INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng. **44**, 593–613 (1999)

A NEURAL NETWORK GRAPH PARTITIONING PROCEDURE FOR GRID-BASED DOMAIN DECOMPOSITION

C. C. PAIN*, C. R. E. DE OLIVEIRA AND A. J. H. GODDARD

Applied Modelling and Computation Group, Centre for Environmental Technology, Imperial College of Science, Technology and Medicine, Prince Consort Rd, London SW7 2BP, U.K.

SUMMARY

This paper describes a neural network graph partitioning algorithm which partitions unstructured finite element/volume meshes as a precursor to a parallel domain decomposition solution method. The algorithm works by first constructing a coarse graph approximation using an automatic graph coarsening method. The coarse graph is partitioned and the results are interpolated onto the original graph to initialize an optimization of the graph partition problem. In practice, a hierarchy of (usually more than two) graphs are used to help obtain the final graph partition. A mean field theorem neural network is used to perform all partition optimization. The partitioning method is applied to graphs derived from unstructured finite element meshes and in this context it can be viewed as a multi-grid partitioning method. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: graph partitioning; mean field theorem; neural network; multi-grid; domain decomposition; unstructured finite element

1. INTRODUCTION

Domain Decomposition Methods (DDM) are often exploited to solve in parallel sets of simultaneous equations formed with the application of a Finite Element Method (FEM) or other grid-based discretization. To use a DDM the mesh must first be partitioned into a number (usually equal to the number of processors) of 'balanced' subdomains such that the interprocessor communication is minimized. A simple method of balancing subdomains is to ensure that the number of nodes of a FEM mesh, in each subdomain, is approximately equal. To minimize the interprocessor communication the number of edges of the graph associated with the FEM mesh between the subdomains is minimized, subject to the complementary constraint of balancing the load. Our approach to this partitioning problem is to use optimization techniques applied to a functional which gauges the quality of the partition and can contain a great deal of partitioning information. The overall aim of producing an efficient DDM solution method is achieved, in part, through the information injected into the functional.

The resulting graph partitioning problem is NP complete. Fortunately, it is not necessary to find an optimal solution as a good quality sub-optimal partition is usually adequate. The recall

* Correspondence to: C. C. Pain, Applied Modelling and Computation Group, Centre for Environmental Technology, Imperial College of Science, Technology and Medicine, Prince Consort Road, London SW 7 2BP, U.K. E-mail: c.pain@ic.ac.uk

CCC 0029–5981/99/050593–21$17.50

Received 12 February 1997
Revised 23 April 1998

mechanism of the Mean Field Theorem (MFT) neural network (used here to optimize the partitions) has the advantage that much is known about its behaviour, but can, for large graphs, produce poor partitionings. This is because global information does not travel well enough across the architecture of the neural network. It is shown here how to improve this situation with the use of a graph hierarchy or multi-grid approach. The coarse graph-mesh levels will be obtained using an automatic coarsening technique suitable for unstructured meshes. Graph coarsening algorithms have been used before[1] to improve the quality of graph partitioning.

A more powerful optimization engine than the neural network (such as gradient annealing[2]) can be used with this graph hierarchy method in order to improve the final partition, possibly to obtain an optimal one. However, for our purposes the MFT network is more than adequate and provides a pleasing compromise between efficiency and partition quality. This partitcular neural network has been used before to partition graphs,[3, 4] but with a different method of obtaining the critical temperature and load balancing parameter to that presented here. In Reference 5 a MFT neural network was applied using nested bi-section and empirically obtained parameters, to partition finite element meshes. They enhanced the spread of partition information across the network using a series of increasingly finer meshes obtained with solution adaptivity. The current paper generalizes the recursive bisection approach and can be used with or without solution adaptivity.

A great deal of effort has been invested in studying the graph partitioning problem and as a result a large number of successful partitioning methods have been developed, these include: methods based on heuristic local searches most notably the Farhat's Greedy,[6, 7] and Lin-Kernighan heuristics;[8] methods based on geometric partitioning [9–11] such as some graph growing algorithms; neural networks[3, 4, 12–16] and spectral bisection.[17, 18] Spectral bisection has established a reputation for producing high-quality partitionings, but since it requires the calculation of the (Fiedler vector) eigenvector associated with the second largest eigenvalue of a Laplacian matrix it is computationally expensive. Genetic algorithms[19] and simulated annealing methods,[20] although robust optimization methods, have not as yet proven valuable tools for decomposing graphs.

However, there seems to be a consensus that partitioning methods that use multi-level graph partitioning[1, 21] are the most computationally efficient. In these methods coarse graph representations of finer graphs are typically obtained by vertex collapsing.[1] Partitions on the coarsest graph level are quickly obtained even when relatively expensive partitioning methods are used. These partitions are then mapped onto a finer mesh after which typically local searches (e.g Greedy, Lin-Kernighan heuristics) are performed to improve the partition quality, before proceeding to map the resulting partition onto the next finest graph and so on until the target graph is partitioned. These are typical options used in partitioning codes such as JOSTLE,[22] METIS[21] and RALPAR.[9] CHAGO[23] is another notable code based on spectral bi-section (although METIS and RALPAR include this as an option). Much work is being performed on harnessing the power of parallel processing to partition graphs using computer codes like these.[24]

We present the MFT neural network as a local search method for finding or improving graph partitionings. It provides a theoretical foundation so that efficient partitioning methods can be derived (including non-neural network partitioning methods). Our computational implementation of this method is geared towards partitioning graphs into a small number of subdomains and as such must be used with nested-dissection to partition graphs a large number of parts.

The remainder of the paper is organised as follows: in the next section the cost function to be optimized is derived; Section 3 shows how a multi-grid-graph approach can be used to aid the optimization; this is followed by the automatic graph coarsening algorithm. Section 5 introduces the neural network with analysis that attempts to extract optimal performance from it and Section 6

covers the applications. Section 7 summarises the advantages and disadvantages of the neural network partitioning method and conclusions are drawn in Section 8.

2. COST FUNCTIONS

The function to be optimized, which gauges the quality of the partition, should ideally be a curve fit found from numerous numerical experiments. However this is not practical because there are too many permutations to take into account, including the hardware (how the communication is organised, etc.), the iterative DDM solver, the shape of the subdomains, the coefficients in the governing equation(s) and the source terms.

There are two ways in which a finite element mesh can be subdivided. One is to assign elements to processors or partitions and the other is to assign nodes. Assigning elements is perhaps the natural approach if a set of interface nodes between the subdomains is sought, e.g. Schur complement methods.[25] Element based partitioning methods are derived by using a graph which describes the communication between elements as opposed to nodes, each vertex in the graph representing an element. Methods that do not use interface nodes (e.g. Block Explicit), but use a halo, would be more suited to node assigning methods.

A partitioning without interface 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 that is K_{ij}. Typically,

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

where $R_i = \sum_j w_{ij}$. The scalars w_{ij} represents the communication cost of node i communicating with node j, when they are adjacent. In the applications presented here (Section 6), $w_{ij} = 1$. The nodes are adjacent if they are adjacent in the graph associated with the mesh or grid. The term $S_p R_i$ in the matrix K is added for numerical convenience when using some continuous optimization methods. This is similar to the selection pressure $S_p \in [-1, 1]$ used in Reference 26. If $S_p < 0$ then the nodes are discouraged, in the optimization, from deciding which subdomain they will belong to. (at $S_p = -1, K$ becomes a diffusion like matrix) and if $S_p > 0$ then the nodes are encouraged to decide which subdomain they will belong to. Some iterative methods may benefit from gradually increasing S_p. However, we choose to use $S_p = 0$.

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

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

When continuous optimization methods are used then $x_i^\mu \in [0, 1]$ becomes the probability of node i being in partition μ. The probability constraint is thus

$$\sum_{\mu=1}^{S} x_i^\mu = 1 \tag{2}$$

Putting the probability constraint aside for the moment, an example of a functional to be optimized is

$$F = F_1 + F_2$$

This functional F is to be maximized or minimized, depending on the choice of F_1 and F_2, to produce the desired partitioning. Optimizing the term involving F_1 should minimize interpartition communication, while optimizing F_2 should balance the number of nodes in, or load associated with, each partition.

The function $x^{\mu\mathrm{T}}Kx^{\nu}$ is an expression of the amount of communication between partitions μ and ν and thus minimization of $x^{\mu\mathrm{T}}Kx^{\nu}$ minimizes the communication between partitions μ and ν; conversely maximization of $x^{\mu\mathrm{T}}Kx^{\nu}$ maximizes the communication between partitions μ and ν.

To construct a functional to be minimized, a function that expresses the amount of communication between partitions is used. To this end a matrix $\mathbf{K_f}$ is defined with submatrices $\mathbf{K_{fij}}$ given by

$$\mathbf{K_{fij}} = \begin{cases} 0 & \text{if } i = j \\ K & \text{otherwise} \end{cases}$$

and F_1 is then

$$F_1 = \tfrac{1}{2}x^{\mathrm{T}}\mathbf{K_f}x = \tfrac{1}{2}x^{\mathrm{T}}\mathbf{K_d}(b - x)$$

Due to the probability constraint $\mathbf{K_f}x = \mathbf{K_d}(b - x)$, where b is a vector of length $N \times S$ with scalar entries of unity.

If maximizing

$$F_2^{\max} = \alpha\tfrac{1}{2}x^{\mathrm{T}}\mathbf{C_f}x \tag{3}$$

balances the number of nodes in each subdomain, a candidate for the matrix $\mathbf{C_f}$ comprises of $S \times S$ square submatrices $\mathbf{C_{fij}}$ is given by

$$\mathbf{C_{fij}} = \begin{cases} 0 & \text{if } i = j \\ B & \text{if } i \neq j \end{cases}$$

where B and 0 are $N \times N$ matrices with all their entries equal to unity and zero, respectively.

Therefore minimizing $F_2 = -F_2^{\max} = -\alpha x^{\mathrm{T}}\mathbf{C_f}x$ balances the number of nodes in each partition, where $F_2^{\max}$ is given by equation (3). The scalar α gives a priority to the resulting functional for balancing the number of nodes in each subdomain relative to minimizing the number of edges or communication between subdomains. Taking the probability constraint (2) into consideration it is easy to see that

$$F_2 = -\tfrac{1}{2}\alpha x^{\mathrm{T}}\mathbf{C_f}x = \tfrac{1}{2}\alpha x^{\mathrm{T}}\mathbf{C_d}(x - b)$$

where the submatrices $\mathbf{C_{dij}}$ of $\mathbf{C_d}$ are defined by

$$\mathbf{C_{dij}} = \begin{cases} B & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

A minimum of $F_2 = \tfrac{1}{2}\alpha x^{\mathrm{T}}\mathbf{C_k}x$ balances the number of nodes in each partition with the matrix $\mathbf{C_k}$ defined by its $S \times S$ submatrices $\mathbf{C_{kij}}$:

$$\mathbf{C_{kij}} = \begin{cases} \dfrac{S-1}{S}B & \text{if } i = j \\ -\dfrac{1}{S}B & \text{if } i \neq j \end{cases}$$

It does this because it is a statment of diffusion, that is it diffuses (makes the same) the number of vertices in each partition, much like a discretization of the diffusion operator. Furthermore,

$N\mathbf{C_k}\mathbf{C_k} = \mathbf{C_k}$ and thus $\mathbf{C_k}$ is positive semi-definite and so $x^{\mathrm{T}}\mathbf{C_k}x$ is never negative. Again, by taking advantage of the probability constraint, an equivalent expression can be found, thus

$$F_2 = \tfrac{1}{2}\alpha x^{\mathrm{T}}\mathbf{C_k}x = \tfrac{1}{2}\alpha x^{\mathrm{T}}\left(\mathbf{C_d}x - \frac{N}{S}b\right)$$

Putting these ideas together, a useful functional to be minimized is

$$F = F_1 + F_2 = \tfrac{1}{2}x^{\mathrm{T}}\mathbf{K_f}x + \tfrac{1}{2}\alpha x^{\mathrm{T}}\mathbf{C_k}x \tag{4}$$

or equivalently

$$F = \tfrac{1}{2}x^{\mathrm{T}}\mathbf{K_d}(b - x) + \tfrac{1}{2}\alpha x^{\mathrm{T}}\left(\mathbf{C_d}x - \frac{N}{S}b\right) \tag{5}$$

subject to the probability constraint (2). If two subdomains are required, half of the vector x, that is x^2, can be removed along with the probability constraint. Then, when $x_i^1 = 1$, node i belongs to subdomain 1 and, when $x_i^1 = 0$, node i belongs to subdomain 2. The penalty term F_2 used as described above to constrain the problem tends to compete with, or work against, optimization of F_1. Thus, it can be of advantage to keep α small.

For the benefit of an iterative DDM solver it may be advisable to discourage the partition from crossing areas of discontinuous coefficients in the governing equations or expected high activity. This can be achieved by setting the cost k_{ij} of communicating between two adjacent nodes i and j in these regions to a high value. A second related problem is that some areas of the domain may have poorer conditioning or greater activity than in others. In these areas it may be advisable to form subdomains containing less nodes than the average so that an iterative solution within each subdomain will take approximately the same CPU time. The functional (4) can distinguish between these two problems.

In order to assign weights u_i to the nodes and force the optimization to balance the sum of the weights in each subdomain, the matrix $\mathbf{C_k}$ in equation (4) can be replaced with the matrix $\hat{\mathbf{C}}$ defined by $\hat{\mathbf{C}} = \mathbf{D}\mathbf{C_k}\mathbf{D}$ in which the diagonal matrix $\mathbf{D}$ has entries $d_{ii\mu\mu} = u_i, \forall\mu, i$. The requirment of having more nodes (or sum of u_i) in some domains than in others can also be achieved by pre and post multiplying $\mathbf{C_k}$ (or even $\hat{\mathbf{C}}$) by a diagonal matrix $\mathbf{D}$ with diagonal entries $d_{ii\mu\mu} = v_\mu, \forall\mu, i$. Increasing v_μ decreases the number of nodes in subdomain μ, so, for example, if $S = 2$ and $v_1 = 2, v_2 = 1$ then there would be twice as many nodes in subdomain two as in one. Thus in general the diagonal matrix $\mathbf{D}$ can be defined by $d_{ii\mu\mu} = u_i v_\mu, \forall\mu, i$.

By introducing higher-order information into the matrix K some of the local minima or maxima in F can be removed. This can make the optimization of F using an iterative method such as the neural network approaches described here less demanding (see also Reference 26).

For example,

$$K_{ij} = \begin{cases} w_{ij} & \text{if the shortest path length between } i \text{ and } j \text{ is less than or equal to } R \\ 0 & \text{otherwise} \end{cases}$$

where w_{ij} is a measure of the cost of communication between distinct nodes i and j. An approximate measure of this is $w_{ij} = R-$ (path length between nodes i and j). The path length between nodes i and j would equal the minimum number of edges, in the associated graph, that need to be traversed in order to travel from node or vertex i to j. Ultimately, if R was large enough, K would be a fully populated matrix. Thus, the following can be used

$$K_{ij} = R - d_{ij}$$

where d_{ij} is a measure of the distance from node i to node j and R is the maximum value of d_{ij}. For efficient coding and with a loss of rigour the distance measures $d_{ij} = \mid x_i - x_j \mid + \mid y_i - y_j \mid$, or $d_{ij} = \left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2$, can be used. The matrix K could then take the form $K_{ij} = d_{ij}$. The x, y co-ordinates can be normalised, so that no excessively large numbers are generated. As pointed out in Reference 17, the eigen vector of the second largest eigen value of the diffusion like matrix, defined by equation (1) with $S_p = -1$, contains distance information. The d_{ij} may therefore be formed by, for example, $d_{ij} = |v_i - v_j|$ where v_i and v_j are the i and j'th entries in this particular eigen vector, respectively.

3. THE USE OF A GRAPH HIERARCHY

A technique analogous to multi-grid methods of solution of simultaneous equations can be used to improve the quality of the partition optimization. This can be achieved by grouping together small numbers of adjacent nodes to form coarse grid nodes. If this is done across the entire grid a coarse grid results. The edge weight between two coarse grid nodes might then take on the value of the sum of the weights of all the edges runing on the fine grid between the two coarse grid nodes. The optimization would then have a smaller search space, and thus the quality of the optimization that can be performed will often be better than on the fine grid. The partition on the coarse grid is then mapped onto the finer grid and the optimization on the finer grid is initialised with the resulting partition. A minnor generalization of the above grid coarsening method will be used here so that some fine graph vertices can be represented by more than one coarse graph vertex. This is analogous to multi-grid solution techniques.[27]

Once a sequence of coarsened graphs have been obtained these can be used to help find the partition of the largest graph. This is achieved by first partitioning the coarsest graph then smoothing the resulting partition onto the next graph level to initialize the optimization on that level. That is $x^{\mu}_{\text{fine}} = P x^{\mu}_{\text{coarse}}, \forall \mu$, in which the prolongation operator P is defined such that a fine graph neuron value $x^{\mu}_{i\ \text{fine}}$, for a given subdomain μ and associated with a given node i, is the mean of the coarse graph vertex neuron values for subdomain μ that are connected to (have an edge between) them and vertex i. For simplicity, this is the method we will use, but, the optimization algorithm might be improved by periodically refering back to the coarser graphs. We briefly discuss two ways of doing this before continuing.

The first alterative is the use of a higher-order communication matrix which is defined as

$$K = \sum_{m=1}^{g} \lambda_m K_m$$

and K_m is the communication matrix associated with the graph on the *mth* level of coarsening, but sumperimposed onto the original graph. Thus supposing that two coarse graph vertices on the fine graph at coarsening level m superimposed onto the fine graph, communicate if there is an edge between them on graph coarsening level m. Here $m = 1$ is the finnest graph level (original graph) and $\lambda_m = 1 \geqslant \lambda_{m-1} \geqslant \cdots \geqslant \lambda_g$ in which g is the number of graph levels. It is suggested that $\lambda_m = 1, \forall m$ might be used. The local communication matrix K_m only involves communication between vertices that exist on graph level m.

The second alternative is to introduce coarse graph vertices into the functional. For example, with one level of coarsening the vector of unknowns is $\hat{x} = \begin{pmatrix} x_{\text{coarse}} \\ x_{\text{fine}} \end{pmatrix}$ in which x_{coarse} and x_{fine}

are the neurons associated with the coarse and fine graphs, respectively. The functional for this example now becomes

$$F = \tfrac{1}{2}\hat{x}^{\mathrm{T}}\hat{A}\hat{x}$$

where

$$\hat{A} = \begin{pmatrix} \begin{pmatrix} K_{\mathrm{c}} & 0 \\ 0 & K_{\mathrm{c}} \end{pmatrix} & 0 \\ 0 & \begin{pmatrix} K & 0 \\ 0 & K \end{pmatrix} \end{pmatrix} + \begin{pmatrix} \alpha_{\mathrm{c}}\frac{1}{2}\begin{pmatrix} B_{\mathrm{c}} & -B_{\mathrm{c}} \\ -B_{\mathrm{c}} & B_{\mathrm{c}} \end{pmatrix} & 0 \\ 0 & \alpha\frac{1}{2}\begin{pmatrix} B & -B \\ -B & B \end{pmatrix} \end{pmatrix}$$

$$+\lambda \begin{pmatrix} 0 & \begin{pmatrix} P^{\mathrm{T}} & 0 \\ 0 & P^{\mathrm{T}} \end{pmatrix} \\ \begin{pmatrix} P & 0 \\ 0 & P \end{pmatrix} & 0 \end{pmatrix}$$

The last term of this matrix couples the coarse and fine graphs and thus forces the optimization of the fine graph partitioning to consider coarse graph information. In this functional the prolongation matrix P is defined as above and the scalar λ must be chosen such that the optimization of the resulting functional produces a good quality fine graph partition. It is suggested that $\lambda = 1$ might be used. The main disadvantage of this approach is that an optimal partition of the functional defined in the previous Section 2 equation (4), whose optimization is ultimately sought, may not be an optimal partition using the functional above.

4. AUTOMATIC GRAPH COARSENING

The strategy of the graph coarsening exercise is to choose coarse graph vertices from the graph to be coarsened, such that the coarse graph vertices are evenly distributed throughtout the original graph. Their position and connectivity (represented by the edges between the coarse graph vertices) define a graph which is a good coarse representation of the original graph.

In this paper, the coarse graph vertices are found with the aid of a graph colouring technique in which the coarse vertices are chosen as those vertices that are coloured with the base colour. The base colour will be a colour that is used as much as possible in the colouring of the graph, (see colouring algorithms latter on in this section). To obtain the base colour thus requires a solution to the maximal independent set problem.[28] A colouring algorithm typically works by repeated application of a maximum independent set algorithm. Conversely, a maximum independent set can often be found from a vertex colouring method.

4.1. Graph colouring and discretization

If the finite element equations are discretised using a 7-point discretization on a regular triangular grid, as shown in Figure 1(a), then three colours are sufficient to colour the graph and uncouple all the equations. A suitable grid colouring is also shown in Figure 1(a). If the equations are discretized using a 9-point discretization on a regular grid, as shown in Figure 1(b), then four colours, as shown in 1(b), are sufficient to uncouple all the equations. Such a discretization can be realized using the four-node quadrilateral element.

For regular grids it is usually apparent how to colour the grid so that equations centred on the same colour are uncoupled. It is rather less obvious how to colour most unstructured grids.

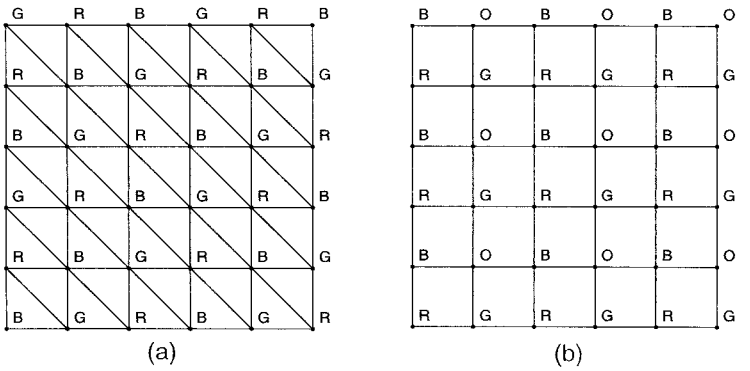

Figure 1. Graph colouring of regular finite element meshes. (a) Colouring of a regular finite element mesh of three node triangular elements—three colours. (b) Colouring of a regular finite element mesh of four node quadrilateral elements—four colours

However, one does not need to minimize the number of colours, although it would often be preferable, and even a fairly unsophisticated colouring algorithm might be good enough to colour an irregular grid.

4.2. Graph colouring algorithms

The method we recommended to colour a graph G, say, is non-iterative, but often produces a colouring with the minimum number of colours or very close to that. For completeness a neural network method is also included, in this section, as this can reduce the number of colours over the heuristic approach albeit at a greater CPU cost.

4.2.1. A heuristic procedure for graph colouring. In this graph colouring method the graph colouring of the mesh will be grown from a vertex. This is achieved by colouring a vertex of the graph G with minimum valency using the base colour 1, where each colour has a unique number associated with it, from one to the number of colours used. All the vertices connected to (have an edge directed from) this root vertex and the edges between these vertices form a sub-graph. The vertices in this subgraph are then coloured. Then the vertices that are connected to this subgraph, that have not been coloured, form another subgraph. These vertices are coloured and the whole process is repeated until all the vertices are coloured.

To colour the sub-graph the uncoloured vertex in the subgraph with the maximum number of coloured neighbouring vertices, in the sub-graph, is chosen for colouring. If more than one vertex has the same number of coloured neighbouring vertices, then from these vertices, the vertex with the minimum valency in the sub-graph is chosen. A vertex is coloured by looking at all the vertices in G that it is connected to, which have already been coloured, and choosing a different colour. If a colour that has already been used can not be used because it will violate the graph colouring constraint (using a colour already used is preferable) then an additional colour is introduced. Another uncoloured vertex is chosen from the subgraph and the process is repeated until all verticies in the subgraph are coloured. Each colour is assigned a number from one to the number

of colours used and colours with the lowest number are used when ever possible in the colouring process.

Once a colouring has been obtained in this manner, it would be a simple matter to balance the number of vertices of a given colour (often desirable in parallel and vector processing applications).

The algorithm is expected to colour the graph with optimal or near optimal number of colours. The four colour theorem gives an indication of the number of colours needed, in this case approximately four. If the graph constructed from a 2-D finite element is itself 2-D i.e. comprising only of triangles, then the four colour theorem is applicable. In 3-D for a graph with edges that form the outline of tetrahedrals, intuitive thinking suggests 8. Other colouring methods may be derived from more general optimization methods an efficient example of which is a neural network.

4.2.2. Graph colouring functional for the MFT neural network. The colouring method described here is intended for symmetrically connected graphs, for example graphs derived from FE or FD grids. It may be applicable to non-symmetric graphs, but the lack of symmetry must be taken into account when differentiating the functional. Alternatively, if the functional $F = \frac{1}{2}x^{\mathrm{T}}Ax + u^{\mathrm{T}}x$ is to be minimized, where A is non-symmetric, the problem could be symmetrized by minimizing instead $F = \frac{1}{2}x^{\mathrm{T}}A^{\mathrm{T}}Ax + A^{\mathrm{T}}u^{\mathrm{T}}x$.

To be more specific the functional that will be minimized is of the form,

$$F = \gamma x^{\mathrm{T}}\mathbf{K_f}x + \sum_{\mu=1}^{M} \mu\, x^{\mu\mathrm{T}}b$$

where b is a vector with entries equal to unity, γ is a number greater than M (e.g. $\gamma = M + 1$) and the term involving γ discourages vertices of the same colour being neighbours in the graph, M is the maximum number of colours expected.

The communication matrix K is defined by its entries k_{ij} where

$$k_{ij} = \begin{cases} 1 & \text{if } i \text{ is connected to } j \\ 0 & \text{otherwise} \end{cases}$$

The matrix $\mathbf{K_f}$ is a $M \times M$ block matrix with block entries $\mathbf{K_f}_{ij}$ given by

$$\mathbf{K_f}_{ij} = \begin{cases} K & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

The 0 in this equation represents an $N \times N$ zero matrix, where N is the number of vertices in the graph G. The entries x_i^{μ} in the matrix x are such that

$$x_i^{\mu} = \begin{cases} 1 & \text{if node } i \text{ has colour } \mu \\ 0 & \text{otherwise} \end{cases}$$

The normalized neural network can be applied to minimize this functional, (see Section 5). Normalization ensures that $\sum_{\mu=1}^{M} x_i^{\mu} = 1$.

If it is found that M is not sufficiently large, so the graph is coloured with M colours and the colouring constraint is violated (there exists two neighbouring vertices coloured with the same colour), then the optimization is performed again with 1 added to M, perhaps using the configuration already obtained, to initialize it. However, $M = 2$ is sufficient to solve the maximum independent set problem.

4.3. *Defining the coarse graph from the base colour*

The vertices of the coarse graph have now been defined as the vertices on the fine graph of the base colour. It remains to define the direct links or edges between the coarse graph vertices. To this end let L_{ij}^{f} be the minimum number of edges to be traversed in the fine graph between the two vertices i and j. There exists an edge in the coarse graph between two coarse grid nodes if the corresponding fine grid nodes i and j are such that $L_{ij}^{\mathrm{f}} = 2$.

5. NEURAL NETWORKS

A simulated electrical circuit or neural network can be used to solve general optimization problems.[29] Traditionally, a neural network is a directed-graph whose vertices or nodes act on data passed to them from other nodes. Input nodes can be used as output nodes. However, information can only travel in the way that the directed-graph suggests. Each directed edge has a weight associated with it that governs the overall importance of the information travelling along it.

To solve many optimization problems, including the partitioning problem, it is not necessary to train the net, that is iteratively obtain the weights, as expressions can be derived for the weights. For the partitioning problem, the architecture of the neural net can be based on the structure of the mesh with S layers and N neurons per layer, and with at least as many links or edges within each layer as the graph to be partitioned. The graph has N vertices and is to have S partitions.

The equation of motion characterising the time evolution of a circuit is

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} \approx \frac{u_i' - u_i}{\triangle t} = -u_i + \sum_{j=1}^{N} w_{ij} x_j + z_i \tag{6}$$

where the scalars z_i contain a source term for each neuron, $x_i = g(u_i)$, $u_i = u_i(t), u_i' = u_i(t+1)$ and t is the time level. Updating the variables u_i^{μ} one at a time (asynchronous iteration) will result in improved convergence. The iteration takes the form of (6) with appropriately chosen $\triangle t$. For many problems, including the partitioning problem, it is convenient to partition x so that $x = (\, x^{1^{\mathrm{T}}} \quad x^{2^{\mathrm{T}}} \quad \cdots \quad x^{S^{\mathrm{T}}} \,)$ (see Section 2) where the vector x^{μ} contains the values of the neurons in layer μ. If the connections are symmetric, that is $w_{ij} = w_{ji}$ the stable states of this equation are the local minima of the quantity,

$$F = -\frac{1}{2} x^{\mathrm{T}} W x - x^{\mathrm{T}} z + \sum_i \int^{x_i} g^{-1}(y)\, \mathrm{d}y \tag{7}$$

where $g(u_i)$ is the output voltage of neuron i, due to an input voltage u_i. This F was derived in Hopfield's.[13, 14] In the high gain limit (when the width of the gain in g is narrow) the last term disappears. Suitable choices for the weights w_{ij} in K and z are found in order that the minimization of F optimizes a given problem. The vector x will contain the resulting neuron values, while the vector z contains a source term for each neuron. Candidates F for the partitioning problem are given in Section 2.

Sigmoid functions are frequently used for $g(\xi)$; some examples are:

$$g(\xi) = \frac{1}{2}\left(1 + \tanh\left(\frac{\xi}{c}\right)\right), \quad g(\xi) = \frac{1}{1 + \mathrm{e}^{-\xi/c}}$$

The scalar $1/c$ is the gain in $g(\xi)$ and is analogous to selection pressure[26]—too large a value of c (low gain) will result in illegitimate solutions (the converged solution may not be at the hypercube corners) and too small a value of c (high gain) will result in poor solutions.

A memory can be stored in the net with appropriate w_{ij} and z. In the symmetric W case F, see (7) above, determines the stable states of the system. If the stable states of the system are a particular set of memory states, W and z must be chosen so that F is the local minima when the system is in one of those states.

The Mean Field Theory (MFT) neural updating procedure can be derived from the Hopfield net[3] and when combined with a time derivative results in

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} \approx \frac{x_i' - x_i}{\triangle t} = -x_i + g(f_i) \tag{8}$$

where $f_i = \partial F(x)/\partial x_i$.

It has been shown[12] that the neural updating procedure below is equivalent to the Hopfield nets with the temperature T corresponding to the inverse of the gain. This net has been enhanced[15] with annealing (similar to gradually increasing the selection pressure in the replicator equation approach[26]) and neural normalization (similar to carefully choosing the time step in the replicator equation approach). That is, the function $g(\xi)$ is chosen so that some of the constraints are satisfied automatically.

At a given temperature T the outputs x_i^μ obey the Boltzmann distribution $x_i^\mu = \beta_i \exp(-f_i^\mu/c)$. The normalization takes the form

$$x_i^\mu = \frac{\exp(-f_i^\mu/T)}{\sum_{v=1}^{S} \exp(-f_i^v/T)} \tag{9}$$

Thus ensuring that $\sum_{v=1}^{S} x_i^\mu = 1, \forall i$. The variables $x_i^\mu, \forall \mu$ at each i are updated asynchronously using equation (9) in a predefined order, typically sweeping across the net, or i may be chosen at random and the variables $x_i^\mu, \forall \mu$ updated according to equation (9).

A functional for graph partitioning can be generated by setting the matrix W in equation (7) equal to the matrix $\frac{1}{2}(\mathbf{K_f} + \mathbf{C_k})$ in equation (4) and $z = 0$. Equivalently, the computationally more efficient functional given by equation (5) 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 Reference 3 where $W = -K + \alpha I$ in equation (7), the matrix K is the communication matrix (equation (1) with $S_p = 0$), I is the identity, α is a load balancing parameter and the vector $z = 0$. Then equation (8) is used for the iteration with $g(\xi) = \tanh(\frac{\xi}{T})$ and the neurons saturate to ± 1. If a node is -1 then the node belongs to partition one, if it equals 1 it 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 equation (5) might be used to the same effect.

As in choosing the gain in the Hopfield net the choice of the value of T is crucial in obtaining good viable solutions (too small a value of T will result in poor solutions and too large a value will result in $x_i^\mu \neq 0$ or 1 for some i and μ). One may overcome this difficulty by gradually reducing T from a relatively large value, using $T = \gamma T$, as 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 $x_i^\mu = 0$ or 1) without degradation of the solution. This annealing method can also be used with the Hopfield net.

In Reference 15 it was shown how using non-uniform T can be of advantage in solving the travelling salesman problem; this can also be applied to the partitioning problem, as will be demonstrated in the next section.

5.1. 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.[16]

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 $\frac{1}{S}$. A small perturbation $\triangle x_i^k$ is made to the neuron associated with node i and subdomain k and corresponding perturbations of

$$\triangle x_i^v = -\frac{\triangle x_i^k}{S-1}, \quad \forall v \neq k$$

From the neural updating, equation (9)

$$\frac{\partial x_i^k}{\partial f_i^m} = \begin{cases} \dfrac{x_i^k(x_i^k-1)}{T} \approx \dfrac{1-S}{S^2T} & \text{if } m = k \\ \dfrac{x_i^k x_i^k}{T} \approx \dfrac{1}{S^2T} & \text{if } m \neq k \end{cases} \tag{10}$$

The functional F that will be minimized by the neural network is given by equation (5) or, equivalently, (4) that is

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

with the $S \times S$ submatrices $\mathbf{K_{fij}}$ of $\mathbf{K_f}$ defined in section 2 and the communication matrix K given by equation (1) with $S_p = 0$.

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

$$\frac{\partial f_i^k}{\partial x_j^m} = \begin{cases} \dfrac{\alpha(S-1)}{S} & \text{if } m = k \\ k_{ij} - \dfrac{\alpha}{S} & \text{if } m \neq k \end{cases}$$

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

$$\triangle f_j^v = \sum_{n=1}^{N}\sum_{\mu=1}^{S} \frac{\partial f_j^v}{\partial x_n^\mu} \triangle x_n^\mu = f_{j\,\mathrm{old}}^v - f_{j\,\mathrm{new}}^v$$

and, therefore,

$$\triangle f_j^k = \sum_{\mu=1}^{S} \frac{\partial f_j^k}{\partial x_i^\mu} \triangle x_i^\mu = -(k_{ij} - \tfrac{\alpha}{S}) \triangle x_i^k$$

$$\triangle f_j^m = \left(k_{ij} - \frac{\alpha}{S}\right) \frac{\triangle x_i^m}{S-1}, \quad m \neq k$$

Changes in the mean field cause (from equation (10)) changes in x_j^μ of

$$\triangle x_j^k = \frac{\partial x_j^k}{\partial f_j^k}\triangle f_j^k + \sum_{m\neq k}^{S}\frac{\partial x_j^k}{\partial f_j^m}\triangle f_j^m = \frac{1}{ST}\left(k_{ij} - \frac{\alpha}{S}\right)\triangle x_i^k$$

$$\begin{aligned}\triangle x_j^m &= \frac{\partial x_j^m}{\partial f_j^k}\triangle f_j^k + \frac{\partial x_j^m}{\partial f_j^m}\triangle f_j^m + \sum_{\mu\neq k,m}^{S}\frac{\partial x_j^m}{\partial f_j^\mu}\triangle f_j^\mu \\ &= -\frac{1}{ST(S-1)}\left(k_{ij} - \frac{\alpha}{S}\right)\triangle x_i^k, \quad m \neq k\end{aligned}$$

It is assumed that the iteration is performed synchronously and that the perturbation will grow if $\sum_{j=1}^{N} x_j^k$ grows in the local vicinity of node i. The local vicinity of node i is defined by those 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.[16] This can be seen by plotting T_c against α. In the vicinity of node i the load balancing term has little effect. Therefore, at the critical temperature T_{c_i}

$$\sum_{j=1}^{N}\frac{1}{ST_{c_i}}k_{ij}\triangle x_i^k = \triangle x_i^k$$

or

$$T_{c_i} = \frac{\sum_{j=1}^{N} k_{ij}}{S} \tag{11}$$

Equation (11) 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. We do not examine the perturbation at a single node, as in Reference 16, because the perturbation tends to decrease and then increase for the general graphs of interest here. This method would necessitate the calculation of $\triangle x_j^k$ at many iteration levels to find an accurate T_{c_i}.

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

$$\alpha_i = \frac{S}{N}\sum_{j=1}^{N} k_{ji} \tag{12}$$

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

$$T_c = \frac{\sum_{j=1}^{N}\sum_{i=1}^{N} k_{ij}}{NS} \tag{13}$$

$$\alpha = \frac{S}{N^2}\sum_{j=1}^{N}\sum_{i=1}^{N} k_{ij} \tag{14}$$

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_{c_i}. The ability of equation (13)

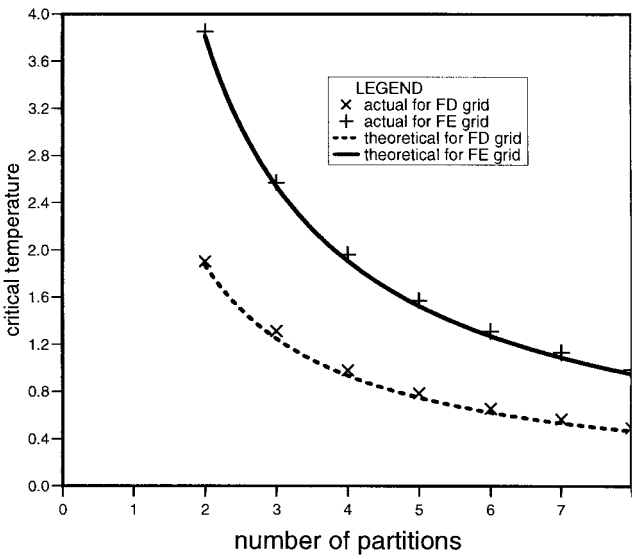

Figure 2. A comparison of the actual and theoretical critical temperatures for a graph derived from a 16×16 finite difference grid (5-point stencil) and a graph derived from a structured 32×32 finite element mesh comprising of four-node quadrilateral elements (9-point stencil)—variation with number of partitions

to predict the critical temperature is demonstrated in Figure 2 with an asyncronous neural network.

5.2. *How to update the neurons*

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 equation (9). The temperature is then constant and near the global critical temperature say 0.95 of T_c. At convergence one can anneal the temperature down e.g.) $T = 0.95 \times T$. At convergence, at this new temperature the temperature is further reduced and this process is repeated until all the neurons are saturated. A different temperature for each neuron might be used in order that the neurons saturate more uniformly. These temperatures can be found by, for example, initially taking the temperatures of the neurons associated with vertex i to be $0 \cdot 95$ of the T_{c_i} given by equation (11).

All neurons associated with a particular node are updated synchronously. The iteration can be continued until convergence is reached at that node. The advantage is that the resulting neural updating method does more work in parts of the mesh that have not converged. Yet another alternative is, at each iteration, to iterate on neurons associated with a given node, gradually reducing the temperature at that node until a steady and saturated solution at that node is obtained.

We visit the nodes in a predefined order sweeping across the network, but we could update the neurons in some random order, possibly visiting all the neurons exactly once, each iteration.

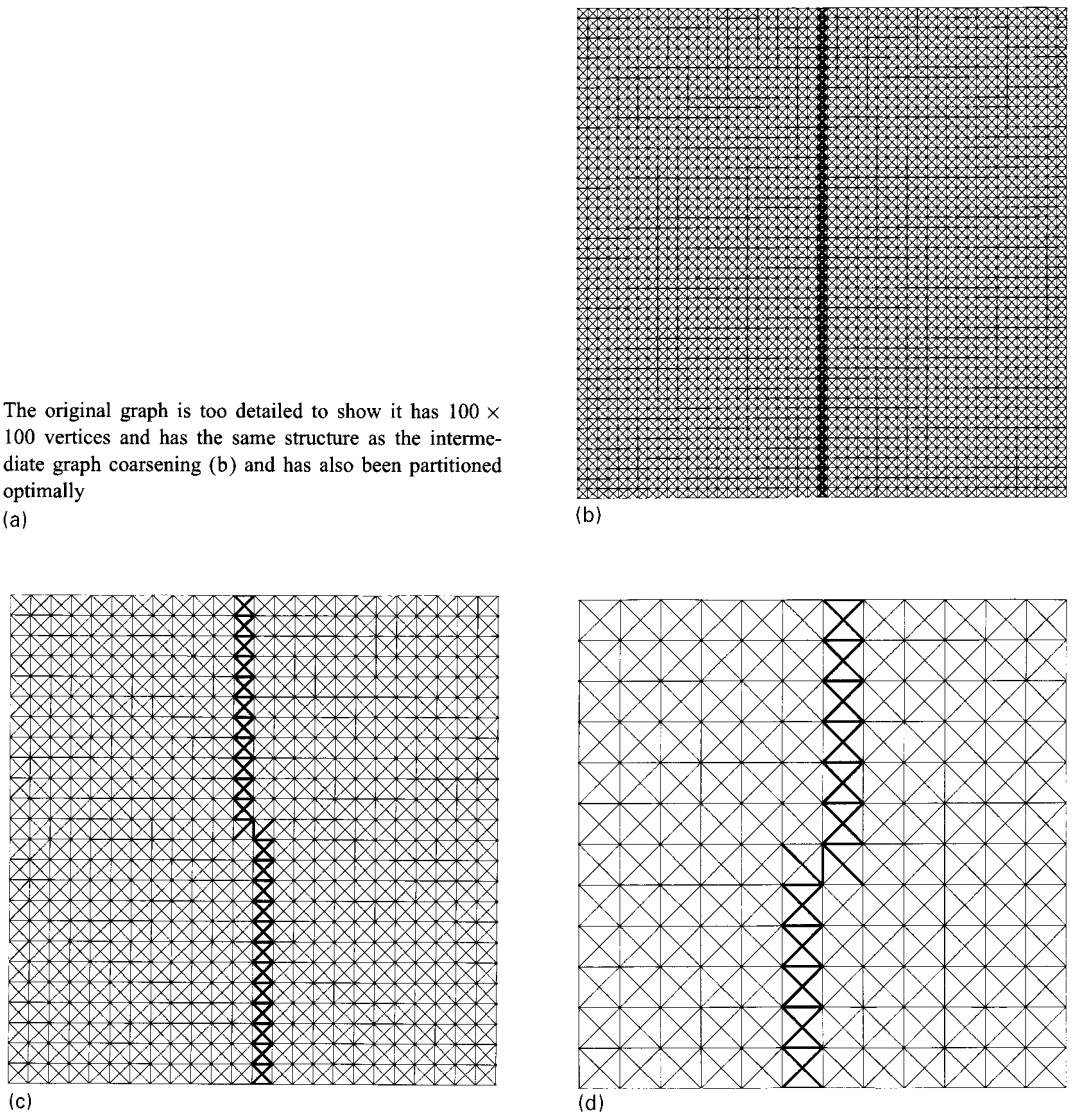

Figure 3. Graphs derived from a structured finite element mesh comprizing 99 × 99 four node quadrilateral elements (9 point stencil). Diagram shows graph and partition into two subdomains at various levels of graph coarsening (automatically coarsened)—the highlighted edges represent edges between nodes belonging to different subdomains: (a) finest graph level; (b) intermediate coarsening level; (c) intermediate coarsening level; (d) coarsest graph

6. APPLICATIONS

In this section we will apply the hierarchical neural network partitioning method to two partitioning problems. The first problem is to partition a structured finite element mesh comprising $99 \times 99 =$

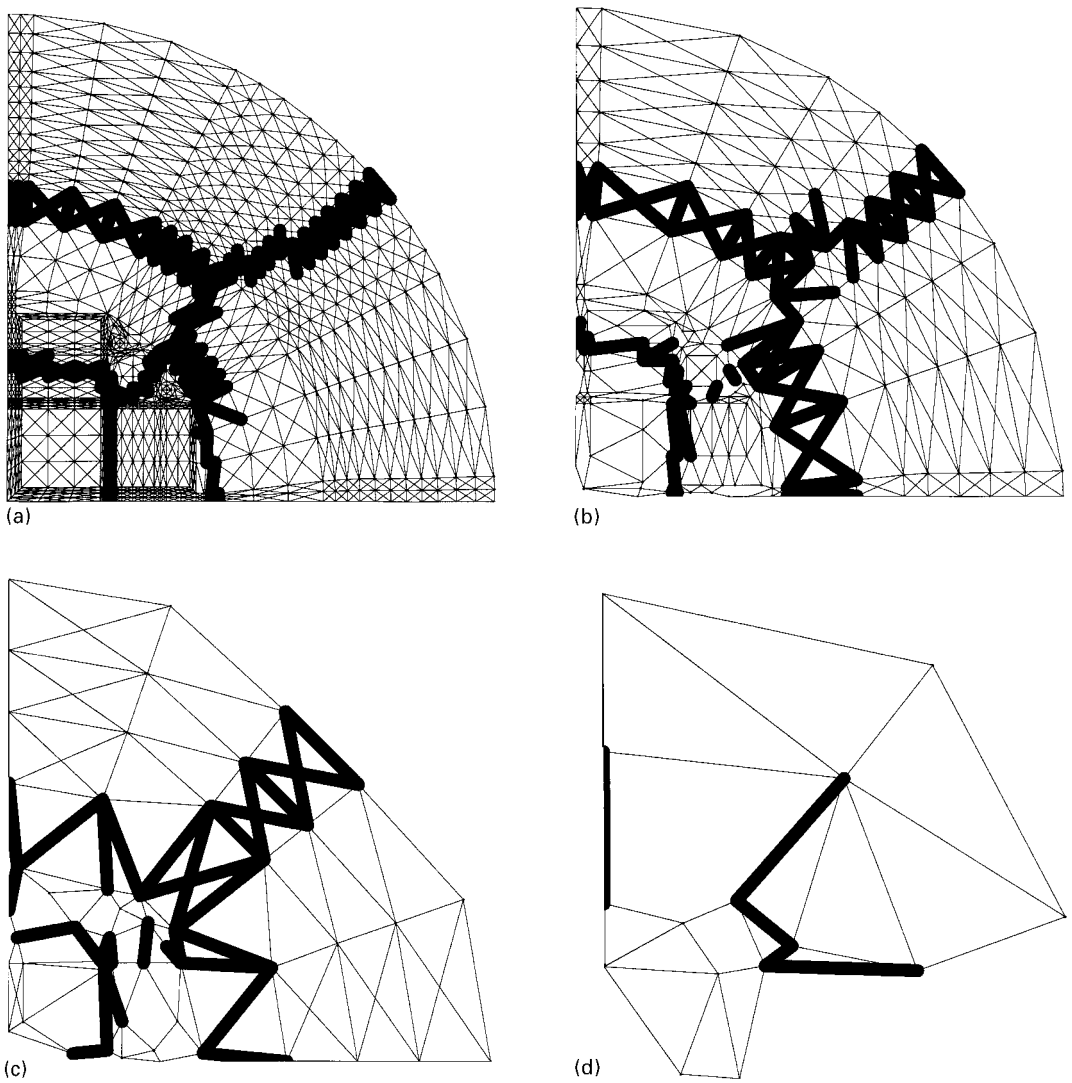

Figure 4. Graphs derived from an unstructured finite element mesh compromizing 697 elements including four node quadrilateral elements and three node triangular elements and 683 nodes. Diagram shows graph and partition into five subdomains at various levels of graph coarsening (automatically coarsened)—the highlighted edges represent edges between nodes belonging to different subdomains: (a) finest graph level; (b) intermediate coarsening level; (c) intermediate coarsening level; (d) coarsest graph

9801 four-node quadriliateral elements and $100 \times 100 = 10\,000$ nodes into two subdomains. For this example, the optimal solutions are known and in fact in this case the neural network has found optimal partitions, (see Figure 3(a)–(d))—the edges between nodes belonging to different partitions are highlighted in the figure. Figues 2(a)–(c) show the various levels of coarsening and the partitions found on the coarsened graphs. The graphs are coloured by first applying the heuritic

Table I. Results for graph partitioning into two subdomains using a neural network

Coarsening level	No of iterations for convergence	Nodes in subdomains	Mean neuron saturation
1 (original graph)	124	5000, 5000	0·754
2	62	1250, 1250	0·745
3	28	312, 313	0·727
4	398	85, 84	0·848

Note: The graph is derived from a structured finite element mesh comprizing 99×99 four node quadrilateral elements. The table shows the number of iterations required for convergence, the mean saturation of the neurons at convergence and the number of nodes in each subdomain of the optimized partition

procedure of Section 4 and then attempt to improve the colouring with the application of the MFT neural network applied to the functional of Section 4. The colouring of the graphs allows them to be coarsened.

Table I shows the number of neural network iterations or sweeps across the entire network needed to obtain convergence (convergence is assumed when the maximum difference of the neuron values between two consecutive iterations is less than 10^{-4}). It also shows the number of nodes on each graph coarsening level in each subdomain at convergence. In none of the examples presented here, do we anneal the temperature down, at convergence, in order to obtain highly saturated neurons, as in our experience the gain in partition quality is only marginal.

The load balancing parameter is defined on each graph level according to equation (14). The critical temperature T_{c_i} calculated from equation (11) for each vertex of each graph and the temperature (different for each vertex) used in the neuron updating equation (9) is taken to be 0.9 of this value in order to ensure neuron saturation. The optimization on the coarsest graph level is initialized using a uniform neuron saturation with the addition of a small quantity of random noise.

The second partitioning problem aims to decompose a graph arising from an unstuctured FEM mesh of four-node quadrilateral and three-node triangular elements into five. This mesh has 711 elements and 687 nodes and is a representation of a nuclear fuel transport flask. Figures 4(a)–(d) show the partition on various levels of graph coarsening in which Figure 4(d) is the desired partition. Table II shows the number of neural network iterations needed for convergence on each graph coarsening level and the number of nodes in each subdomain. Notice that on the coarsest graph level some of the subdomains have no nodes belonging to them: this is because the load balancing parameter given by equation (14) is not accurate when the quotient: (number of vertices)/(number of subdomains) is small; less than 6 in our experience.

6.1. Comparison with other partitioning methods

In this section the neural network partitioning method is compared with partitioning methods implemented in JOSTLE (version 2.0 by C. Walshaw, M. Cross and M. Everett)[22] and KMETIS (version 2.0 by G. Karypis and V. Kumar)[21] codes, using a DEC ALPHA 433 workstation in single precision. Graphs derived from four 3-D finite element meshes are partitioned using these two methods as well as the neural network. A brief description of the graphs and the problems they were derived from can be found in Table III.

Table II. Results for graph partitioning into five subdomains using a neural network

Coarsening level	No of iterations for convergence	Nodes in mesh	Nodes in subdomains	Mean neuron saturation
1 original graph	191	687	134, 137, 138, 141, 137	0.977
2	228	175	33, 33, 37, 37, 34	0.964
3	93	51	9, 10, 10, 11, 11	0.938
4	42	15	8, 0, 0, 7, 0	0.811

Note: The graph is derived from an unstructured finite element mesh compromizing 687 nodes, 711 elements including both four node quadrilateral and three node triangular elements. The table shows the number of iterations required for convergence, the mean saturation of the neurons at convergence and the number of nodes in each subdomain of the optimized partition

Table III. Brief description of graphs used to compare partioning methods. All graphs are derived from 3-D finite element meshes (a brief description of the problem they are used to solve is also included)

Mesh	Problem description	No of nodes	No of edges
Mesh1	Radiation shielding	31482	377939
Mesh2	Advanced gas cooled nuclear reactor	37789	392681
Mesh3	Flow over a sea-mount	73065	896804
Mesh4	Flow past a sphere	135081	1702099

Table IV displays the partitioning results using the three methods. The functional F used to gauge the quality of the partitioning is given in equation (4) with the load balancing parameter α defined by equation (14). The first term in this functional is equal to the number of edges E in the partition which is also given in Table IV. Thus, in the table, the difference between the functional value F and the number of edges E provides a measure of the load balancing achieved by the methods. The time to obtain the partitioning is also given in the table.

The neural networks performance has not been optimized in terms of the convergence criterion on each mesh level. The tolerance for the network was set to 10^{-4} as before. To speed up the optimization a second convergence criteria was used, that is the domain decomposition, calculated from the neuron saturations, remaining unchanged for the last 10 neural network iterations. This means that much of the calculations are wasted as neuron values inside subdomains do not vary significantly. This contributes to the fact that the neural network is considerably slower than the other methods. However, for small numbers of processors, it usually obtains better quality partitions than the other methods.

The neural network partitions were obtained without using a nested dissection procedure with the exception of partitions into 128. Nested bi-section was used to obtain these and as such the resulting partitions are generally of poorer quality than those produced by the other methods. A local search on these relatively poor partitions would improve their quality. This could in principle be performed by the neural network presented here. However, this would require another computational implementation geared towards efficiency when the number of subdomains is large.

Table IV. Comparison of partitioning methods for four graphs derived from finite element meshes—F is the functional value, gauging the quality of the partition and E is the number of edges between partitions. The smaller the value of F the better the partition quality

	Neural network			JOSTLE			KMETIS		
Mesh/parts	F	E	CPU (sec)	F	E	CPU (sec)	F	E	CPU (sec)
Mesh1									
2	4921	4921	1·99	5290	5290	0·98	5491	5491	0·91
4	16108	16085	29·2	16515	16515	1·15	16239	16239	0·98
8	25276	25263	38·3	25560	25361	1·35	26511	26511	1·10
128	98195	95927	17·8	92539	92204	6·50	94220	94080	1·63
Mesh2									
2	5284	5291	2·61	5434	5425	1·00	6132	6132	0·96
4	12567	12558	25·8	15424	15217	1·15	14618	14618	1·01
8	21718	21554	33·8	21616	21439	1·93	23496	23496	1·15
128	85544	80379	12·1	80568	79041	7·02	79544	79357	1·73
Mesh3									
2	12737	12728	11·7	13116	13116	2·50	12857	12857	2·19
4	23747	23740	18·1	26045	26045	2·75	28233	28233	2·32
8	38372	38362	60·6	42041	42041	3·28	41623	41623	2·45
128	176019	173429	47·1	164314	164266	12·47	167038	166996	3·35
Mesh4									
2	26212	26132	36·7	26612	26612	4·87	31139	31139	4·50
4	47171	46791	111·1	53417	51029	5·45	54964	54964	4·55
8	80972	80876	176·5	81843	81838	6·22	81849	81849	4·84
128	298323	295731	117·2	275442	275389	20·3	281178	281178	5·83

7. ADVANTAGES/DISADVANTAGES OF NEURAL NETWORK PARTITIONING

In the light of the above theory, applications and comparisons, a list of the advantages and disadvantges of the neural network partitioning method can be compiled:

Advantages

(1) Direct optimization of a functional which gauges the quality of a partition.
(2) A great deal of partitioning information can be injected into functional so that number of nodes does not necessarily represent load and partition positions can be discouraged/encouraged from/to cross certain areas of the solution domain in finite element applications.
(3) The neural network is easy to code and works in a similar way to matrix solvers, allowing standard libraries to be used e.g. for parallel communication of halo neuron values.
(4) Provides a theoretical framework for studying graph partitioning.
(5) Produces high-quality partition optimizations into small numbers of subdomains.
(6) There is scope for considerable improvement in run-times e.g. adaptive neuron updates - doing work just between subdomain boundaries.
(7) Automatically uses more CPU for difficult partitioning problems.
(8) Is guaranteed to converge.[30]

(9) Neuron saturation provides a level of certainty which can be used cheaply to more closely balance the load. Alternatively, the load balancing parameter can be annealed, incrementally increasing it as the optimization progresses, to improve partition balancing.

Disadvantages

(1) Unpredictable run times, see advantage 7 above.
(2) Often long-run times particularly for k-way partitions greater than two (in out implementa tions)—reflecting substantial increase in difficulty of partitioning problem as the number of domains increases.
(3) Efficient programming of the neural network for k-way partitioning into a large number of subdomains is lengthy.

8. CONCLUSIONS

Parallel computing has proven to be an effective means of solving large finite element problems. However, a large finite element mesh must first be decomposed into a number of subdomains. As shown in this paper, multi-grid partitioning methods can help provide the global partitioning information necessary to efficiently partition these problems and, when combined with automatic grid coarsening, can be applied to unstructured meshes.

Optimization by the MFT neural network provides a good compromise between efficiency and partition quality. It provides a theoretical basis on which the graph partitioning problem can be studied as evident by the derivation of the critical temperature and the load balancing parameters. Although our computer implementation was considerably slower than the other methods it was compared against, it generally produced better quality partitions into a small number of subgraphs.

The overall aim of producing an efficient DDM solution method can be achieved, in part, through the information injected into the functional. Dynamic partitioning can discourage subdomain boundaries from crossing areas in the solution domain of high activity or poor conditioning, as well as improving load balance. It is through the functional that such dynamic partitioning can be achieved.

REFERENCES

1. G. Karypis and V. Kumar, 'Parallel multi-level graph partitioning', *Proc.* Int. Parallel Processing Symp., April 1996.
2. C. C. Pain, C. R. E. de Oliveira and A. J. H. Goddard, 'Gradient annealing: an approach to combinatorial optimisation', submitted to *Parallel Computing* (1997).
3. C. Peterson and J. Anderson, 'Neural networks and NP-complete optimization problems; A performance study on the graph bisection problem', *Complex Systems*, **2,** 59–89 (1988).
4. A. H. Gee, S. Aiyer and R. Prager, 'An analytical framework for optimizing neural networks', *Neural Nets*, **6,** 77–97 (1993).
5. A. Bahreininejad, B. H. V. Topping and A. I. Khan, 'Finite element mesh partitioning using neural networks', *Adv. Eng. Software*, **27,** 103–115 (1996).
6. C. Farhat, 'A simple and efficient automatic FEM domain decomposer', *Comput. Struct.*, **28,** 579–602 (1988).
7. W. Chen, M. F. M. Stallmann and E. F. Gehringer, 'Hypercube embedding heuristics: an evaluation', *Int. J. Paral. Prog.*, **18**(6), (1989).
8. B. Kernighan and S. Lin, 'An efficient heuristic procedure for partitioning graphs', *Bell System Technical J.*, 291–307 (1970).
9. C. Greenough and R. F. Fowler, 'Partitioning methods for unstructured finite element meshes', *Proc. 1994, World Transputer Cong.*, Como, Italy, 1994, pp. 748–762
10. G. Miller *et al.*, A. George, J. Gilbert and J. Liu, 'Automatic mesh partitioning', in A. George, J. Gilbert and J. Liu (eds.), *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, Springer, New York, 1993.

11. B. Nour-Omid *et al.*, 'Solving finite element equations on concurrent computers', A. K. Noor, (ed.), *Am. Soc. Mech. Engng*., 291–307 (1986).
12. G. Bilbro, R. Mann, T. Miller, W. Snyder, D. Van den Bout and M. White, 'Mean field annealing and Neural networks', *Advances in Neural Information Processing*, Morgan-Kaufmann, 1989, 91–98.
13. J. J. Hopfield, 'Neurons with graded response have collective computational properties like those of two state neurons', *Proc. Natl. Acad. Sci. USA*, **81,** 3088–3092 (1984).
14. J. J. Hopfield and D. W. Tank, 'Neural computations of decisions in optimization problems', *Biol. Cybernet.*, **52,** 141–152 (1985).
15. D. Van den Bout and T. Miller, 'Improving the performance of the Hopfield-Tank neural network with normalization and annealing', *Biol. Cybernet.*, **62,** 129–139 (1989).
16. D. E. Van Den Bout and T. K. Miller, 'Graph partitioning using annealed neural networks', *IEE Trans. Neural Networks*, **1** (1990).
17. A. Pothen, H. D. Simon and K. Liou, 'Partitioning sparse matrices with eigenvectors of graphs', *SIAM J. Matrix Anal. Appl.* **11**(3), 430–452 (1990).
18. A. Pothen *et al.*, 'Towards a fast implementation of spectral nested dissection', *Proc. Supercomputing*, 1992, pp. 42–51.
19. H. Muhlenbein, M. Schomisch and J. Born, 'The parallel genetic algorithm as function optimizer', *Parallel Comput.*, **17,** 619–632 (1991).
20. E. H. L. Aarts and P. J. M van Laarhoven, 'Simulated annealing: a pedestrian review of the theory and some applications', *Pattern Recognition Theory and Applications*', Springer, Berlin, 1987, pp. 179–192.
21. G. Karypis and V. Kumar, 'A fast and high quality multilevel scheme for partitioning irregular graphs', *Int. Conf. Parallel Processing* (1995).
22. C. Walshaw, M. Cross and M. Everett, 'A localised algorithm for optimising unstructured mesh partitions', *Int. J. Supercomputer Appl.*, **9**(4), 280–295 (1995).
23. Bruce Hendrickson and Rober Leland, The chaco user's guide, *Technical Report SAND93-2339*, Sandia National Laboratories, 1993.
24. C. Walshaw *et al.*, 'Parallel dynamic load-balancing for adaptive unstructured meshes: Extended abstract', *Proc. Parallel CFD 97*, May 19 1997, Manchester England, 1997.
25. D. E. Keyes and W. G. Gropp, 'A comparison of domain decomposition techniques for elliptical partial differential equations and their parallel implementations', *SIAM. J. Statist. Comp.*, **8,** 166–202 (1987).
26. H. Muhlenbein, M. Gorges-Schleuter and O. Kramer, 'Evolution algorithms in combinatorial optimization', *Parallel Computing*, **7,** 65–85 (1988).
27. R. Webster, 'An algebraic multigrid solver for Navier-Stokes problems', *Int. J. Numer. Meth. FLuids*, **18,** 761–780 (1994).
28. M. Luby, 'A simple parallel algorithm for the maximal independent set problem', *SIAM J. Comput.*, **15,** 1036–1053 (1986).
29. F. J. Pineda, 'Generalization of backpropagation to recurrent and higher order neural networks', *American Inst. Phys.*, 602–610 (1988).
30. Dan W. Patterson, *Artificial Neural Networks—Theory and Applications*, Prentice Hall, Englewood cliffs, N.J., ISBN 0-13-295353-6.
31. T. Chockalingam and S. Arunkumar, 'A randomized heuristics for the mapping problem: The genetic approach', *Parallel Computing*, **18,** 1157–1165 (1992).
32. D. Whitley, T. Starkweather and C. Bogart, 'Genetic algorithms and neural networks optimizing connections and connectivity', *Parallel Comput.*, 347–361 (1990).
33. G. Miller *et al.*, 'A unified geometric approach to graph separators', *Proc. 31st Annual Symp. on Foundations of Computer Science*', 1991, pp. 538–547.

