Biol. Cybern. 80, 227–234 (1999)

Biological Cybernetics
© Springer-Verlag 1999

K-way neural network graph partitioning with separator vertices

C.C. Pain, C.R.E. de Oliveira, A.J.H. Goddard, A.P. Umpleby

Computational Physics and Geophysics Group, T.H. Huxley School of Environment, Earth Sciences and Engineering, Imperial College of Science, Technology and Medicine, Prince Consort Rd, London SW7 2BP, UK

Received: 14 January 1998 / Accepted in revised form: 24 November 1998

Abstract. A neural network for partitioning graphs into any number of subgraphs using a k-way procedure is presented. The resulting neural network optimises all subgraphs so that they contain approximately the same number of vertices with the exception of a 'separator' subgraph. The latter separates all the other subgraphs and is optimised to contain a minimum number of vertices. Expressions for the neuron link weights for such a network are derived analytically, and the recall mechanism of the mean field theorem neural network is used to obtain the graph partitioning. Applications focus on partitioning graphs associated with finite element meshes which, together with parallel domain decomposition solution methods, provide the motivation for this work.

1 Introduction

Graph partitioning problems arise in many areas of engineering science, for instance, circuit design and numerical modelling on computational meshes – the latter is an example of task scheduling on a parallel computer. With the advent of parallel computing, a greater need has arisen for robust and computationally efficient techniques for scheduling tasks and assigning load to processors. In recent years a number of successful algorithms have been applied to solve these problems: spectral bi-section [16], geometric partitioning [3], Greedy and Kernighan-Lin heuristics [6]. These algorithms are typically applied with a multi-level approach [11] akin to multi-grid solutions methods [8]. JOSTLE [17] and METIS [12] are current state of the art computer codes which use many of the above graph partitioning techniques. Various authors have studied graph partitioning using neural networks, see [2, 5, 9, 14] for example. A good description of graph partitioning using neural networks can be found in [7]. In [1] a neural network was presented for partitioning finite element meshes. However, to our knowledge, the neural network described here is the first one to be designed to produce a set of separator vertices.

In this paper solutions (optimisations) are sought to a particular type of graph partitioning problem. In the solutions, the number of vertices in each sub-graph (partition) is 'balanced', with the exception of one of the sub-graphs which separates all other sub-graphs. An additional aim of the optimisation, other than balancing the number of vertices, is to minimise the number of vertices in the separator subgraph. The term k-way is used to indicate that the 'k' balanced partitions are obtained simultaneously. For reasons of computational efficiency, all the vertices are first assigned to partitions, and then, using this as the initial condition, the 'separator finding' neural network is applied.

This paper focuses on the graph partitioning problem associated with the dissection of a finite element mesh. The resulting dissection can be used with a domain decomposition method [13] to solve field equations on a parallel computer. There are two ways in which a finite element mesh comprising nodes and elements [3] can be subdivided. One is to assign elements to processors or partitions, and the other is to assign nodes. Element-based partitioning methods are derived by using a graph which describes the communication between elements as opposed to nodes, with each vertex in the graph representing an element. Assigning elements is perhaps the natural approach if a set of separator nodes between the resulting partitions (sub-domains) is sought, e.g. Schur complement methods [13]. However, if triangular or tetrahedral elements are used, there are many more elements than nodes, and thus assigning elements to processors can increase the search space of the optimisation compared with assigning nodes to processors or partitions. Therefore, the latter optimisation problem is solved here, and the problem of assigning nodes to processors is considered. To do this, a functional is first derived that provides a measure of the quality of a graph partitioning with separator vertices (nodes). The separator neural network

Correspondence to: C.C. Pain
(e-mail: c.pain@ic.ac.uk)

is then presented as a method of optimising this functional. Domain decomposition methods that do not use separator nodes (e.g. multi-block-explicit), but use a halo (nodes belonging to a subdomain that are adjacent to nodes belonging to different subdomains), would be more suited to node-assigning methods.

Section two of this paper covers, functions that attempt to gauge the quality of a partition. Section three explains the mean field theorem neural network and, in section four, the results obtained using this network are presented.

2 Functional for gauging graph partitioning quality

A partitioning of a graph without separator nodes can be arranged as follows. Let us introduce a communication matrix K which contains the cost of communication from node i to node j, K_{ij}, and suppose that there are N nodes. Typically,

$$K_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and nodes } i \text{ and } j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Now define a matrix K_f with submatrices $K_f{}^{\mu\nu}$ given by

$$K_f{}^{\mu\nu} = \begin{cases} 0 & \text{if } \mu = \nu \\ K & \text{if } \mu \neq \nu \end{cases}, \quad \forall \mu, \nu \in [1, 2, \dots, S] \tag{2}$$

and a matrix K_d with submatrices $K_d{}^{\mu\nu}$ given by

$$K_d{}^{\mu\nu} = \begin{cases} K & \text{if } \mu = \nu \\ 0 & \text{if } \mu \neq \nu \end{cases}, \quad \forall \mu, \nu \in [1, 2, \dots, S] \tag{3}$$

If it is necessary to divide a graph into S partitions, then it is convenient to define S vectors $\mathbf{x}^{\mu}, \mu \in \{1, 2, \dots, S\}$ of length equal to the number of nodes N and

$$\mathbf{x}_i^{\mu} = \begin{cases} 1 & \text{if node } i \text{ is in partition } \mu \\ 0 & \text{otherwise} \end{cases}$$

When continuous optimisation methods are used, then $\mathbf{x}_i^{\mu} \in [0, 1]$ becomes the probability of node i being in partition μ. The probability constraint is thus

$$\sum_{\mu=1}^{S} \mathbf{x}_i^{\mu} = 1 \tag{4}$$

An example of a functional to be optimised is

$$F = F_1 + F_2 \tag{5}$$

This functional F is to be minimised to produce the desired partitioning. Optimising the term involving F_1 should minimise inter-partition communication, while optimising F_2 should balance the number of nodes in, or load associated with, each partition.

Next the submatrices $C_d{}^{\mu\nu}$ of C_d are defined as

$$C_d{}^{\mu\nu} = \begin{cases} B & \text{if } \mu = \nu \\ 0 & \text{if } \mu \neq \nu \end{cases}, \quad \forall \mu, \nu \in [1, 2, \dots, S] \tag{6}$$

where B is a square matrix of order N and unit entries. A minimum of $F_2 = \frac{1}{2}\alpha \mathbf{x}^T C_k \mathbf{x}$ can balance the number of nodes in each partition with the matrix C_k defined by its $S \times S$ submatrices $C_k{}^{\mu\nu}$:

$$C_k{}^{\mu\nu} = \begin{cases} \frac{S-1}{S} B & \text{if } \mu = \nu \\ -\frac{1}{S} B & \text{if } \mu \neq \nu \end{cases} \tag{7}$$

Note that $C_k C_k = N C_k$ and thus C_k is positive semi-definite, and thus $\mathbf{x}^T C_k \mathbf{x}$ is never negative. Putting these ideas together, a useful functional whose minimum produces the desired partition is

$$F = F_1 + F_2 = \frac{1}{2}\mathbf{x}^T K_f \mathbf{x} + \frac{1}{2}\alpha \mathbf{x}^T C_k \mathbf{x} \tag{8}$$

Taking advantage of the probability constraint, an equivalent expression can be found, thus

$$F = \frac{1}{2}\mathbf{x}^T K_d (b - \mathbf{x}) + \frac{1}{2}\alpha \mathbf{x}^T \left(C_d \mathbf{x} - \frac{N}{S} b \right) \tag{9}$$

subject to the probability constraint 4. If two partitions are required, half of the vector $\mathbf{x}$, that is $\mathbf{x}^2$, can be removed along with the probability constraint. Then, when $\mathbf{x}_i^1 = 1$, node i belongs to partition 1 and, when $\mathbf{x}_i^1 = 0$, node i belongs to partition 2.

For finite element meshes comprising triangular or tetrahedron elements, the graph representing the communication of the elements is considerably larger, in terms of number of vertices, than the graph representing the communication of nodes. This increases the size of the search space, which can make the problem harder to solve. One way to overcome this is to search for both a set of separator vertices and partition vertices at the same time. To this end, minimising F defined by

$$\begin{aligned} F &= F_1 + F_2 + F_3 \\ &= \frac{1}{2}\mathbf{x}^T \begin{pmatrix} K_f & 0 \\ 0 & 0 \end{pmatrix} \mathbf{x} + \frac{1}{2}\alpha \mathbf{x}^T \begin{pmatrix} C_k & 0 \\ 0 & 0 \end{pmatrix} \mathbf{x} + \frac{1}{2}\gamma \mathbf{x}^T Q \mathbf{x} \end{aligned} \tag{10}$$

with suitable α, γ will result in the desired partitioning. Q is defined by its $N \times N$ block entries $Q^{\mu\nu}$ in which $Q^{\mu\nu} = I, \ \forall \mu, \nu \in [1, 2, \dots, S+1]$. The probability constraint becomes

$$\sum_{\mu=1}^{S} \mathbf{x}_i^{\mu} + \mathbf{x}_i^{\mathscr{I}} = 1, \quad \forall i \tag{11}$$

where $\mathscr{I} = S + 1$, $\mathbf{x}_i^{\mathscr{I}}$ is the probability that node i is a separator node, $y^T = (\mathbf{x}^{1^T} \ \mathbf{x}^{2^T} \ \dots \ \mathbf{x}^{s^T})$, $\mathbf{x}^T = (y^T \ \mathbf{x}^{\mathscr{I}^T})$, and F_3 minimises the number of separator nodes. Using (11) and the identity $Q\mathbf{x} = b$, a computationally more efficient, but equivalent form is found to be

$$\begin{aligned} F = &-\frac{1}{2}\mathbf{y}^T K_d \mathbf{y} - \frac{1}{4}\sum_{\mu=1}^{S} (\mathbf{x}^{\mu} K \mathbf{x}^{\mathscr{I}} + \mathbf{x}^{\mathscr{I}} K \mathbf{x}^{\mu}) \\ &+ \frac{1}{2}\mathbf{y} K_d b + \frac{1}{2}\mathbf{y}^T C_d \mathbf{y} + \frac{1}{4S}\sum_{\mu=1}^{S} (\mathbf{x}^{\mu} B \mathbf{x}^{\mathscr{I}} + \mathbf{x}^{\mathscr{I}} B \mathbf{x}^{\mu}) \\ &- \frac{1}{2}\frac{\alpha N}{S}\mathbf{y}^T b + \frac{1}{2}\gamma \mathbf{x}^T b = \frac{1}{2}\mathbf{x}^T A \mathbf{x} + \mathbf{x}^T \mathbf{r} \end{aligned} \tag{12}$$

for appropriate vector $\mathbf{r}$ and symmetric matrix A, thus $\frac{\partial F}{\partial \mathbf{x}} = A\mathbf{x} + \mathbf{r}$.

The product $\frac{1}{2}\gamma$ can be chosen to be just less than the smallest non-zero entry in the local communication matrix K, ensuring that no two nodes belonging to different partitions μ, ν, where $\nu \leq S, \mu \leq S$, are adjacent to each other. In fact, if there is considerable variation in the non-zero entries in the matrix K, then it may be advisable to replace the term $\gamma \mathbf{x}^T b$ with $\mathbf{x}^T \Gamma b$ in which Γ is a diagonal matrix with diagonal entries $\Gamma_{ii} = \rho \min_j K_{ij}$. Non-zero values of K_{ij} are only considered in this expression, and ρ is a scalar less than 1.

3 Mean-field theorem neural network

This section explains how neural networks can be used to solve the graph partitioning problem with and without separator vertices.

3.1 The neural network

A simulated electrical circuit or neural network can be used to solve general optimisation problems [15]. Traditionally, a neural network is a directed-graph whose vertices or neurons act on data passed to them from other neurons. Information can travel only in the way suggested by the directed-graph. Symmetric bidirectional links will be used in the neural networks presented here. Each directed edge has a weight associated with it that governs the overall importance of the information travelling along it. To solve many optimisation problems, including the mapping problem, it is not necessary to train the net, as expressions can be derived for these.

The neural network used in this study is the mean field theory (MFT) neural network in which the outputs $\mathbf{x}_i^\mu$ obey the normalised Boltzmann distribution,

$$\mathbf{x}_i^\mu = \frac{\exp(-f_i^\mu/T)}{\sum_{\nu=1}^S \exp(-f_i^\nu/T)} \tag{13}$$

at a given temperature T and where the mean field is given by

$$f_i^\mu = \frac{\partial F(\mathbf{x})}{\partial \mathbf{x}_i^\mu} \tag{14}$$

thus ensuring that $\sum_{\nu=1}^S \mathbf{x}_i^\mu = 1, \forall i$. See [2, 4, 9, 10, 14] for more details about the MFT neural network and the related Hopfield network.

The variables $\mathbf{x}_i^\mu, \forall \mu$ at each i are updated asynchronously using (13) in a predefined order, typically sweeping across the net. Alternatively, i may be chosen at random and the variables $\mathbf{x}_i^\mu, \forall \mu$ updated according to (13).

A functional F for graph partitioning is given by (8). Equivalently, the computationally more efficient functional given by (9) can be used (this is the functional actually used in our neural networks). If one wishes to decompose a graph into two (or use a nested dissection approach), it may be beneficial to use the net of [14] where the functional is given by $F = \mathbf{x}^T(-K + \alpha I)\mathbf{x}$; the matrix K is the communication matrix (1), I is the identity, and α is the load balancing parameter. Then the equation $\mathbf{x}_i^\mu = \tanh(f_i^\mu/T)$ is used to iteratively update the neurons, and the neurons saturate to ± 1. If a neuron value is -1, then the corresponding node belongs to partition one; if it equals 1, the node belongs to partition two. This method has the advantage of needing only N neurons, where N is the number of nodes in the graph to be bi-partitioned. Alternatively, the N neuron version of the functional given by (9) might be used to the same effect.

As in choosing the gain in the Hopfield net, the choice of the temperature T is important in enabling good viable solutions to be obtained (too small a value of T will result in poor solutions, and too large a value will result in $\mathbf{x}_i^\mu \neq 0$ or 1 for some i and μ). One may overcome this difficulty by gradually reducing T from a relatively large value, using $T = \beta T$ (as performed with the annealing method). The advantage in using annealing is that, as T is reduced, the solution saturates to one of the hypercube corners (that is, $\mathbf{x}_i^\mu = 0$ or 1) without degradation of the solution. This annealing method can also be used with the Hopfield net.

The simplest method of updating the neurons is to visit each vertex in the graph to be partitioned in a predefined order, and update the associated neurons according to the rule given by (13). At convergence the temperature is annealed down; for example, $T = 0.7T$. At convergence for this new temperature, the temperature is further reduced, and this process is repeated until all the neurons are saturated.

In [4] it was shown how using a non-uniform T can be advantageous in solving the travelling salesman problem; non-uniform T can also be advantageous in solving the mapping problem, as will be demonstrated in the next section.

3.2 Suitable values of temperature and penalty parameter

The critical temperature, T_c, is the value of the temperature above which all the neurons have equal values. Below the critical temperature the neurons start to saturate; that is, the neuron values become non-uniform and will move towards 0 or 1. The critical temperature is usually clearly defined, (see, for example, [5]). For the separator-finding neural network it will be assumed that all the neurons are fully saturated and below the critical temperature; for this network, the separator neurons will start to saturate to either 1 or 0 with some separator neurons saturating to 1.

3.2.1. Neural network – no separator vertices

In this section the critical temperature will be calculated by examining the conditions that allow a small perturbation of the network to grow. To this end, suppose all the neurons have equal values of $1/S$. A small perturbation $\triangle \mathbf{x}_i^k$ is made to the neuron associated with vertex i and partition k and corresponding perturbations of $\triangle \mathbf{x}_i^\nu = -\triangle \mathbf{x}_i^k/(S-1), \forall \nu \neq k$. Then, from the neural updating, (13)

$$\frac{\partial \mathbf{x}_i^k}{\partial f_i^m} = \begin{cases} \frac{\mathbf{x}_i^k(\mathbf{x}_i^k-1)}{T} \approx \frac{1-S}{S^2T} & \text{if } m=k \\ \frac{\mathbf{x}_i^k\mathbf{x}_i^k}{T} \approx \frac{1}{S^2T} & \text{if } m\neq k \end{cases} \tag{15}$$

The functional F that will be minimised by the neural network is given by (9) or, equivalently, (8). That is,

$$F = \frac{1}{2}(\mathbf{x}^T K_f \mathbf{x} + \alpha \mathbf{x}^T C_k \mathbf{x})$$

with the $S \times S$ sub-matrices $K_{f\,ij}$ of K_f given by (2) and the communication matrix K given by (1) with $\beta = 1$ and $S_p = 0$.

The mean field f is given by $f = K_f\mathbf{x} + \alpha C_k \mathbf{x}$ and therefore

$$\frac{\partial f_i^k}{\partial \mathbf{x}_j^m} = \begin{cases} \frac{\alpha(S-1)}{S} & \text{if } m=k \\ K_{ij} - \frac{\alpha}{S} & \text{if } m\neq k \end{cases}$$

The result of the perturbation is a perturbation of the mean field of

$$\triangle f_j^v = \sum_{n=1}^{N}\sum_{\mu=1}^{S} \frac{\partial f_j^v}{\partial \mathbf{x}_n^\mu} \triangle \mathbf{x}_n^\mu = f_{j\,\text{old}}^v - f_{j\,\text{new}}^v$$

and therefore:

$$\triangle f_j^k = \sum_{\mu=1}^{S} \frac{\partial f_j^k}{\partial \mathbf{x}_i^\mu}\triangle \mathbf{x}_i^\mu = -\Big(K_{ij} - \frac{\alpha}{S}\Big)\triangle \mathbf{x}_i^k$$

$$\triangle f_j^m = \Big(K_{ij} - \frac{\alpha}{S}\Big)\frac{\triangle \mathbf{x}_i^m}{S-1}, \quad m \neq k$$

Changes in the mean field cause [from (15)] changes in $\mathbf{x}_j^\mu$ of

$$\begin{aligned}\triangle \mathbf{x}_j^k &= \frac{\partial \mathbf{x}_j^k}{\partial f_j^k}\triangle f_j^k + \sum_{m\neq k}^{S} \frac{\partial \mathbf{x}_j^k}{\partial f_j^m}\triangle f_j^m \\ &= \frac{1}{ST}\Big(K_{ij} - \frac{\alpha}{S}\Big)\triangle \mathbf{x}_i^k \end{aligned} \tag{16}$$

$$\begin{aligned}\triangle \mathbf{x}_j^m &= \frac{\partial \mathbf{x}_j^m}{\partial f_j^k}\triangle f_j^k + \frac{\partial \mathbf{x}_j^m}{\partial f_j^m}\triangle f_j^m + \sum_{\mu\neq k,m}^{S} \frac{\partial \mathbf{x}_j^m}{\partial f_j^\mu}\triangle f_j^\mu \\ &= -\frac{1}{ST(S-1)}\Big(K_{ij} - \frac{\alpha}{S}\Big)\triangle \mathbf{x}_i^k, \quad m \neq k \end{aligned} \tag{17}$$

It is assumed that the iteration is performed synchronously and that the perturbation is growing if $\sum_{j=1, j\neq i}^{N} \mathbf{x}_j^k$ is growing in the local vicinity of node i, that is, nodes j for which $K_{ij} \neq 0$. Empirical evidence suggests that the critical temperature T_c of the network, when the neuron values are updated asynchronously, is nearly identical to the T_c resulting from synchronous neuron updates. For the range of α of interest, T_c is almost independent of α, see also [5]. This can be seen by plotting T_c against α. In the vicinity of node i the load-balancing term has little effect. Therefore, summing (16) over all j and setting $\sum_j^N \triangle \mathbf{x}_j^k = \triangle \mathbf{x}_i^k$ (that is the perturbation does not grow or contract), at the critical temperature T_{ci}

$$\sum_{j=1}^{N} \frac{1}{ST_{ci}} K_{ij} \triangle \mathbf{x}_i^k = \triangle \mathbf{x}_i^k$$

or

$$T_{ci} = \frac{\sum_{j=1}^{N} K_{ij}}{S} \tag{18}$$

Equation (18) suggests that neurons with fewer connections within each layer of the network will be less saturated at a given temperature – as the critical temperature is lower for these neurons. This is observed in practice, see Fig. 4. We do not examine the perturbation at a single node, as in [5], because this tends to decrease and then increase for the general graphs of interest here. This would necessitate the calculation of $\triangle \mathbf{x}_j^k$ at many iteration levels to find an accurate T_{ci}.

For load balancing it will be assumed that $\sum_{j=1}^{N} \mathbf{x}_j^k$ does not grow globally. Thus, in the limit for node i, $\sum_{j=1}^{N}(K_{ij} - \frac{\alpha}{S}) = 0$ or

$$\alpha_i = \frac{S}{N}\sum_{j=1}^{N} K_{ij} \tag{19}$$

A global value of α and T_c may be required. This can be obtained by taking some kind of average of $\alpha_i, T_{ci}, \forall i$ or the maximum, or most common of these values. Therefore, T_c and α can be found from

$$T_c = \frac{1}{SN}\sum_{j=1}^{N}\sum_{i=1}^{N} K_{ij} \tag{20}$$

$$\alpha = \frac{S}{N^2}\sum_{j=1}^{N}\sum_{i=1}^{N} K_{ij} \tag{21}$$

Exhaustive numerical experiments have shown that the value of α obtained from this equation results in good load balancing. However, our experiments suggest that the global critical temperature should be weighted more towards the most common values of T_{ci}. This may be avoided by using a non-uniform temperature given by (18).

3.2.2 Neural network – with separator vertices

The neural updating (13) for the separator finding neural network, with probability constraint (11) is

$$\mathbf{x}_i^\mu = \frac{\exp(-f_i^\mu/T)}{\sum_{v=1}^{S+1}\exp(-f_i^v/T)}, \quad \mu \in \{1,2,\ldots,S,S+1\} \tag{22}$$

The separator finding network minimises (10), so the mean field f is then given by

$$f = \begin{pmatrix} K_f & 0 \\ 0 & 0 \end{pmatrix}\mathbf{x} + \alpha\begin{pmatrix} C_k & 0 \\ 0 & 0 \end{pmatrix}\mathbf{x} + \gamma Q\mathbf{x} \text{ and therefore}$$

$$\frac{\partial f_i^k}{\partial \mathbf{x}_j^m} = \begin{cases} \frac{\alpha(S-1)}{S} + \gamma & \text{if } m=k \text{ and } m\neq \mathscr{I} \\ K_{ij} - \frac{\alpha}{S} + \gamma & \text{if } m\neq k \text{ and } m \neq \mathscr{I} \\ \gamma & \text{if } m = \mathscr{I} \end{cases}$$

The result of a perturbation $\triangle \mathbf{x}_i^{\mathscr{I}}$ (and the corresponding perturbations $\triangle \mathbf{x}_i^{\nu} = \frac{\triangle \mathbf{x}_i^{\mathscr{I}}}{S}, \forall \nu \neq \mathscr{I}$) is a perturbation of the mean field,

$$\begin{aligned}\triangle f_j^{\nu} &= \sum_{n=1}^{N}\sum_{\mu=1}^{S}\frac{\partial f_j^{\nu}}{\partial \mathbf{x}_n^{\mu}}\triangle \mathbf{x}_n^{\mu} \\ &= \sum_{\mu=1,\ \mu\neq\mathscr{I}}^{S+1}\frac{\partial f_j^{\nu}}{\partial \mathbf{x}_j^{\mu}}\triangle \mathbf{x}_j^{\mu} + \frac{\partial f_j^{\nu}}{\partial \mathbf{x}_j^{\mathscr{I}}}\triangle \mathbf{x}_j^{\mathscr{I}} \qquad (23)\end{aligned}$$

and therefore,

$$\begin{aligned}\triangle f_j^{\mathscr{I}} &= \left(-K_{ij} + \frac{\alpha}{S}\right)\triangle \mathbf{x}_i^{\mathscr{I}} \\ \triangle f_j^{\nu} &= -K_{ij}\frac{(S-1)}{S}\triangle \mathbf{x}_i^{\mathscr{I}}\end{aligned} \qquad (24)$$

Suppose $\hat{\mu}$ is such that $\mathbf{x}_i^{\hat{\mu}} = 1$, and $\mathbf{x}_i^{\nu} = 0, \forall \nu \neq \hat{\mu}$. This is assumed to correspond to the initial values of $\mathbf{x}_i^{\mu}$. This is the case in practice because, as a precursor to applying the separator neural network, each vertex will be assigned to a partition, as described previously. Therefore, from (15),

$$\frac{\partial \mathbf{x}_i^k}{\partial f_i^m} = \begin{cases}\frac{1}{T} & \text{if } k = \hat{\mu} \neq m \\ 0 & \text{otherwise}\end{cases}$$

Using this in (23) leads to,

$$\triangle \mathbf{x}_j^{\hat{\mu}} = \sum_{m\neq\hat{\mu},\ m\neq\mathscr{I}}^{S+1}\frac{1}{T}\triangle f_j^m + \frac{1}{T}\triangle f_j^{\mathscr{I}}$$

Substituting (24) into this equation then gives

$$\triangle \mathbf{x}_j^{\hat{\mu}} = \frac{\triangle \mathbf{x}_i^{\mathscr{I}}}{TS}((-S-(S-1)(S-1))K_{ij} + \alpha)$$

The following operations and substitutions are applied to this equation: summing this equation over j; using the corresponding perturbations at node i of $\triangle \mathbf{x}_i^{\hat{\mu}} = -\triangle \mathbf{x}_i^{\mathscr{I}}/S$; assuming the limiting case of no growth in the perturbation, that is $\sum_j \triangle \mathbf{x}_j^{\hat{\mu}} = \triangle \mathbf{x}_i^{\hat{\mu}}$; assuming the load-balancing constraint has no effect, so let $\alpha = 0$. The result is the critical temperature at node i, $\hat{T}_{ci}$ of the separator-finding neural network,

$$\hat{T}_{ci} = (S^2 - S + 1)\sum_j K_{ij}$$

Since this temperature is higher than that obtained for the non-interface finding network (18), the temperature of the separator finding neural network must also be less than that given by (18).

To ensure that the neurons saturate, the initial temperature used with this neural network is,

$$T_0 = \frac{(S-1)\sum_{i=1}^{N}\sum_{j=1}^{N}K_{ij}}{N(S+1)^2}$$

However, since this temperature is lower than that defined in (20), we ensure that the network in the vicinity of the interface remains saturated. The value of γ in (12) is given by $\gamma = \frac{9}{20}\min_{(i,j)\in O}K_{ij}$ where O is the set of all pairs (i,j) such that i is adjacent to j. The value of α is given by (21). For this neural network the temperature above is used as the initial temperature T_0. This is annealed down according to $T = 0.7T$ until all the neurons have become fully saturated.

4 Applications

All of the optimisations in this section were performed on a DEC Alpha workstation in single precision. The normalised neural network explained in Sect. 3 is used, which attempts to minimise the functional given by (9). Normalisation reduces to two the number of parameters that need to be defined, the temperature T of the neural updating (13) and the penalty parameter α which is used with the load-balancing constraint. The performance of the neural network depends on these parameters. The temperature T_i at each node i is set to $0.9T_{ci}$ in order to be sure that the neurons start to saturate. Relatively high temperatures are used, as the resulting partitions are generally of better quality than those obtained at lower temperatures. The critical temperatures T_{ci} and the load-balancing parameter α are found from (20) and (21), respectively. These values of α and T work well with graphs derived from structured and unstructured two-dimensional (2D) finite element meshes.

Figure 1 compares the theoretically predicted critical temperature with the actual critical temperature for the graphs derived from several meshes: the unstructured finite element mesh shown in Fig. 2 and containing 2089 nodes; a 16×16 node structured finite element mesh of four node quadrilateral elements, and a 16×16 node structured finite element mesh of three node triangular elements. Figure 1 demonstrates that the critical temperature is accurately predicted by (20) for a range of graphs with differing valancies, when the temperature

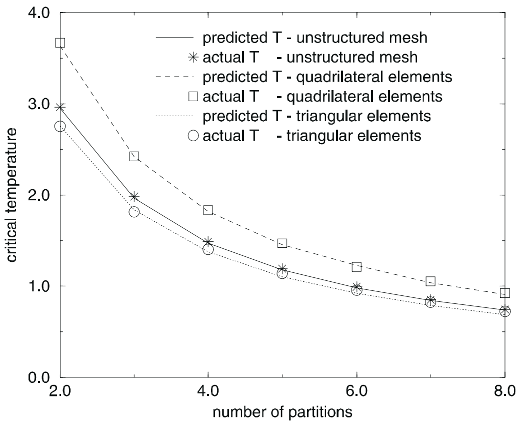

Fig. 1. A comparison of the actual and theoretical critical temperatures variation with number of partitions for three graphs derived from finite element meshes: the unstructured mesh is shown in Fig. 2 and has 2089 nodes (each vertex in the graph is directly connected to about 6 other vertices); a mesh comprising of a square grid of 15×15 quadrilateral elements with 16×16 nodes (each vertex in the graph is directly connected to about 8 other vertices); a structured mesh of triangular elements with 16×16 nodes (each vertex in the graph is directly connected to about 6 other vertices)

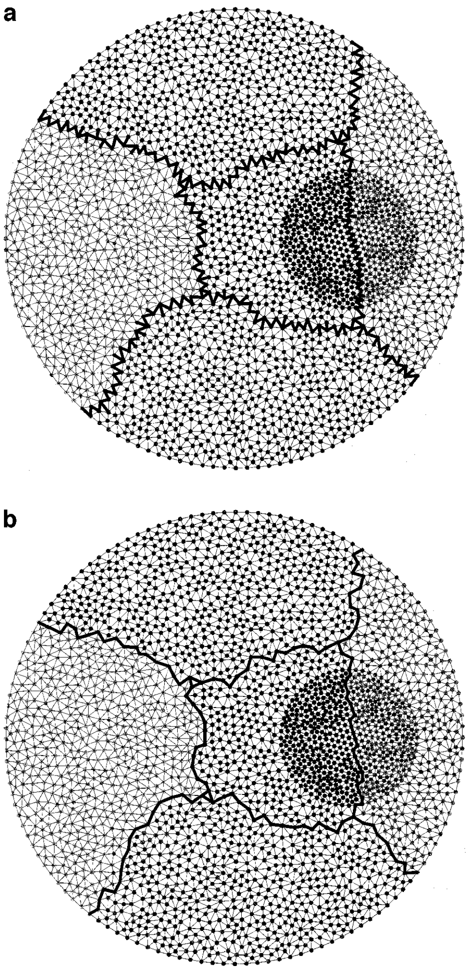

Fig. 2a,b. An unstructured 2089-node finite element mesh is partitioned into 5 using subdomains and a set of separator nodes. **a** Initial configuration for the separator node neural net with 516, 515, 513, 516, 514 nodes in each subdomain. **b** Final partitioning with 494, 486, 478, 495, 489 nodes in each subdomain; 132 separator nodes; 2.06 s spent in optimisation; the chain lengths are 42, 7, 6, 5, 3, 3, 1

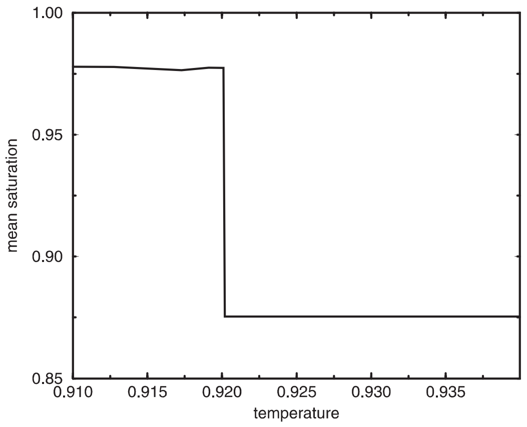

Fig. 3. The mean neuron values for a graph derived from finite element mesh of 15×15 quadrilateral elements (16×16 nodes) against temperature used in the network to split the graph into 8 parts – the predicted critical temperature is 0.908, the actual critical temperature is 0.920

used in the neural network is uniform. Figure 3 shows the mean values of the neurons at convergence for the graph derived from a 16×16 node structured finite element mesh of four node quadrilateral elements, for different values of temperature (uniform temperature). The abrupt change in the saturation of the neurons at the critical temperature is seen clearly. The predicted critical temperature of 0.908, given by (20), is slightly lower than the actual critical temperature of 0.920 because the global critical temperature should be weighted more towards the most common values of T_{ci}, given by (21).

We have found that larger values of T result in less vigorously oscillating convergence than small values of T. For small values of T, the neurons can oscillate wildly, and their values change frequently from 0 to 1, with little time spent at intermediate values. With the larger values of T, some neurons fail to saturate; these neurons are likely to be in the vicinity of the boundaries of the partitions or subdomains, see Fig. 4. Neurons with fewer neighbouring neurons are also likely to be less saturated (see Fig. 4). Thus, if a non-uniform temperature T is to be used, then neurons with fewer links should have a lower temperature. In this case the critical temperature given by (18) could be used.

By annealing the temperature down from near the critical temperature T_c, both a good solution and saturated neurons can be obtained. If annealing is not used and not all the neurons have saturated completely, then we find the maximum value of $\mathbf{x}_i^{\mu}$ in $\{\mathbf{x}_i^{\nu} | \nu \in \{1, 2, \ldots, S\}\}$ and conclude that node i is in partition μ. One could of course use the unsaturated neurons to balance the number of nodes more closely within each subdomain, but we do not do this as this could bias the results.

As with the other decomposition methods, much of the insight into how the network works has been gained by the study of partitioning of graphs derived from finite difference formulations. For a given value of α, as the temperature T is varied, it is discovered that the best quality partition is not necessarily that which minimises the functional. The best partitions are obtained at rela-

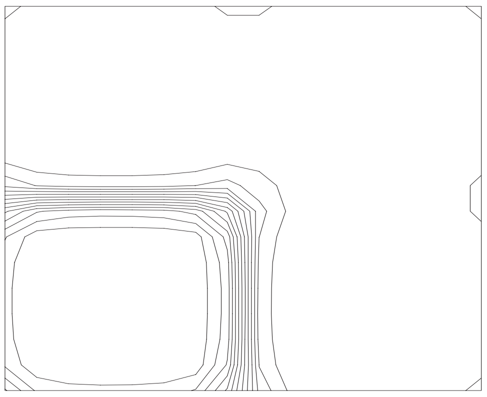

Fig. 4. A 16×16 finite difference grid is partitioned into four by the neural network with $\alpha = 0.0586$, $T = 0.9T_c = 0.844$ – the graph shows the contours of the values of the neurons associated with subdomain one – contours at equal increments

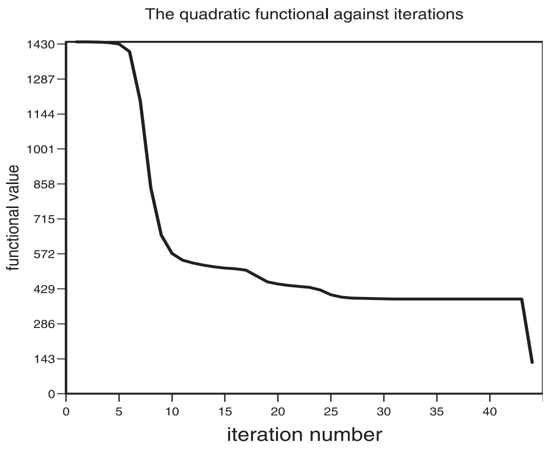

Fig. 5. A 16 × 16 finite difference grid is partitioned into four by the neural network with $\alpha = 0.0586$, $T = 0.9T_c = 0.844$ – graph showing the variation of the functional with the number of iterations – at convergence the value of T is reduced to 0.1 so that the neurons saturate

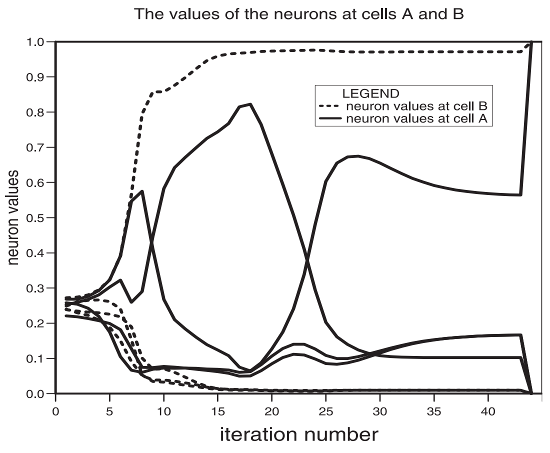

Fig. 7. A 16 × 16 finite difference grid is partitioned into four by the neural network with $\alpha = 0.0586$, $T = 0.9T_c = 0.844$ – graph showing the values of the neurons associated with nodes A and B (see Fig. 1) against number of iterations – at convergence the value of T is reduced to 0.1 so that the neurons saturate

tively high values of T, in which case the functional may be large because not all the neurons are saturated, see Fig. 5.

Convergence is assumed when the change in functional from two consecutive iterations is less than 0.0001. This can lead to problems for some values of α and T as the network will never converge; the network then seems unable to decide on a partition and oscillates between partitions.

In Fig. 6 the values of the two sets of four neurons associated with cells A and B (see Fig. 7) are plotted against the number of iterations. The neurons associated with cell A are amongst the most saturated in the network, and those associated with cell B are amongst the least saturated. The neurons at cell B are relatively unsaturated because the optimisation was performed at a high temperature of $0.9T_c$.

The neural network that minimises the functional 12 and searches for a set of separator nodes has, in our computer programs, a default initial temperature of $T_i = (1/2(S+1)) \sum_{j=1}^{N} K_{ij}$ for each node i. At convergence the temperature is annealed down according to $T_i = (7/10)T_i$. The load-balancing parameter α is set to $\alpha = (S/2N^2) \sum_{i,j=1}^{N} K_{ij}$, and γ in (12) is set to 0.45. Annealing and non-uniform temperatures are used with this neural network as it conducts a difficult search.

Figure 2a shows the initial configuration or partitioning (found using a neural network) used with the separator finding neural network; the results are shown in Fig. 2b. It is seen in Fig. 2b that the separator nodes are situated roughly at the boundaries of the subdomains of the initial configuration. This allows the separator node finding neural network to converge quickly. Figure 8 shows a decomposition of a structured 50 × 50 node finite element mesh decomposed into five with a set of separator nodes found using the separator node neural network. The resulting partition has a pleasing symmetry.

Figure 9 shows a decomposition of a finite element mesh of a 3D domain. The mesh comprises 46432 elements and 32145 nodes and is decomposed into 32 parts

2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
1	1	1	1	1	1	1	1*A*	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3
1	1	1	1*B*	1	1	1	1	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3

Fig. 6. A 16 × 16 finite difference grid partitioned into four by the neural network – optimal partition – nodes A and B are highlighted

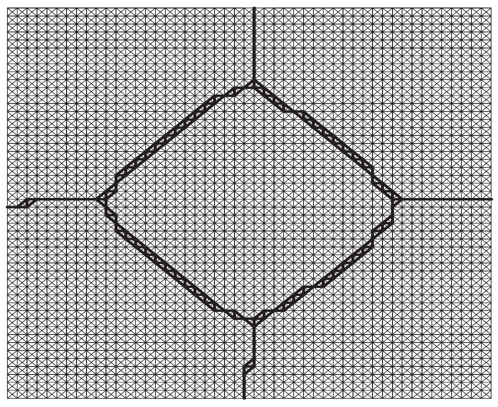

Fig. 8. Graph of structured 50 × 50 node finite element mesh of four node quadrilateral elements decomposed into five subdomains and separator nodes – *bold lines* are used to join the separator nodes. The number of nodes in each subdomain is 463, 472, 468, 467, 472; number of separator nodes is 158; time spent in separator network is 0.41 s; the chain lengths are 5, 4, 3, 1, respectively

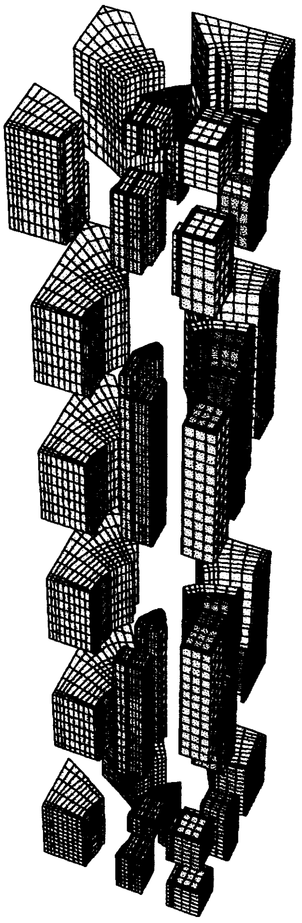

Fig. 9. An exploded view of a finite element mesh which has been partitioned into 32 parts with interface nodes – the number of interface nodes is 6722, and the maximum and minimum number of nodes in a partition are 850 and 708, respectively. The total number of nodes is 32145 – the mesh was partitioned using recursive bi-section. The surfaces of elements containing interface nodes and those that are on the surface of the domain are shown in the diagram

with interface nodes separating the parts. The mesh was decomposed using a recursive bi-section approach. This demonstrates the use of the separator neural network in practical problems.

5 Conclusions

In this paper, expressions for the weights of a k-way separator-finding neural network were derived analytically for the first time. The recall mechanism of the mean field theorem neural network is used to perform the associated optimisation, and expressions for the critical temperature and load-balancing penalty parameter were derived. These expressions can help us to understand and extract a good performance from the neural network methods.

To extract the best performance from the neural network method that finds a set of separator nodes, it is best to decompose the domain using another technique. Spectral bi-section is commonly used – we have used another neural network in this paper. Once nodes have been assigned to subdomains, the separator neural network can be applied to find the separator nodes. This means that we are not heavily reliant on the separator neural network's optimisation abilities, which can be lacking.

The dynamics of the separator finding neural network are particularly interesting and worthy of further investigations.

Acknowledgements. We would like to thank the anonymous reviewers for their helpful comments and the Engineering and Physical Sciences Research Council, UK, for funding this work through grant GR/K73466.

References

1. Bahreininejad A, Topping BHV, Khan AI (1996) Finite element mesh partitioning using neural networks. Adv Eng Software 27:103–115
2. Bilbro G, Mann R, Miller T, Snyder W, Van den Bout D, White M (1989) Mean field annealing and neural networks. (Advances in neural information processing) Morgan-Kaufmann, New York, pp 91–98
3. Birnbaum RS, Oliveirs CRE de, Goddard AJH (1994) A parallel cholesky factorisation method based on nested dissection for the solution of radiation transport problems. In: Valero M, et al. (eds) Parallel computing and transputer applications. Barcelona
4. Van den Bout D, Miller T (1989) Improving the performance of the Hopfield-Tank neural network with normalization and annealing. Biol Cybern 62:129–139
5. Van den Bout DE, Miller TK (1990) Graph partitioning Using Annealed Neural Networks. IEE Transactions on Neural Networks 1:192–203
6. Chen W, Stallmann MFM, Gehringer EF (1989) Hypercube Embedding heuristics: an evaluation. Int J Paral Prog 18:505–549
7. Hertz J, Krogh A, Palmer RG (1990) Introduction to the theory of neural computation. Addison Wesley, New York
8. Hirsch C (1999) Numerical computation of internal and external flows, Vol 1. Wiley, New York
9. Hopfield JJ (1984) Neurons with graded response have collective computational properties like those of two state neurons. Proc Natl Acad Sci USA 81:3088–3092
10. Hopfield JJ, Tank DW (1985) Neural computations of decisions in optimization problems. Biol Cybern 52:141–152
11. Karypis G, Gupta A, Kumar V (1994) A parallel formulation of interior point algorithms. Supercomputing
12. Karypis G, Kumar V (1995) A fast and high quality multilevel scheme for partitioning irregular graphs. Int Conf Parallel Processing
13. Keyes DE, Gropp WG (1987) A comparison of domain decomposition techniques for elliptical partial differential equations and their parallel implementations. SIAM J Stat Comp 8:166–202
14. Peterson C, Anderson J (1988) Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem. Complex Systems 2:59–89
15. Pineda FJ (1988) Generalization of backpropagation to recurrent and higher order neural networks. Am Inst Physics, pp 602–610
16. Pothen A, Simon HD, Liou K (1990) Partitioning sparse matricies with eigenvectors of graphs. SIAM J Matrix Anal Appl 11:430–452
17. Walshaw C, Cross M, Everett M (1995) A localised algorithm for optimising unstructured mesh partitions. Int J Supercomputer Appl 9:280–295