the computational cost of the multiscale network.

The proposed $P[\vec{V}]$ is inexpensive because B is very sparse: it has just one nonzero entry per fine-scale variable (i.e. per row). On the other hand, P must be recomputed on successive cycles of the multiscale method – whenever an intervening fine-scale relaxation step alters $\vec{v}^0$ and therefore the gradient $\vec{z}(\vec{v}^0)$ as well. So, unlike the coarse-scale network connection matrix RTP discussed earlier, the coarse-scale matrix here must be recomputed for each aggregation phase of coarse-scale relaxation:

$$\begin{aligned} \textstyle\sum_{ij} T_{ij} P[\vec{V}]_i P[\vec{V}]_j &= \textstyle\sum_{ij} T_{ij} \left(\sum_a B_{ia} z_i(\vec{v}^0) V_a \right) \left(\sum_b B_{jb} z_j(\vec{v}^0) V_b \right) + \text{linear terms} \\ &= \textstyle\sum_{ab} \hat{T}_{ab} V_a V_b + \ldots, \\ \text{where} \qquad \hat{T}_{ab} &= \textstyle\sum_{ij} B_{ia} B_{jb} T_{ij} z_i(\vec{v}^0) z_j(\vec{v}^0) \end{aligned} \tag{7}$$

The cost of recomputing $\hat{T}_{ab}$ and the rest of $\hat{E}$ is small compared to the cost of the fine-scale partial relaxation which immediately preceeds it. This network construction cost is due to the variable $\vec{v}^0$ and $\vec{z}(\vec{v}^0)$ vectors only. B remains constant and defines a fixed topology for the construction algorithm, which can be considered to be a feed-forward neural net that computes $\hat{T}$ according to equation (7). In comparison the fine scale relaxation net for E has about the same number of connections but it contains feedback and requires many iterated relaxation steps. Even for a hard-wired circuit implementation, the wiring cost of recomputing $\hat{T}_{ab}$ would about the same as that of the fine-scale relaxation and therefore in balance with it.

The cubic, quadratic, and linear terms of $\hat{E}$ may be computed from the corresponding terms of E as in equation (7). It remains to consider the effect of the prolongation map $P[\vec{V}]$ on E's single-neuron potential term, $\sum_i \phi_i(v_i)$.

2.2 The Potential Term

Low-order polynomial summands in $\phi_i(v_i)$ can be efficiently transferred to the rest of the objective; assume this has been done. We confine our discussion to singularites in ϕ_i, such as those of the barrier functions that correspond to sigmoidal neural transfer functions $v = g(u)$ through $g^{-1}(v) = \phi'(v)$ (Hopfield, 1984). It may be possible to handle some nonsigmoidal neural transfer functions in a similar way.

We will assume a restricted form of the potential term's dependence on the neuron index i:

$$E_\phi[\vec{v}] = \sum_i \phi_i(v_i) = \sum_i \left[c_{i-}\phi_-(a_{i-}v_i + b_{i-}) + c_{i+}\phi_+(a_{i+}v_i + b_{i+})\right] \quad (a_{i\pm} \geq 0) \tag{8}$$

where $\phi_-(w)$ has a singularity at the origin and provides an infinite penalty for negative values of w, so that relaxation algorithms are restricted to positive w; similarly $\phi_+(w) = \phi_-(-w)$ restricts its argument to negative values. For example, one could take $\phi_-(w) = w^{-p}$ or $-(1/2)\log w$ for $w > 0$. We assume $-b_{i-}/a_{i-} \leq 0 \leq -b_{i+}/a_{i+}$ for consistency, so that there is an allowable region of v_i. Equation (8) imposes a standard form for $\phi_i(v_i)$ (in terms of $\phi_\pm$) that can easily be generalized to a small number of standard "species" of potentials, $\phi_\pm^\lambda$, indexed by λ, one of which is used for each neuron.

Using equation (2) directly for the potential term in $\hat{E}$ is too expensive for a multiscale method, but can be simplified as follows:

$$E_\phi[P[\vec{V}]] = \sum_i \left[c_{i-}\phi_-\left(a'_{i-}\sum_a B_{ia}V_a + b'_{i-}\right) + c_{i+}\phi_+\left(a'_{i+}\sum_a B_{ia}V_a + b'_{i+}\right)\right] \tag{9}$$

where $a'_{i\pm} = a_{i\pm}z_i^0$ and $b'_{i\pm} = a_{i\pm}v_i^0 + b'_{i\pm}$. Using the fact that B is a partition matrix (as discussed in the previous section), equation (9) may be written as

$$E_\phi[P[\vec{V}]] = \sum_{ia} \left[B_{ia}c_{i-}\phi_-(a'_{i-}V_a + b'_{i-}) + B_{ia}c_{i+}\phi_+(a'_{i+}V_a + b'_{i+})\right]. \tag{10}$$

Since $a_{i\pm} \geq 0$, a'_{i-} and a'_{i+} have the same sign. If they are negative we can change their signs using $\phi_+(w) = \phi_-(-w)$, in the process interchanging the roles of ϕ_+ and ϕ_-. Then we obtain

$$E_\phi[P[\vec{V}]] = \sum_{ia} \left[B_{ia}\tilde{c}_{i-}\phi_-(\tilde{a}_{i-}V_a + \tilde{b}_{i-}) + B_{ia}\tilde{c}_{i+}\phi_+(\tilde{a}_{i+}V_a + \tilde{b}_{i+})\right] \quad (\tilde{a}_{i\pm} \geq 0) \tag{11}$$

where

$$\begin{aligned}
\tilde{a}_{i\pm} &= \begin{cases} a'_{i\pm} & \text{if } a'_{i-} > 0 \\ -a'_{i\mp} & \text{otherwise} \end{cases} \\
\tilde{b}_{i\pm} &= \begin{cases} b'_{i\pm} & \text{if } a'_{i-} > 0 \\ -b'_{i\mp} & \text{otherwise} \end{cases} \\
\tilde{c}_{i\pm} &= \begin{cases} c'_{i\pm} & \text{if } a'_{i-} > 0 \\ c'_{i\mp} & \text{otherwise} \end{cases}
\end{aligned} \tag{12}$$

Equation (11) has a form similar to its fine-scale counterpart, equation (8), but takes just as many ϕ evaluations to compute. Fortunately there is another strategy available. To guarantee a favorable result for a phase of coarse-scale relaxation, equation (2) is not necessary. It suffices to choose $\hat{E}$ so that

$$\begin{array}{lrcl} (a) & \hat{E}[\vec{V}] & \geq & E[P[\vec{V}]], \quad \text{and} \\ (b) & \hat{E}[\vec{0}] & = & E[P[\vec{0}]]. \end{array} \tag{13}$$

Subtracting (13a) from (13b) gives

$$0 \geq \Delta\hat{E} \geq \Delta E[P[\vec{V}]] \tag{14}$$

so relaxation in $\hat{E}$ implies *at least as much relaxation* in E, when $\vec{V}$ is prolonged back to $\vec{v}$ (see Remark 1 of section 2.1).

We establish (13a) and (13b) for each summand $\phi_\pm$ of $E_\phi[P[\vec{V}]]$ in equation (11). The idea is to bound all the ϕ_- terms in one partition block by the ϕ_- term whose singularity is *closest* to the initial value $V_a = 0$, and likewise with the ϕ_+ terms. So we look for a function $\hat{\phi}$ for which

$$\begin{array}{lrcl} (a) & \tilde{c}_{i-}\phi_-(\tilde{a}_{i-}V_a + \tilde{b}_{i-}) & \leq & \hat{\phi}_{i-}(V_a) \equiv C_{i-}\phi_-(\hat{a}_{a-}V_a + \hat{b}_{a-}) + D_{i-} \\ (b) & \tilde{c}_{i-}\phi_-(\tilde{b}_{i-}) & = & \hat{\phi}_{i-}(0) \equiv C_{i-}\phi_-(\hat{b}_{a-}) + D_{i-} \end{array} \tag{15}$$

where $-\hat{b}_{a-}/\hat{a}_{a-} = \max_{i\in\pi(a)} -\tilde{b}_{i-}/\tilde{a}_{i-}$, and C_{i-} and D_{i-} are to be adjusted to satisfy equations (15). Then summing over i and a as in equation (11) shows that (15) implies (13).

For some potentials ϕ_- (e.g. $\phi_-(w) = w^{-p}$ or $-(1/2)\log w$), equations (15) can be assured by demanding equality of the two functions and their V_a-derivatives at $V_a = 0$, and solving for C_{i-} and D_{i-}:

$$\begin{array}{rcl} C_{i-} & = & \tilde{c}_{i-}\tilde{a}_{i-}\phi'_-(\tilde{b}_{i-})/[\hat{a}_{a-}\phi'_-(\hat{b}_{a-})] \\ D_{i-} & = & \tilde{c}_{i-}\phi_-(\tilde{b}_{i-}) - C_{i-}\phi_-(\hat{b}_{a-}). \end{array} \tag{16}$$

Then

$$\begin{array}{rcl} \hat{E}_\phi[\vec{V}] & = & \sum_{ia} B_{ia}\left[\hat{\phi}_{i-}(V_a) + \hat{\phi}_{i+}(V_a)\right] \\ & = & \sum_a \left(\sum_i B_{ia}C_{i-}\right)\phi_-(\hat{a}_{a-}V_a + \hat{b}_{a-}) + \sum_i D_{i-} \\ & & + \sum_a \left(\sum_i B_{ia}C_{i+}\right)\phi_+(\hat{a}_{a+}V_a + \hat{b}_{a+}) + \sum_i D_{i+} \\ & \equiv & \sum_a \left[\hat{c}_{a-}\phi_-(\hat{a}_{a-}V_a + \hat{b}_{a-}) + \hat{c}_{a+}\phi_+(\hat{a}_{a+}V_a + \hat{b}_{a+})\right] + \hat{d} \quad (\hat{a}_{i\pm} \geq 0), \end{array} \tag{17}$$

which is the coarse-scale version of equation (8). Notice that calculating $\hat{E}_\phi$ now requires as many evaluations of ϕ as there are coarse-scale variables, not fine-scale variables; thus the cost of the coarse-scale neural net has become affordable for a multiscale method.

3 Discussion

Having presented the proposed multiscale optimization method for neural networks, we now make a few observations about it before presenting experimental results in section 4.

3.1 Benefits

We have presented a very conservatively designed multiscale method: regardless of the particular optimization problem being solved, the method can be applied and the restriction, prolongation

and coarse-scale relaxation steps are each guaranteed to have $\Delta E[v] \leq 0$ so that they at least do no harm to the fine-scale minimization process. Also, we showed that the cost of the method is low. The potential benefit is in speedier convergence: if the coarse-scale relaxations make a lot of progress in minimizing E, they can be called upon to do most of the work at very little computational cost. Ordinarily multigrid methods are studied on problems defined on spatial domains. For linear problems the modal techniques of Fourier analysis are used to prove that some speedup will occur (Brandt, 1977). We know of no such proof for our method; it must simply be tried out. Spatial-domain multiscale techniques can also lead to better local minima for problems with nonquadratic and multimodal objective functions; for example, scale-space continuation methods in computer vision may have this desirable property (Leclerc, 1989). We do not expect such an improvement in the local minima reached by our multiscale method because it never takes any uphill steps in the original, multimodal E. This suggests using the multiscale method to speed up convergence within a continuation method for minimizing E.

3.2 Saddle Points

As mentioned in the introduction, any polynomial summand of an objective function can be reduced to a cubic polynomial, so equation (1) is rather general (Mjolsness and Garrett, 1990). But this reduction occurs at the expense of replacing minima with saddle points which have the characteristic that each variable is classified ahead of time as requiring maximization or minimization. Thus one can consider a two-phase algorithm for finding such saddle points: alternately maximize E with respect to all the maximization variables, then minimize E with respect to all the minimization variables, and iterate. For each phase one can use the multiscale method we have presented to speed up the calculation. But the number of max/min cycles required for convergence may be large. Alternatively one could seek saddle points rather than minima or maxima within the multiscale algorithm, but this would destroy the built-in property that the restriction, prolongation and coarse-scale relaxation steps do not undo any of the optimization progress made at the fine scale. So the algorithm may be less effective on saddle point problems than on minimization problems.

3.3 The Partition, B_{ia}

One may obtain the required 0/1 partition matrix B_{ia} in several ways. For example, one might preprocess the original network of equation (1) so as to group together neurons that are strongly connected (large $|T_{ij}|$). This calculation may be carried out by another optimizing neural net similar to the graph-partitioning networks studied in (Anderson and Peterson, 1988). A particularly cheap (and approximate) way to do this is to bisect the net into two "modules" with minimal inter-module connections, and recursively bisect the modules. If the cost of such preprocessing is still regarded as too high for its benefits, one might simply guess a partition of the neurons based for example on a partition of some spatial domain loosely associated with the net, as we do in the experiments reported in section 4. Finally, one might attempt to learn an effective partition through experience in repeatedly running the network for different problems and optimizing B on a slow time scale while respecting its sparseness.

3.4 An Alternate Network Notation

We will not consider in great depth the question of implementing our multiscale neural nets as analog circuits or other special-purpose hardware. But a slight change of notation can bring this

question into the domain of a circuit-design method that uses rewrite rules by which one objective function can be algebraically transformed into another, more implementable one (Mjolsness and Garrett, 1990). We will describe this alternate notation for the multiscale optimization method.

In our case we want to transform a generic neural net objective function $E[\vec{v}]$, given by equation (1), into a two-level optimization scheme using $E[\vec{v}]$ and $\hat{E}[\vec{V}]$. The problem is that the result of this transformation is not a single objective function, but two objective functions (E and $\hat{E}$) and also the dynamic relationship between them. At some times we optimize E; at other times we optimize $\hat{E}$. We therefore allow the result of transforming an objective function to be a "clocked objective function", whose argument list and functional form depend on time through nonoverlapping clock functions $\psi_\alpha(t) = 0$ or 1 (with $\sum_\alpha \psi_\alpha(t) \leq 1$). Such clocked objective functions can be written as

$$E_{\text{clocked}}[\vec{x}, t] = \sum_\alpha \psi_\alpha(t) E_\alpha[\mathcal{X}_\alpha^{\text{free}} | \mathcal{X}_\alpha^{\text{fixed}}] \tag{18}$$

where $\mathcal{X}_\alpha^{\text{free}}$ and $\mathcal{X}_\alpha^{\text{fixed}}$ are subsets of variables from the entire set $\{x_i\}$. During phase α (i.e. when $\psi_\alpha(t) = 1$) $E_{\text{clocked}} = E_\alpha[\mathcal{X}_\alpha^{\text{free}} | \mathcal{X}_\alpha^{\text{fixed}}]$ is to be minimized with respect to all variables in $\mathcal{X}_\alpha^{\text{free}}$, while all variables in $\mathcal{X}_\alpha^{\text{fixed}}$ are to be held fixed or "clamped".

With this notation, the algebraic tranformation which encodes our multiscale method can be written as follows (assuming $\hat{T}_{abc} = 0$ for simplicity):

$$E[\vec{v}] \equiv E[\vec{v}|T, h] \rightarrow E_{\text{clocked}} \tag{19}$$

where

$$\begin{aligned}
E_{\text{clocked}} &= \psi_1(t) \quad E[\vec{v}] \\
&+ \psi_2(t) \quad \Big(\tfrac{1}{2}\textstyle\sum_a V_a^2 + \tfrac{1}{2}\sum_i (v_i - v_i^0)^2 + \tfrac{1}{2}\sum_i (z_i + \tfrac{\partial E}{\partial v_i})^2\Big)[\vec{z}, \vec{v}^0, \vec{V}|\vec{v}] \\
&+ \psi_3(t) \quad \Big(\textstyle\sum_{ab} (\tfrac{1}{2}\hat{T}_{ab}^2 - \hat{T}_{ab} \sum_{ij} B_{ia} B_{jb} z_i z_j T_{ij}) \\
&\qquad + \textstyle\sum_a (\tfrac{1}{2}\hat{h}_a^2 - \hat{h}_a \sum_i B_{ia} h_i a_i - \hat{h}_a \sum_{ij} B_{ia} z_i v_j^0 (T_{ij} + T_{ji})) \\
&\qquad + e_{\text{aggregate}}[\hat{\vec{a}}_\pm, \hat{\vec{b}}_\pm, \hat{\vec{c}}_\pm]\Big)[\hat{T}, \hat{\vec{a}}_\pm, \hat{\vec{b}}_\pm, \hat{\vec{c}}_\pm | \vec{z}] \\
&+ \psi_4(t) \quad \hat{E}[\vec{V}|\vec{z}, \vec{v}^0, \hat{T}, \hat{h}, \hat{\vec{a}}_\pm, \hat{\vec{b}}_\pm, \hat{\vec{c}}_\pm] \\
&+ \psi_5(t) \quad \tfrac{1}{2}\textstyle\sum_i (v_i - v_i^0 + \sum_a B_{ia} z_i V_a)^2 [\vec{v}|\vec{z}, \vec{v}^0, \vec{V}].
\end{aligned} \tag{20}$$

Phases 1 through 5 occur in cyclic order. Phase 1 performs fine scale relaxation; phase 2 is the restriction step, incorporating equation (6) and the definition of z as the negative gradient; phase 3 creates the coarse-scale net by means of equations (7); phase 4 is the coarse-scale relaxation step; and phase 5 is the prolongation step corresponding to equation (5). Phase 3 also involves the straightforward computation of the parameters of $\hat{\phi}$, summarized by $e_{\text{aggregate}}[\hat{\vec{a}}_\pm, \hat{\vec{b}}_\pm, \hat{\vec{c}}_\pm]$, by means of simple maximum-picking and analog arithmetic networks which may be designed using methods described in (Mjolsness and Garrett, 1990), for example. Phases 2, 3, and 5 are associated with simple quadratic objectives (except for $e_{\text{aggregate}}$), and we assume that the optimization dynamics can almost completely optimize these objectives in the time allowed by $\psi_\alpha(t)$. By contrast phases 1 and 4 have nonquadratic objectives which are just partially relaxed during each cycle. The entire sequence could be done recursively for more than two levels of optimization.

4 Experimental Results

We have applied the proposed multiscale optimization techniques to several nonquadratic objective functions. To illustrate the method's independence of a continuous spatial domain, we used an objective function for inexact graph-matching based on purely structural similarity of two graphs (Hopfield and Tank, 1986; von der Malsburg and Bienenstock, 1986). This problem may have application to problems of model matching in high-level computer vision. The objective is related to the Traveling Salesman Problem objective of (Hopfield and Tank, 1985), and both suffer from a strong increase in the number of local minima as the problem size increases. So we also considered a less problematical but spatially-structured nonquadratic objective function from low-level vision (Koch et al., 1986). It may be used for smooth two-dimensional surface reconstruction from sparse data, modified by nonlinear discontinuity detection processes, all defined on a discretized two-dimensional grid.

The inexact graph-matching problem is defined by two 0/1 incidence matrices g and G which specify the graphs to be matched, and the answer is a sparse 0/1 matrix M of variables specifying a permutation of nodes of g onto nodes of G. The objective function for graph-matching is taken to be

$$\begin{aligned} E[M] &= \tfrac{A}{2}\textstyle\sum_i(\sum_\alpha M_{\alpha i}-1)^2+\tfrac{A}{2}\sum_\alpha(\sum_i M_{\alpha i}-1)^2 \\ &+ \tfrac{B}{2}\textstyle\sum_{\alpha i}(1-M_{\alpha i})M_{\alpha i} \\ &- C\textstyle\sum_{\alpha\beta ij}G_{\alpha\beta}g_{ij}M_{\alpha i}M_{\beta j} \\ &+ \textstyle\sum_{ij}\phi_s(M_{ij}) \end{aligned} \tag{21}$$

where

$$\phi_s(x) = -\frac{1}{2g_0}[\ln x + \ln(1-x)] \tag{22}$$

and $A = 30$, $B = 1$, $C = 3$, and $g_0 = 10$. The first two terms favor unique matches between nodes in the two graphs; the B term favors $M \approx 0$ and $M \approx 1$ over the intervening values $M \in (0,1)$; the C term favors consistent matches between neighboring nodes in the two graphs; and the final term restricts each M variable to the interval $(0,1)$ by raising infinite barriers at the border.

For matching two n-node graphs of constant degree, this objective function has n^2 variables (neurons) and $\mathcal{O}(n^3)$ monomial interactions, each corresponding to a connection in the neural net which we simulated. It is known how to reduce this to $\mathcal{O}(n^2)$ connections in a saddle-point objective function (Mjolsness and Garrett, 1990) (with different temporal behavior). But to avoid the complications of studying the multiscale algorithm in the context of saddle points, we will compare it to the $\mathcal{O}(n^3)$ single-scale network just described. The multiscale method requires a partition of the $M_{\alpha i}$ variables. We used the outer product of partitions of the graph nodes in G and g, which are indexed by α and i respectively. We chose the graph partitions heuristically, intending to minimize the number of graph links that cross the partition boundaries.

Two families of arbitrary-size graphs were considered. First, we constructed a size-n nearly-balanced binary tree with incidence matrix g and the same tree under a different labelling of the nodes had incidence matrix G; in this case the graph partition was obtained by grouping together equal-length segments of the chain of nodes resulting from an inorder traversal of the tree. Second, we considered the sparse two-dimensional graphs of (Anderson and Peterson, 1988), obtained by independenly choosing n points from the unit square with the uniform probability distribution and connecting up all points within a distance d determined by the requirement that the average degree of connectivity be $4nd^2 = 3$. G was obtained similarly, after a randomly selected 20% of the points had been displaced by random vectors with x and y components between -.1 and +.1. In this case g and G were often not exactly isomorphic.

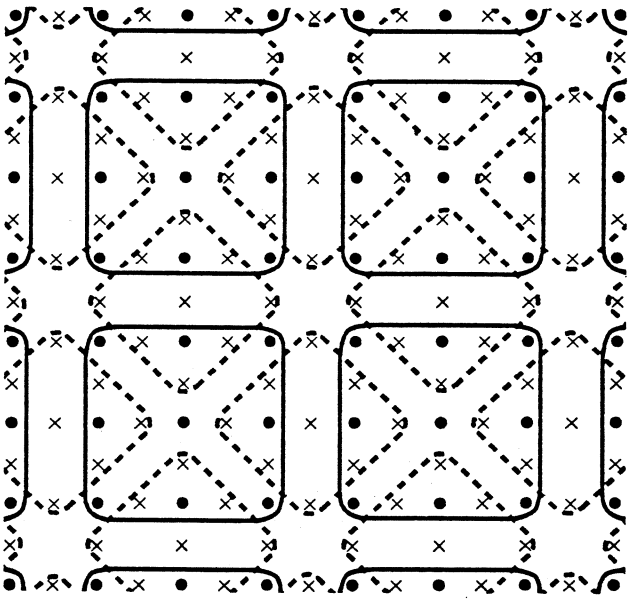

Figure 2: Partition of variables used in the multiscale version of the surface reconstruction network. Dots represent f_{ij} neurons in the plane. Crosses represent the associated horizontal and vertical "line process" neurons l^h_{ij} and l^v_{ij}. Each l neuron is placed symmetrically between the two f neurons with which it interacts. All these neurons occur in the original net and therefore at the finest scale of the multiscale net. The f neurons are aggregated by means of a partition, here into 3×3 blocks with solid outlines. The l neurons are aggregated into their own 3×3 partition blocks, separate from those of the f neurons, as shown by the dotted outlines. Both the l neurons and, at the larger scale, their partition blocks, occur on lattices tilted at 45 degrees to the f_{ij} lattice. In this way the fine-scale network structure is reproduced at the coarse scale.

The objective function for two-dimensional surface interpolation with discontinuities was

$$\begin{aligned} E[f,l] &= A\textstyle\sum_{ij}(1-l^v_{ij})(f_{i+1j}-f_{ij})^2 + A\sum_{ij}(1-l^h_{ij})(f_{ij+1}-f_{ij})^2 \\ &+ B\textstyle\sum_{ij}(f_{ij}-d_{ij})^2 + C\sum_{ij}(l^h_{ij}+l^v_{ij}) \\ &+ \textstyle\sum_{ij}(\phi_s(l^h_{ij})+\phi_s(l^v_{ij})) \end{aligned} \tag{23}$$

where d is the data, f is the real-valued interpolated function, and l^h and l^v form a set of 0/1 variables (called "line processes") indicating the presence of a discontinuity of the reconstructed surface (the f's in the horizontal or vertical directions. The parameters were $A = 1$, $B = 1$, $C = 0.1$, $D = 1$, $g_0 = 100$. The first two terms in E favor smoothness of the reconstructed function f in the horizontal and vertical directions, unless interrupted by a line process variable l. The B term favors consistency between f and the original data d. The C term penalizes a large number of active line processes or discontinuities. And the final term again restricts l to $(0,1)$. The f and l variables were partitioned differently; since they occur on the sites and links (respectively) of a fine-scale grid, their partition blocks were chosen to occur on the sites and links (respectively) of a coarse-scale grid. This partition scheme is shown in Figure 2.

In our experiments the numerical relaxation step at any given scale consisted of repeated univariate minimization along the gradient direction (a "line search") (Luenberger, 1984). This

strategy is standard in parallel optimization algorithms but slightly different from the continuous steepest methods often used in analog neural nets. Rather than continuously computing the gradient direction and moving the current state vector in that direction, a line search holds the state vector fixed while the gradient is computed, and then the descent direction is held fixed while a continuous displacement along that direction is calculated and taken. Such a two-phase minimization procedure could be implemented as continuous steepest descent in a clocked objective function similar to that described in section 3.4.

Also, the control scheme we used for the multiscale relaxation algorithm is a standard one for multigrid algorithms: multiscale relaxation at level l of the scale hierarchy (where level number increases with coarseness) consists of ordinary relaxation at level l, then multiscale relaxation at level $l+1$, then ordinary relaxation at level l, and a final step of multiscale relaxation at level $l+1$. This recursive "W-shaped" control scheme ensures that the smallest and cheapest networks are called upon most frequently in a completely serial algorithm. Naturally a parallel algorithm could omit the coarsest levels of the network, those which are too small to make effective use of the parallel machine. (Indeed such an algorithm might profitably relax each finer level of the network in optimal-size chunks, sequentially, with efficient relaxation of interactions that cross chunk boundaries delegated to the next coarser level.)

The graph-matching network of equation (21) was used to find the best match between two n-node graphs; we tried $n = 8, 16, 25$ for tree graphs and $n = 8, 16, 25, 36$ for 2-dimensional graphs. This single-scale algorithm was the control experiment. The problem was also given to a three-level multiscale neural net of the design proposed in this paper; its three levels contained roughly n^2, $n^{4/3}$, and $n^{2/3}$ neurons respectively. The two algorithms resulted in about equally good solutions, for three runs that differed in their randomly selected starting points. But their computational costs were different.

The total number of univariate relaxation steps (line searches) required for convergence was similar for the multiscale and the single-scale algorithms. But the multiscale relaxations are much cheaper to perform, on the average, than fine-scale relaxations of the full objective function because most multiscale relaxations occur at the coarsest scales. We estimate the magnitude of this effect by weighting the number of univariate minimization iterations by the number of nonzero connections in the network at each level of the multiscale scheme. This estimate omits the construction cost of the coarse-scale networks and the cost of computing the parameters of $\hat{\phi}(v)$ as well as the effects of highly parallel implementations (which would just truncate the coarsest levels of the network).

The construction cost of the coarse-scale networks and the cost of computing the parameters of $\hat{\phi}(v)$ were previously argued to be small compared to the cost of the fine-scale relaxation which they follow, if many steps of partial relaxation are involved. We tested the multiscale net for the extreme case in which just one univariate minimization was performed during each partial relaxation (more univariate minimizations seemed to be less effective per unit of computational cost, in our graph-matching experiments). In this case the network construction cost may become comparable to (but not more than) the relaxation cost if a univariate minimization requires very few search steps. This is because each coarse network construction, and each search step in a univariate minimization procedure, require a number of operations proportional to the number of nonzero connections at any given network level. Also the cost of computing the parameters of $\hat{\phi}(v)$ is proportional to the number of neurons at the same level - generally a much smaller quantity. To avoid implementation-dependent assumptions about the ratio of the relevant proportionality constants, we omit the (usually smaller) cost of the coarse-scale network construction and potential calculation from the following cost estimates except when reporting on actual running times of

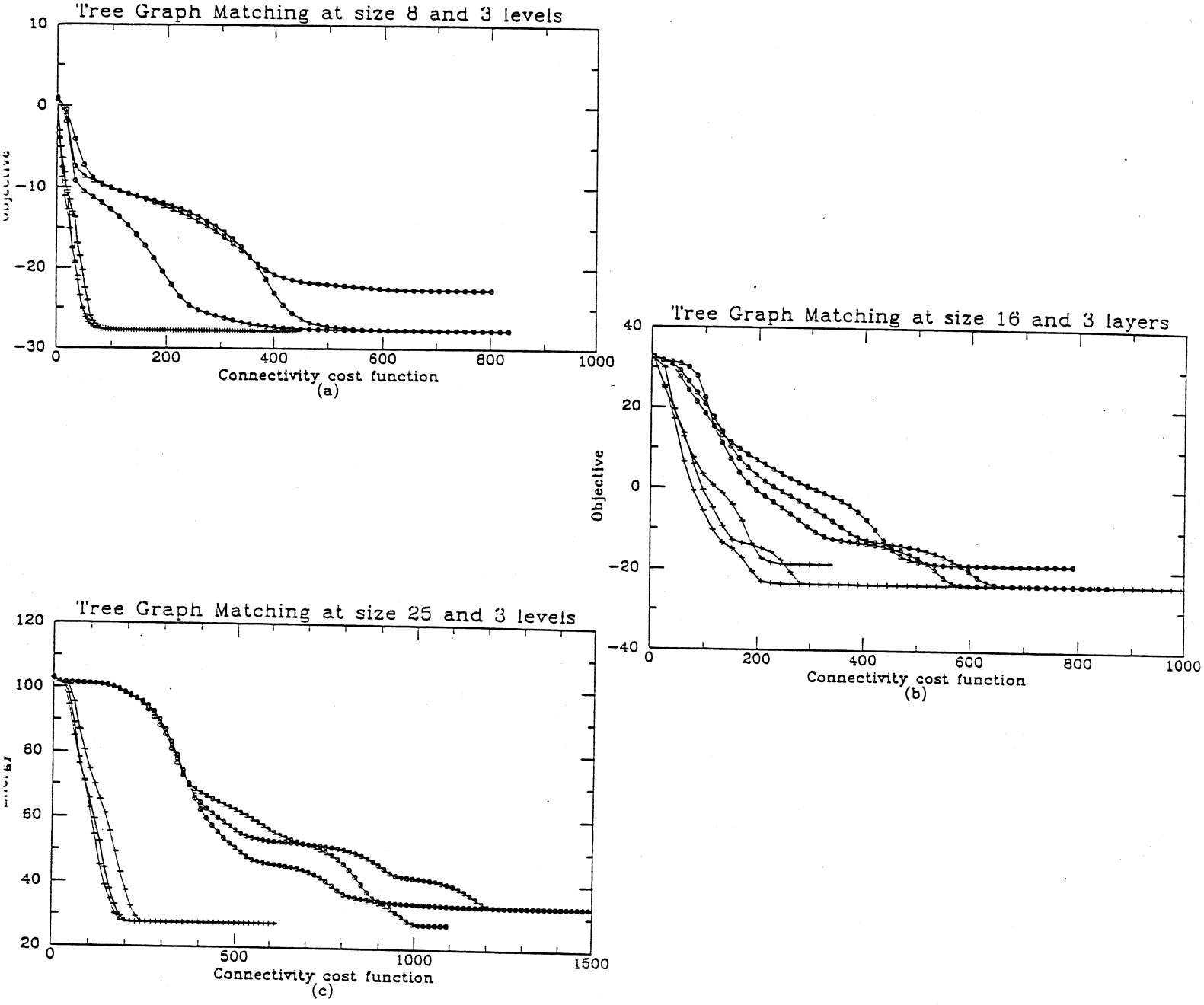

Figure 3: Graph matching network using tree graphs. Values of E vs. estimated cumulative computational cost, as the multiscale and single-scale minimization algorithms proceed. Three runs are shown for each size n. Single-scale runs are plotted with circles and multiscale runs are plotted with '+' signs. Under the multiscale algorithm, E is substantially minimized after roughly 20% to 50% of the effort required in the single-scale control experiment. (a) $n = 8$. Ratio of convergence times ≈ 6. (b) $n = 16$. Ratio of convergence times ≈ 2.5 . Multiply by correction factor .65 to get observed running-time cost ratio. (c) $n = 25$. Ratio of convergence times ≈ 5. Cost correction factor = .51 .

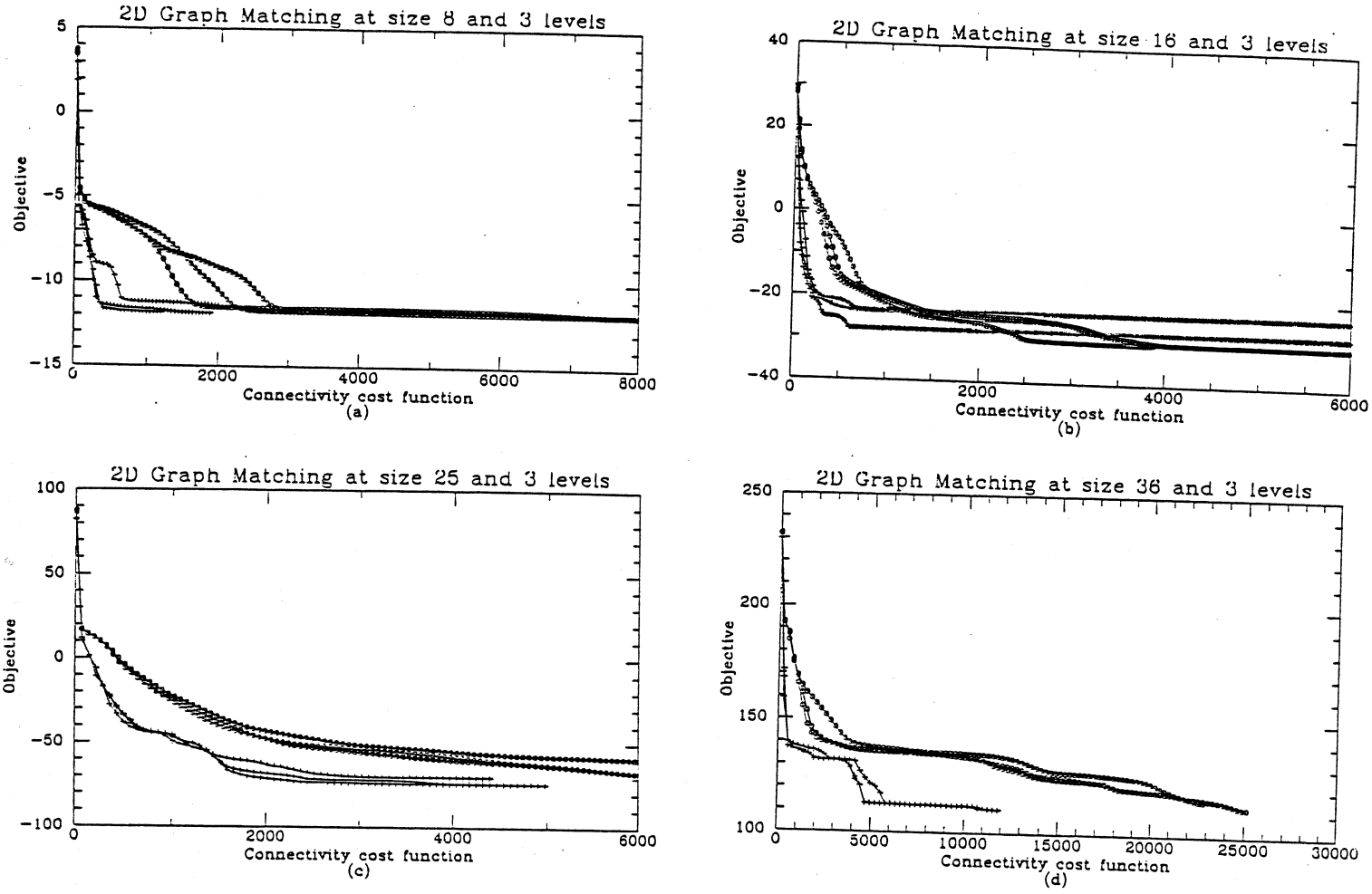

Figure 4: As in figure 3, but here for 2-d graphs. (a) $n = 8$. Ratio of convergence times ≈ 4. (b) $n = 16$. Ratio of convergence times ≈ 5 (worse local minima). Cost correction factor = .79. (c) $n = 25$. Ratio of convergence times $\approx 3 - 10$. Cost correction factor = .89 . Note speedup of late-time convergence tail, which makes the speedup depend sensitively on the convergence criterion. (d) $n = 36$. Ratio of convergence times ≈ 5.

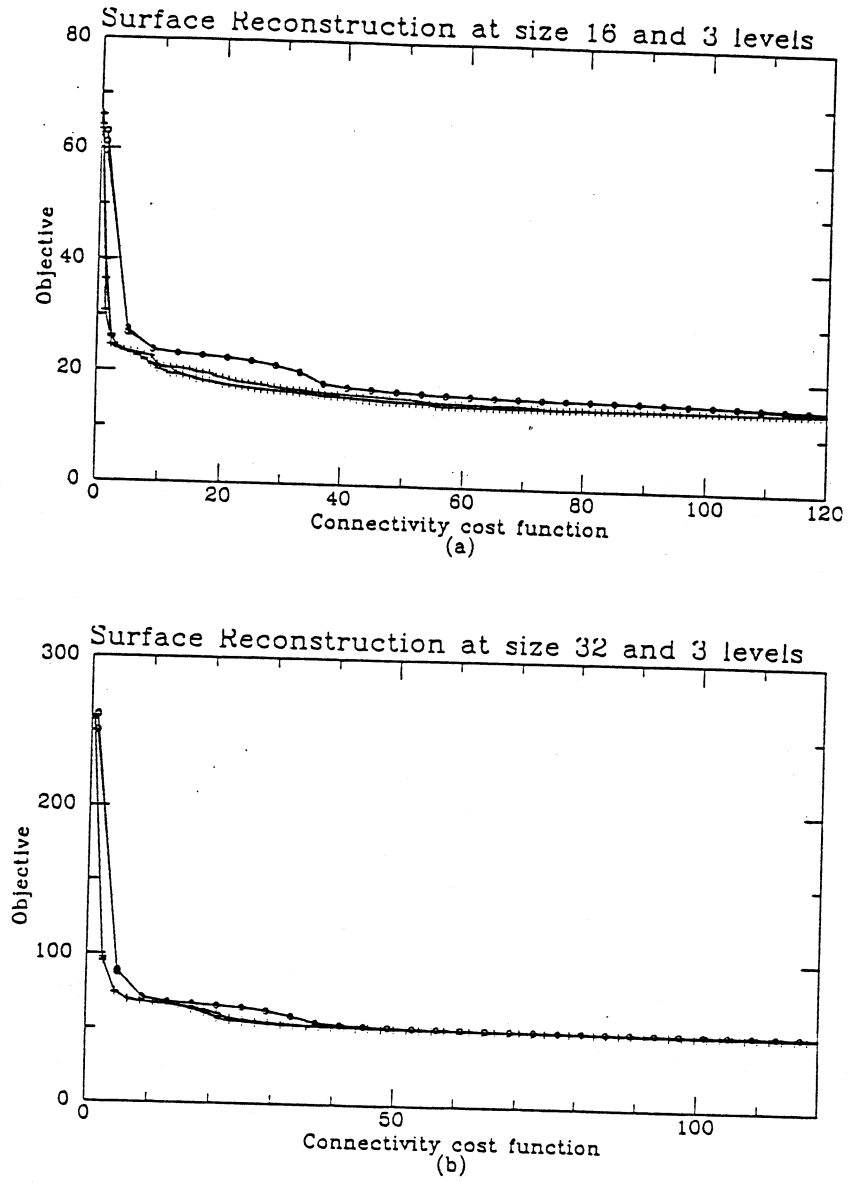

Figure 5: $E(t)$ for smooth surface reconstruction network with discontinuity detection. Three runs are shown for each size n. Single-scale runs are plotted with circles and multiscale runs are plotted with '+' signs. The main effect of the multiscale algorithm for this network is to cut off long convergence tails, thus greatly reducing the amount of computational effort required for the final reduction in E. (a) $n = 16$. (b) $n = 32$.

our serial-computer implementation.

Under this estimate of the cost of the two algorithms, the multiscale algorithm is strongly favored over the single-scale control experiment for matching tree graphs as shown in Figure 3. Similar results were obtained for matching two-dimensional graphs as shown in Figure 4. The simulations generally exhibit punctuated descent towards local minima. Note the roughly constant ratio of convergence times, as a function of n, between single-scale and multiscale simulations. (For larger sizes there may be an additional benefit in cutting off a long tail of convergence at late times.) There is generally about a five-fold improvement in estimated cost under the multiscale method. For three scales and the W-shaped recursive control scheme, a five-fold improvement is quite reasonable since it corresponds to roughly equal effectiveness of the fine-scale and coarse-scale nets. But our attempts to improve this ratio by moving to four levels resulted in only marginal improvements even for $n = 36$. As can be seen from the figures, the ratios of estimated cost are sensitive to convergence criterion.

Including the costs of coarse-scale network construction will reduce the observed five-fold savings in an implementation-dependent way. For our serial implementation using sparse data structures and the C programming language on a Sparcstation 1 computer, the ratio of running times for single-scale and multiscale tests was observed to be well predicted by the estimated cost function used in the Figures in the following sense: for each problem there is a robust "cost correction factor" which can be used to multiply the estimated cost ratio to get the observed running time ratio, independent of iteration number and, to a lesser extent, problem size. The correction factors are a little less than unity, so the observed cost ratios are a little less favorable to the multiscale method than estimated ones. For two-dimensional graph matching the observed cost correction factor is .79 ($n = 16$) or .89 ($n = 25$). For tree graph matching the correction factor is .65 ($n = 16$) or .52 ($n = 25$).

In the case of the two-dimensional discontinuous interpolation network of equation (23), the multiscale method cut off a long tail of slow convergence, allowing much quicker convergence in the final stages of minimization as shown in Figure 5.

To summarize the experimental results, the multiscale method offers a decrease in computational cost by a factor that depends on the problem and on the convergence criterion, is roughly independent of problem size for the three problems we tested, and is between two and five for these problems. The advantage may be larger than this when the original network has a slow convergence tail.

5 Conclusion

We have developed a fairly general, low-cost multiscale method for neural net optimization. It transforms a neural network into a multiscale neural network of a similar form. In the particular networks to which it was applied, we observed a nontrivial speedup by a constant factor (between two and five) independent of problem size. Further improvements in computational cost are very likely to be available, especially for problem-specific multiscale neural net methods, since the method proposed here is conservatively designed in order to be more generally applicable. The conservative design ensures that the multiscale network never accepts steps that move the original objective function in the wrong direction; thus convergence is unobstructed. The method applies to highly nonlinear networks, without underlying geometric domains or known descent directions, and for which no particular problem structure is assumed or exploited except for a user-supplied hierarchical partition of the optimization variables.