Multilevel Mesh Adaptivity for Elliptic Boundary Value Problems in Two and Three Space Dimensions

by

Rashid Mahmood

Submitted in accordance with the requirements

for the degree of Doctor of Philosophy

The University of Leeds

School of Computing

January 2002

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Acknowledgements

I thank The Almighty, for the courage and strength He granted me to complete this project.

I would like to thank my supervisor Dr. Peter K. Jimack for his keen supervision, unlimited availability and encouragement throughout this project. I would also like to thank Professor Martin Berzins and Dr. Mark Walkley for their helpful advice and fruitful discussions.

I would like to acknowledge Ministry of Education, Pakistan for the financial support throughout this project in the form of the 100 merit scholarships scheme. My thanks also go to the General Office and Support staff of the School for the help provided to me throughout my stay.

Thanks also to my colleagues Idrees Ahmad, Muhammad Rafiq Asim, Sharifullah Khan, Sarfraz Ahmad Nadeem, Allah Nawaz and Nasir Touheed for matters not necessarily relating to the research.

I especially wish to thank my parents, my wife and my son for their constant love and support over the years. They have always been there for me and for that I thank them. It is to them that I dedicate this thesis.

Abstract

In this work we have developed, implemented and tested a new multilevel hybrid algorithm for the adaptive finite element solution of a general class of variational problems. Our multilevel hybrid algorithm is a combination of node movement, edge swapping and local h-refinement. The adaptive strategy used in our hybrid algorithm is based upon the construction of a hierarchy of locally optimal meshes starting with a coarse grid for which the location and the connectivity of the nodes is optimised. The grid is then locally refined and the new mesh is optimised in the same manner. Our hybrid algorithm does not need any global solution of the problem, it uses only local information to update the nodal solution values by solving the local variational problems on a relatively small domain with only few unknowns.

The node movement strategy is based upon knowledge of a steepest descent direction for each node found by a gradient calculation. A derivation of the gradient of stored energy with respect to the position of nodes is provided. A strategy for the movement of interior as well as boundary nodes is then given. Edge/face swapping in two and three space dimensions is explained and algorithms for node movement and edge swapping are given. Detailed descriptions of the possible local refinement strategies in two and three space dimensions are provided. Possible variants of our hybrid algorithm are considered and aspects of our hybrid algorithm regarding the quality of the meshes achieved and the computational work undertaken are discussed with some preliminary results.

We have applied our hybrid algorithm on a number of test problems: considering linear, nonlinear and system of equations in two and three space dimensions. A detailed comparison of the results produced by our hybrid algorithm with other adaptive approaches has been made for all of our test problems. Results presented indicate that our hybrid algorithm can produce better meshes, in both two and three space dimensions, than is possible by more conventional adaptive strategies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mathematical modelling of many real life situations, whether of a physical, chemical, biological, economic or other nature almost always involves differential equations. Most of the differential equations involved in these mathematical models are either so complex that analytical solutions to them do not exist or it is too difficult to find an analytical solution. Then come numerical techniques which are used to get an approximate solution by discretising the original problem. Popular numerical methods are the finite difference method, the finite element method, the finite volume method and the boundary element method. This work is devoted exclusively to the Finite Element Method (FEM) for the solution of boundary value partial differential equations (PDEs). The literature about this method is vast, and we cite the standard textbooks by Ciarlet [40], Johnson [70] and Zienkiewicz and Taylor [112] for a complete introduction and overview.

1.1 Finite element method

The finite element method is a numerical technique for obtaining approximate solutions to a wide variety of differential equations arising in engineering and mathematical physics. According to [88], some key features of the FEM were first introduced in 1943, but it is formally presented by Clough, Martin and Top in 1956 and the term "Finite Element" was first introduced by Clough in 1960.

Differential equations in which the dependent variable, and possibly its derivatives, are required to take specified values on the boundary are called boundary value problems or BVPs. In the finite element method, a given BVP is first transformed into a weak form and the domain for the BVP is viewed as an assemblage of subdomains (known as elements). An approximate solution is sought over each subdomain in the same way as in classical variational methods such as the Ritz method, the Galerkin method and the weighted residual method.

1.1.1 Weak form or variational form

A weak form is a weighted integral statement of a differential equation in which the differentiation is distributed among the dependent variable and the weight function, and includes the natural boundary conditions of the problem. The weak formulation has two desirable characteristics. First, it requires weaker continuity of the dependent variable by distributing the differentiation between the solution and the weight function w (due to its

weaker requirement of continuity, it has been given the name weak form). Second, the natural boundary conditions of the problem are included in the weak form, and the solution is required to satisfy only the essential boundary conditions of the problem. Whenever the classical solution exists it coincides with the weak solution of the problem.

We briefly describe how we can transform a classical boundary value problem to a variational boundary value problem. A variational boundary value problem is one of the form: find a function u that belongs to a Hilbert space V and satisfies the equation

$$a(u, v) = \langle l, v \rangle \tag{1.1}$$

for all functions $v \in W$, another Hilbert space. Here the space V is known as the space of admissible functions and W is known as the space of test or weight functions. They are both subspaces of a Sobolev space $H^m(\Omega)$ (i.e. functions in $L^2(\Omega)$ whose m^{th} derivatives are also in $L^2(\Omega)$ for some integer $m \gg 1$) such that

$$V = \{u \in H^m(\Omega) : u \text{ satisfies all essential boundary conditions}\}, \tag{1.2}$$

$$W = \{v \in H^m(\Omega) : v \text{ is zero on the essential boundary}\}.$$

We begin with a simple linear example of a Dirichlet problem, namely Poisson's equation with homogeneous Dirichlet boundary conditions. This is,

$$-\underline{\nabla}^2 u = f \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega, \tag{1.3}$$

with $f \in L^2(\Omega)$. Here Ω is the domain and $\partial\Omega$ is the boundary of the domain. We wish to derive the corresponding variational boundary value problem (VBVP). First we select V to be $H_0^1(\Omega)$ (i.e. $m = 1$ in (1.2) and the subscript 0 implies that all functions in this space are zero on $\partial\Omega$); next, we multiply by an arbitrary function v from $H_0^1(\Omega)$ (note $V = W$ for this problem) and integrate over Ω, to obtain

$$-\int_\Omega (\underline{\nabla}^2 u) v d\underline{x} = \int_\Omega f v d\underline{x}. \tag{1.4}$$

Green's theorem is now applied to the left-hand side of (1.4), to reduce this to,

$$-\int_\Omega (\underline{\nabla}^2 u) v d\underline{x} = -\int_{\partial\Omega} (\frac{\partial u}{\partial n}) v ds + \int_\Omega \underline{\nabla} u . \underline{\nabla} v d\underline{x}. \tag{1.5}$$

Since $v \in H_0^1(\Omega)$ the boundary integral vanishes and so we get,

$$\int_\Omega \underline{\nabla} u . \underline{\nabla} v d\underline{x} = \int_\Omega f v d\underline{x}, \quad \text{for all} \quad v \in H_0^1(\Omega). \tag{1.6}$$

This is the VBVP corresponding to the BVP (1.3). Where,

$$a(u,v) = \int_\Omega \underline{\nabla} u . \underline{\nabla} v d\underline{x} \quad and \quad \langle l, v \rangle = \int_\Omega f v d\underline{x}. \tag{1.7}$$

A weak form exists for all problems, linear or nonlinear that are described by second and higher order differential equations, and since the VBVP (1.6) is derived from the equation (1.3), every solution of (1.3) is a solution of the VBVP. Conversely it can be shown that every solution of (1.6) that has sufficient regularity solves the classical problem.

1.1.2 Energy minimisation

Traditionally the variational form means that the problem is formulated as one in which it is required to minimise a particular functional. These are

often called variational problems, and may be posed in the following form (or similar, according to the precise nature of the boundary conditions):

$$\min_{v \in V} \int_{\Omega} F(\underline{x}, v, \underline{\nabla} v) d\underline{x} \tag{1.8}$$

for some energy density function $F : \mathcal{R}^d \times \mathcal{R} \times \mathcal{R}^d \rightarrow \mathcal{R}$. Physically this variational form may be used to model problems in linear and nonlinear elasticity, heat and electrical conduction, motion by mean curvature and many more (see, for example, [19, 61, 62]).

Now we show that minimising an energy functional of the form (1.8) is equivalent to solving a VBVP. To illustrate this we continue with the simple problem of Poisson's equation subject to homogeneous Dirichlet boundary conditions. The first stage is to introduce a functional J, that is an operator, which maps a function $v(\underline{x}) \in H_0^1(\Omega)$ to a real number. For the Poisson problem the corresponding functional is defined as,

$$J(v) = \int_{\Omega} F(\underline{x}, v, \underline{\nabla} v) d\underline{x} = \frac{1}{2} \int_{\Omega} (\underline{\nabla} v)^2 d\underline{x} - \int_{\Omega} f v d\underline{x} \tag{1.9}$$

and the VBVP corresponding to the Poisson problem will be equivalent to the following;

$$\text{Find } \; u \quad \text{such that } J(u) \leq J(v) \; \text{ for all admissible } \; v \in H_0^1(\Omega).$$

To show the equivalence we assume that a minimum of (1.9) does exist and that this minimum is achieved at the function u. If we replace v by $u + \epsilon v$, where v is arbitrary, although a member of $H_0^1(\Omega)$, then we may treat $J(u + \epsilon v)$ as a function of the single variable ϵ, and write $J(u + \epsilon v) \equiv H(\epsilon)$

say. A minimum of H is then achieved at $\epsilon = 0$, so this condition for a minimum is therefore that

$$\frac{d}{d\epsilon}H(\epsilon)\mid_{\epsilon=0}= 0. \tag{1.10}$$

From (1.9),

$$H(\epsilon) = \left[\frac{1}{2}\epsilon^2\int_\Omega(\underline{\nabla}v)^2 + \epsilon\int_\Omega \underline{\nabla}u.\underline{\nabla}v + \frac{1}{2}\int_\Omega(\underline{\nabla}u)^2 - \epsilon\int_\Omega fv - \int_\Omega fu\right] d\underline{x}. \tag{1.11}$$

Differentiation with respect to ϵ, and evaluation at $\epsilon = 0$ results in the equation (1.10) reducing to

$$\int_\Omega \underline{\nabla}u.\underline{\nabla}v d\underline{x} = \int_\Omega fv d\underline{x}. \tag{1.12}$$

However this is derived for an arbitrary choice of $v \in H_0^1(\Omega)$ and so it is the corresponding weak form of Poisson's equation. Hence finding a minimum of the functional (1.9) or solving the VBVP (1.6) are equivalent.

A similar argument to the above shows that a solution of the general minimisation problem (1.8), if it exists, must satisfy the weak form of the following classical differential equation:

$$-F_{,2}(\underline{x}, u, \underline{\nabla}u) + \frac{d}{dx_j}F_{,3j}(\underline{x}, u, \underline{\nabla}u) = 0, \tag{1.13}$$

for summation of j from 1 to n. Equations (1.13) are known as the Euler-Lagrange equations, where $F_{,k}(.,.,.)$ represents differentiation of F with respect to its k^{th} dependent variable (and $F_{,3j}$ is the j^{th} component of $F_{,3}$).

As mentioned earlier, a weak form for every differential equation of order two or higher exists, however not all equations admit a functional formulation such as (1.8). On the other hand variational methods and the finite

element method do not require a functional; a weak form is sufficient. If one does have a functional at hand then the weak form may also be obtained by taking its first variation (as illustrated above).

1.1.3 Variational methods

In these methods the boundary value problem is first transformed into a variational boundary value problem or weak form and then an approximate solution of the type,

$$u \approx \sum_{i=1}^{N} u_i \psi_i, \tag{1.14}$$

is sought where u_i are coefficients to be determined and ψ_i are chosen approximating functions. The approximate solutions found via these methods are continuous functions of position in the domain (and may be smooth if the functions ψ_i are). The main disadvantage of general variational methods is the difficulty encountered in selecting appropriate approximating functions. Clearly there is no unique way of constructing them, even given the constraints of the boundary conditions. The selection becomes even more difficult when the domain is geometrically complex and/or the boundary conditions are unusually complicated. Further details of these methods can be found in textbooks by Oden [82] and Forray [47].

1.1.4 Formulation of the FEM

The finite element method not only overcomes the above shortcomings of general variational methods, but it is endowed with the features of an effective computational technique. A finite element model of a problem gives a

piecewise polynomial approximation to the governing equations. The basic idea of the finite element method is that a solution region can be approximated by replacing it with an assemblage of discrete elements, usually called a mesh. Since these elements can be put together in a variety of ways, they can be used to represent very complex domain shapes. The mesh consists of line segments in one dimension, in two dimensions it may consist of triangles or quadrilaterals and in three dimensions it may consist of tetrahedra or hexahedra. All these are known as finite elements or simply elements.

A variety of element shapes may be used, and, with care different element shapes may be employed in the same solution region. If we partition the domain Ω into a finite number E of elements Ω_1,Ω_2,...,Ω_E, then these elements should be non overlapping and cover the domain Ω in the sense that,

$$\Omega_e \bigcap \Omega_f = \phi \quad for \quad e \neq f \quad and \quad \bigcup_{e=1}^{E} \bar{\Omega}_e = \bar{\Omega}. \tag{1.15}$$

The number and the type of elements to be used in a given problem are matters of mathematical or engineering judgement (however see Section 1.4 for a discussion of error estimation).

The finite element method works by expressing the unknown field variable in terms of assumed approximating functions within each element. The approximating functions (sometimes called interpolation functions) are defined in terms of linear combinations of algebraic polynomials called basis functions and the values of the field variables at specified points called nodes or nodal points. Nodes usually lie on element boundaries where adjacent ele-

ments are considered to be connected. The nodal values of the field variable and the basis functions for the elements completely define the behaviour of the field variable within the elements. For the finite element representation of a problem the nodal values of the field variable become unknowns to be determined.

The choice of algebraic polynomials as a basis function has two reasons. First the interpolation theory of numerical analysis can be used to develop the approximate functions systematically over an element. Second, numerical evaluation of integrals of algebraic polynomials is easy. The degree of the polynomial chosen depends on the number of nodes assigned to the element, the nature and number of unknowns at each node and certain continuity requirements imposed at the nodes and along the element boundaries. For example, in two dimensions on triangles the field variables may be approximated by linear polynomials $p = \alpha_1 + \alpha_2 x + \alpha_3 y$, with three nodes at the vertices of the triangle or by quadratic polynomials $p = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 x^2 + \alpha_5 xy + \alpha_6 y^2$, with six nodes, three at the vertices and three at the mid points of the triangle edges. Basis functions ψ_i have the following properties.

1. The functions ψ_i are bounded and continuous, that is, $\psi_i \in C(\bar{\Omega})$.

2. The total number of basis functions is equal to the number of nodes present in the mesh and each function ψ_i is nonzero only on those elements that are connected to node i: $\psi_i(\underline{x})|_{\Omega_e} \equiv 0$ if $i \notin \bar{\Omega}_e$.

3. ψ_i is equal to 1 at node i, and equal to zero at the other nodes,

$$\psi_i(x_j) = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{otherwise.} \end{cases}$$

For a homogeneous Dirichlet boundary value problem[1] the finite element solution can be written as,

$$u^h = \sum_{i=1}^{N} u_i \psi_i, \tag{1.16}$$

where the values of u_i are unknown (to be determined) for $i = 1, ..., N$. Now we are in a position to apply the finite element method to approximate the solution of the VBVP (1.1). A Galerkin approximation u^h to the solution of (1.1) may be sought by constructing a finite dimensional subspace V^h of V, which is spanned by a finite number of basis functions ψ_i, and we then pose the problem of finding $u^h \in V^h$ that satisfies,

$$a(u^h, v^h) =< l, v^h > \quad \forall \quad v^h \in V^h. \tag{1.17}$$

For the Galerkin method to solve for u^h we simply note that both u^h and v^h must be linear combinations of the basis functions of V^h so that,

$$u^h = \sum_{i=1}^{N} u_i \psi_i \quad \text{and} \quad v^h = \sum_{j=1}^{N} c_j \psi_j. \tag{1.18}$$

Of course, since v^h is arbitrary so are the coefficients c_j. Substitution of (1.18) in (1.17) and use of the fact that a is bilinear and l is linear, lead to the equation,

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a(\psi_i, \psi_j) u_i c_j = \sum_{j=1}^{N} < l, \psi_j > c_j, \tag{1.19}$$

[1] For the simplicity of this explanation we describe the FEM for a homogeneous Dirichlet problem on a mesh with N vertices in the interior of the domain Ω.

or,

$$\sum_{j=1}^{N} c_j \left(\sum_{i=1}^{N} K_{ij} u_i - f_j \right) = 0, \tag{1.20}$$

where

$$K_{ij} = a(\psi_i, \psi_j) \quad and \quad f_j =< l, \psi_j > . \tag{1.21}$$

Since the coefficients c_j are arbitrary, it follows that (1.20) only holds if the term in brackets is zero for each j from 1 to N. The problem is thus reduced to one of solving the set of simultaneous equations,

$$\sum_{i=1}^{N} K_{ij} u_i = f_j, \quad j = 1, ..., N. \tag{1.22}$$

In Matrix form this is

$$K\underline{u} = \underline{f}, \tag{1.23}$$

where the matrix K is known as the stiffness matrix and $\underline{f}$ is known as the load vector.

Since $\psi_i = 0$ for all elements that do not have node i as a node, it follows that this property of basis functions will result in the matrix K having a sparse structure or, with an appropriate ordering, a banded structure in which all nonzero entries are clustered around the main diagonal.

1.2 Solution of the FEM equations

In engineering problems the size of the system of equations is generally quite large. It is important to design the solution algorithm so that the arithmetic operations in the solution phase can be performed as efficiently as possible. The finite element mesh required for the solution of a PDE depends on the

nature of the problem. There are two general types of mesh: structured meshes and unstructured meshes, and there are two main approaches to solving the system of simultaneous equations: direct methods or iterative methods, and there are many variants of each method.

1.2.1 Structured meshes

These type of mesh are also known as regular meshes. Structured meshes are easy to generate for simple domain shapes and require no storage. They are mostly used for simple geometries and high order elements because it is difficult to generate a structured mesh for a complex geometry (and it is also difficult in structured mesh to increase the number of elements just in the regions where the variation in solution is sharp). Discussions of the application and use of structured meshes can be found in [55] and [102] and the references there in.

1.2.2 Unstructured meshes

These are also known as irregular meshes. The theory and applications of numerical methods for unstructured meshes have been improved significantly in recent years. Complicated geometrical domains can be discretised relatively easily using unstructured meshes. Unstructured meshes can be used easily to increase the number of elements in the regions where a solution exhibits sharp features, either by moving nodes or by increasing the number of elements in that region, for a better approximation of the solution. Our work is solely focused on these meshes.

1.2.3 Direct methods

Direct methods obtain the exact solution (in real arithmetic) for the system of simultaneous linear equations (1.23). Direct methods use a finite number of operations which can be specified in advance, so that the number of operations performed is independent of the accuracy desired. An exact solution would be obtained if there were no round off errors.

Direct methods are usually based on Gaussian elimination: however the algorithms are constructed so that the sparsity of the matrix is exploited in the interest of computational efficiency (see for example [44, 56]). In general direct solvers are preferred when the stiffness matrix is not very large and there are several load vectors. Also, direct solves are often used with structured meshes, as the sparsity pattern can be exploited more efficiently in these cases.

For very large order matrices, direct methods are too costly and require too much storage. Also, when solving finite element equations it is not the aim to solve them exactly because it would not affect the approximate solution significantly to solve beyond the discretisation error.

1.2.4 Iterative methods

For solving finite element equations iterative methods are the more common choice because, when carefully applied, less computational effort is required to compute a solution than with direct methods. An iterative method generates a sequence of approximations which should converge to an exact solution of the discrete system. The accuracy of the solution obtained de-

pends upon the number of iterations performed, the condition number of the matrix, its size and the accuracy of the arithmetic performed by the computer (as well as the particular algorithm used of course)

Iterative methods are particularly efficient for solving finite element equations when the system is very large, unstructured meshes are used and if there are few load vectors. Iterative methods are also a popular choice for parallel implementation, although this is not considered here. There are many iterative methods (having their own advantages and disadvantages). Some of them are the Jacobi method, the Gauss-Seidel method, the successive over relaxation method, the conjugate gradient method, the generalised minimal residual method (GMRES) (see for example [95]). A potential drawback of iterative methods however is that they sometimes suffer from the problem of failure to converge (or extremely slow convergence) when the system of linear equations is very ill conditioned.

1.2.5 Condition number

The need for local resolution in physical models occurs frequently in practice, either when the boundary of the domain is not smooth, or to take care of special local features of the solution, or when the operator coefficients of the partial differential equation have singularities. To reduce the error from the FEM discretisation significantly and to approximate the solution efficiently it is necessary to refine the finite element mesh locally in the neighbourhood of the singularities or the non-smooth features.

Unfortunately, this objective of efficiently and accurately adapting to

local features is often in conflict with the solution process. This is because it often makes the linear system of equations ill conditioned, leading to slow convergence of iterative methods. Equation solvers can degrade or even fail to converge in the presence of small or varying discretisation scales. The situation becomes even worse when elements are refined anisotropically (see Berzins [29]), because anisotropic refinement makes the interior angles of some elements very small and it is a known result that when the element interior angles become small the condition number of the element matrix increases (see Fried [49]).

The condition number is a value that is used to measure the tendency of a problem to be ill-posed, see [100] for example. The convergence properties of iterative methods for solving systems of linear equations can be estimated in terms of the condition number of the linear system and the sensitivity of the solution to a perturbation in the right-hand side can be bounded using the condition number (see [21] and the references there in). In the L_2 norm the condition number, κ, is defined as the ratio between the maximum and minimum eigenvalues of the stiffness matrix K, i.e.

$$\kappa = \frac{\lambda_{max}(K)}{\lambda_{min}(K)} \geq 1. \tag{1.24}$$

Since convergence rates are determined by this condition number, if $\kappa \approx 1$ one generally obtains fast convergence and accurate results whereas if $\kappa \gg 1$ slow convergence and inaccurate results are usually observed.

It is a well known result (see Strang [101]) that, for each variational problem, (1.17) say, and each choice of finite element there is a constant C

such that,

$$\kappa \leq C h_{min}^{-2m}, \tag{1.25}$$

where $2m$ is the order of the corresponding strong form of the PDE. The constant depends inversely on the smallest eigenvalue λ, of the given continuous problem, and it increases if the geometry of the elements become degenerate. From this result we can see that when the mesh is refined near singularities to approximate a non-smooth solution accurately and efficiently, it will increase the condition number of the stiffness matrix significantly making the iterative solver's convergence potentially very slow.

1.2.6 Preconditioning

The term *preconditioning* is often used in numerical methods as an extra step designed to accelerate the convergence. Preconditioning is an important technique in iterative methods for solving large systems of linear equations especially when the condition number deteriorates as described above. Both the efficiency and robustness of iterative techniques can be improved by using it. Preconditioning is simply a means of transforming the original linear system into one which has the same solution, but which is better conditioned and is therefore likely to be easier to solve with an iterative solver. For the system of linear equations (1.23), a left preconditioner T is constructed such that the following system is solved

$$T^{-1} K \underline{u} = T^{-1} \underline{f}.$$

An equivalent right preconditioned system of the following form

$$KT^{-1}\underline{v} = \underline{f}, \quad \underline{u} = T^{-1}\underline{v},$$

could also be solved instead of the original system (1.23). The problem of finding an efficient preconditioner lies in identifying a matrix T that should satisfy the following properties.

- T should be a good approximation of K.
- The system $T\underline{u} = \underline{v}$ should be much easier and cheaper to solve than the original system.

If a carefully constructed preconditioner is used, it is possible to obtain a much faster convergence of the iterative method for the preconditioned system and the solution in less time, than for the original system. For a general introduction of the preconditioning see for example [95].

1.3 Mesh adaptivity

Finite element grid adaptation can be defined as an algorithmic procedure for the generation of a finite element discretisation that aims to yield a required accuracy for close to the minimum amount of computational effort. If the solution is smooth everywhere in the domain then a uniform mesh will generally be adequate to approximate the solution provided that it contains a sufficient number of elements. On the other hand when the solution exhibits sharp features such as steep gradients, boundary layers, shock waves, singularities or discontinuities, then a uniform mesh will not

be a good mesh for the approximation of the solution. This is because uniformly spaced nodes often waste computational effort in order to obtain acceptable truncation errors in regions of large solution variation where much smaller node separation than is necessary in regions of slow solution variation must be used.

During the last decade adaptive mesh generation has become an indispensable tool for the efficient numerical solution of partial differential equations having large local variations. Differential equations having such large local variations arise in many branches of computational mathematics such as fluid dynamics, hydraulics, combustion, heat transfer and external and internal aerodynamics. In many cases only a very small portion of the domain requires small node separation; thus significant economies can be obtained by adapting the nodes so that they remain concentrated about areas of large solution variation.

For the accurate and efficient resolution of these problems one needs a tool for assessing the error in the computed approximation and also a strategy to improve the mesh (and hence the solution) in the light of this information. Two different approaches may be used for assessing the error: *a priori* error assessment (based on a prior knowledge of the nature of the solution), see for example [5, 34], and *a posteriori* error assessment (which uses the numerical solution to drive the refinement), see for example [1, 2, 12].

The main strategies to improve solution accuracy are p-adaptivity (poly-

nomial enrichment), see for example [16, 28, 43], r-adaptivity (mesh movement and edge and face swapping), see for example [17, 35, 36, 38, 48], and h-adaptivity (mesh refinement), see for example [74, 75, 98, 109, 111]. In practice we can combine different types of adaptivity: the popular adaptive schemes are hp-adaptivity, see for example [10, 12, 53, 106, 113] and hr-adaptivity, see for example [36, 55, 84, 89, 104].

In principle any of the approaches for error assessment can be combined with any of the strategies for adapting the spatial discretisation to produce an adaptive strategy for finite element computations. The next two sections consider these issues of assessment of the error and the possible adaptivity techniques respectively.

1.4 Refinement criteria

In traditional finite element analysis, as the number of elements increases, the accuracy of the solution improves. The accuracy of the problem can be measured quantitatively or qualitatively with various physical entities, such as strain energies, displacements and stresses, as well as with various error estimation methods, such as described in [4] for example.

The goal is to be able to have control over the behaviour of the actual finite element model by using error analysis methods. Refinement criteria are based on the estimated approximation error of the discretisation method and how this error is measured in some suitable norm. Since the exact solution is unknown the error can not be measured exactly but only assessed

by lower and upper bounds.

1.4.1 *A priori* error assessment

These criteria use knowledge of the exact solution behaviour and/or the input data. They allow the prediction of the decrease of the error norm as the discretisation becomes better. Knowledge of the numerical solution is not necessary. For example, it has been shown by Apel [5] that isotropic as well as anisotropic meshes graded *a priori* are useful for treating edge singularities; both diminishing the error and achieving the optimal approximation order.

As this approach does not make use of knowledge of the numerical solution, the size of the error in this solution can not be assessed. The technique is valuable when there is some information of the solution behaviour, however it is not always the case that the locations of singularities etc. are known *a priori*. Likely candidates for *a priori* refinement are re-entrant corners, holes in the domain or regions where the forcing function is non-smooth. For reliability however it is generally necessary to inspect the distribution of an *a posteriori* error estimate to find locations of large error and to refine the FEM mesh in a more efficient way.

1.4.2 *A posteriori* error assessment

A posteriori error assessment requires and employs the approximate solution once it has been calculated. This error assessment criterion is an indispensable ingredient of adaptive solution algorithms. Such algorithms

typically consist of the following steps.

Algorithm 1 *:*

1. *Start with an initial mesh.*
2. *Solve the corresponding discrete system.*
3. *Compute a local a posteriori error estimate or indicator for each element of the mesh.*
4. *When the global error is small enough then stop, otherwise obtain information for a new better mesh i.e. identify the regions where solution error is largest.*
5. *Refine the mesh based on the information obtained in steps 3 and 4 then goto 2.*

A posteriori assessment criteria can be divided into two general types: error estimates and error indicators. Error estimators approximate a measure of the actual error in a given norm i.e. they estimate the error quantitatively (e.g. [13, 14]). Error indicators on the other hand are based on heuristic considerations. An error indicator only gives relative information about the error, i.e. it measures the error qualitatively. For each particular application, a readily available quantity is chosen, in an ad-hoc manner, as an indicator of error (e.g. [36, 104]).

The history of *a posteriori* error assessment starts with Babuska and Rheinboldt who presented a so called "residual error estimator" [11] and "a local problem error estimator" [12]. Since then it has been applied in virtually every area of computational PDEs, and there is a great wealth of

papers in this area. Surveys of the traditional approach to *a posteriori* error assessment are given by Verfuth [107] and in the book by Ainsworth and Oden [4] for example.

1.4.3 Error estimators

Error estimators are based on firm mathematical foundations and are usually more expensive to evaluate than error indicators. In exchange for that they have a major advantage. They provide objective and quantitative information about the error. Moreover the range of applicability of a certain error estimator is usually larger than for corresponding error indicators. Error estimators may be classified into two groups: flux projection or recovery based estimators (e.g. [113, 114]) and residual type estimators (e.g. [3, 11, 12, 15]).

Zienkiewicz and his coworkers propose error estimators based on a recovered gradient [113, 115] (nowadays often called ZZ or Zienkiewicz-Zhu error estimators). Later an improved estimator has been designed by means of super-convergent patch recovery [114]. The basis for recovery based error estimators is the fact that, on many occasions, the solution is most accurately sampled at the nodes defining an element and that the gradients are best sampled at some interior points. Such points often exhibit the quality known as super-convergence (i.e. the values sampled at these points show an error which decreases more rapidly than elsewhere). These methods perform well on highly regular grids but for irregular grids the existence of super-convergent points is questionable (see Zienkiewicz [112]). Neverthe-

less ZZ-type error estimates often work very well on unstructured meshes and are very cheap to compute.

An introduction into the general concept of residual based error control for finite element methods has been given in the survey article by Eriksson, Estep, Hansbo and Johnson [46]. In the methods proposed by Babuska and coworkers the error estimate at each step is based on solving local problems involving a local residual, and the refinements are carried out according to the size of the solution of these local error problems. Bank and Weiser [24] also derived different local problem error estimators. Eriksson, and Johnson [45] have also contributed to the methodology of error estimation for various differential equations. Verfurth [107] derived local lower bounds of the error within a unified theory of error estimators, and considered numerous other aspects of error estimation.

1.4.4 Error indicators

Various choices of error indicator can be found in the literature. From a geometrical point of view, for instance, the element aspect ratio [48], or more generally, the distortion can be used. For elliptic equations a common choice is the energy functional or its gradient [104]. Error indicators are attractive because of their simplicity: they are based on very simple intuitive considerations (geometrical, mechanical, physical etc.) and can be computed easily and efficiently. Qualities used as error indicators are always readily available in the finite element computation, so the overhead cost is minimum. The draw back is that the judgement of the user, for

defining an appropriate indicator, is crucial and may be very specific to each particular application. Moreover, error indicators only give relative information. Since the error is not quantified an error indicator only tells where the spatial discretisation must be richer, but not how much richer should it be.

1.4.4.1 Geometrical criteria

To use the finite element method the domain must be decomposed into simple geometric elements. Very complex domains can be discretised using an automatic mesh generator, which uses adaptive mesh refinement, i.e. it is possible to refine the mesh near holes and corners of the domain, or where we can *a priori* predict that this area of the domain needs more concentration of elements and nodes. For this purpose there are many mesh generators available which can deal with very complex domains: for example GRUMMP [52], Easymesh [105] and Meshme [69]. Combining automatic mesh generation and adaptive refinement sometimes produces poorly shaped elements that cause numerical difficulties when computing the solution.

There are well known results that as an element's maximum interior angle becomes too large, the discretisation error in the finite element solution increases [9] and as angles become too small, the condition number of the element matrix increases [49]. Thus for meshes with highly distorted elements, the solution is potentially less accurate and more difficult to compute. The problem is more severe in three dimensions than in two dimensions, because

tetrahedra can be distorted to poor quality in more ways then triangles can. But these poorly shaped elements can be improved using mesh movement and edge or face swapping, by taking into account some geometric quality measure. For a detailed description of geometric quality measures see [48] and the references there in.

However for solution based optimisation procedures, which are the main topics of this thesis, there are some occasions where stretched elements may be desirable for certain solution shapes. The resulting mesh may include some poor geometric quality elements, as discussed in [90], such that long and thin elements can form part of a good mesh for strongly anisotropic solutions.

1.4.4.2 Physical quantity criteria

Some times there are physical quantities present in the mathematical model of a physical situation which may be used to dictate the refinement criteria. For example, for flow problems if the edges of the mesh are aligned in the direction of flow a better approximation of the solution may be obtained [60, 93]. Also for computational fluid dynamics problems it is possible to refine the mesh according to the behaviour of the flow variables (see for example [85]).

1.5 Refinement schemes

Having briefly discussed some of the main refinement criteria used in adaptive algorithms, we now present a discussion of the main refinement schemes

that are available.

1.5.1 h-Adaptivity

The most widely used form of mesh adaptivity is the approach referred to as h-adaptivity. This method of adaptivity consists of subdividing an existing mesh, using the same type of element, so that the local element size matches the requirements of solution. A generalisation of this approach also allows elements to be merged together where the initial mesh is unnecessarily fine.

There are two general approaches to this type of refinement: isotropic, [75, 98, 109] for example, and anisotropic, as in [6, 8, 26] for example. Isotropic refinement implies that child elements are created by dividing all of the edges of the element in half. This creates four child elements in 2D and eight in 3D. Anisotropic refinement creates a minimum of two child elements in a selected direction depending on the solution or according to some geometrical requirement. Anisotropic refinement sometimes produces similar accuracies to isotropic refinement but with fewer elements, particularly for problems where solution is strongly oriented along a particular direction. Such schemes can easily create highly nonuniform meshes adapted to the singularities of a given solution.

Perhaps the biggest drawback of h-refinement is that the exact locations of grid points depend on the geometry of the father element (e.g., the midpoint of an edge) and, inductively, on the structure of the initial coarse triangulation. Often this presents no problem, but in other cases (e.g., that of a steep front) it might be the case that moving the grid points a little

(to better align the mesh with the front) can reduce the error substantially. Without such movement excessive subdivision of elements can sometimes occur.

1.5.2 p-Adaptivity

The idea of p-adaptivity is to increase the order of the polynomials where a richer interpolation is needed, and maintain polynomials of low order where it is already rich enough. One obvious advantage of p-type finite elements is that they allow solution adaptivity without requiring mesh refinement. Furthermore this approach provides the fastest rate of convergence as the number of degrees of freedom increases provided the solution is sufficiently smooth (see Zienkiewicz [112]). In a p-type finite element method, the basis functions are constructed in such a way that the maximum polynomial order of approximation can be selected independently for each edge, face and solid in the mesh. The goal of p-type adaptivity is to select appropriate polynomial orders for each edge which will lead to an overall solution providing good quality results in an acceptable elapsed time for the minimal computing resources. Prescribed accuracies can also be obtained via this method when used with an appropriate a posteriori error estimator.

There are two main drawbacks of p-type adaptivity. Firstly, the implementation is very tedious since special care is needed to match two adjacent elements of different order. Secondly, the computational cost of solving the linear equations (1.23) is high since the stiffness matrix becomes much less sparse as p is increased. Finally, we note that this strategy is not appropri-

ate in the vicinity of shocks or other singularities in the solution.

1.5.3 r-Adaptivity

In this method the objective is not to alter the number of elements in the finite element mesh, but to adapt their shape and position instead. The meshes produced through the use of r-refinement are frequently capable of allowing extremely accurate representation of sharp solution features, such as cracks, shocks, boundary layers etc. This method of adaptivity can be divided into two types: one in which the position of the nodes is changed by relocating them but not changing the topology of the mesh, commonly known as mesh movement techniques, and the other in which the position of the nodes is fixed but the topology of the mesh is changed, commonly known as edge swapping.

1.5.3.1 Mesh movement

In this technique an initial mesh is selected and this is allowed to evolve so as to reduce (or equidistribute [57, 77, 87]) some measure of the error. This measure may depend upon residual [18, 80] or energy [62, 104]. A number of algorithms have been proposed for allowing such a relocation of mesh points both for problems which are time dependent [18, 61] and those which are steady state [36, 62]. Descriptions of some of these appear in Section 1.5.4 below.

The possible limitations associated with node movement methods are: element distortion, geometric complexity and slow convergence of the alge-

braic equation solver. Also, as the number of nodes are fixed in this form of adaptivity, it tends to lack of the robustness of h or p-refinement. It may not be possible to reduce the error to within a required tolerance for a given starting mesh for example. Once the location of nodes is optimal (according to some measure of the error), a more accurate solution can only be achieved by increasing the number of degrees of freedom either by h or p-adaptivity. Also many techniques of movement are susceptible to falling into local minimum traps which provide grids that may well be far from a truly globally optimal mesh.

1.5.3.2 Edge swapping

In this variation of the method we change the topology of the mesh by swapping the edges or faces to improved positions, i.e. we find an orientation of edges or faces in the triangulation which reduces some measure of the error. Edge swapping in two and higher dimensions is based on the work by Lawson [72, 73]. The work by Gunzberger, Fix and Nicolaides [54] for example demonstrates that the convergence rate of the FEM depends on the connectivity of the mesh. For this problem, a commonly used uniform direction triangulation yields a suboptimal convergence rate, where as, with the same vertices but with a "criss cross" connectivity the optimal convergence rate is achieved. It has been shown in [93] and [108] that for highly directional flow problems (e.g. [86, 90, 97]), aligning the edges in the direction of the flow improves the quality of the mesh with respect to solution of the mesh. Iliescu [60] has shown that in 2D and 3D the solution accuracy for a linear

hyperbolic problem can be increased by simple mesh reconnection to better align the mesh with the convection direction.

For the two spatial dimensions edges are swapped to get connectivity improvements, see for instance, the work by Lawson [72], Chui and Hang [39] and Ripa and Schiff [91]. One begins with an initial triangulation of domain Ω and we know that there are finitely many V-triangulations of Ω. If e is the common edge of two triangles in the triangulation and the quadrilateral formed by these two triangles is strictly convex then the edge e can be swapped to obtain another triangulation. If this new triangulation is better for approximating the solution locally (for some given quality criterion) then the triangulation before swapping is replaced by the swapped one. An optimal triangulation in which each edge is at its optimal position can be obtained by sweeping through all the internal edges in the mesh repeatedly, and swapping them whenever this yields an improvement.

In three dimensions not only edges but faces can also be swapped to get improved connectivities for tetrahedral meshes. Edge or face swapping is much more complicated in three space dimensions, without adding more nodes the number of elements can be increased by adding edges between two adjacent elements having a common face. Similarly the number of elements can be decreased by eliminating edges. We provide a detailed description of edge and face swapping in three dimensions in Chapter 5.

The limitations associated with edge swapping are: the number of degrees of freedom are fixed, the outcome may depend heavily on the choice

of initial mesh, and that no prescribed accuracies can be obtained just by edge swapping.

1.5.4 Moving mesh schemes

Mesh movement strategies seek to improve the accuracy of a solution without adding more basis functions; instead it is the support of each basis function which is adjusted. There are three main techniques for the movement of nodes. One is based on residual minimisation (e.g. [18, 80, 81]), another is based on equidistribution principles (e.g. [57, 77, 78]) and the third is based on some form of optimisation (e.g. [36, 42, 104]). However it should be noted that these three techniques overlap in many formulations. For example, the moving finite element (MFE) method was originally formulated as an L_2 minimisation (see [80, 81]). However, for steady equations of variational type, it has been shown in [63] that this technique corresponds to satisfaction of a minimisation principle for the mesh.

In the following subsections we describe the above three generic mesh movement approaches, which can be applied to a wide range of problems. The fact that, when nodes move, the basis functions depend on the vertex locations in a nonlinear fashion implies that all of these mesh movement procedures are inherently nonlinear, even for linear PDEs, whilst h and p-adaptive approaches can remain linear for linear problems.

1.5.4.1 Moving finite element

The moving finite element method is a numerical solution scheme for the solution of time dependent partial differential equations, which allows the automatic adaption of the finite element approximation space with time. The MFE method was introduced by Gelinas, Dross and Miller [50] , Miller and Miller [80] and Miller [81] in 1981, to obtain solutions of problems with steep moving fronts.

It uses piecewise linear finite elements and generates the augmented ODE system from the minimisation of the L_2 norm of the residual of the PDE over the time derivatives of both the nodal positions and the solution parameters. In this scheme a spatial mesh with a constant number of degrees of freedom is allowed to deform continuously in time. The positions of the nodes are treated as unknown time-dependent variables which, just like the conventional finite element degrees of freedom, must be evaluated as part of the solution procedure. This procedure is designed to simultaneously determine at each time both a suitable spatial mesh and a finite element solution on that mesh.

This scheme can also be applied to static problems with the aid of an artificial time parameter (for detail see [62]). It has been proved that the refinement which is induced by this approach can lead to an optimal minimiser of the energy functional for an elastostatic problem, over all variations in the finite element mesh as well as variations in the nodal displacement values.

The MFE method is widely applicable, however the method has a number of significant drawbacks. The method becomes singular when elements collapse (mesh tangling) or when the gradient of approximation is constant in neighbouring elements. There are various ways of preventing these singularities, for instance by using penalty functions (see Miller [80]) or element removal (see Baines [18]). By allowing the nodal positions to become degrees of freedom the size of the discrete problem that must be solved is doubled in one dimension and quadrupled in three dimensions. Also the equations added by the unknown positions of nodes are inevitably nonlinear even for a linear problem. This substantially increases the computational cost involved. Finally, the connectivity of the mesh is fixed: one possible way to overcome this problem however is to undertake occasional remeshing of the grid to alter the connectivity and/or local density of the mesh (e.g. see [37]).

1.5.4.2 Equidistributional principle

The use of an equidistributional principle (EP) provides another class of moving mesh method in which the mesh point locations may either be computed simultaneously with the solution via an augmented system of differential equations or computed iteratively. Equidistributional principles were first introduced by de Boor [32, 33] for solving boundary value problems for ordinary differential equations. It turns out to be an excellent approach for formulating moving mesh equations too. This involves the selection of mesh points such that some measure of the solution error is equalised over

each subinterval, which is usually done by the construction of a monitor function which uses the discrete solution. This is constructed in such a way that it captures the main characteristics of the error. A popular choice in one spatial dimension is the arc length monitor,

$$M(x) = \sqrt{1 + u_x^2}, \tag{1.26}$$

which concentrates the nodes in the regions where the solution changes rapidly.

A number of moving mesh methods have been developed depending on equidistribution principles, for example see [27, 59, 78]. The approach has been successfully applied in one spatial dimension but its extension to higher dimension is still the subject of ongoing research [58, 77]. The drawbacks of this method are the computational cost as the mesh equations are nonlinear. Also how best to construct a monitor function for a given problem is not obvious. Again for this method mesh tangling can appear while solving the mesh equations if care is not taken. Some progress has been made by Huang and Russell [58, 59] and Mackenzie et al. [77, 78] regarding these issues, however further research is required in this area to overcome all of these problems satisfactorily: especially in greater than one space dimension.

1.5.4.3 Direct minimisation principle

Since minimisation principles, such as (1.8) or (1.9), provide a functional to monitor and reduce, it is possible to take advantage of standard optimisation procedures in generating local minima with respect to mesh vertex locations as well as solution values at these vertices. As has been proved in

Section 1.1.2 minimising the energy functional and solving the corresponding VBVP are equivalent. Hence, where it exists, it is natural to use the energy functional as a quality measure for the approximate solution and to build an adaptive procedure which attempts to construct an approximation to the solution yielding a functional value as close to the global minimum as possible. An early attempt to include mesh adaptation into such a minimisation principle was due to Delfour et al. [42] who sought a finite element solution with free nodes for the variational formulation of a linear elliptic PDE.

This approach has been considered by other authors since then, see for example [36, 104]. Nevertheless, direct minimisation has many of the same drawbacks as the MFE and EP methods, i.e. a high computational cost and the need to avoid mesh tangling. Again, the coupling of the mesh and the discrete equations results in a large nonlinear algebraic system, even for linear problems, whose computational solution can be prohibitively expensive. A final difficulty concerning this technique is that it is often susceptible to falling into local minimum traps, which may well be far from a globally optimal mesh.

Chapter 2

Multilevel r and h-Refinement

In Chapter 1 we have briefly described a variety of different strategies for mesh refinement. The aim of this chapter is to provide details of our new multilevel r and h-refinement algorithm that combines local h-refinement with movement of vertices and with the local swapping of edges in order to attempt to obtain locally optimal finite element meshes for variational problems which accept the energy formulation. Details, and some applications of variational problems have been given in Section 1.1.2. In Section 2.1 we introduce the notation to be used in the rest of this thesis and consider how variational problems may be transformed into finite element problems. In Section 2.2 we describe our node movement algorithm, including local gradient calculations and the solution procedure for the local problems that arise when this approach is taken. This culminates in Subsection 2.2.4 where we summarise the complete algorithm for node movement. Section 2.3 then discusses the generalisation to systems of PDEs and then in Section 2.4 and 2.5 we describe details of the edge swapping and local h-refinement algorithms respectively. Finally, in Section 2.6, we put all of these components

together to arrive at our new hybrid algorithm.

2.1 Formulation of problem and notation

As explained above, in this thesis we restrict our attention to variational problems which accept an energy minimisation formulation, and which may therefore be posed in the following form.

$$\text{Find} \quad u \in V \quad \text{such that} \quad E(u) = \min_{v \in V} E(v) = \min_{v \in V} \int_{\Omega} F(\underline{x}, v, \underline{\nabla} v) d\underline{x}, \tag{2.1}$$

where V is a linear space consisting of functions defined on a bounded spatial domain $\Omega \subset \mathcal{R}^d$ such that $F : \mathcal{R}^d \times \mathcal{R} \times \mathcal{R}^d \rightarrow \mathcal{R}$, the energy density function, has a finite integral. In this work we are concerned with the construction of an approximation, say u^h, to the solution u. For the finite element method this continuous problem is replaced by a discrete problem whereby the infinite dimensional space V is replaced by finite dimensional subspace $V^h \subset V$. Hence, the discrete problem then takes the following form.

$$\text{Find} \quad u^h \in V^h \quad \text{such that} \quad E(u^h) = \min_{v \in V^h} \int_{\Omega} F(\underline{x}, v, \underline{\nabla} v) d\underline{x}. \tag{2.2}$$

Furthermore, it will be assumed that at least one isolated solution $u \in V$ exists for (2.1), and that this solution has sufficient regularity that it can be approximated satisfactorily by piecewise polynomials on appropriate finite element meshes. For the time being it will be assumed that n=2, and that the domain Ω is polygonal (however the case $n = 3$ with a polyhedral domain is considered in Chapter 5). Furthermore an arbitrary point in the

closure of Ω will be denoted by $\underline{x} = (x_1, x_2)$ and $d\underline{x} = dx_1 dx_2$ will denote an infinitesimal area in the domain Ω. The boundary of Ω will be denoted by $\partial\Omega$ and it will be assumed that this boundary is the union of two disjoint sets:

$$\partial\Omega = \partial\Omega_D \cup \partial\Omega_N.$$

Here $\partial\Omega_D$ denotes the Dirichlet part of the boundary and $\partial\Omega_N$ denotes the (possibly empty) Neumann part of boundary. As implied by (2.1), the functional is assumed to be of the form

$$E(v) = \int_\Omega F(\underline{x}, v, \underline{\nabla} v) d\underline{x}, \tag{2.3}$$

where the integrable function F is a continuously differentiable function of its arguments. In the finite element method the space V^h is constructed by subdividing the domain into non-overlapping computational cells, known as elements, and choosing a convenient set of piecewise polynomial functions. For this work however, only piecewise linear finite element trial functions are considered, since we are interested in r and h (rather then p) refinement. We use the term *triangulation* to refer to the set of elements. We consider only meshes of triangles in two dimensions (and tetrahedra in three dimensions) and we assume that all nodes lie at the vertices of triangles only (i.e. we do not allow so called *hanging nodes* in the mesh). The phrase "mesh optimisation" refers to the local minimisation of the discrete energy functional $E(u^h)$ over all allowable choices of the triangulations with a given number of nodes.

2.2 Node movement

As described in Section 1.5.3 node movement may be used to relocate the position of the nodes of the mesh such that some measure of the error $u - u^h$ is reduced. Our aim is to move the nodes such that the energy functional (2.3) is locally minimised. We therefore define a locally optimal mesh *with respect to position of nodes* for the finite element solution of (2.2) to be a mesh with the following property.

- There exists some number $\epsilon > 0$ such that if any node is displaced by any distance $\delta \leq \epsilon$ in any direction (subject to the constraint that a boundary node remains on the boundary and the domain is not altered) the finite element solution on the new mesh has an energy which is no less than the energy of the finite element solution on the locally optimal mesh with respect to the position of nodes.

When $v = u^h$, i.e. the solution of (2.2), the energy functional clearly depends upon the choice of V^h, making it an implicit function of the position of the nodes. As such, the position of nodes may be viewed as degrees of freedom for the minimisation of the energy functional, thus ensuring that the mesh optimisation problem is generally highly nonlinear. This nonlinearity implies that it will be almost impossible to find globally optimal positions for the nodes and so only locally optimal meshes are sought here[1]. Our node movement algorithm (which is a generalisation of [48] and [104]) achieves

[1] As we will demonstrate in later chapters however the quality of these local optima may be improved considerable through the use of a good algorithm

this through the use of an iterative procedure that moves only one vertex at a time. Hence if there are N nodes in a given triangulation, the nonlinear system consisting of $O(N)$ equations, which must be solved fully (i.e. all solutions found) in order to obtain a globally optimal solution, is replaced by a sequence of smaller problems, that each involve only $O(1)$ equations at each iteration. Our algorithm achieves this by considering each node in turn and combining a steepest descent method for the node relocation with the solution of a local variational problem to update the approximation.

For each node in the mesh, j say, we consider only those elements in the mesh which contain the node j, i.e. the polygon formed by the union of all triangular elements which have j as a vertex. We refer to this polygon as Ω_j. The aim of relocating node j is to find a new admissible position in Ω_j such that the energy functional, evaluated on the modified mesh, is reduced. Figure 2.1 shows a typical patch of elements surrounding a node to form a polygon.

Like the approach of [104] our algorithm seeks to reduce the energy functional monotonically by moving each node in turn until the derivative with respect to the position of each node is zero (in practice it should be less than some chosen threshold value). The algorithm is designed in such a way that changes resulting from moving a node affect only the interior of its patch. The algorithm consists of a number of sweeps through all nodes in the mesh until convergence is achieved. For each sweep each node is visited one at a time while keeping every other node fixed; thus yielding a kind of

nonlinear Gauss-Seidel iteration.

To find a new location for the node j, the local energy for the patch containing the node j is first calculated by the expression,

$$E_j(v) = \int_{\Omega_j} F(\underline{x}, v, \underline{\nabla} v) d\underline{x}, \tag{2.4}$$

where v is the current approximation to the solution on the domain. The algorithm then calculates the gradient of the energy functional with respect to the position of node j (this is described in detail in Section 2.2.2 below). The negative of the gradient is then used to provide the steepest descent direction, which we use to determine along which line the node should be moved within the patch to attempt to reduce the energy.

The node j can be moved anywhere on the line of steepest descent between the points $\underline{x}_j$ (initial position of the node j) and $\underline{x}_t$ (point of intersection of the steepest descent direction and the polygon containing the node j) for a valid triangulation to be maintained. However, as a form of under-relaxation, we impose a maximum admissible limit $\underline{x}_a$ up to which the node can be moved by multiplying the distance from its initial position to the point of intersection $\underline{x}_t$ by a factor w $(0 < w < 1)$ (see Figure 2.1). A one dimensional energy minimisation problem is then solved to find the point on the line between $\underline{x}_j$ and $\underline{x}_a$ where the energy will be minimum (with the solution value, say u_j, at j held fixed).

Once a new position for the node has been found the value of u_j must be updated by solving the local problem

$$\min_{v \in V_0^h} \int_{\Omega_j} F(\underline{x}, v, \underline{\nabla} v) d\underline{x}. \tag{2.5}$$

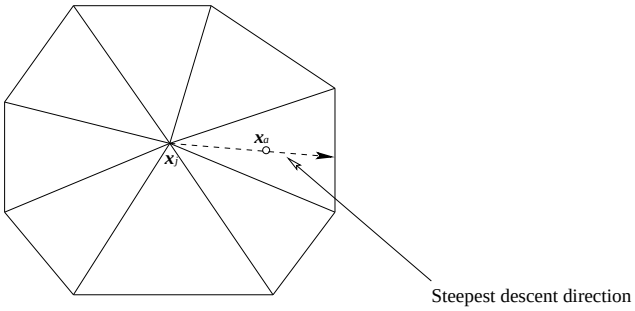

Figure 2.1: An illustration of local node movement

Here V_0^h is the space of piecewise linear functions on the patch Ω_j with fixed solution values on the boundary of Ω_j (given by the latest estimate of the solution). Hence this is a one dimensional problem when there is a single dependent variable. Once the update for the node j is complete the same process is undertaken for the next node. When each node has been visited the sweep is complete. Provided convergence has not been achieved the next sweep may then begin.

When the 2-norm of the gradient for each node is less than some predefined tolerance the position of that node is not updated in that sweep. Convergence is therefore considered to have been achieved when a complete sweep through the nodes takes place without updating any of their positions.

2.2.1 Movement of boundary nodes

It is necessary to distinguish between nodes on the boundary of Ω and interior nodes since a slight modification is required to the above movement algorithm in the case of boundary nodes. The interior nodes are permitted to move in any direction within the polygon surrounding them, while

boundary nodes may only be moved tangentially along the boundary and even this is subject to the constraint that the domain remains unchanged.

A boundary node n_{bj} in two dimensions has to satisfy the following two constraints if it is to be moved.

1. $n_{bj} \notin \overline{\partial\Omega_D} \cap \overline{\partial\Omega_N}$.

2. There is a straight line along which the vertex can be moved without modifying the computational domain. In particular, this excludes points at corners or on curved boundaries.

Where these constraints are not violated the downhill direction of motion along the boundary is easily computed by projecting the steepest descent vector onto the boundary. Although, in general, this projection does not coincide with the direction of steepest descent, it is expected that it will still provide a downhill direction.

The one dimensional minimisation in this direction is then computed as for any interior node. On Dirichlet boundaries the updated value of the solution is of course prescribed, however on any other type of boundary it must be computed by solving a local problem of the same form as of (2.5).

2.2.2 Derivation of gradient

In this section we provide an expression for the gradient of energy,

$$E = \int_\Omega F(\underline{x}, u^h, \underline{\nabla} u^h) d\underline{x}, \tag{2.6}$$

with respect to the position of a node j. Note that u^h is assumed to satisfy (2.2). The approach which we follow here to derive an expression for the

gradient of the energy functional is based on [63] and is slightly different from the approach adopted by [104]. However we shall then show that these apparently different expressions for the gradient of the energy functional are equivalent.

For simplicity and brevity we shall provide the derivation for polygonal domains in two dimensions, assuming that the problem has a single dependent variable and that the solution is zero on the boundary of Ω. However the extensions to polyhedral domains in greater than two dimensions, to problems with more than one dependent variable and to more general boundary conditions are all relatively straightforward.

To get the expression for the gradient of the energy functional we first note that the solution to (2.1) satisfies the corresponding Euler-Lagrange equation:

$$-F_{,2}(\underline{x}, u, \underline{\nabla} u) + \frac{d}{dx_k} F_{,3k}(\underline{x}, u, \underline{\nabla} u) = 0. \tag{2.7}$$

Here $F_{,i}(.,.,.)$ denotes differentiation with respect to the i_{th} dependent variable of F, other suffices refer to vector components (hence $F_{,3k}$ is the k_{th} component of $F_{,3}$) and repeated suffices denote summation from 1 to 2.

Let $M = (\underline{v}, \mathcal{C})$ be a mesh of non-overlapping triangles which exactly cover the domain Ω. Here $\underline{v} = (\underline{v}_1, ..., \underline{v}_N, \underline{v}_{N+1}, ..., \underline{v}_{N+B})$ is an ordered set of the position vectors of the vertices of the mesh (the first N being interior points and the remaining B points being on the boundary), and $\mathcal{C}$ is a list of all the edges. The finite element method seeks to approximate u, the solution of (2.7) by a piecewise linear function, u^h, defined on the mesh

$M = (\underline{v}, \mathcal{C})$. As the connectivity of the mesh remains fixed in our node movement algorithm, we shall refer to the mesh $M = (\underline{v}, \mathcal{C})$ only by the ordered set $\underline{v}$ for notational convenience. We may write our approximation u^h in the form

$$u^h(x) = \sum_{m=1}^{N} u_m \psi_m(\underline{x}, \underline{v})$$

where ψ_m is the usual piecewise continuous linear basis function on the mesh $\underline{v}$:

$$\psi_m(\underline{v}_n, \underline{v}) = \delta_{mn}, \qquad m = 1, ..., N; \qquad n = 1, ..., N + B.$$

Before deriving the expression for the gradient it is first necessary to establish some more notation and then to quote two preliminary lemmas. The proofs of these lemmas can be found in [66] and [61] respectively.

- Suppose $j \in \{1, \ldots, N\}$ is the number of an internal node of the triangulation of the domain Ω. Then we denote by $N(j)$ the number of elements in the triangulation that have this node as a vertex (i.e. in Ω_j). Further, for $t = 1, \ldots, N(j)$, let $T(j,t)$ be a unique ordering of these $N(j)$ elements which have a vertex at $\underline{v}_j$, let $\Omega_{T(j,t)}$ be the region occupied by the triangle numbered $T(j,t)$ and let $A_{T(j,t)}$ be the area of this region.

- Given any triangle within a finite element mesh we may represent the vertices of that triangle by a local numbering as $\underline{\hat{v}}_0, \underline{\hat{v}}_1$ and $\underline{\hat{v}}_2$.

- We may also define a standard triangle, $\triangle$, as the triangle whose vertices are $\underline{e}_0 = (0,0)^T, \underline{e}_1 = (1,0)^T$ and $\underline{e}_2 = (0,1)^T$.

- Now define a mapping from an arbitrary element within the triangulation onto that standard triangle by

$$\underline{\xi}(\underline{x},\underline{v}) = \sum_{\mu=0}^{2} \underline{e}_\mu \psi_\mu^l(\underline{x},\underline{v}) \tag{2.8}$$

where $\psi_\mu^l(\underline{x},\underline{v})$ is the usual linear basis function (but with a local numbering corresponding to a particular triangle) such that

$$\psi_\mu^l(\underline{v}_\nu,\underline{v}) = \delta_{\mu\nu}, \text{for} \quad \mu,\nu \in \{0,1,2\}.$$

- The inverse of the mapping (2.8) is

$$\underline{x}(\underline{\xi},\underline{v}) = \sum_{\mu=0}^{2} \underline{\hat{v}}_\mu \tilde{\psi}_\mu(\underline{\xi}), \tag{2.9}$$

where the $\tilde{\psi}_\mu(\underline{\xi})$ are the piecewise linear basis functions on the standard triangle such that $\tilde{\psi}_\mu(\underline{e}_\nu) = \delta_{\mu\nu}$. (Note that $\tilde{\psi}_\mu(\underline{\xi}) \equiv \psi_\mu^l(\underline{x}(\underline{\xi},\underline{v}),\underline{v})$.)

Lemma 2.1

$$\frac{\partial \psi_l}{\partial v_{md}} = -\psi_m \frac{\partial \psi_l}{\partial x_d} \qquad \text{for} \quad l = 1,...,N; \quad d = 1,2$$

hence

$$\frac{\partial u^h}{\partial v_{md}} = -\psi_m \frac{\partial u^h}{\partial x_d} \qquad \text{for} \quad d = 1,2.$$

Lemma 2.2

Given a triangle with vertices $\underline{\hat{v}}_0, \underline{\hat{v}}_1, \underline{\hat{v}}_2$ and area $A(\underline{\hat{v}}_0, \underline{\hat{v}}_1, \underline{\hat{v}}_2)$, then

$$\frac{\partial A}{\partial \hat{v}_{\nu d}} = \frac{\partial \psi_\nu^l}{\partial x_d} A \qquad \text{for} \quad \nu \in \{0,1,2\} \quad \text{and} \quad d \in \{1,2\}.$$

Now to get an expression for the gradient of the energy with respect to the nodal positions $\underline{v_j}$, assuming $d = 1, 2$, we have

$$\frac{\partial E}{\partial v_{jd}} = \frac{\partial}{\partial v_{jd}} \int_\Omega F(\underline{x}, u^h, \underline{\nabla} u^h) d\underline{x}.$$

As for node j the integral is zero on all the elements which do not contain this node we can write the above expression as

$$\begin{aligned}
\frac{\partial E}{\partial v_{jd}} &= \frac{\partial}{\partial v_{jd}} \sum_{t=1}^{N(j)} \int_{\Omega_{T(j,t)}} F(\underline{x}, u^h, \underline{\nabla} u^h) d\underline{x} \\
&= \sum_{t=1}^{N(j)} \frac{\partial}{\partial v_{jd}} \int_\triangle F(\underline{x}(\underline{\xi}, \underline{v}), u^h(\underline{x}(\underline{\xi}, \underline{v}), \underline{v}), \underline{D}(\underline{u}, \underline{v})) \left| \frac{d\underline{x}}{d\underline{\xi}} \right| d\underline{\xi} \\
&= \sum_{t=1}^{N(j)} \left\{ \int_\triangle F \frac{\partial}{\partial v_{jd}} \left| \frac{d\underline{x}}{d\underline{\xi}} \right| d\underline{\xi} + \int_\triangle F_{,1i} \frac{\partial x_i}{\partial v_{jd}} \left| \frac{d\underline{x}}{d\underline{\xi}} \right| d\underline{\xi} + \right. \\
&\quad \left. \int_\triangle F_{,2} \left[\frac{\partial u^h}{\partial x_k} \frac{\partial x_k}{\partial v_{jd}} + \frac{\partial u^h}{\partial v_{jd}} \right] \left| \frac{d\underline{x}}{d\underline{\xi}} \right| d\underline{\xi} + \int_\triangle F_{,3k} \frac{\partial D_k}{\partial v_{jd}} \left| \frac{d\underline{x}}{d\underline{\xi}} \right| d\underline{\xi} \right\}.
\end{aligned}$$

Here $\underline{x}(\underline{\xi}, \underline{v})$ is given by (2.9), $\triangle$ is the standard triangle, and D_l represents the value of $\frac{\partial u^h}{\partial x_l}$ restricted to the triangle $T(j,t)$. (Note that this value is independent of $\underline{x}$ since we are using piecewise linear finite elements.)

We may now make use of the fact that the Jacobian, $\left| \frac{d\underline{x}}{d\underline{\xi}} \right|$, in the above transformation is equal to $2A_{T(j,t)}$ on each triangle and so, by using Lemma 2.2, we can write,

$$\frac{\partial}{\partial v_{jd}} \left| \frac{d\underline{x}}{d\underline{\xi}} \right| = \frac{\partial \psi_j}{\partial x_d} \left| \frac{d\underline{x}}{d\underline{\xi}} \right|.$$

In addition we may use Lemma 2.1 and the fact that $u^h(\underline{x})$ is piecewise linear to deduce that, on each triangle $T(j,t)$,

$$\frac{\partial D_k}{\partial v_{jd}} = \frac{\partial}{\partial v_{jd}} \left[\frac{\partial u^h}{\partial x_k} \right] = \frac{\partial}{\partial x_k} \left[\frac{\partial u^h}{\partial v_{jd}} \right] = \frac{\partial}{\partial x_k} \left[-\psi_j \frac{\partial u^h}{\partial x_d} \right] = -\frac{\partial \psi_j}{\partial x_k} \frac{\partial u^h}{\partial x_d}.$$

We can now deduce that

$$\frac{\partial E}{\partial v_{jd}} = \sum_{t=1}^{N(j)} \int_\triangle \left\{ F \frac{\partial \psi_j}{\partial x_d} + F_{,1i} \delta_{id} \psi_j + F_{,2} \left[\frac{\partial u^h}{\partial x_k} \delta_{kd} \psi_j + \frac{\partial u^h}{\partial v_{jd}} \right] - \right.$$

$$F_{,3k}\frac{\partial\psi_j}{\partial x_k}\frac{\partial u^h}{\partial x_d}\Bigg\}\left|\frac{d\underline{x}}{d\underline{\xi}}\right| d\underline{\xi}$$

(using the fact that, by (2.9), $\frac{\partial x_i}{\partial v_{jd}} = \delta_{id}\psi_j$)

$$= \sum_{t=1}^{N(j)} \int_{\Omega_{T(j,t)}} \left\{ F\frac{\partial\psi_j}{\partial x_d} + F_{,1d}\psi_j - F_{,3k}\frac{\partial\psi_j}{\partial x_k}\frac{\partial u^h}{\partial x_d} \right\} d\underline{x}$$

(again using Lemma 2.1).

As we know that ψ_j is zero for all the elements which do not contain the node j, the integral in the above equation can be taken over the whole domain Ω. Rearranging the terms in the above equation we get,

$$\frac{\partial E}{\partial v_{jd}} = \int_\Omega \left[F\delta_{dk} - \frac{\partial u^h}{\partial x_d}F_{,3k} \right] \frac{\partial\psi_j}{\partial x_k} d\underline{x} + \int_\Omega F_{,1d}\psi_j d\underline{x}. \tag{2.10}$$

This expression may be used to provide the gradient of energy functional with respect to the position of nodes.

To demonstrate how the gradient with respect to position of nodes is computed in practice we consider the energy functional,

$$E = \int_\Omega F(\underline{x}, u^h, \underline{\nabla}u^h) d\underline{x} = \int_\Omega \left[\frac{1}{2}(\underline{\nabla}u^h)^2 - fu^h \right] d\underline{x}, \tag{2.11}$$

corresponding to Poisson's equation (see for reference (1.9) where v is replaced by u^h). For the energy density function $F(\underline{x}, u^h, \underline{\nabla}u^h)$ in (2.11) we have $F_{,1} = -\underline{\nabla}fu^h$, $F_{,2} = -f$, $F_{,3} = \underline{\nabla}u^h$. By plugging these values in (2.10) we can get the gradient of energy functional, corresponding to the Poisson's equation. This gradient vector may be assembled in a single pass through the elements.

We now prove here that the expression for the gradient of energy given by (2.10) is equivalent to the expression derived by Tourigny and Hulsemann

in [104]. They derive the following expression for the gradient of the energy functional with respect to the position of node j (expressed in the notation of [104]):

$$\begin{aligned}\frac{\partial E}{\partial \underline{v}_j} = \Phi_j \quad = \quad & \sum_{K\in\Omega_j}\left\{\int_{\partial K}\left[F(u,\underline{\nabla}u)\psi_j\underline{n} - \frac{\partial F(u,\underline{\nabla}u)}{\partial\underline{\nabla}u}\cdot\underline{n}\psi_j\underline{\nabla}u\right]ds + \right. \\ & \left.\int_K\left[\underline{\nabla}\cdot\left(\frac{\partial F(u,\underline{\nabla}u)}{\partial\underline{\nabla}u}\right) - \frac{\partial F(u,\underline{\nabla}u)}{\partial u}\right]\psi_j\underline{\nabla}u d\underline{x}\right\} + \\ & \int_{\partial\Omega_N\cap\partial\Omega_j} g\nabla u\psi_j ds.\end{aligned}$$

Here $\underline{n}$ is the unit normal vector, K is a triangle having node j as its vertex and Ω_j is the union of all triangles having node j as a vertex, as above. In the above expression it has been assumed that $F = F(u,\underline{\nabla}u)$ rather than $F(\underline{x},u,\underline{\nabla}u)$ as is considered for the derivation of (2.10). The last term in the expression is due to the presence of a Neumann boundary condition, and so we discard this as we have assumed in our derivation of (2.10) that the solution is zero throughout $\partial\Omega$. Also we replace u with u^h, a piece-wise linear function, in the above expression as, in our derivation of (2.10), u is assumed to be a piecewise linear function. Multiplying the above expression by an arbitrary constant vector $\underline{c}$ and rearranging terms we get,

$$\begin{aligned}\underline{c}\cdot\Phi_j \quad = \quad & \underline{c}\cdot\sum_{K\in\Omega_j}\left\{\int_{\partial K}F\psi_j\underline{n}ds - \int_K\frac{\partial F}{\partial u^h}\psi_j\underline{\nabla}u^h d\underline{x}\right. \\ & \left. - \int_{\partial K}\frac{\partial F}{\partial\underline{\nabla}u^h}\cdot\underline{n}\psi_j\underline{\nabla}u^h ds + \int_K\underline{\nabla}\cdot(\frac{\partial F}{\partial\underline{\nabla}u^h})\psi_j\underline{\nabla}u^h d\underline{x}\right\} \\ = \quad & \sum_{K\in\Omega_j}\left\{\int_{\partial K}F\psi_j\underline{c}\cdot\underline{n}ds - \int_K\frac{\partial F}{\partial u^h}\psi_j\underline{\nabla}u^h\cdot\underline{c}d\underline{x}\right. \\ & \left. - \int_{\partial K}\left(\psi_j(\underline{c}\cdot\underline{\nabla}u^h)\frac{\partial F}{\partial\underline{\nabla}u^h}\right)\cdot\underline{n}ds + \int_K\underline{\nabla}\cdot(\frac{\partial F}{\partial\underline{\nabla}u^h})\psi_j\underline{c}\cdot\underline{\nabla}u^h d\underline{x}\right\}.\end{aligned}$$

Applying Green's theorem on the boundary integral terms in the above

equation we get,

$$\begin{aligned}
\underline{c}\cdot\Phi_j &= \sum_{K\in\Omega_j}\left\{\int_K \underline{\nabla}\cdot(F\psi_j\underline{c})d\underline{x} - \int_K \frac{\partial F}{\partial u^h}\psi_j\underline{\nabla}u^h\cdot\underline{c}d\underline{x}\right.\\
&\quad \left. -\int_K \underline{\nabla}\cdot\left(\psi_j(\underline{c}\cdot\underline{\nabla}u^h)\frac{\partial F}{\partial\underline{\nabla}u^h}\right)d\underline{x} + \int_K \underline{\nabla}\cdot(\frac{\partial F}{\partial\underline{\nabla}u^h})\psi_j(\underline{c}\cdot\underline{\nabla}u^h)d\underline{x}\right\}\\
&= \sum_{K\in\Omega_j}\left\{\int_K\left[\underline{c}\cdot\underline{\nabla}(F\psi_j) - \frac{\partial F}{\partial u^h}\psi_j\underline{c}\cdot\underline{\nabla}u^h\right]d\underline{x}\right.\\
&\quad \left. -\int_K\left[(\underline{c}\cdot\underline{\nabla}u^h)\underline{\nabla}\cdot\left(\psi_j\frac{\partial F}{\partial\underline{\nabla}u^h}\right) - \underline{\nabla}\cdot\left(\frac{\partial F}{\partial\underline{\nabla}u^h}\right)\psi_j(\underline{c}\cdot\underline{\nabla}u^h)\right]d\underline{x}\right\}
\end{aligned}$$

Applying the chain rule we get,

$$\begin{aligned}
\underline{c}\cdot\Phi_j &= \sum_{K\in\Omega_j}\left\{\int_K\left[\underline{c}\cdot\underline{\nabla}F\psi_j + \underline{c}\cdot\underline{\nabla}\psi_j F - \frac{\partial F}{\partial u^h}\psi_j\underline{c}\cdot\underline{\nabla}u^h\right]d\underline{x}\right.\\
&\quad -\int_K\left[\underline{c}\cdot\underline{\nabla}u^h\underline{\nabla}\psi_j\cdot\frac{\partial F}{\partial\underline{\nabla}u^h} + \underline{c}\cdot\underline{\nabla}u^h\psi_j\underline{\nabla}\cdot\frac{\partial F}{\partial\underline{\nabla}u^h}\right.\\
&\quad \left.\left. -\underline{\nabla}\cdot\frac{\partial F}{\partial\underline{\nabla}u^h}\psi_j\underline{c}\cdot\underline{\nabla}u^h\right]d\underline{x}\right\}\\
&= \sum_{K\in\Omega_j}\int_K\left[\psi_j\underline{c}\cdot(\frac{\partial F}{\partial u^h}\underline{\nabla}u^h) + \psi_j\underline{c}\cdot(\frac{\partial F}{\partial\underline{\nabla}u^h}\underline{\nabla}\cdot\underline{\nabla}u^h) + \underline{c}\cdot\underline{\nabla}\psi_j F\right.\\
&\quad \left. -\frac{\partial F}{\partial u^h}\psi_j\underline{c}\cdot\underline{\nabla}u^h - (\underline{c}\cdot\underline{\nabla}u^h)\underline{\nabla}\psi_j\cdot\frac{\partial F}{\partial\underline{\nabla}u^h}\right]d\underline{x}.
\end{aligned}$$

Cancelling out the same terms with opposite sign and because $\underline{\nabla}\cdot\underline{\nabla}u^h = 0$ on an element we get,

$$\begin{aligned}
\underline{c}\cdot\Phi_j &= \sum_{K\in\Omega_j}\left\{\int_K \underline{c}\cdot\underline{\nabla}\psi_j F d\underline{x}\right.\\
&\quad \left. -\int_K \underline{c}\cdot\left(\underline{\nabla}u^h\left(\underline{\nabla}\psi_j\cdot\frac{\partial F}{\partial\underline{\nabla}u^h}\right)\right)d\underline{x}\right\}\\
&= \underline{c}\cdot\sum_{K\in\Omega_j}\int_K\left[F\underline{\nabla}\psi_j - \left(\frac{\partial F}{\partial\underline{\nabla}u^h}\cdot\underline{\nabla}\psi_j\right)\underline{\nabla}u^h\right]d\underline{x}.
\end{aligned}$$

But since this is true for any choice of constant vector $\underline{c}$ and ψ_j will be zero for all the elements which do not contain the j^{th} node, the integral

in the above equation can be taken over the whole domain Ω, so that the above equation can be written as,

$$\underline{\Phi}_j = \int_\Omega \left[F \underline{\nabla} \psi_j - \left(\frac{\partial F}{\partial \underline{\nabla} u} \cdot \underline{\nabla} \psi_j \right) \underline{\nabla} u^h \right] d\underline{x}.$$

By rearranging the terms and writing in subscript notation the above equation can be written as,

$$\underline{\Phi}_j = \int_\Omega \left[F \delta_{dj} - \frac{\partial u^h}{\partial x_d} F_{,3j} \right] \frac{\partial \psi_n}{\partial x_j} d\underline{x},$$

which is exactly the same as the expression (2.10) derived before, in the special case where $F = F(u^h, \underline{\nabla} u^h)$ rather than $F(\underline{x}, u^h, \underline{\nabla} u^h)$.

2.2.3 Solution of the local problem

In this section we demonstrate that by solving a sequence of local variational problems on a fixed mesh and updating the solution value for each node in a Gauss-Seidel fashion it is possible to obtain the solution to the finite element problem (2.2), at least in the case where F is quadratic. This observation is used to motivate our algorithm which, as described in the Section 2.1, aims to reduce the energy functional by moving each node and then solving a local Dirichlet form of the variational problem (2.2) for that node by considering only the patch Ω_j as the domain. Hence there is no need to solve the whole global problem (2.2) after moving each node.

For this demonstration we take a specific example by considering Poisson's equation,

$$-\underline{\nabla}^2 u = f \text{ in } \Omega, \qquad u = 0 \text{ on } \partial\Omega. \tag{2.12}$$

Let u^h be the finite element solution of this problem given by

$$u^h = \sum_{j=1}^{N} u_j \psi_j, \tag{2.13}$$

where u_j are the solution values at nodes and ψ_j are the usual basis functions. As we have seen in Section 1.1.4, applying the finite element method to (2.12) gives a system of linear equations,

$$\sum_j K_{ij} u_j = F_i, \quad i = 1, ..., N. \tag{2.14}$$

For this example of Poisson's equation the solution can also be determined by minimising the energy functional

$$I(\underline{u}) = E(u^h) = \frac{1}{2} \int_\Omega (\underline{\nabla} u^h)^2 d\Omega - \int_\Omega f u^h d\Omega, \tag{2.15}$$

where $\underline{u}$ is the vector of nodal unknowns $(u_1, u_2, ..., u_N)^T$. The necessary condition for the minimisation of $I(u_1, u_2, ..., u_N)$ is that

$$\frac{\partial I}{\partial u_1} = \frac{\partial I}{\partial u_2} = ... = \frac{\partial I}{\partial u_N} = 0. \tag{2.16}$$

These give N linear algebraic equations in the N unknowns $\underline{u}$ which are exactly the same as in (2.14). In order to solve this system of equations (2.16) by an iterative method, such as Gauss-Seidel iteration, one starts with an initial approximation and finds an update of the unknown u_j from the j^{th} equation by using the latest values of the other unknowns involved in that equation.

If we now consider the local Dirichlet variational problem for the patch Ω_j containing the node j we are required to minimise

$$I_j(u_j) = \frac{1}{2} \int_{\Omega_j} (\underline{\nabla} u^h)^2 d\Omega_j - \int_{\Omega_j} f u^h d\Omega_j, \tag{2.17}$$

with all values of $\underline{u}$ other than u_j held fixed. This implies solving $\frac{\partial I_j}{\partial u_j} = 0$, which is a single equation and is exactly the same as single Gauss-Seidel update of u_j in the system (2.16). Hence if we keep on solving the local problems for a *fixed mesh* and keep on updating the solution values for each node iteratively we obtain a sequence of approximations which coincides with that obtained using the Gauss-Seidel procedure to obtain the global solution of the variational problem.

2.2.4 Node movement algorithm

We have now introduced all of ingredients contained within our node movement algorithm. This is now presented.

Algorithm 2 *:*

Input: An initial mesh, an approximation of the solution of the variational problem (2.2) on that mesh, and a list of nodes available for movement.

1. *Find the global energy E_{old} of this solution.*
2. *Put $j = 0$.*
3. *If j is not equal to the last node available for movement put j equal to the next node available for movement else, go to step 10.*
4. *Solve the local variational problem for the node j by assuming the Dirichlet boundary conditions on the polygon Ω_j containing the node j and find local energy of the solution for the polygon Ω_j.*
5. *Find the gradient of the energy with respect to the position of node j by using expression (2.10).*
6. *If the magnitude of the gradient is less than the threshold chosen for this magnitude, go to step 3.*

7. *Find the maximum admissible limit along the straight line (in the direction of the negative gradient) in the polygon up to which the node can be moved.*

8. *Find the new position for node j using one dimensional minimisation of the energy for the polygon Ω_j, and then update u_j by solving the local problem with this new node position.*

9. *Go to step 3.*

10. *Find the global energy E_{new} of the latest estimate of the solution on the latest mesh.*

11. *If $abs(E_{new} - E_{old})$ is greater than the threshold value chosen set $E_{old} = E_{new}$ and go to step 2.*

12. *End of node movement.*

For each node j to be moved, we first minimise the energy over the position of the node within Ω_j and then minimise it over the solution value u_j by solving the corresponding local problem (2.5). This node movement algorithm is a major component of our new multilevel hybrid algorithm presented in Section 2.6. It not only moves the nodes to their optimal positions but also updates the solution values by solving the local problems (2.5) for each node. Furthermore, it is guaranteed to be energy reducing at each step.

2.3 Formulation for system of equations

All the above discussion of node movement and the derivation of the gradient with respect to node position, was for scalar problems (that is for one

dependent variable). However in later sections of this thesis we also demonstrate the application of our new hybrid algorithm to systems of equations. We briefly describe here the main changes that appear when dealing with systems of q partial differential equations in d space dimensions (in the following case we consider $q = d$). The energy minimisation problem will now become the following.

$$\text{Find} \quad \underline{u} \in \mathbf{V} \quad \text{such that} \quad E(\underline{u}) = \min_{\underline{v} \in \mathbf{V}} E(\underline{v}) = \min_{\underline{v} \in \mathbf{V}} \int_{\Omega} F(\underline{x}, \underline{v}, \underline{\nabla v}) d\underline{x}, \tag{2.18}$$

where $\mathbf{V}$ is a linear space consisting of the cartesian product of d functions defined on a bounded spatial domain $\Omega \subset \mathcal{R}^d$ such that $F : \mathcal{R}^d \times \mathcal{R}^d \times \mathcal{R}^{d \times d} \rightarrow \mathcal{R}$, the energy density function, has a finite integral. The corresponding discrete problem will be,

$$\text{Find} \quad \underline{u}^h \in \mathbf{V}^h \quad \text{such that} \quad E(\underline{u}^h) = \min_{\underline{v} \in \mathbf{V}^h} \int_{\Omega} F(\underline{x}, \underline{v}, \underline{\nabla v}) d\underline{x}, \tag{2.19}$$

where $\mathbf{V}^h \subset \mathbf{V}$. For the node movement, the new position for each node is again found by solving a one dimensional energy minimisation problem in the direction of steepest descent (see below). After moving the node j, the solution vector is updated by solving the local variational problem

$$\min_{v \in \mathbf{V}_0^h} \int_{\Omega_j} F(\underline{x}, \underline{v}, \underline{\nabla v}) d\underline{x} \tag{2.20}$$

Here $\mathbf{V}_0^h \subset \mathbf{V}^h$ is the space of piecewise linear functions on the patch Ω_j with fixed solution values on the boundary of Ω_j (given by the latest estimate of the solution). To get the expression for the gradient of the energy functional, for system of equations we note that the solution to (2.18) satisfies the

corresponding Euler-Lagrange equations

$$-F_{,2m}(\underline{x},\underline{u},\underline{\nabla u}) + \frac{d}{dx_k}F_{,3mk}(\underline{x},\underline{u},\underline{\nabla u}) = 0, \quad \text{for m=1,...,d.} \tag{2.21}$$

Here $F_{,i}(.,.,.)$ denotes differentiation with respect to the i_{th} dependent variable of F, other suffices represent tensor components in the usual manner and repeated suffices denote summation from 1 to d. The generalisation of (2.10) for the gradient of the energy with respect to position of node j, now becomes

$$\frac{\partial E}{\partial v_{je}} = \int_\Omega \left[F\delta_{ek} - \frac{\partial u^h_m}{\partial x_e}F_{,3mk} \right] \frac{\partial \psi_j}{\partial x_k} d\underline{x} + \int_\Omega F_{,1e}\psi_j d\underline{x}. \tag{2.22}$$

Here the index e is for dimension 1 to d and repeated suffices are again summed from 1 to d.

2.4 Edge swapping

The definition of mesh optimisation presented in Section 2.2 refers to "all allowable choices of the triangulation with a given number of nodes". The algorithm in the previous section considers only the location of the nodes in the triangulation. In this section we consider the positions of the edges (i.e. the topological connectivity of the mesh) and how this may be optimised. We define a locally optimal mesh *with respect to the position of edges* for the finite element solution of (2.2) to be a mesh with the following property.

- If the position of any internal edge in the mesh is swapped (provided the triangulation remains valid) then the finite element solution on this modified triangulation has an energy which is no less than the energy of the finite element solution on the unmodified mesh.

A brief introduction to how edges are swapped (for two dimensional triangulation) is given in Section 1.5.3 and is shown in Figure 2.2. However edge swapping in three dimensions is far more complicated than in two dimensions and we shall discuss edge or face swapping in three dimensions in Chapter 5.

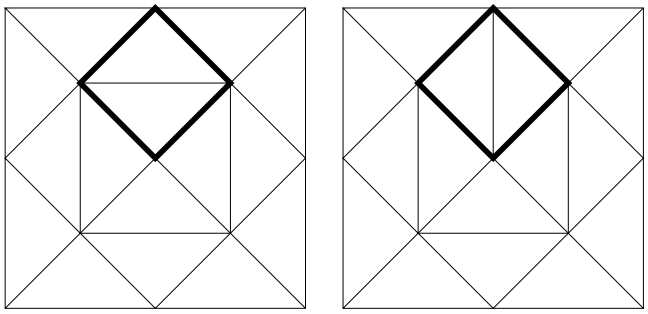

Figure 2.2: An illustration of the modification of a mesh by the swapping of a single edge.

The algorithm which we use for edge swapping is based on Ripa and Schiff [91]. It is undesirable to recompute the global solution after each change to the mesh, as this will increase the computational cost so much that the whole approach becomes useless. In [91] it is demonstrated that it is possible to improve the mesh topology significantly (i.e. reduce the total energy) by applying a simple edge swapping algorithm on the interpolant of the current solution, thus eliminating the need to solve the global problem repeatedly. A loop through each of the internal edges is completed and for each swappable edge the local energy in the two triangles on either side is computed. The edge is then swapped in the manner as described above and the new local energy over the two triangles on either side is computed. If this energy is less than the original local energy on the quadrilateral then

the edge swap is kept; otherwise it is rejected. Once the loop through each of the edges has been completed it is repeated until there is an entire pass for which no edges are swapped. At this point convergence to a locally optimal mesh with respect to the position of the edges has been achieved. Note that just like the node movement algorithm this approach reduces the global energy monotonically.

2.4.1 Edge swapping algorithm

We now present the edge swapping algorithm that is used for the numerical results presented in Chapter 3. For the three dimensional case in Chapter 5 a significantly modified form of this algorithm (where faces are also swapped) is used and discussed there.

Algorithm 3 *: Input: An initial mesh, an approximation of the solution of variational problem (2.2) on that mesh, and a list of internal edges.*

1. *Find the global energy E_{old} of this solution.*
2. *Put $e_j = 0$.*
3. *If e_j is not equal to the last internal edge put e_j equal to the next internal edge, else go to step 9.*
4. *If the quadrilateral formed by the two triangles which share the edge e_j is not convex go to step 3.*
5. *Find the local energy E_1 of the quadrilateral formed by the two triangles which share the edge e_j.*
6. *Swap the edge and find the local energy E_2 of this new quadrilateral.*
7. *If $E_2 < E_1$ keep the swapped edge, else reject the swap.*

8. *Go to step 3.*

9. *Find the global energy E_{new} on the latest mesh.*

10. *if $abs(E_{new} - E_{old})$ is greater than the threshold value chosen set $E_{old} = E_{new}$ and go to step 2.*

11. *End of edge swapping.*

A very small value is chosen as the threshold for the edge swapping algorithm (step 9) which effectively ruled out any edge swap after the application of this algorithm, hence this is in agreement with the definition for a converged mesh given above in this section. Node movement and edge swapping are the two components of our r-refinement strategy which is combined with local h-refinement to form our hybrid algorithm.

2.5 Local h-refinement

The main difficulty with the mesh optimisation strategy introduced in the previous sections is that it is impossible to know *a priori* how many nodes or elements will be required in order to get a sufficiently accurate finite element solution to any given variational problem. For this reason some form of h-refinement is necessary in addition to the above use of r-refinement.

Following the pattern of the rest of this chapter we discuss here local h-refinement for two dimensional problems only (for three dimensional problems it will be discussed in Chapter 5). Two different local h refinement algorithms have been widely used for two dimensional problems. The most popular one is the combination of *red* (regular) and *green* (irregular)

refinements which have been proposed by Bank et al. in [22] and then implemented into the well known multigrid code PLTMG [20]. A red refinement subdivides all triangles which are to be refined into four children by joining the mid points of all the three edges as illustrated in the top half of Figure 2.3. Green refinements are only used to close the triangulations (to remove the "hanging nodes") and consist of simple bisections of the neighbouring elements by connecting the hanging nodes to the opposite vertices of the neighbouring elements (again see Figure 2.3). The second refinement method is based on bisection only (see e.g. [25, 92]), as shown in the bottom half of Figure 2.3. The edge to be bisected is important and can be chosen on some geometric criteria or on some directional refinement criteria. To remove any hanging nodes, the same green refinement strategy is used as discussed above.

We have implemented both of these refinement schemes i.e. one-to-four and one-to-two. For our one-to-four refinement algorithm we adopted the strategy of [36]. We first mark those elements to be red refined. If any element is not marked for red refinement but two of its edges are marked for refinement (due to neighbours), we mark it for red refinement. After applying the red refinement green refinement is applied to remove any hanging nodes. For our one-to-two refinement algorithm we adopted the strategy of [92] i.e. for each element to be refined we choose the longest edge for bisection. Hanging nodes left at the end are removed by bisecting the neighbouring elements (see Figure 2.3).

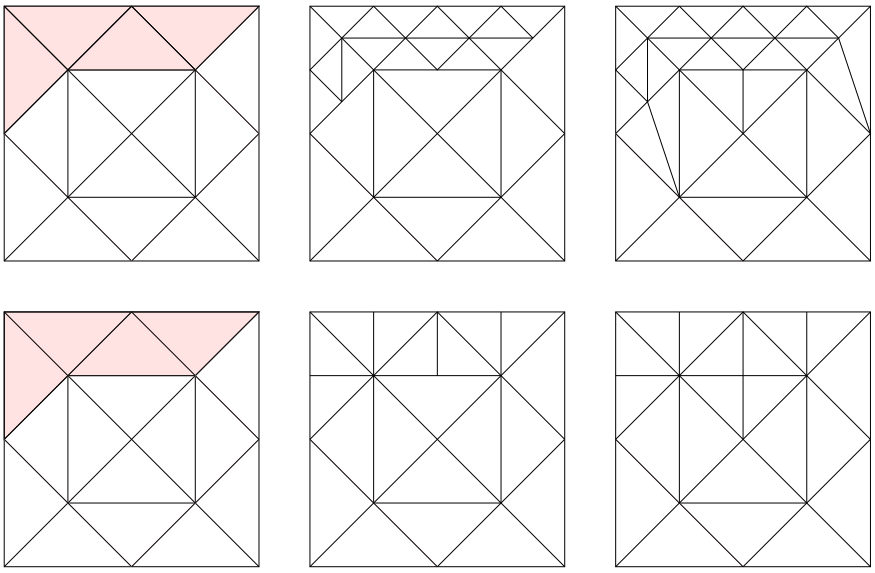

Figure 2.3: An illustration of the refinement of certain (shaded) elements of a mesh using one-to-four subdivision (top) and one-to-two subdivision (bottom).

There are at least two strategies for applying local adaptivity. The first subdivides all elements for which the error is greater than $X\%$ of the largest error value calculated on any single element. The second subdivides $X\%$ of the elements (e.g. by making a list of elements in descending order with respect to error magnitude, and then subdividing the first $X\%$ of the elements in this list). Typically X is chosen to be somewhere between 40 and 80. We have used the former approach based on the energy over each element rather than the error (i.e. using energy as the error indicator). The solution values for the newly inserted nodes are found by linear interpolation; i.e. by taking the average of the solution values at the two ends of the edge which has been bisected.

2.6 Multilevel hybrid algorithm

Combining the r-refinement and h-refinement approaches in an appropriate manner should allow locally optimal meshes to be obtained which are better (in terms of energy minimisation) than using either strategy alone. There are many possible ways to do this and we conclude this chapter with an outline of the approach that we adopted for our hybrid algorithm. A more detailed explanation of this approach is given in Chapter 3.

We begin with a very coarse mesh which is optimised using r-refinement. Once convergence with respect to the position of the nodes has been achieved a further reduction in the energy of the solution is sought by applying the edge swapping algorithm. After reaching convergence with respect to edges the grid is no longer likely to be locally optimal with respect to the position of nodes, since the edge swapping will generally cause the node locations to become suboptimal. Hence it is necessary to alternate between the node movement and the edge swapping algorithms until the whole process converges. The down hill nature of each step in the process guarantees that this will eventually occur (since we assume a solution to (2.1) exists and we are approximating this solution from above).

At this stage we consider applying local h-refinement, as a locally optimal mesh with a given number of nodes may not be adequate for obtaining a solution of a desired accuracy. By applying h-refinement to the locally optimal mesh we get a new mesh at the next level of refinement. In general this is not locally optimal so we again apply the node movement and edge

swapping algorithms alternately to obtain a locally optimal mesh at this new level. Hence, we are applying a hierarchical approach: starting with a coarse grid and then optimising, refining, optimising, refining etc., to get a final optimised mesh with a desired accuracy and a minimal number of nodes.

There are of course other ways in which h-refinement might be combined with r-refinement to produce a hybrid algorithm. Our experience suggests that a robust approach is to always refine an optimised mesh and then to interpolate the coarser solution onto the refined mesh as a starting point for the next level of optimisation. Details of this observation are presented in Chapter 3.

Having provided an overview the algorithms for r and h refinement we now present our new hybrid algorithm explicitly.

Algorithm 4 *:*

Input: An initial coarse mesh which can reasonably approximate the solution of the variational problem (2.2).

1. *Solve the problem on the initial mesh to get an approximation* u^h *on this mesh.*
2. *Apply the node movement, Algorithm 2.*
3. *Calculate the global energy* E_{old} *for the current approximation* u^h *on the current mesh.*
4. *Apply the edge swapping, Algorithm 3.*
5. *Apply the node movement, Algorithm 2.*

6. *Calculate the global energy E_{new} for the current approximation u^h on the current mesh.*

7. *If $abs(E_{new} - E_{old})$ is greater than the threshold value chosen set $E_{old} = E_{new}$ and go to step 4.*

8. *If the desired accuracy achieved go to step 10.*

9. *Apply local insertion algorithm and go to step 2.*

10. *End of mesh refinement process.*

Output: *A refined mesh and solution u^h with a desired accuracy.*

The above algorithm is entirely local, i.e. there is no need to solve any global finite element systems at intermediate levels. After every step of edge swapping and local h-refinement there is a step of node movement which not only moves nodes but also has the effect of updating the solution estimate. However global solves can be added to the algorithm as part of Step 9, local refinement. This is discussed in Chapter 3. There is also an intermediate option of doing some local solves without moving nodes to get better initial approximation to the solution immediately after local refinement. This may be cheaper than applying global solves and such local solves can eliminate the need of any global solves at all in the algorithm (even for the initial coarse mesh).

Chapter 3

Implementation and Numerical Results in 2-Dimensions

In this chapter we describe implementation details of our hybrid algorithm and some numerical results using this algorithm. This chapter has three main sections. In Section 3.1 we define our notion of mesh quality and discuss the choices available within the algorithm regarding this quality. We consider specific strategies which we have chosen during the implementation to achieve best quality meshes. In Section 3.2 we provide the details of the numerous adjustable parameters used in our hybrid algorithm. Possible ranges of values for these parameters are given there, along with the specific values used in Section 3.3, which is devoted to sample numerical results. Four test problems are considered in order to show the most important features of the new hybrid algorithm. For each of the test problems comparison has been made between our hybrid algorithm and more conventional adaptive strategies.

3.1 Mesh quality

Our principle aim in this thesis is to achieve meshes of the best possible quality, i.e. *meshes having as small a number of elements as possible for a desired accuracy (with respect to the energy of the solution on these meshes).* At this stage it is not necessarily our intention to make our hybrid algorithm the most efficient algorithm (i.e. to achieve a desired accuracy with the minimum possible number of floating point operations), instead we are concerned with what is achievable in terms of the meshes produced and the robustness of the hybrid algorithm. Hence we do not consider the computational costs involved in achieving these meshes in this chapter (but see Chapter 4 for a discussion of the main cost issues), however we do try to keep them feasible.

The main ingredients and the mathematical basis of our hybrid algorithm have already been presented in Chapter 2. Many possible choices occur when these ingredients are implemented using a computer program. These possible choices directly affect the quality of the mesh achieved, and usually also the overall computational expense. These various possible choices within the hybrid algorithm are explained below.

3.1.1 Exact / Inexact line search

Our node movement algorithm involves the solution of a one dimensional energy minimisation problem (2.5) for each node, i.e., to find a point on the line of steepest descent, where the node should be moved to minimise the

local energy functional (2.4), while holding the solution value fixed. There is another possibility of moving the nodes by solving a two dimensional energy minimisation problem. That is to minimise the local energy (2.4) over the position of nodes and the solution value by modifying the steepest descent direction to get a gradient which reduces this extra variable also. However, to solve a two dimensional minimisation problem is computationally expensive and to keep the computational cost feasible we do not experiment with this.

To solve a one dimensional energy minimisation problem there are two possible choices for this: exact or inexact line minimisation. An exact line search involves more computational work, but it may provide a better position of the node for the local energy minimum. Conversely, an inexact line search requires less computational effort but the new position of node found may not be as good as via an exact line search. The motivation behind using an inexact line search is the fact that, since this is just an intermediate step, it is not really essential that the new position found for a node be optimal at each iteration. Instead we expect to get this eventually as our algorithm relocates each node iteratively, reducing the energy monotonically. We now provide implementation details for each of these approaches.

Let $\underline{n} = -\underline{\nabla} E(\underline{x}_j)/\|\underline{\nabla} E(\underline{x}_j)\|$ be the unit vector in the direction of steepest descent for the node j ($\underline{n} = \underline{0}$ if $\underline{\nabla} E(\underline{x}_j) = \underline{0}$), where $\underline{\nabla} E(\underline{x}_j)$ is the gradient of the energy with respect to the position, $\underline{x}_j$, of node j (2.10). Let

s_{max} be the distance between the points $\underline{x}_j$ and $\underline{x}_a$. Where $\underline{x}_a$ is the point on the line of steepest descent up to which the node j is allowed to move. We have to find a new position,

$$\underline{x}_j^{new} = \underline{x}_j + s * \underline{n}, \tag{3.1}$$

such that the local energy functional (2.4) is minimised, where s will take its value in the interval $[0, s_{max}]$. If there are more than one minima in this interval we aim to choose the one closest to $s = 0$.

The algorithm that we use for exact line searches is based on the bisection method, taken from [110], and is given as follows.

Algorithm 5 *:*

Input: Node j, its position $\underline{x}_j$, the point $\underline{x}_a$ on the line of steepest descent up to which the node j is allowed to move, the unit vector $\underline{n}$ in the direction of the steepest descent for the node j and the local energy (2.4) $E_j(\underline{x}_j)$ for the node j positioned at $\underline{x}_j$.

1. *Set $itn = 0$ and $\underline{x}_j^{old} = \underline{x}_j$.*
2. *$itn = itn + 1$.*
3. *If $itn > 100$, set $\underline{x}_j = \underline{x}_j^{old}$ and stop.*
4. *Set $\underline{x}_m$ as the mid point of segment $\underline{x}_j$ and $\underline{x}_a$.*
5. *Set two real numbers $D1 = \underline{n} \cdot \underline{\nabla} E_j(\underline{x}_m)$ and $D2 = \underline{n} \cdot \underline{\nabla} E_j(\underline{x}_a)$ as the scalar product of $\underline{n}$ and the gradients of energies with node j positioned at $\underline{x}_m$ and $\underline{x}_a$ respectively.*
6. *If $[D1 > 0]$ then*

   $\underline{x}_a = \underline{x}_m$

go to step 2

end if (see Figure 3.1 (top left)).

7. *If [abs(D1) < tolerance] then*

if $[E_j(\underline{x}_m) < E_j(\underline{x}_j)]$ *then*

$\underline{x}_j = \underline{x}_m$

stop

end if (see Figure 3.1 (top right))

if $[E_j(\underline{x}_m) \geq E_j(\underline{x}_j)]$ *then*

$\underline{x}_a = \underline{x}_m$

go to step 2

end if.

8. *If [D1 < 0 and D2 > 0] then*

if $[E_j(\underline{x}_m) < E_j(\underline{x}_j)]$ *then*

$\underline{x}_j = \underline{x}_m$

go to step 2

end if (see Figure 3.1 (bottom left))

if $[E_j(\underline{x}_m) \geq E_j(\underline{x}_j)]$ *then*

$\underline{x}_a = \underline{x}_m$

go to step 2

end if.

9. *If [D1 < 0 and D2 < 0] then*

if $[E_j(\underline{x}_a) < E_j(\underline{x}_j)]$ *then*

$\underline{x}_j = \underline{x}_a$

stop

end if (see Figure 3.1 (bottom right))

if $[E_j(\underline{x}_a) \geq E_j(\underline{x}_j)]$ *then*

$\underline{x}_a = \underline{x}_m$ *then*

go to step 2

end if.

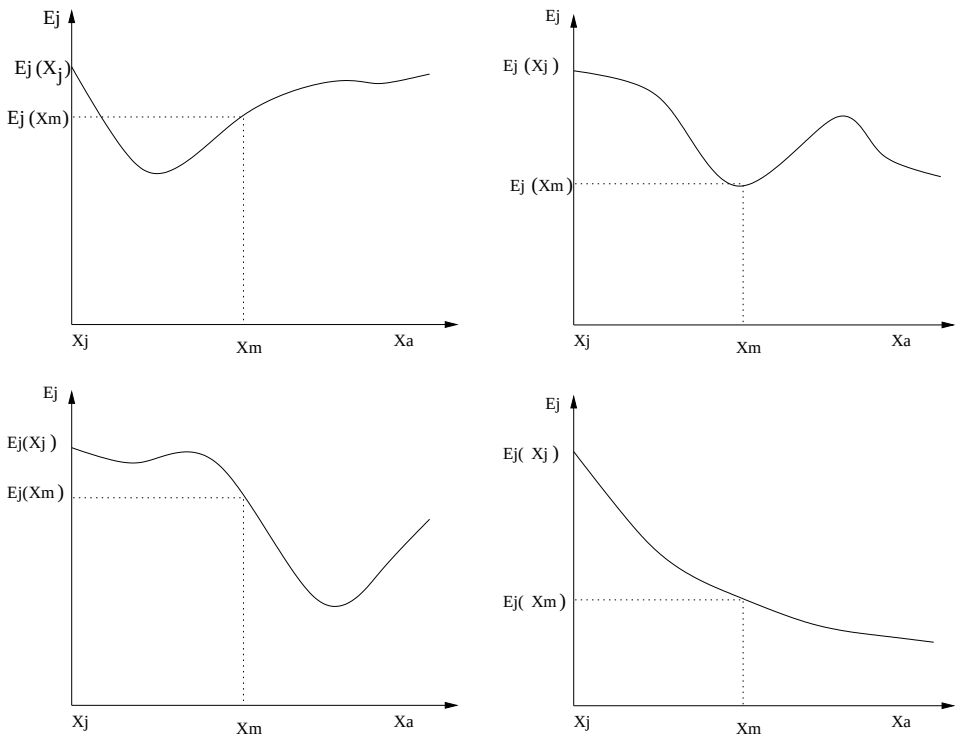

Figure 3.1: Some possibilities for the exact line search.

The algorithm that we use for inexact line searches is based on an idea taken from Freitag and Gooch [48]. They move the nodes to minimise some geometrical quality measure so as to improve the geometric quality of the mesh (see Section 1.4.4). We generalised this idea and used it to minimise the energy functional. The idea works by fitting a quadratic between the points $\underline{x}_j$ and $\underline{x}_a$. The minimum of this quadratic is then found and the node j is moved to this minimum provided it lies between the two points $\underline{x}_j$ and $\underline{x}_a$. If the minimum lies beyond $\underline{x}_a$ then node j is only moved to $\underline{x}_a$. In each case this relocation will depend upon the condition that it reduces the energy, otherwise the distance of $\underline{x}_a$ is reduced to half of its previous distance from $\underline{x}_j$, and the whole process is repeated. The algorithm for the inexact line minimisation is therefore as follows.

Algorithm 6 *:*

Input: Node j, its position $\underline{x}_j$, the point $\underline{x}_a$ on the line of steepest descent up to which the node j is allowed to move, the unit vector $\underline{n}$ in the direction of the steepest descent for the node j and the local energy (2.4) $E_j(\underline{x}_j)$ for the node j positioned at $\underline{x}_j$.

1. *Set $itn = 0$ and $\underline{x}_j^{old} = \underline{x}_j$.*

2. *$itn = itn + 1$.*

3. *If $itn > 10$ set $\underline{x}_j = \underline{x}_j^{old}$ and stop.*

4. *Set $E_0 = E_j(\underline{x}_j)$ and $E_a = E_j(\underline{x}_a)$.*

5. *Define a quadratic $\overline{E}(s) = ps^2 + qs + r$, where the real numbers p,q and r are to be determined such that $\overline{E}(0) = E_0$, $\overline{E}(s_{max}) = E_a$ and $\overline{E}'(0) = \underline{n} \cdot \underline{\nabla} E_j(\underline{x}_j)$.*

6. *Find the minimum of the quadratic by putting $E'(s) = 0 \Rightarrow s = -q/2p$ (see Figure 3.2 (left)).*

7. *If $[p \leq 0]$ then*

   if $[(E_j(\underline{x}_a) < E_j(\underline{x}_j))]$ then

   $\underline{x}_j = \underline{x}_a$

   stop

   end if (see Figure 3.2 (right))

   if $[(E_j(\underline{x}_a) \geq E_j(\underline{x}_j))]$ then

   $\underline{x}_a = \underline{x}_a - (\underline{x}_a - \underline{x}_j)/2$

   go to step 2

   end if.

8. *If $[p > 0]$ then*

   $\underline{x}_j^{new} = \underline{x}_j + s * \underline{n}$ where

   $s = min\{-q/2p, distance\ between\ \underline{x}_j\ and\ \underline{x}_a\}$,

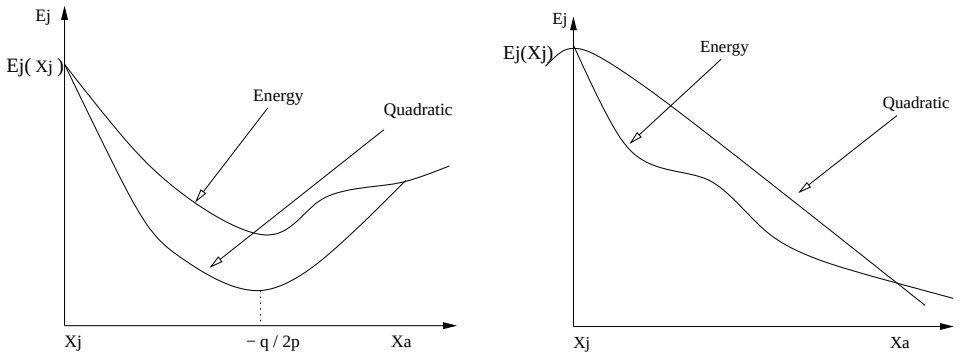

Figure 3.2: Some possibilities for the inexact line search.

if $[E_j(\underline{x}_j^{new}) < E_j(\underline{x}_j)]$ *then*

$\underline{x}_j = \underline{x}_j^{new}$

stop

end if

if $[E_j(\underline{x}_j^{new}) \geq E_j(\underline{x}_j)]$ *then*

$\underline{x}_a = \underline{x}_a - (\underline{x}_a - \underline{x}_j)/2$

go to step 2

end if.

In both of the above algorithms we allow a node to be displaced only if this reduces the local energy, otherwise its previous position is retained: with the assumption that this will be improved in the next node movement sweep. A comparison of both the line minimisation strategies with respect to the quality of meshes achieved and the computational work involved in each is given in Chapter 4. Generally, exact line search provides better quality meshes compared to using an inexact line search, however this involves more computational work. Since our aim in this chapter is to achieve the best possible quality meshes, we use exact line minimisation for the test problems presented in Section 3.3.

3.1.2 One-to-two and one-to-four element refinement

For the local refinement part of our two dimensional algorithm we have implemented both one-to-two and one-to-four element refinement schemes as discussed in Section 2.5. Each of these have their benefits and their drawbacks. For the one-to-two refinement scheme the geometric quality of the mesh (as given in Section 1.4.4) deteriorates as the number of local insertions increases. Long and thin elements may appear more frequently than when using the one-to-four refinement scheme and this can reduce the geometric quality of the mesh. However, for some problems long and thin elements can be beneficial for the accurate approximation of some sharp features of the solution. In this case the one-to-two refinement scheme can perform better. Also the number of nodes inserted in one pass of the one-to-two refinement scheme is much less than with the one-to-four refinement scheme and it may be possible to achieve a desired accuracy with fewer elements.

With the one-to-four refinement scheme the geometric quality of the mesh can be better maintained. However, more elements may be required to obtain a desired accuracy, as compared to the one-to-two refinement scheme. For our test problems presented in Section 3.3, we have used both of the approaches discussed here for comparison.

3.1.3 Local solves vs. global solves

It is shown in Section 2.2.3 that by solving a sequence of local problems iteratively on a fixed mesh it is possible to obtain a global solution of the

minimisation problem (2.2) on that mesh. Our hybrid algorithm is therefore designed in such a way that it never needs a global solve of the problem, except possibly on the initial fixed coarse mesh. Instead, it updates the nodal solution values during the node movement step by solving the local problems (2.5) for each node iteratively immediately after the position of that node has been updated. After every application of the edge swapping, Algorithm 3, and the local h-refinement step, a node movement step is undertaken which has the effect of updating the solution estimate.

It has been observed however that, although not necessary to achieve convergence, if the global problem is solved at intermediate points in the Algorithm 4 the convergence of the optimisation process may be accelerated. Furthermore, this may lead to a different down hill energy path being followed which can lead to a desired accuracy being obtained with fewer elements or with fewer refinement levels. Since it is our intention to achieve the best quality mesh that we can, even at the expense of some additional computational work, we modify the hybrid algorithm to incorporate a global solve after every step of local h-refinement. We consider the issue of the quality of the mesh achieved and the additional computational work involved through the inclusion of such global solves in Chapter 4.

3.1.4 Order of nodes and edges

In our node movement algorithm, the nodes which are available for movement are moved one by one, in a Gauss Seidel fashion, to reduce the energy functional. The order in which the nodes should be moved is significant as

changing this order may lead to convergence to a different local minimum and thus produce a change to the converged mesh and the solution obtained. There are many possible strategies which can be used to determine the order in which we update the nodes. One possible strategy is to sort the nodes by the magnitude of the gradients with respect to their positions, and then update them in descending order of the magnitude of this gradient. Alternatively we could sort them by the value of the initial local energy for each node and then update the nodes in descending order of their local energy. Perhaps the easiest choice is simply to update them in lexicographic order.

For the test problems considered in Section 3.3 we sort the nodes by the magnitude of their gradients at the start of the sweep and update them in descending order of this magnitude. We have adopted this strategy as a result of numerical tests which indicate that the quality of the meshes produced were at least no worse than those from any of the other strategies considered. It is to be noted that after moving the first node the sorted list may not remain valid. However, to sort the nodes again after moving each node is likely to increase the computational cost too significantly.

Similarly, for the edge swapping, Algorithm 3, we swap internal edges by considering them one by one. In contrast to node movement, Algorithm 2, however it is observed that changing the order in which the edges are visited does not produce such a significant change in the converged mesh. However, we do note that if we sort the edges by the energy of the quadrilateral in which they are contained, then swapping first the edge of that quadrilateral

which has the largest energy and so on, appears to reduce the number of iterations needed to converge. This must be balanced against the additional computational work that is needed to sort the edges of course. For the test problems featured in Section 3.3 of this chapter we sort the edges with respect to the energy of the quadrilateral containing each edge. The quality of the meshes achieved and the computational cost associated with sorting the nodes and the edges is discussed in Chapter 4.

3.1.5 Order of refinement strategies

Our hybrid algorithm combines r and h-refinement. There are many possible orders in which these refinement strategies can be combined. The classical approach to refinement typically uses h-refinement first, to create a mesh with reasonable topology, and then applies node movement for fine tuning [23]. However we apply these algorithms in the reverse order for the reasons explained below.

We start with a coarse initial mesh, which can crudely approximate the solution. It is highly unlikely that such a coarse initial mesh will permit all of the sharp features of the solution to be captured. If we apply local h-refinement first then it is possible that some of the regions where the solution variation is very sharp may be missed and the majority of the new nodes may be inserted in regions where the solution variation is not as sharp as elsewhere. Our hope is that if we first apply the node movement algorithm then when we come to do h-refinement the base mesh will be the best possible for its size and so the possibility of picking out the correct

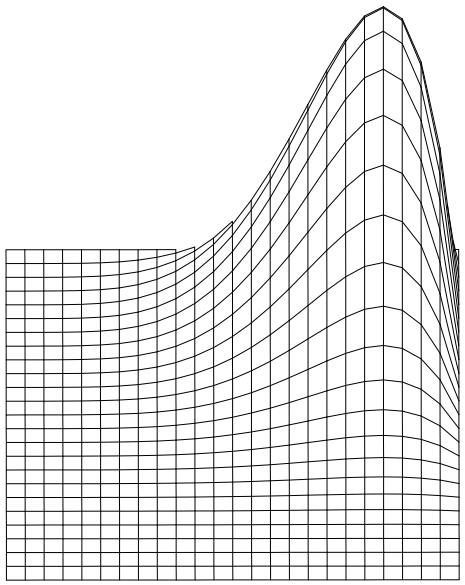

Figure 3.3: $U = 10^3 x^5(1-x)y^5(1-y)$

regions in which to refine the mesh will be increased.

We observe this effect during a typical test problem. The problem is such that it has two regions of solution variation, one is very sharp and the other is less sharp (see Figure 3.3). First we applied one-to-four local h-refinement on the initial coarse mesh of 128 elements by refining only those elements whose local energy exceeded $X = 40\%$ of the maximum local energy on any element. As the element size in the region of sharp solution variation is so large, the local refinement scheme is unable to detect the need for refinement in that region, however it does detect the region where the solution variation is less sharp. As a result all of the refinement takes place in this region (see Figure 3.4), which is far from ideal. When we optimise this mesh (solving Poisson's equation) the energy of the solution on that mesh is -55.878820. When local h-refinement (one-to-four with $X = 40\%$) is applied after node movement however, the realignment of the initial mesh

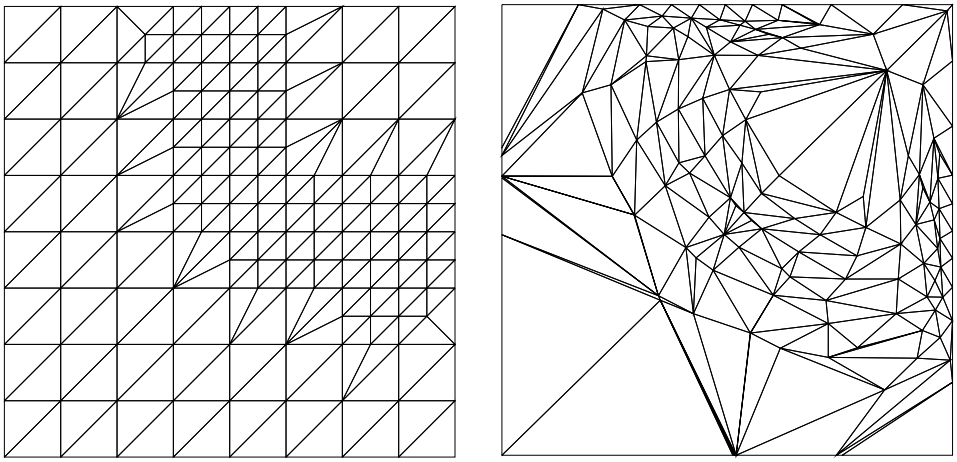

Figure 3.4: Initial coarse mesh after applying local h-refinement and corresponding locally optimised mesh of 248 elements.

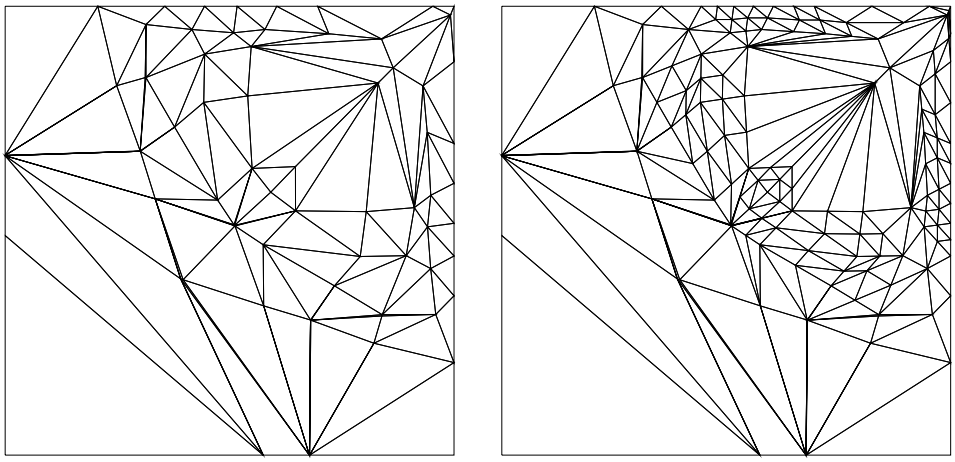

Figure 3.5: Initial coarse mesh after applying r-refinement and the corresponding locally refined mesh having 242 elements.

ensures that the additional nodes are inserted at better locations, and the energy of the solution on that mesh reduces to -56.657943 (see Figure 3.5). Note that the second mesh in Figure 3.4 is locally optimal but the second mesh in Figure 3.5 (which is not locally optimal) permits a significantly better solution.

Order of edge swapping and node movement does not make such a significant difference however we always apply the node movement algorithm

after every application of edge swapping algorithm. This is because it integrates the two tasks of moving the nodes and updating the solution values with the help of local solves. Hence after each application of edge swapping algorithm an application of node movement algorithm eliminates the need of a global finite element solve. Overall therefore, our final hybrid algorithm consists of optimising the mesh first using node movement and then edge swapping and node movement (interleaved), then refining this optimised mesh and interpolating the coarser solution onto the refined mesh as a starting point for the next level of optimisation. At this point we also include the possibility of applying a global solve on the new mesh so as to improve upon the interpolated solution values.

3.2 Adjustable parameters

Having presented a justification for the structure of Algorithm 4 we now note that there are numerous adjustable parameters within the algorithm, that the user must select. So far as possible however we do not treat these as *tuning parameters*, i.e. it is not necessary to change these parameters for different problems. Some of these parameters have already been introduced briefly in our discussion of the quality of the mesh in Section 3.1. We discuss all of them in detail below.

3.2.1 Minimum area condition

During the implementation of the node movement algorithm we have enforced a constraint imposing a minimum allowable area of elements in the

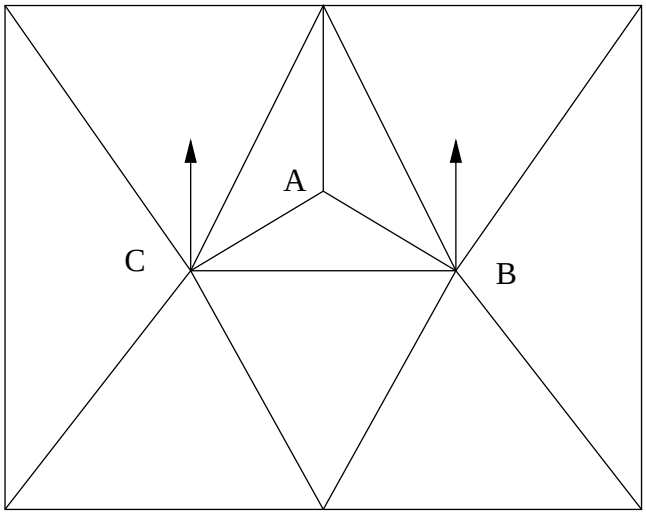

Figure 3.6: A situation where interior element ABC can flip to assign it negative area

mesh. This requires that the area of each of element be bounded below by a positive number A_{min}. During node movement when the meshes become highly distorted tangling can occur quite quickly and it is insufficient just to monitor the area of the element into which the node is moving. Figure 3.6 demonstrates this. Here the position of node A has been locally optimised and nodes B or C are moved next. Either of these could be moved far enough upwards to cause triangle ABC to flip over, giving it a negative area and causing the mesh to be tangled. Even without such occurrences, when the number of node movement sweeps is large, some of the nodes in the mesh can get so close to each other that elements formed by them can become arbitrarily small.

To avoid these situations, for each node j to be moved we allow its move to a new position in Ω_j only if the area of each element in Ω_j does not violate the minimum area condition after this move. For all of our

test problems a very small value for $A_{min} = 10^{-10}$ has been chosen. For this value of A_{min}, and the test problems considered in this chapter, this constraint never becomes active when the mesh is coarse. However at the third or fourth level of local/global refinement this constraint does become active for some of the cases. We tried different values of A_{min} and found that if a large value is chosen the node movement algorithm converges in fewer iterations, however with a larger value of energy. If a very small value is chosen the node movement algorithm takes more iterations to converge, however the reduction in energy is not significant. For example, for the first test problem in Section 3.3, when we use $A_{min} = 10^{-6}$ the energy at the final level (with global refinement, see Table 3.1) comes out as 50.0242 and when $A_{min} = 10^{-14}$ the energy comes out to be 50.0157 (as opposed to 50.0158 when $A_{min} = 10^{-10}$. Hence we recommend A_{min} should be in the range 10^{-8} to 10^{-12}.

3.2.2 Threshold for gradient

The gradient of the energy with respect to the position of node j (2.10), is used to decide whether or not node j should be moved and in which direction. If the magnitude of this gradient is more than some chosen threshold value the node is moved, otherwise it is not. The reason for including this condition is that moving a node for which the magnitude of the gradient vector is very small is unlikely to cause a significant reduction in the energy functional. Furthermore, at convergence to a local minimum all of these gradients should be zero. The design of the node movement algorithm is

such that, at each sweep through the nodes, it tries to reduce the energy functional. As we approach a local minimum this will begin to cause a reduction in the magnitude of the gradient vector for each node. As the number of sweeps increases, more and more nodes come to their locally optimal positions and by including this condition we can skip those nodes for which the gradient of the energy functional is very small.

In principal, knowledge of the gradient only does not provide any information about whether we are near a local maximum, a saddle point or a local minimum. It is possible therefore that we could skip a node which is at a local energy maximum rather than a minimum. In practice however this does not appear to present a problem since this situation is unstable and, by moving neighbouring nodes in their steepest descent directions, any nodes near to a local maximum are likely to be available for movement at the next sweep.

For all of our test problems we have chosen this threshold to be equal to 10^{-5}. Again a very small value for this parameter will cause unnecessary repetitions of the node movement algorithm (for very small gain), and a larger value will freeze the nodes at suboptimal positions. For our test problems we tried different values for this parameter and we recommend that a suitable range for this parameter should be between 10^{-4} to 10^{-7}.

3.2.3 Convergence criteria

As our hybrid algorithm is a combination of node movement, edge swapping and local h-refinement, it requires some stopping or convergence criteria for

each of these components. We describe each of these in turn. Since our hybrid algorithm and its components are iterative processes by increasing or decreasing the convergence criteria the desired accuracy can be decreased or increased. In each case outlined below we have used a tolerance of 10^{-6} for the results presented in Section 3.3.

3.2.3.1 Convergence criterion for node movement

Our node movement algorithm is an iterative algorithm, rather like the Gauss-Seidel iterative method for solution of linear or nonlinear equations. We consider one node at a time and decide whether it is to be moved or not and if it is moved we then update the local solution for it immediately. One node movement sweep, or iteration, is completed when we have visited through each of the nodes. The global energy of the solution is then computed after completing each iteration. The algorithm guarantees that this energy functional is reduced monotonically by each node movement and therefore each full iteration. The convergence criterion for the node movement algorithm is deemed to be satisfied when the difference in the energy functional computed in two consecutive iterations is less than some constant value chosen by the user (given above as 10^{-6} in our examples). We stop our node movement process when either this convergence criterion is satisfied or the maximum number of iterations (set by the user) is reached.

3.2.3.2 Convergence criterion for edge swapping

The edge swapping algorithm is also an iterative process. We swap all internal edges in a Gauss-Seidel fashion, considering one edge at a time and

deciding whether it is to be swapped or not. Unlike the node movement algorithm we do not update any local solution after swapping an edge. One edge swapping iteration is completed when we have visited each of the internal edges and the global energy of the solution is computed after each iteration. The convergence criterion for the edge swapping algorithm is deemed to be satisfied when the difference between global energy functional computed at two consecutive iterations is less than the tolerance given above. In practice, such a small value of the convergence parameter tends to ensure that no edge is swapped in the last iteration before convergence.

3.2.3.3 Convergence criterion for the optimised mesh

By optimising the mesh we mean that each of its nodes as well as each of its internal edges are at their locally optimal positions. After getting a locally optimal mesh with respect to the position of the nodes (as defined in Section 2.2) we apply the edge swapping algorithm to make the mesh locally optimal with respect to position of the edges too (as defined in Section 2.4). But swapping edges to their locally optimal positions will generally make the positions of nodes suboptimal. Hence we apply the node movement and edge swapping algorithms in turn making an iterative process. One iteration is completed when both of the algorithms have been applied. We compute the global energy before and after applying the combination of these two algorithms. The combination of these two is converged when the difference of global energy functionals computed at two consecutive iterations is less than the convergence tolerance chosen.

3.2.3.4 Stopping criterion for the hybrid algorithm

We optimise the mesh at the first level and then we refine the mesh locally. After applying local h-refinement, the position of the nodes and edges are no longer locally optimal. To make the new mesh locally optimal we re-apply the combination of node movement and edge swapping, which leads to an optimised mesh at the second level. To get the mesh at next level we again refine locally. We repeat the whole process of r-refinement and local h-refinement until the difference between two energies at consecutive levels is less than the convergence criterion or the maximum mesh size is reached.

3.2.4 Admissible limit factor for node movement

The maximum admissible limit factor w, as explained in Section 2.2, plays an important role in the convergence of the node movement algorithm. By changing its value one can significantly change the nature of the locally optimal mesh and the value of global energy on this converged mesh, as well as altering the number of iterations needed to converge. We have tried a number of different values of w for different problems for the purpose of experimentation. We observed that the algorithm never fails for any value of w lying in the interval $(0, 1)$. However, for different values of w it either converges to a different mesh with a different value of energy, or it converges to the same mesh but in a different number of iterations.

We observed from our experiments that when an exact line search is used a high value of w, around 0.9 produced better results, but when an inexact line search is used a more cautious value of around 0.5 appeared

best. For all of the test problems in Section 3.3 we use exact line search with $w = 0.9$, except for the last one where a different choice is explained.

For example, for the third test problem in Section 3.3, when exact line searches are used and $w = 0.1, 0.5$ and 0.9 are chosen we get energies for the optimised initial mesh of 0.550093, 0.549725 and 0.549242 respectively. When inexact line searches are used and $w = 0.1, 0.5$ and 0.9 are chosen we get energies for the optimised initial mesh of 0.551730, 0.549901 and 0.551321 respectively.

3.2.5 Number of edges attached to one node

In the algorithm for edge swapping, the only criterion for accepting an edge swap is that it reduces the energy. Typically it turns out that application of the edge swapping algorithm usually improves the geometrical quality of the mesh produced, after each application of the node movement algorithm (even though the criteria for each edge swap is energy minimisation). This observation is made by comparing the area of the smallest element before and after applying the edge swapping algorithm, which almost always increases the area of the smallest element. However, in the absence of any additional conditions for edge swapping, in some very rare cases, the algorithm tries to attach very large numbers of edges to one node. This situation is probably not particularly desirable as it may well increase the number of long and thin elements unnecessarily and will certainly damage the sparsity pattern of the finite element stiffness matrix.

To overcome this potential problem we introduce an integer E_{max}, the

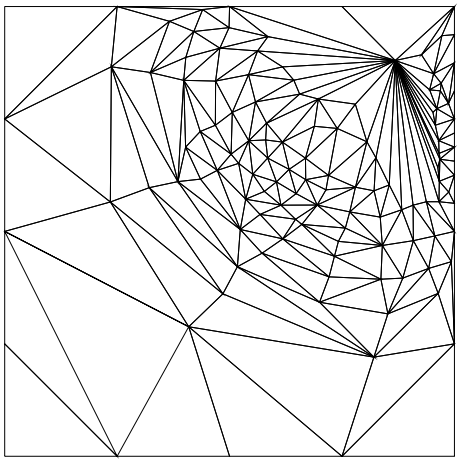

Figure 3.7: A situation where too many edges attached to one node

maximum number of edges that may be attached to one node. For each edge to be swapped, we count the number of edges attached to the other two nodes of the quadrilateral (to which the edge would be attached after swapping). If for either of these nodes this number is greater than or equal to E_{max}, we do not allow the edge swap. For each of our test problems in Section 3.3 we take E_{max} to be equal to 15. For all of these test problems the situation where this constraint becomes active is very rare (with this choice of upper limit). We believe that $E_{max} = 15$ will not cause our edge swapping algorithm to become trapped in any local minima (that it would not otherwise have found) and that it will improve the geometric quality of the mesh.

3.3 Numerical results

Now we present the numerical results for the full hybrid algorithm applied to a number of test problems. All of the parameters involved in the hybrid algorithm have been described in the previous sections, except the details

of the quadrature rule used for evaluating integrals numerically. For all of the test problems considered in this chapter, we use the 13-point rule given in [41], which is exact for polynomials of degree less than or equal to 7.

3.3.1 Problem one

The first test problem that we consider is the simple two-dimensional reaction-diffusion equation

$$-\Delta u + \frac{1}{\varepsilon^2} u = 0 \ , \qquad \underline{x} \in \Omega = (0,1) \times (0,1) \ , \tag{3.2}$$

with Dirichlet boundary conditions,

$$u = e^{-x_1/\varepsilon}, \tag{3.3}$$

assumed throughout $\partial\Omega$. This problem is a little artificial since the solution (also given by (3.3)) is essentially one dimensional. Nevertheless, we feel this is a good test of our hybrid algorithm. A moderate value of $\varepsilon = 0.01$ is chosen to produce a weak boundary layer in the solution near $x_1 = 0$. It can be easily verified that (3.3) is also the exact solution of (3.2) over the whole domain Ω. The energy functional corresponding to the problem (3.2) is given by

$$E = \frac{1}{2} \int_\Omega \left[\frac{\partial u}{\partial x_i} \frac{\partial u}{\partial x_i} + \frac{u^2}{\varepsilon^2} \right] \, d\underline{x} \ , \tag{3.4}$$

and this is clearly of the form of problem defined in Section 2.1. As the exact solution for this problem is known,

$$E = \frac{1}{2\varepsilon}[1 - e^{-2/\varepsilon}] \tag{3.5}$$

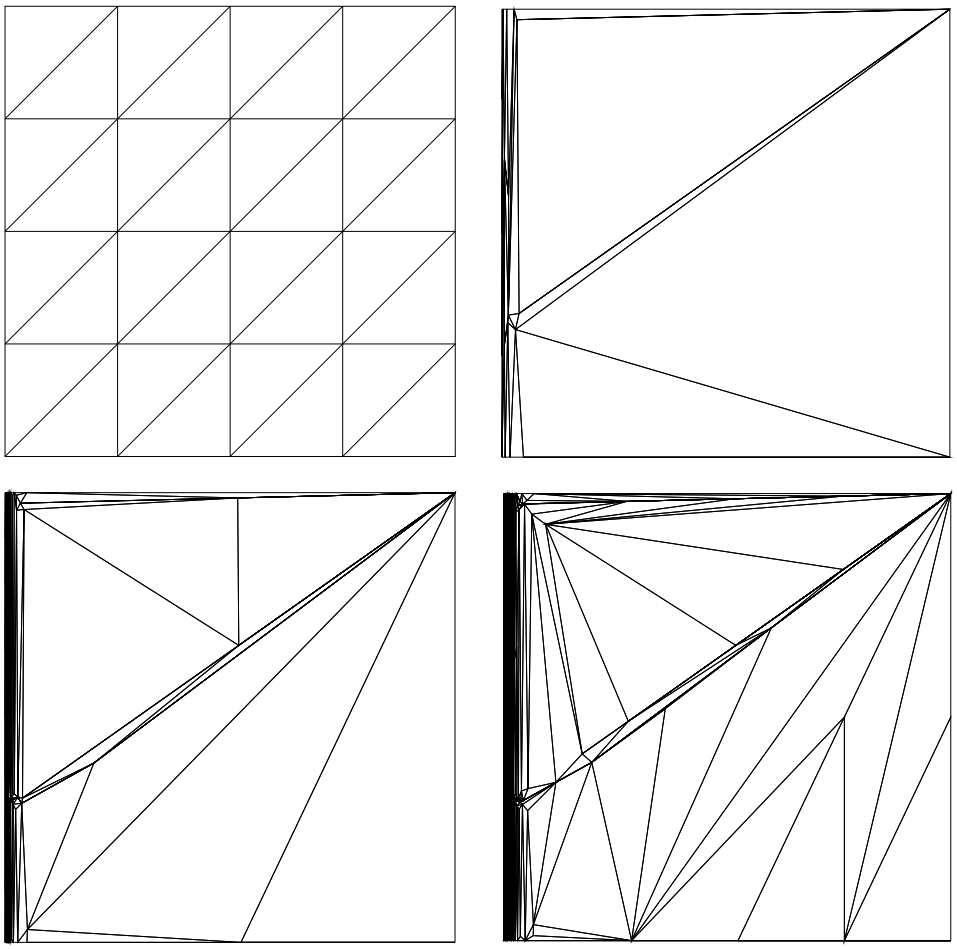

Figure 3.8: An initial mesh (top left) followed by a sequence of meshes obtained by r-refinement and then combinations of global h-refinement with r-refinement.

is the minimum of (3.4), which can be obtained by substituting the exact solution into (3.4). By putting the chosen value of $\varepsilon = 10^{-2}$ in (3.5) we get $E = 50.0000$.

Initially the problem is solved on a uniform coarse mesh containing just 32 elements. This mesh is then optimised using r-refinement, i.e. the combination of node movement and edge swapping. The total energy reduces from 374.473 to 50.8937, reflecting the fact that before optimisation there were no degrees of freedom in the boundary layer near $x_1 = 0$. Following

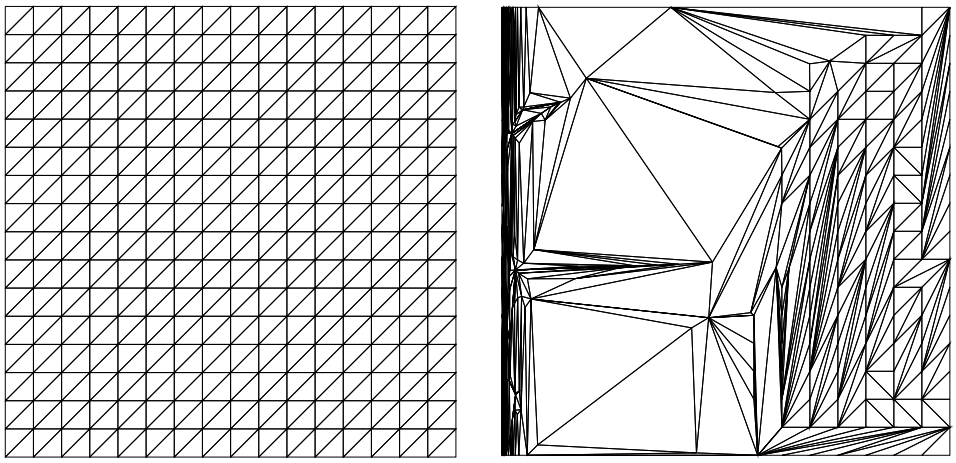

Figure 3.9: A globally refined mesh of 512 elements and the corresponding locally optimised mesh.

[104] this optimal mesh may now be uniformly refined to produce 128 elements which may themselves be optimised. This leads to a solution with a total stored energy of 50.1137. A further global refinement and optimisation then leads to a solution with a total stored energy of 50.0158 on a mesh of 512 elements: this sequence of locally optimal meshes is shown in Figure 3.8.

Figure 3.9 illustrates two further meshes of 512 elements: the first obtained by global refinement of the initial uniform mesh and the second by optimising this mesh directly. The energies of the solutions on these meshes are 103.630 and 50.2311 respectively thus illustrating, for this example at least, the superiority of the hierarchical approach when r-refinement is combined with global h-refinement. It is clear from this pair of meshes that although the second mesh is locally optimal it suffers from the problem that too many of the degrees of freedom, inherited from the first mesh, lie in a part of the domain that is far from the boundary layer.

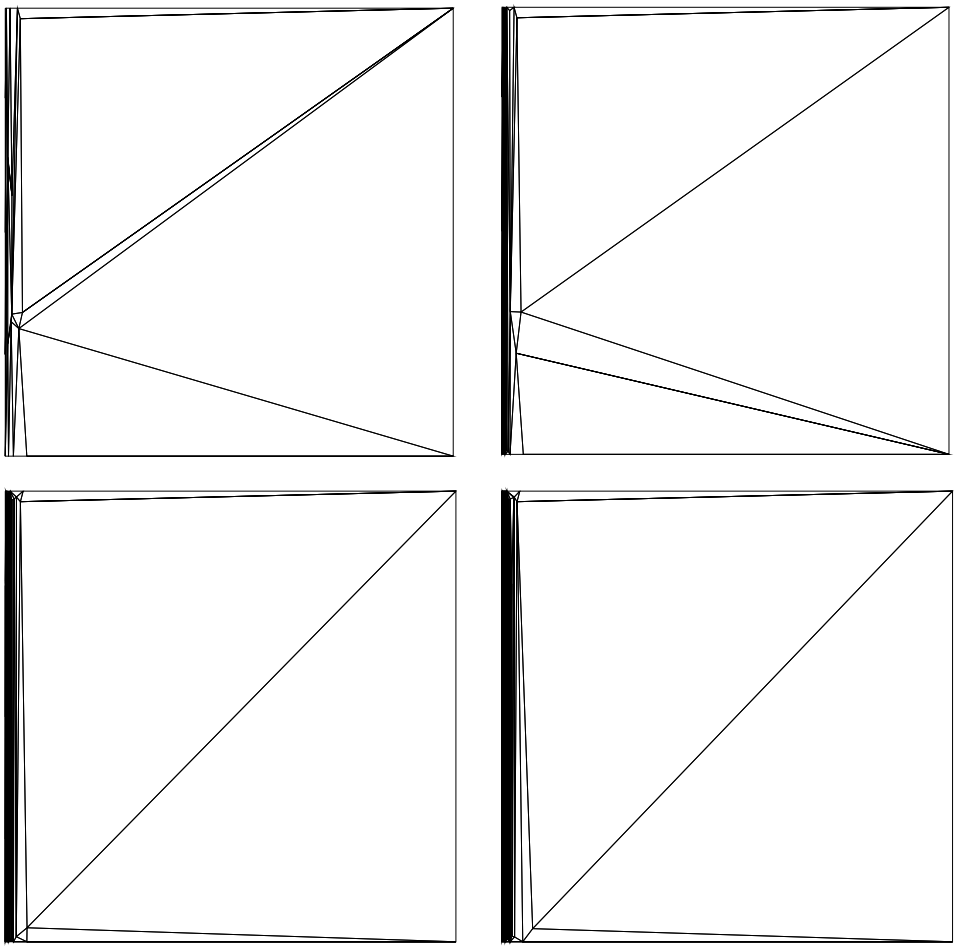

Figure 3.10: A sequence of meshes obtained by r-refinement of an initial coarse mesh (top left) and then combinations of local h-refinement followed by r-refinement.

Figure 3.10 shows a sequence of meshes obtained by applying our hybrid algorithm which combines r-refinement with local h-refinement. The first mesh is the same one, containing 32 elements, that appears in Figure 3.8, whilst the second, third and fourth meshes contain 42, 94 and 323 elements respectively and were obtained by refining into 2 children only those elements whose local energy exceeded $X = 60\%$ of the maximum local energy on any element. The total energies of the solutions on these four meshes are 50.8937, 50.3408, 50.1010 and 50.0085 respectively: clearly illustrating

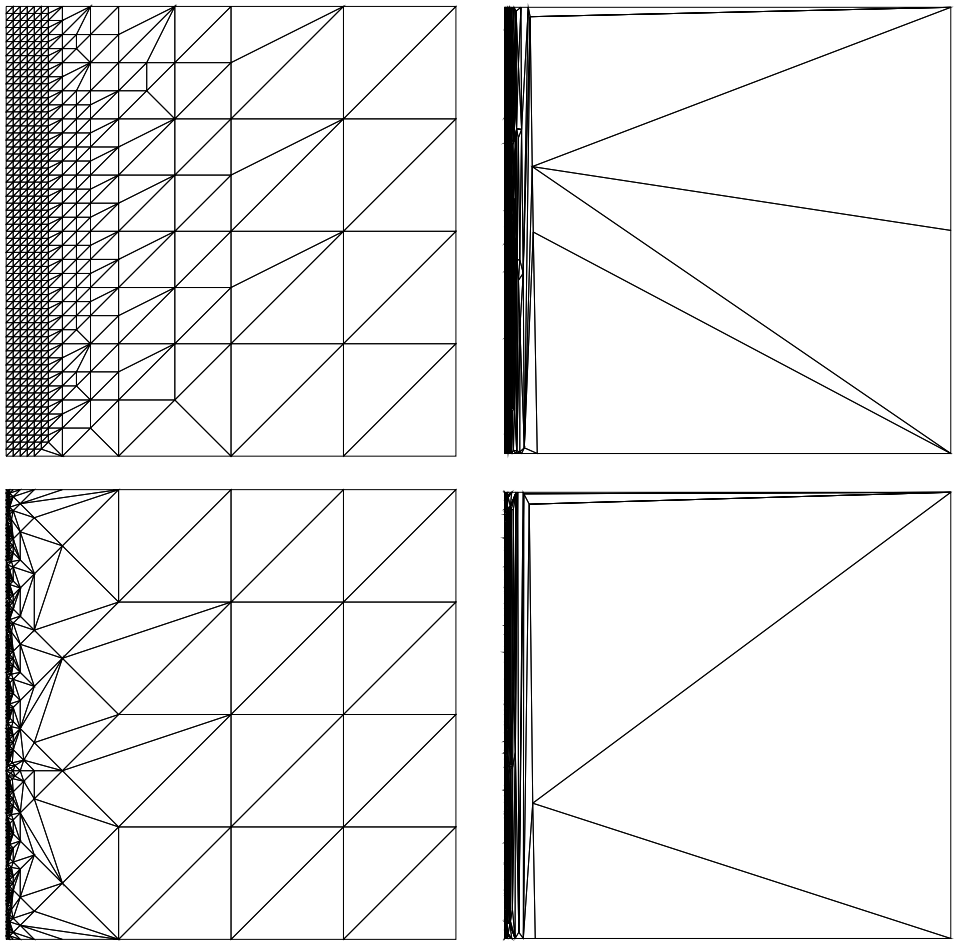

Figure 3.11: A pair of meshes of 1048 elements obtained using local one-to-four h-refinement (top left) followed by optimisation and a pair of meshes of 784 elements obtained using local one-to-two h-refinement (bottom left) followed by optimisation.

the superiority of the use of local rather than global h-refinement within the hybrid algorithm.

To conclude our discussion of this example we illustrate the advantage of applying the hybrid approach hierarchically by contrasting it with the use of local h-refinement alone, possibly followed by a single application of r-refinement. Figure 3.11 shows two meshes of 1048 elements and two meshes of 784 elements that were obtained in this manner (again using a threshold

Elements	Energy	Description
32	374.473	Figure 3.8 (top left)
32	50.8937	Figure 3.8 (top right)
128	50.1137	Figure 3.8 (bottom left)
512	50.0158	Figure 3.8 (bottom right)
512	103.630	Figure 3.9 (left)
512	50.2311	Figure 3.9 (right)
32	50.8937	Figure 3.10 (top left)
42	50.3408	Figure 3.10 (top right)
94	50.1010	Figure 3.10 (bottom left)
323	50.0085	Figure 3.10 (bottom right)
1048	54.8553	Figure 3.11 (top left)
1048	50.0536	Figure 3.11 (top right)
784	51.4939	Figure 3.11 (bottom left)
784	50.0714	Figure 3.11 (bottom right)

Table 3.1: Summary of the results obtained for Problem one (the global energy minimum is 50.0000).

of $X = 60\%$ for the local refinement). The total energies of the solutions on the 1048 element meshes (obtained by local one-to-four refinement alone and then a single application of the mesh optimisation at the end) are 54.8553 and 50.0536 respectively, whilst the total energies of the solutions on the 784 element meshes (obtained by local one-to-two refinement plus a final optimisation) are 51.4939 and 50.0714 respectively.

We see that in both cases, despite the fact that the second of each pair of meshes is locally optimal, the quality of these local optima are not as good as that obtained using the hierarchical approach. A summary of all of the computations made for this test problem is provided in Table 3.1.

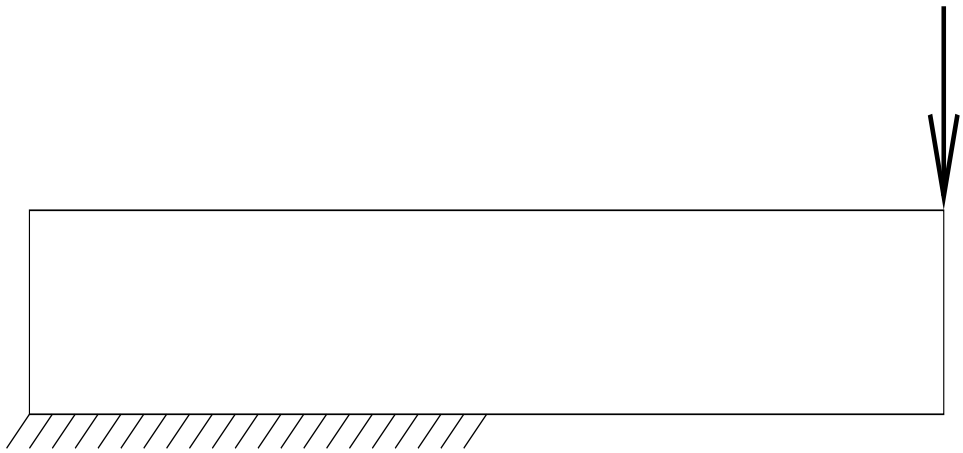

Figure 3.12: An illustration of the overhanging cantilever beam with a vertical point load at the end of the cantilever.

3.3.2 Problem two

We now consider the more challenging problem of calculating the displacement field for a two-dimensional linear elastic model of an overhanging cantilever beam supporting a vertical point load at the end of the cantilever, as illustrated in Figure 3.12. This problem is taken from [62] and [103] and corresponds to a system of PDEs as outlined in Section 2.3.

For this problem $m = n = 2$ and the energy functional is given by

$$E = \frac{1}{2}\int_\Omega \frac{\partial u_i}{\partial x_j} C_{ijk\ell} \frac{\partial u_k}{\partial x_\ell}\, d\underline{x} - \int_\Omega \rho b_i u_i \, d\underline{x} - \int_{\partial_\theta} \theta_i u_i \, ds \,. \tag{3.6}$$

Here, all repeated suffices are summed from 1 to 2, $\mathbf{C}$ is the usual fourth order elasticity tensor (in this case corresponding to a Young's modulus $E = 100$ and a Poisson ratio $\nu = 0.001$), $\rho\underline{b}$ provides the external body forces due to gravity and $\underline{\theta}$ represents the traction boundary condition (in this case a point load as illustrated in Figure 3.12). The left half of the lower boundary is fixed whilst the rest of the boundary, ∂_θ say, is free. Unlike for

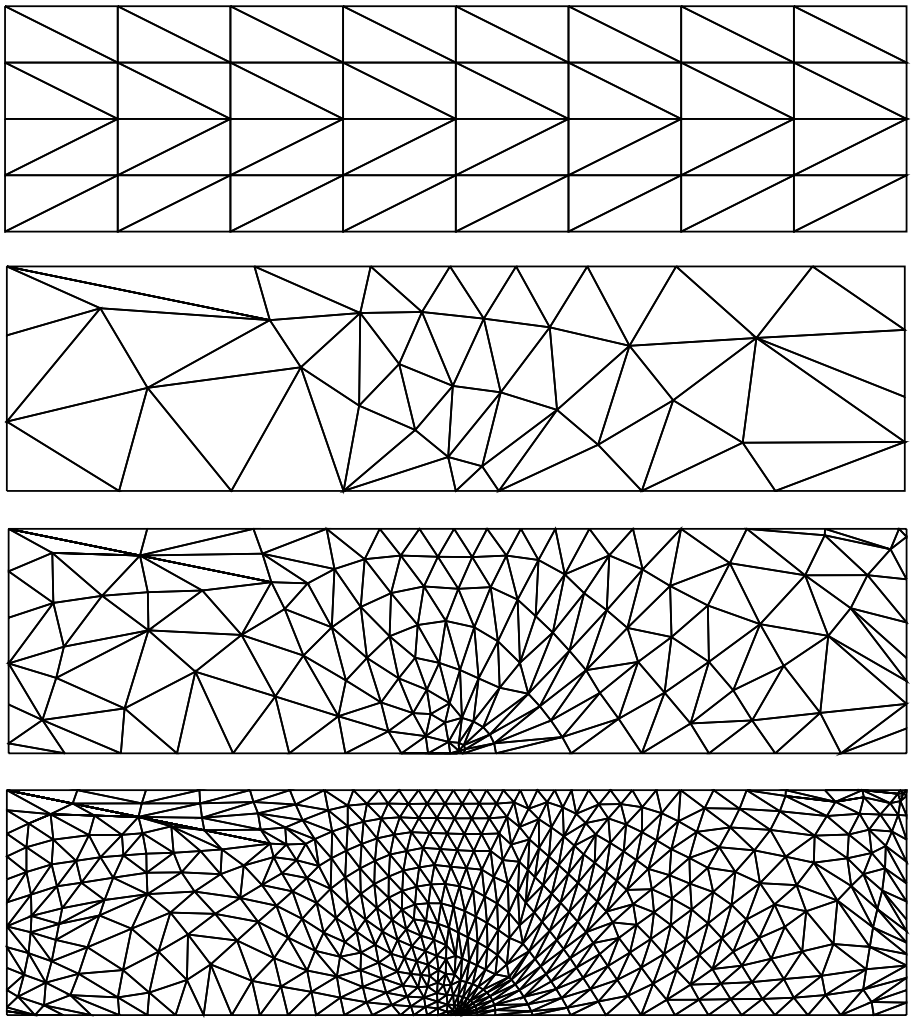

Figure 3.13: An initial mesh followed by a sequence of meshes obtained by r-refinement and then combinations of global h-refinement with r-refinement.

the first example we do not know an exact solution for this problem and so the optimal value for the stored energy is not known *a priori*.

As before we begin by solving the problem on a uniform coarse mesh, this time containing 64 elements. This mesh is then optimised using the r-refinement algorithm to reduce the total energy from -0.201352 to -0.253210. This optimal mesh may now be uniformly refined to produce 256 elements which are also optimised, leading to a solution with a total stored energy

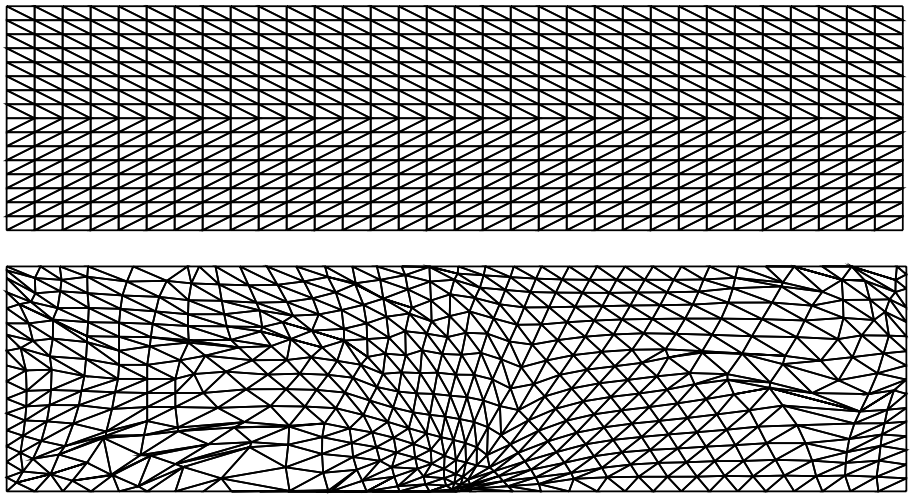

Figure 3.14: A globally refined mesh of 1024 elements and the corresponding locally optimised mesh.

of -0.302353. One further global refinement and optimisation then leads to a solution with a total stored energy of -0.338964 on a mesh of 1024 elements. This sequence of locally optimal meshes is shown in Figure 3.13.

Figure 3.14 illustrates two further meshes of 1024 elements. The first of these is obtained by global refinement of the initial uniform mesh and the second by optimising this mesh directly. The energies of the solutions on these meshes are -0.306791 and -0.329249 respectively and so we again observe the superiority of the hierarchical approach when r-refinement is combined with global h-refinement.

As for the previous example, our goal is to assess the hybrid algorithm that combines r-refinement with local h-refinement hence Figure 3.15 shows a sequence of meshes obtained in this manner. The first mesh is the same one, containing 64 elements, that appears in Figure 3.13, whilst the second and third meshes contain 224 and 455 elements respectively and were ob-

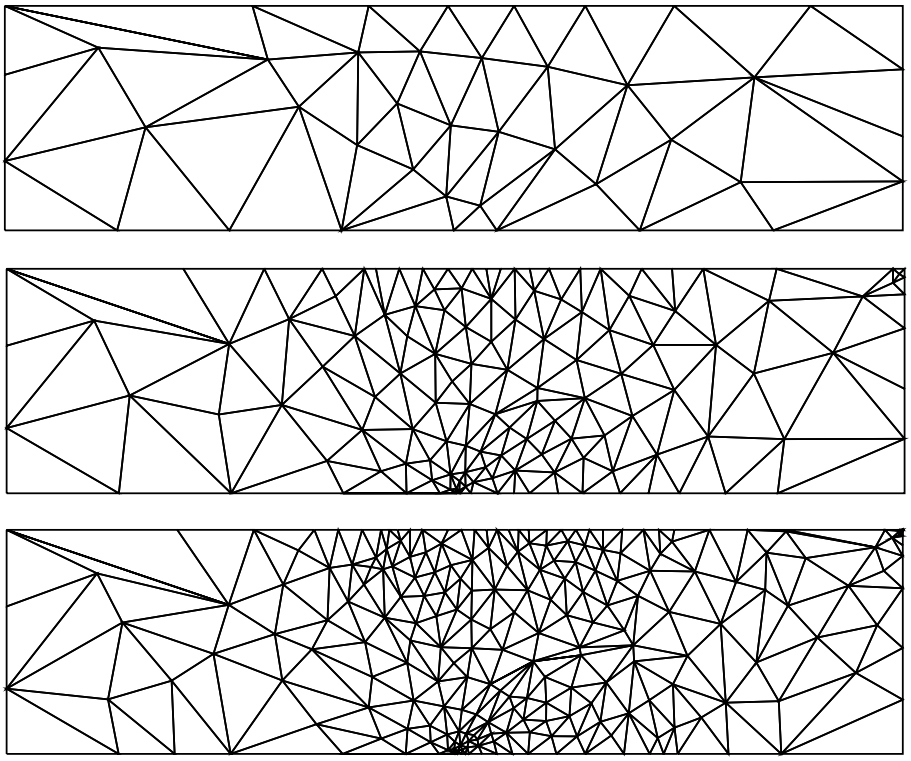

Figure 3.15: A sequence of meshes obtained by r-refinement of an initial coarse mesh and then combinations of local h-refinement followed by r-refinement.

tained by refining into 2 children only those elements whose local energy exceeded 60% of the maximum local energy on any element. The total energies of the solutions on these three meshes are -0.253210, -0.308351 and -0.363313 respectively: again illustrating the superiority of the use of local rather than global h-refinement within the hybrid algorithm.

We again conclude our example by illustrating the advantage of applying the hybrid approach hierarchically by contrasting it with the use of local h-refinement alone, possibly followed by a single application of r-refinement. Figure 3.16 shows two meshes of 674 elements and two meshes of 462 elements that were obtained in this manner (again using a threshold

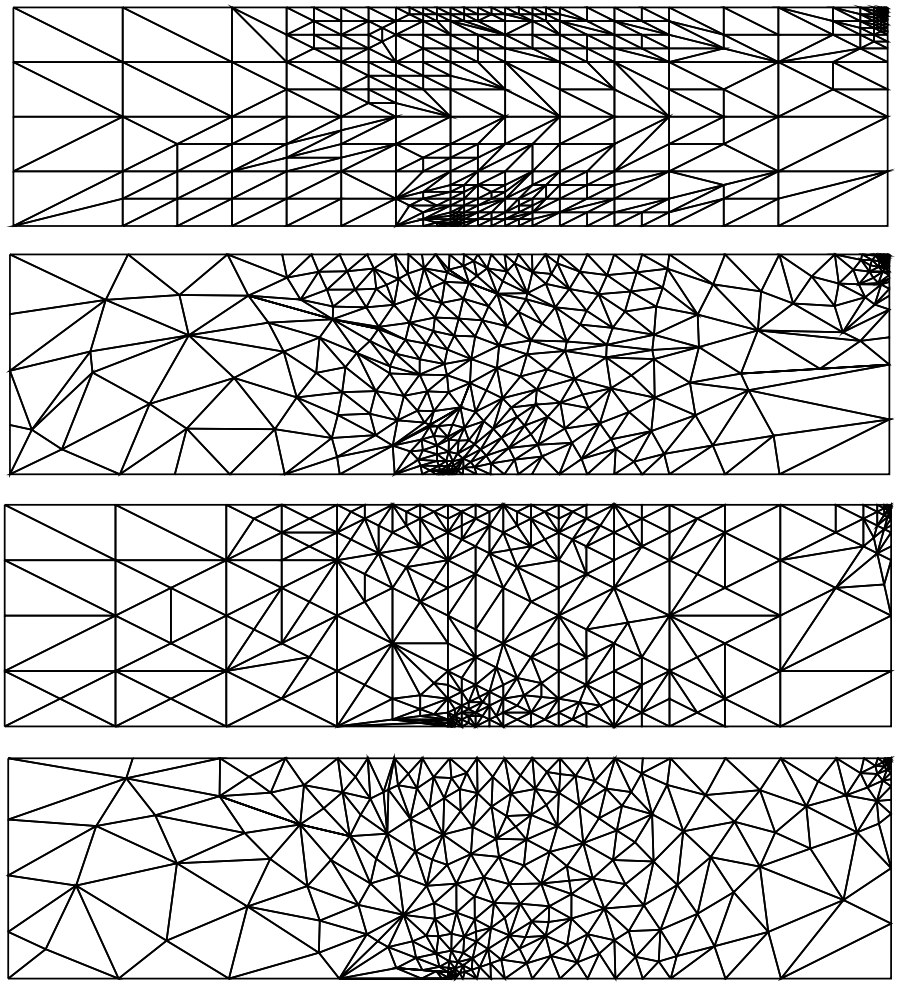

Figure 3.16: A pair of meshes of 674 elements obtained using local one-to-four h-refinement (top) followed by optimisation (second) and a pair of meshes of 462 elements obtained using local one-to-two h-refinement (third) followed by optimisation (bottom).

of $X = 60\%$ for the local refinement). The total energies of the solutions on the 674 element meshes (obtained by local one-to-four refinement alone and then a single application of the mesh optimisation at the end) are -0.325679 and -0.342525 respectively, whilst the total energies of the solutions on the 462 element meshes (obtained by local one-to-two refinement plus a final optimisation) are -0.325879 and -0.342355 respectively. As before it is clear that the quality of the locally optimal meshes obtained in this manner

Elements	Energy	Description
64	-0.201352	Figure 3.13 (top)
64	-0.253210	Figure 3.13 (second)
256	-0.302353	Figure 3.13 (third)
1024	-0.338964	Figure 3.13 (bottom)
1024	-0.306791	Figure 3.14 (top)
1024	-0.329249	Figure 3.14 (bottom)
64	-0.253210	Figure 3.15 (top)
224	-0.308351	Figure 3.15 (middle)
455	-0.363313	Figure 3.15 (bottom)
674	-0.325679	Figure 3.16 (top)
674	-0.342525	Figure 3.16 (second)
462	-0.325879	Figure 3.16 (third)
462	-0.342355	Figure 3.16 (bottom)

Table 3.2: Summary of the results obtained for Problem two (the global energy minimum is unknown).

is inferior to that of meshes obtained using the hierarchical approach. A summary of all of the computations made for this test problem is provided in Table 3.2.

3.3.3 Problem three

For the third two-dimensional problem that we consider, we return to an example with just one dependent variable, however it features a solution which is singular at the origin. This problem appears in [20, 70, 104].

The energy functional corresponds to the Laplacian operator and is given by

$$E = \frac{1}{2}\int_{\Omega} \frac{\partial u}{\partial x_i}\frac{\partial u}{\partial x_i} d\underline{x} \,, \tag{3.7}$$

where the presence of repeated suffices again implies summation from 1 to 2. The domain, Ω, is the unit disc with a 45° sector removed, as illustrated in Figure 3.17, and Dirichlet boundary conditions consistent with the exact

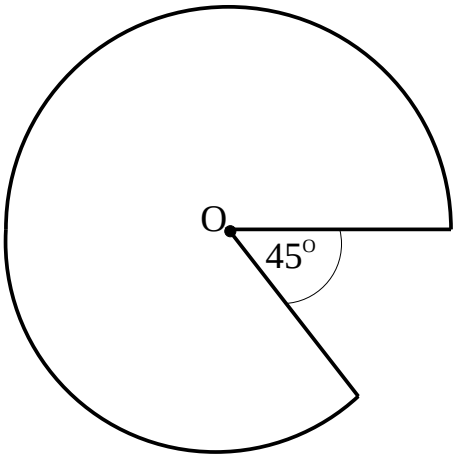

Figure 3.17: An illustration of the domain for the singular problem.

solution $u = r^{2/7} \sin \frac{2\theta}{7}$ are applied throughout $\partial\Omega$. Since the exact solution is known in this case, so is the true value of the global minimum of E in (3.7): 0.392699.

As with the previous examples the problem is first solved on a coarse initial mesh, in this case with just 28 elements, which is then optimised. This locally optimal mesh is then refined globally and optimised to three further levels, giving meshes of 112, 448 and 1792 elements respectively. These meshes are shown in Figure 3.18 and their corresponding solutions have energies of 0.549242, 0.434828, 0.404352 and 0.396215.

Once again, it may be observed that the approach of optimising the mesh at each level after global refinement is superior to applying global h-refinement alone and then optimising the resulting mesh. Figure 3.19 shows two meshes, each containing 1792 elements, that were obtained by this method. The energies of the solutions on these meshes are 0.438164 (uniform h-refinement only) and 0.405547 (after optimisation), which are

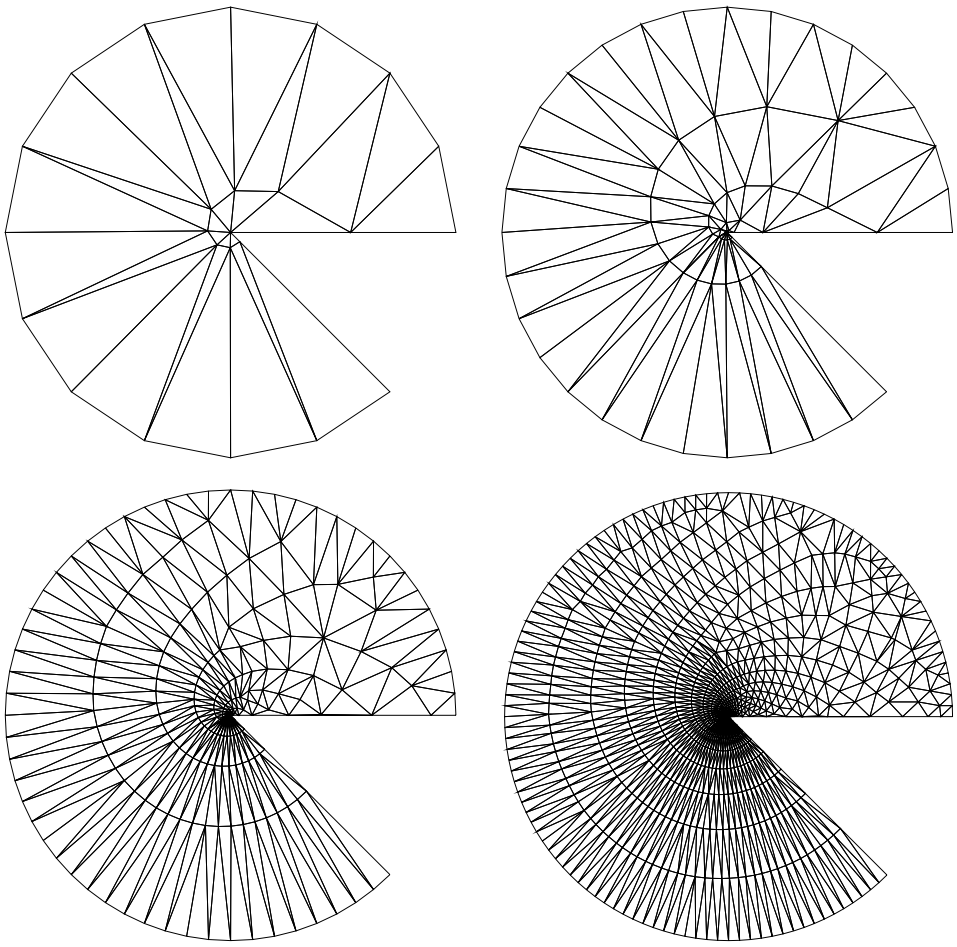

Figure 3.18: A sequence of meshes obtained by r-refinement of an initial coarse mesh (top left) and then combinations of global h-refinement followed by r-refinement.

significantly worse than for the final mesh of Figure 3.18.

To conclude this example, we now consider the application of local h-refinement in our hybrid algorithm. Figure 3.20 shows a sequence of four meshes of 28, 107, 255 and 1275 elements respectively. In order to contrast the solutions on these meshes with those obtained on the meshes shown in Figure 3.18 we have forced refinement of each of the edges on the circular boundary so that the domains correspond to the four domains in Figure 3.18. Further refinement (one element to two children) has then been per-

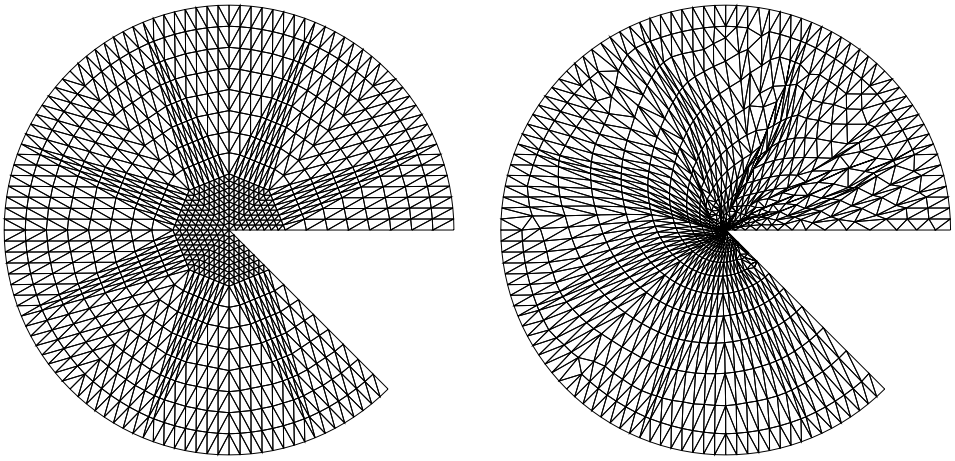

Figure 3.19: A globally refined mesh of 1792 elements and the corresponding locally optimised mesh.

mitted locally for any elements whose energy is greater than $X = 60\%$ of the maximum energy on any single element. This local refinement is executed repeatedly on each domain until it is necessary to refine the boundary elements again. The total energies of the solutions on the four meshes shown in Figure 3.20 are 0.549242 (the same mesh as in Figure 3.18), 0.431777, 0.402413 and 0.395183 respectively.

Again we have seen the advantage of using the hierarchical mesh optimisation approach with local, rather than global, refinement. Furthermore, when local h-refinement is used on its own, even if this is followed by mesh optimisation, the resulting meshes are not as good. Two pairs of such meshes, containing 1437 (one-to-four refinement) and 1413 (one-to-two refinement) elements respectively, are illustrated in Figure 3.21. For these examples the corresponding finite element solutions have total energies of 0.407613 and 0.398523 (1437 elements before and after optimisation) and 0.402199 and 0.398123 (1413 elements before and after optimisation) re-

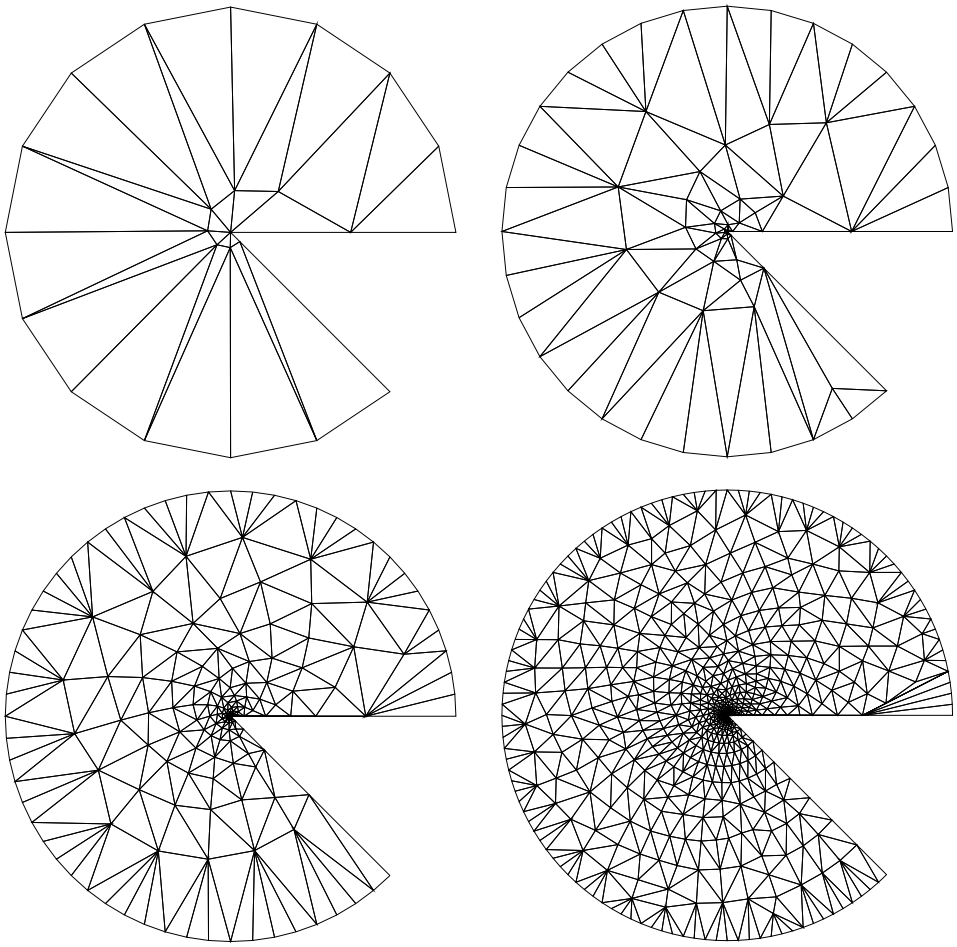

Figure 3.20: A sequence of meshes obtained by r-refinement of an initial coarse mesh (top left) and then combinations of local h-refinement followed by r-refinement.

spectively. (For the purposes of comparison, we have artificially refined those edges on the circular boundary so as to ensure that the domains are identical to the final domains in Figures 3.18 to 3.20.) A summary of all of the computations made for this test problem is provided in Table 3.3.

3.3.4 Problem four

The final two-dimensional problem that we consider also involves just one dependent variable. This problem is taken from [104]. The energy functional

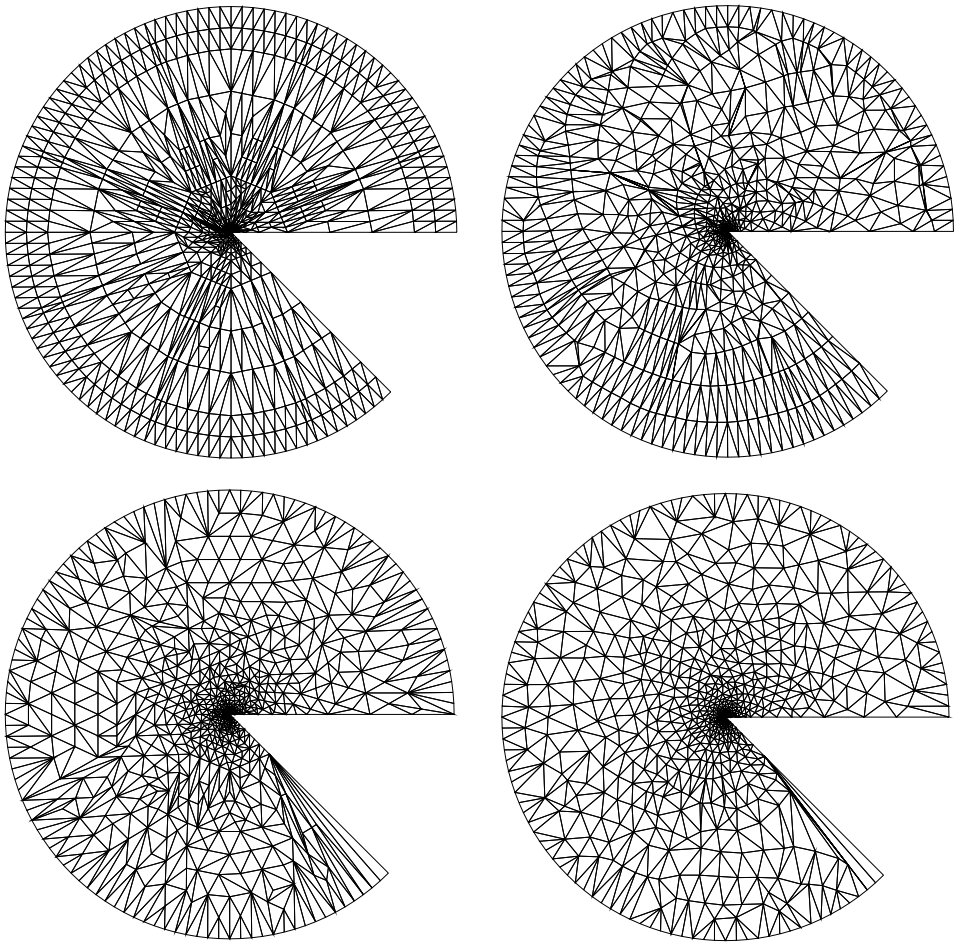

Figure 3.21: A pair of meshes of 1437 elements obtained using local one-to-four h-refinement (top left) followed by optimisation and a pair of meshes of 1413 elements obtained using local one-to-two h-refinement (bottom left) followed by optimisation.

to be minimised is given by,

$$E = \int_{\Omega} \sqrt{1 + \frac{\partial u}{\partial x_i}\frac{\partial u}{\partial x_i}} d\underline{x} \,, \tag{3.8}$$

where the repeated suffices implies summation from 1 to 2. The domain Ω is given by,

$$\Omega = \{\underline{x} \in \mathcal{R}^2 : 1/4 < |\underline{x}| < 1\}.$$

Dirichlet boundary conditions,

$$u_o(\underline{x}) = a\{cosh^{-1}(|\underline{x}|/a) - cosh^{-1}(1/a)\}, \tag{3.9}$$

Elements	Energy	Description
28	0.549242	Figure 3.18 (top left)
112	0.434828	Figure 3.18 (top right)
448	0.404352	Figure 3.18 (bottom left)
1792	0.396215	Figure 3.18 (bottom right)
1792	0.438164	Figure 3.19 (left)
1792	0.405547	Figure 3.19 (right)
28	0.549242	Figure 3.20 (top left)
107	0.431777	Figure 3.20 (top right)
255	0.402413	Figure 3.20 (bottom left)
1275	0.395183	Figure 3.20 (bottom right)
1437	0.407613	Figure 3.21 (top left)
1437	0.398523	Figure 3.21 (top right)
1413	0.402199	Figure 3.21 (bottom left)
1413	0.398123	Figure 3.21 (bottom right)

Table 3.3: Summary of the results obtained for Problem three (the global energy minimum is 0.392699).

are applied over the boundary, where a is a given parameter in $(0, 1/4)$. The interesting feature of this test problem is the fact that the corresponding Euler-Lagrange equation, whose weak form is given by

$$\int_\Omega \frac{1}{\sqrt{1+(\nabla u)^2}} \nabla u \cdot \nabla v d\underline{x} = 0, \tag{3.10}$$

is nonlinear.

It can be verified that the analytic solution u is given by the extension of the boundary function u_o to the whole domain, i.e.,

$$u(\underline{x}) = u_o(\underline{x}) \quad \forall\, \underline{x} \in \Omega.$$

This exact solution shows that, when $a \to 1/4$, ∇u develops a line singularity along the circle $|\underline{x}| = 1/4$. Therefore for a close to $1/4$, uniform meshes in the radial direction can be expected to provide sub-optimal approximations of the solution.

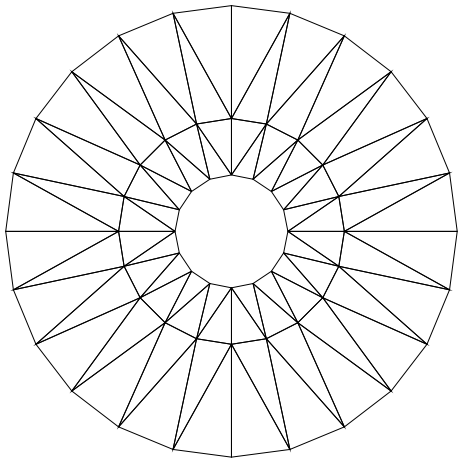

Figure 3.22: An initial mesh for the nonlinear problem.

Since this problem is nonlinear we choose not to solve a global nonlinear problem at any step due to the complexity of this. Instead we use local solves only and rely on the fact that solving a sequence of local problems iteratively eventually provides us with a global solution (as has been explained in Section 2.2.3). We start with an initial approximation of the solution as zero on all of the interior vertices. We then solve the sequence of local problems for each interior node iteratively, to get an improved solution for each node. The iteration continues until we get,

$$\left| \frac{E^i - E^{i-1}}{E^i} \right| < C_{pre}, \tag{3.11}$$

where E^i is the global energy of the solution after the i^{th} iteration. We have chosen the same value of $a = 1/4 - 10^{-8}$ and $C_{pre} = 10^{-5}$, as are chosen in [104]. For this value of a the global minimum of energy is 3.4468778 (for the circular domain).

There was a discrepancy in the results shown in [104] for this problem. The initial mesh shown was of 48 elements and further global refinement

was made to this mesh. However the results tabulated were for an initial mesh of 60 elements, and the corresponding globally refined meshes. Another confusion with this problem is that for the meshes used in [104] they state that they approach the global minimum of energy from below (whereas the Galerkin method always overestimates the global minimum). The explanation for this is that after each level of global refinement in [104] the authors are stretching out the nodes to better approximate the circular domain (which causes the energy minimum to increase at each refinement level). To remove this confusion we deliberately decided not to change the domain at each refinement level in our example. The initial mesh we used (shown in Figure 3.22) is of 80 elements and therefore provides a slightly better approximation to the circular domain than the initial mesh used in [104]. For this non-circular domain the global minimum of the energy is not known analytically however.

For this problem we use $w = 0.25$ (defined in Section 3.2.4). This may be regarded as an under-relaxation parameter to aid convergence. If a node is allowed to move too large a distance, the initial approximation (which is the nodal solution at current node position) required for Newton's method to converge becomes a poor initial guess, and for some of the cases that we encountered the solution for the local problem (2.5) does not converge. To avoid this situation we choose a smaller value of w and $w = 0.25$ works well.

As before the problem is solved on a uniform coarse mesh to begin with, containing 80 elements. This mesh is then optimised by applying r-

refinement, and the energy reduces from 3.455935 to 3.450755. The resulting mesh is then globally refined and optimised, which gives a mesh of 320 elements having energy 3.422683. The mesh at the next level, having 1280 elements, is obtained by globally refining the previous optimised mesh. This mesh is then optimised to reduce the energy further to 3.415145. The mesh at the final level, having 5120 elements, is obtained by globally refining and optimising the optimised mesh at the previous level, and the energy of the solution is reduced to 3.413707. This sequence of locally optimal meshes is shown in Figure 3.23.

We now globally refine the initial coarse mesh to get a mesh of 5120 elements. The global energy of the solution on this mesh is 3.448782. This mesh is then optimised using r-refinement to reduce the energy to 3.447841. The results clearly indicate the superiority of hierarchical approach. These two meshes are shown in Figure 3.24.

To conclude this example, we now consider the application of local h-refinement in our hybrid algorithm. We start with the same optimised mesh of 80 elements shown in Figure 3.23 (top left). Refinement (one element to two children) has then been applied locally for any elements whose energy is greater than $X = 60\%$ of the maximum energy on any single element. The advantage of using the hierarchical mesh optimisation approach with local, rather than global, refinement can be seen from the results presented in Table 3.4. However more computational work has been done than in our previous examples as many more refinement levels have been used. Figure

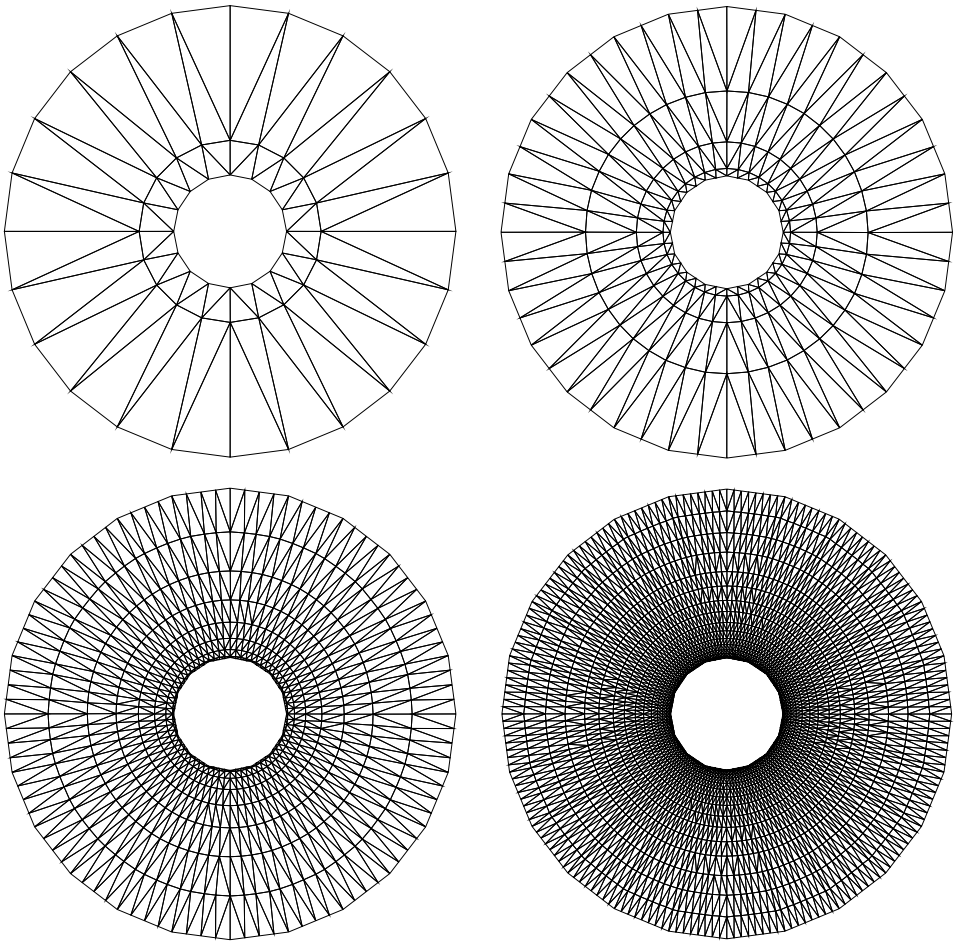

Figure 3.23: A sequence of meshes obtained by r-refinement of an initial coarse mesh (top left) and then combinations of global h-refinement followed by r-refinement.

3.25 shows four meshes of 266, 738, 1720 and 3523 elements at intermediate levels, using this approach.

To complete this set of results, when local h-refinement is used on its own, even if this is followed by mesh optimisation, the resulting meshes are not as good. Two pairs of such meshes, containing 4660 (one-to-two refinement) and 4896 (one-to-four refinement) elements respectively, are illustrated in Figure 3.26. For these examples the corresponding finite element solutions have total energies of 3.421024 and 3.414235 (4660 elements

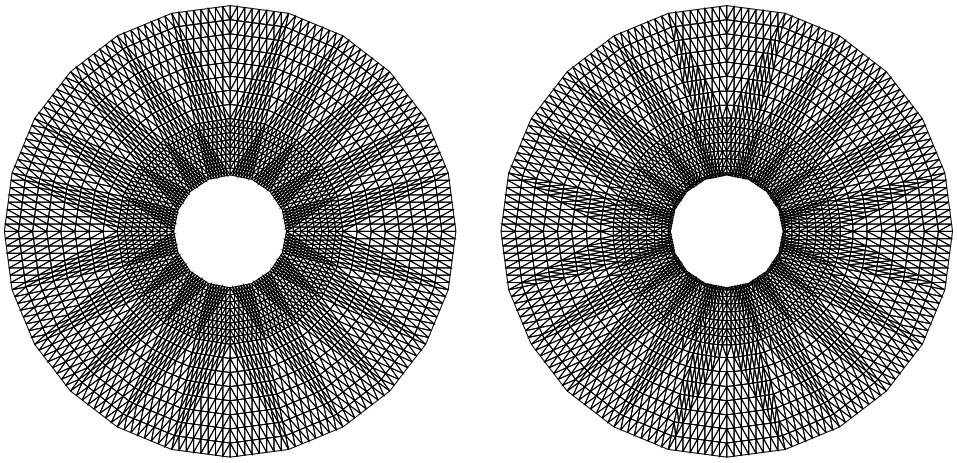

Figure 3.24: A globally refined mesh of 5120 elements and the corresponding locally optimised mesh.

before and after optimisation), and 3.418394 and 3.415132 (4896 elements before and after optimisation) respectively.

A summary of all of the computations made for this test problem is provided in Table 3.4. It is again apparent that a hierarchical approach has led to better local optima than obtained by using multiple levels of h-refinement followed by optimisation. In this particular case, the problem is such that the advantage of local rather than global h-refinement is not so great.

3.4 Summary

In this chapter we have provided details of the possible implementation choices available within our algorithm. A number of adjustable parameters are used within our hybrid algorithm and possible ranges of values for them are provided. A variety of challenging test problems have been considered to test the benefits achieved and the robustness of our multilevel hybrid

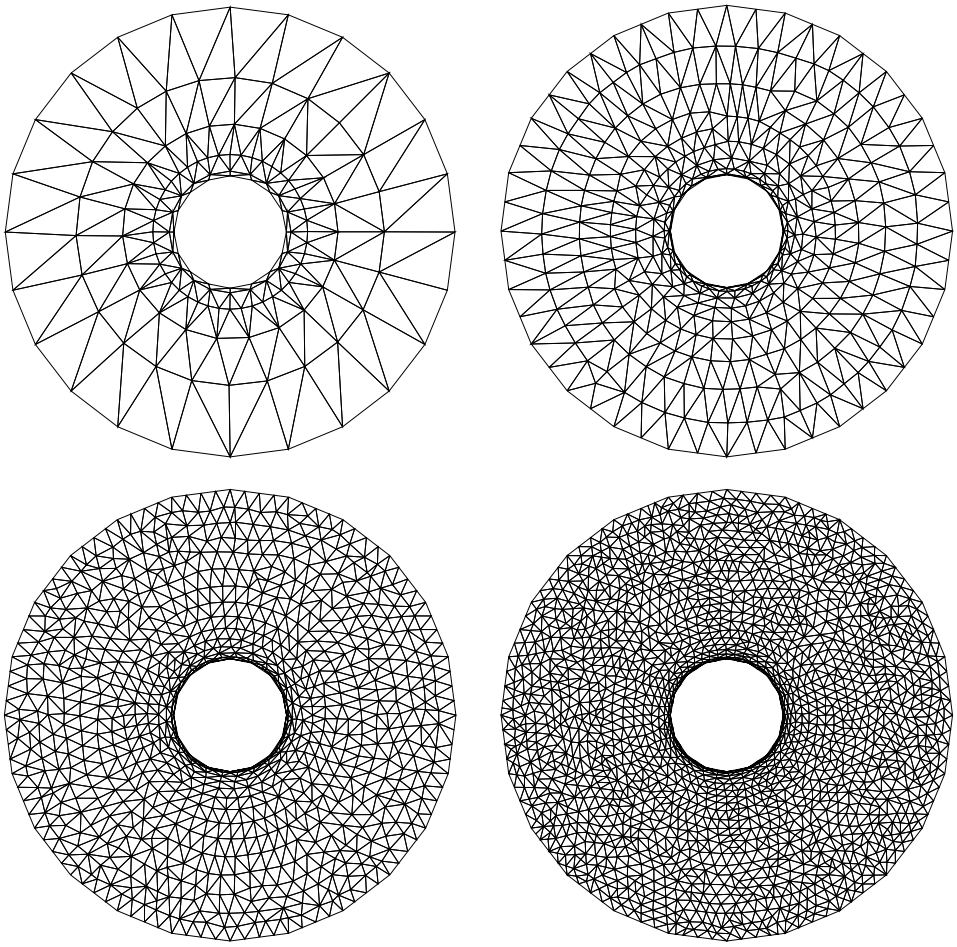

Figure 3.25: A sequence of meshes obtained by applying a combinations of local h-refinement followed by r-refinement on an optimised coarse mesh (shown in Figure 3.23 (top left)).

algorithm.

The test problems considered have clearly illustrated that the quality of the final mesh produced when using the proposed hybrid algorithm is better, in the sense that the finite element solution has a lower energy, than that obtained by using either h-refinement or r-refinement alone. Furthermore it is demonstrated that combining the mesh optimisation with local h-refinement is superior to the global refinement approach used in [104]. Finally, the advantage of using the hierarchical approach, whereby the intermediate level

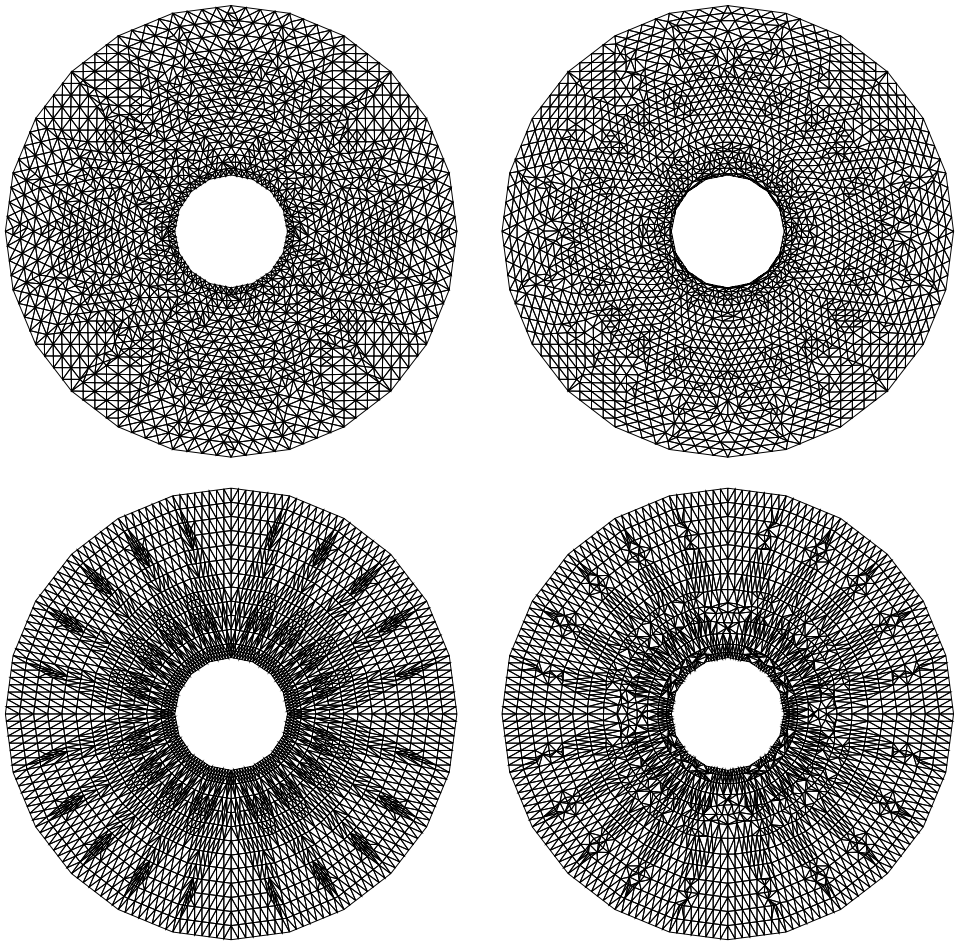

Figure 3.26: A pair of meshes of 4660 elements obtained using local one-to-two h-refinement (top left) followed by optimisation and a pair of meshes of 4896 elements obtained using local one-to-four h-refinement (bottom left) followed by optimisation.

meshes are optimised, is also apparent: an excellent combination of small mesh sizes and low energies for the corresponding finite element solutions being achieved. The fact that no tuning of the adjustable parameters is required for a variety of test problems indicates the robustness of our hybrid algorithm.

When discussing the merits of our proposed algorithm it is important to note that there are some problems for which the benefits may not be

Elements	Energy	Description
80	3.450755	Figure 3.23 (top left)
320	3.422683	Figure 3.23 (top right)
1280	3.415145	Figure 3.23 (bottom left)
5120	3.413707	Figure 3.23 (bottom right)
5120	3.425569	Figure 3.24 (left)
5120	3.414361	Figure 3.24 (right)
80	3.450755	Figure 3.23 (top left)
128	3.429835	
266	3.420375	Figure 3.25 (top left)
442	3.417730	
578	3.415652	
738	3.415428	Figure 3.25 (top right)
1324	3.414515	
1720	3.414160	Figure 3.25 (bottom left)
2454	3.413940	
3523	3.413543	Figure 3.25 (bottom right)
4660	3.421024	Figure 3.26 (top left)
4660	3.414235	Figure 3.26 (top right)
4896	3.418394	Figure 3.26 (bottom left)
4896	3.415132	Figure 3.26 (bottom right)

Table 3.4: Summary of the results obtained for Problem four (the global energy minimum is not known).

quite so substantial as those observed in the first three examples above. A common feature to each of the first three examples is the desirability of clustering the majority of the mesh elements in a relatively small subset of the domain. When a problem is such that the optimal mesh is more uniformly distributed across the domain the local refinement algorithm will show little advantage over the global approach of [104]. Nevertheless, even in this case, our variant of the algorithm performed better than the globally refined approach.

Chapter 4

Further Algorithmic Assessment

As explained previously, the goal of this work is to undertake an investigation into the use of adaptive techniques to obtain high quality locally optimal meshes for variational problems. In particular, our emphasis is on understanding the limits of what meshes it is possible to obtain if one is prepared to invest sufficient computational effort. For this reason we have so far avoided any detailed discussion of the computational costs of the different components of the adaptive techniques that we have considered. In this chapter we take a look at some of the trade-offs between mesh quality and computational time that must be considered when selecting some of the options available within our hybrid algorithm. We also provide an indication of the overall cost of our, unoptimised, implementations of the key components.

4.1 An assessment of the main options in our hybrid algorithm

In this section we consider the main options available within our multilevel hybrid algorithm (which have already been explained in Section 3.1) and their effect on both the mesh quality and computational cost. We continue to measure mesh quality in terms of the total stored energy (2.3) of the computed solution to (2.2), hence for a fixed number of nodes a mesh will be of best quality if the energy of the solution on that mesh is minimum.

The options discussed in this section are concerned mainly with the optimisation part of our hybrid algorithm which is a combination of node movement, Algorithm 2, and edge swapping, Algorithm 3. In particular, we consider whether it is necessary to undertake exact line minimisation when each node is relocated (see Section 3.1.1) or whether an inexact approach (as in Section 3.1.1) is sufficient. We also consider the affect of pre-ordering the nodes and edges before undertaking sweeps of the node movement and edge swapping algorithms respectively. Finally, we consider the value of keeping the hybrid algorithm completely local in nature, or whether some global solves are worthwhile after h-refinement (as suggested in Section 3.1.3)

The choices made for the above mentioned options are significant. Choosing a different option may lead to a different local minimum and/or the number of iterations needed for the algorithms to converge may change. We have no analytical proof to guarantee that any specific option will find a better local minimum or will yield converged mesh and solution in less

computational time. However we provide here some sample numerical results to illustrate the effects of these different options and provide some recommendations on the basis of these empirical results. Wherever a time is quoted it is in CPU seconds on single SG R10000 processor running under the Irix operating system on an Origin 2000 computer.

4.1.1 Line minimisation strategies

As explained in Section 3.1.1, there are two possible choices concerning the movement of a node along its line of steepest descent: exact or inexact line minimisation. It is clear from Algorithms 5 and 6 presented in Section 3.1.1 that the exact line search approach is computationally more expensive than our proposed inexact line search when finding the new position for a single node in general. It is possible however that the exact approach may reduce the overall number of iterations for the node movement algorithm to converge, or it may lead to a better local minimum being found. Of course, we are unlikely to be able to guarantee that one approach will always be superior to the other.

In Table 4.1 results for the solution of Problem One from the previous chapter are presented using both the exact and inexact line search algorithms. We start with an initial coarse mesh of just 32 elements. This is then optimised with respect to the position of the nodes and their solution values, and then global refinement (one-to-four) is undertaken. This gives a mesh of 128 elements which has its solution values updated using a global solve before it too is optimised. This process is repeated to yield a locally

optimal mesh of 512 elements. The times quoted in each row correspond to the combination of the global solution and the optimisation at that level. In each case two sets of results are presented: one which sweeps through the nodes in lexicographic order, and one which pre-orders the nodes at the start of each sweep according to the magnitude of their gradient with respect to the node position.

It is evident from these results that the inexact line search approach is converging in much less time than the exact line search algorithm for both the sorted and the unsorted node orderings. There is no clear pattern for the minimum of the energy functional at each level however. In this particular case the exact line search with the sorted order of nodes finds the lowest value of the energy functional at the final level, however the advantage is very small. Such an outcome is typical but may not always be seen. In view of this very small improvement for a significant difference in computational time we reach the following conclusion.

Recommendation 1: Regarding the efficiency of the mesh movement algorithm, inexact line search is the better choice.

4.1.2 Order of nodes

The number of iterations needed to converge and the particular local minimum of the energy found by our node movement algorithm may depend on the order in which the nodes are visited at each sweep. The movement of nodes is similar to the Gauss-Seidel method since it treats one component of the problem at a time using the most recent information available while

Elements	Energy	time	Description
32	50.902834	23.07	Exact line search
128	50.196339	41.45	with order of nodes
512	50.045417	106.09	
32	50.966222	3.69	Inexact line search
128	50.210542	12.67	with order of nodes
512	50.050063	24.86	
32	50.902128	12.14	Exact line search
128	50.215151	55.71	without order of nodes
512	50.051433	88.06	
32	50.936809	7.413	Inexact line search
128	50.209973	16.76	without order of nodes
512	50.049033	29.13	

Table 4.1: Summary of the results for exact and inexact line searches, obtained for Problem One using exact and inexact line searches.

keeping the other unknowns fixed. Our numerical experiments confirm the expectation that the ordering of the nodes influence the local energy minimum that is found and/or the number of iterations needed to converge. Other than ensuring that convergence is possible for all orderings we have no theory to prove that any particular order will produce a better local minimum, or will reduce the number of iterations needed to converge.

For the results presented in Table 4.1 we see that with the nodes pre-ordered (sorting of the nodes as explained in Section 3.1.4) a better local minimum of energy is found when exact line searches are used. When the inexact line search algorithm is used the unsorted node order produces better results. For $O(N)$ nodes, a good sorting strategy has a complexity of $O(Nlog_2N)$, hence sorting the nodes is a moderate and probably non-negligible expense for large meshes. From the results presented in Table 4.1, the benefit of this sorting overhead is not evident.

Recommendation 2: Sorting of nodes is not necessary to gain efficiency or a significantly improved mesh quality.

4.1.3 Order of edges

As discussed in Section 3.1.4 that the order in which edges are to be swapped may produce a change in the number of iterations needed for convergence or in the local minimum found by the edge swapping algorithm. In this section we provide some typical numerical results to illustrate how the order of the edges affects the performance of Algorithm 3. We considered two orders for the swapping of edges: the lexicographic order and the sorted order, where sorting of the edges is undertaken as explained in Section 3.1.4 (i.e. based upon the stored energy of the latest solution estimate over the quadrilateral in which the edge is contained).

In Table 4.2 results for the solution of Problem One using meshes of 32, 128, 512, 2048 elements are presented. For each regular initial mesh edge swapping, using Algorithm 3, is applied with and without pre-sorting of the edges. For the meshes of 32, 128 and 512 elements sorting the edges produces slightly better results at the expense of a slightly longer time to converge. For the mesh of 2048 elements sorting the edges provides no advantage with respect to either quality or time. The lack of consistency in this, and similar, examples leads us to reach the following recommendation.

Recommendation 3: Sorting of edges is not necessary to gain efficiency and quality of the mesh.

Elements	Initial energy	Final energy	time	Description
32	374.472889	357.666756	0.12	Edge swapping
128	189.646235	180.401346	0.43	with sorting
512	103.630270	99.7761695	1.59	
2048	67.400215	65.1759131	7.00	
32	374.472889	357.670496	0.12	Edge swapping
128	189.647236	180.443140	0.37	without sorting
512	103.630270	99.9357822	1.24	
2048	67.400215	64.9645894	6.30	

Table 4.2: Solutions to Problem One using edge swapping: with and without sorting the edges.

4.1.4 Global solves at intermediate levels

Our hybrid algorithm may be described as purely local since it does not need a global finite element solve at any stage. Global solves can be included in the algorithm at appropriate points however. As suggested in Section 3.1.3, by including a global finite element solve immediately after h-refinement has taken place, it is possible that the convergence process of our hybrid algorithm may be accelerated and/or a better local minimum might be found. Our numerical experiments confirm this expectation. However some extra computational work is required to undertake these global solves at these intermediate stages. On the other hand the node movement algorithm may converge in fewer iterations following a global solve since it updates the nodal solution values (by solving the local problems (2.5)) as well as the nodal positions.

In this subsection we present some typical numerical results to discuss this issue. In Table 4.3 we present timings and energies for the solutions to Problem One with and without global solves at intermediate levels. It is

Elements	Energy	time	Description
32	50.893721	63.72	With global solves at
128	50.113745	133.65	intermediate levels
512	50.015823	256.22	
32	50.893721	63.72	Without global solves
128	50.137345	195.17	at intermediate levels
512	50.035632	327.29	

Table 4.3: Results with and without global solves at intermediate levels

clear from the results that the quality of the mesh achieved at each refined level is better when the option of intermediate global solves is used. In addition, the overall solution time is reduced. When the option of undertaking global solves at intermediate levels is not used the node movement algorithm, which not only moves the nodes but also updates the nodal solution values, is clearly doing more work.

We have used the software SPARSKIT [94] (which uses the Generalised Minimal RESidual (GMRES) method [95]) for the global solution of the problem. This iterative solver, with right ILU preconditioning, is very efficient and so only a small time is required to solve the global problem on these meshes. However for problems where the global solve is more expensive, such as very large nonlinear problems for example, this option may not be as beneficial, with respect to the efficiency gain, as for this example. Nevertheless we reach the following conclusion.

Recommendation 4: The inclusion of an intermediate global solve after h-refinement is a good option.

4.2 Computational cost

It is clear from the results presented in Section 3.3 that, for a given number of elements, the approximations of the solutions obtained using our multilevel hybrid algorithm are more accurate (the accuracy of the approximation being measured in terms of the total stored energy), than the other adaptivity approaches used in Section 3.3 for comparison. We deliberately do not include any efficiency comparison of our hybrid algorithm with the other adaptive approaches since our goal is to strive for the best possible meshes. Furthermore, a fair numerical comparison is only possible when each of the approaches is most efficiently implemented, using the best iterative solvers, data structures, software optimisation, etc. However, for completeness, in this section we provide some indications of the computational cost (in terms of time) involved in various steps of our hybrid algorithm. For all of the results presented in this section the initial mesh considered is a uniform mesh of 512 elements and the test problem solved is again Problem One from Chapter 3.

4.2.1 Cost involved in various steps of our hybrid algorithm

Computationally the most expensive part of our multilevel hybrid algorithm is the node movement algorithm. When the node movement algorithm (with sorting of nodes and exact line searches) is applied on the initial mesh, the time consumed in the first iteration of node movement is 0.96 seconds and the overall time consumed up to the convergence of the node movement

algorithm is 138.12[1] seconds. From the recommendations in the previous section we may use inexact line searches and the lexicographic node ordering however. In this case the time required for a single node movement iteration is 0.10 seconds, and the overall time up to convergence is 59.01 seconds.

Edge swapping is computationally much less expensive than node movement and can produce significant energy reduction. Generally the edge swapping algorithm only takes a few iterations to converge. From our numerical experiments we concluded that for a mesh of E elements less than $log_e E$ iterations are needed for its convergence and this observation is true for all of our test problems. Moreover there are only two valid options available for an edge to be swapped (in two dimensions), determination of the optimum of these two choices is computationally inexpensive. Hence the time consumed in the first iteration of edge swapping is 0.60 seconds and the total time consumed up to convergence of the edge swapping algorithm is 1.59 seconds. Time for the convergence of the combination of node movement and edge swapping algorithm, with inexact line search and no ordering of the nodes and the edges, is 140.83 seconds.

To illustrate the affect of this mesh optimisation, Figure 4.1 shows the energy distribution per element (in a sorted order) before and after applying the iterative combination of the node movement and edge swapping algorithms. Before optimising the mesh, the maximum energy on any one element is 5.13 and after optimising the maximum energy on a single element is 1.37. It is clear from these figures that the optimisation part of our

[1]The initial mesh considered here is a uniform mesh of 512 elements

hybrid algorithm is very effective in reducing the energy for this particular problem. It is also clear that this improvement is extremely computationally expensive. Especially if the node movement is undertaken to full convergence.

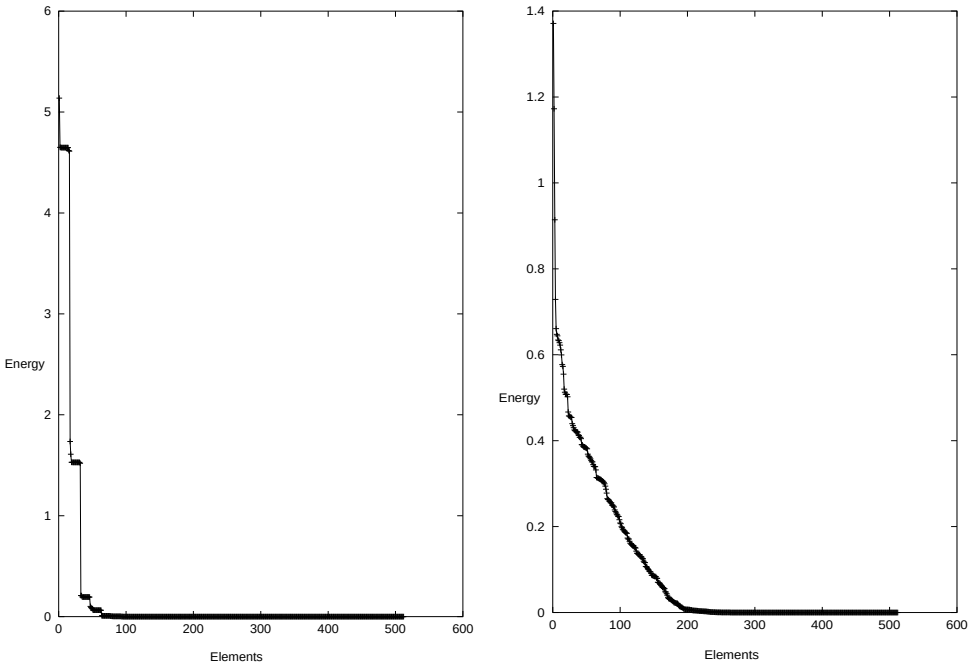

Figure 4.1: Energy distribution per element before and after optimisation (note the difference in the vertical scale in each graph).

4.2.2 Approximate movement of nodes

It may be observed that the energy decreases more rapidly in first few iterations of node movement than in later iterations. This is illustrated in Figure 4.2 where the total stored energy of the solution is plotted against the number of iterations of the node movement procedure. It is clear from this figure that most of the significant change appears only for the first few iterations. After this the change in the energy is relatively small. Given

that the timings quoted in the previous subsection show that most work is undertaken in the node movement algorithm, an efficient version of this algorithm requires a strategy that stops the node movement iteration before full convergence.

The obvious way to overcome this situation is to increase the threshold for convergence of the node movement algorithm. Our node movement algorithm reduces the energy monotonically but the energy does not reduce by a constant factor in each iteration. It is possible that when nodes have moved close enough to a local minimum, they start moving towards another better local minimum and the increased convergence criteria may be satisfied before the nodes have settled down for a local minimum. The other possible ways are to put a reasonable limit on the maximum number of node movement iterations or to monitor the rate at which the energy is being reduced. More research is needed to develop a suitable strategy that stops the node movement algorithm before its full convergence. In this way efficiency of our hybrid algorithm can be increased a lot on a slight loss of the quality of the mesh.

4.2.3 Approximate swapping of edges

We observe in practice that most of the edges are swapped in the first two or three iterations on typical meshes. Figure 4.3 shows the total stored energy of the solution versus the number of iterations of the edge swapping algorithm considered in isolation. It is possible for the implementation of the edge swapping algorithm to provide a stopping criterion before convergence

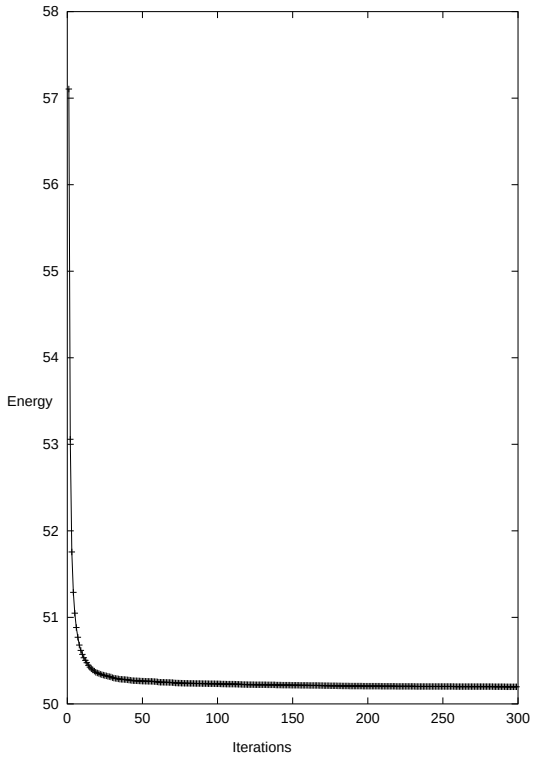

Figure 4.2: Energy versus number of iterations of node movement.

is achieved. This could involve the use of a small tolerance, a reasonable limit on the number of maximum edge swapping iterations or be based upon the rate at which the energy is being reduced. It is likely that less significant efficiency gains can be achieved by adopting this strategy than is the case with the node movement algorithm however.

4.3 Summary

In this chapter we have considered the four main options within our hybrid algorithm and discussed them with respect to the efficiency and the quality of the meshes achieved. We provide some simple empirical recommendations on the basis of these typical computational results. We have also provided

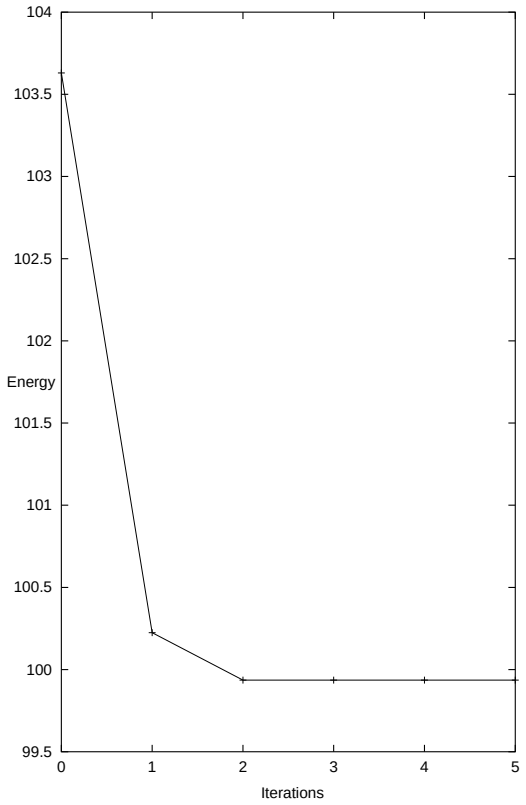

Figure 4.3: Energy versus number of iterations of edge swapping.

some results to give an idea of how much time is consumed in different parts of our hybrid algorithm for a specific problem. These computations show that the expense of obtaining the highest quality optimal meshes and solutions is too great to be included as part of a practical adaptive algorithm. We have therefore outlined a number of ways in which the costs can be substantially reduced with only very small affects on the quality of the final meshes produced.

We have not provided any comparison of our hybrid algorithm with other adaptive approaches with respect to computational cost or efficiency. This is because we have developed our hybrid algorithm keeping in mind primarily

the robustness and the quality of meshes achieved rather than the efficiency. Furthermore, it would be extremely difficult to carry out fair comparisons since a large amount of time and effort would need to be spent optimising the code produced. Nevertheless, we believe that there is considerable scope for developing our hybrid algorithm to enhance the efficiency, as explained in the previous sections: by having inexact line searches and non converged node movement and edge swapping, for example. Theoretically it is possible to achieve any desired accuracy by the use of h-refinement alone (either local or global). However the time consumed to solve a problem increases with the number of elements in the mesh. To achieve roughly the same accuracy, as achieved by our hybrid algorithm for test Problem One with 323 elements (see Table 3.1), more than a million elements are required for a uniformly spaced mesh. For a locally refined mesh about 150000 elements are required to get roughly the same accuracy. It is in this context that the relatively high cost of our hybrid algorithm should be considered.

Chapter 5

Implementation and Numerical Results in 3-Dimensions

In this chapter we consider the extension of the preceding work on multi-level r and h-refinement into three dimensions. This is done by providing implementation details of a generalisation of the two dimensional hybrid algorithm and then presenting some numerical results in three dimensions. Many of the implementation details in three dimensions are the same as in two dimensions, as given in Chapter 3, hence it is the differences in the implementation that will be discussed in this chapter. The most significant of these differences concern the edge swapping and node insertion algorithms, which are far more complex in three dimensions than two, and so much of this chapter is devoted to these issues.

We begin the chapter with a brief description of the differences in the implementation of node movement for three dimensions. These differences are quite minor and therefore require only a short description. Section 5.2 however is devoted to face/edge swapping and Section 5.3 describes the refinement algorithm that we use in three dimensions. To test the

effectiveness of the hybrid algorithm, two sample problems are considered in Section 5.4 and corresponding results are presented for these problems to those presented for the two-dimensional problems in Chapter 3. In Section 5.5 we provide a summary of this chapter.

Unlike in two dimensions, for these three dimensional numerical simulations we have not developed the codes for edge swapping and local h-refinement algorithms ourselves from scratch. Instead, we make use of other available software for edge swapping and mesh refinement as tools, and modify them to fulfill our criteria of energy minimisation. For edge swapping we make use of the software package GRUMMP [52] which is in the public domain as open source. This is based on the improvement of the geometric quality of the mesh and we briefly describe how it works and the alterations that we have made to it in Section 5.2.3. For mesh refinement we use the private package TETRAD [99] which was developed within School of Computing, Leeds. This is based on refinement and derefinement of tetrahedra according to user defined criteria, provided to it through a given interface. Details of TETRAD are given in Section 5.3.1. It should be noted that we must implement our node movement algorithm within the constraints of the existing interface for TETRAD. Fortunately this proved to be reasonably straightforward in practice.

To modify GRUMMP and to provide a suitable interface for TETRAD for use in our hybrid algorithm were not easy tasks, but were definitely more straightforward than developing such codes independently. However

there are two main drawbacks in using these tools. First, each of them uses different data structures and file formats for input, so they can not easily be integrated. The output file generated from one tool must be modified in order to provide input to the other tool. Second, unlike with our implementations of edge swapping and refinement in two dimensions, we have less freedom to choose and contrast different options within the hybrid algorithm, such as the maximum number of edges attached to one node, the use of different types of refinement, the orders in which edges are swapped, etc. Nevertheless, using GRUMMP and TETRAD as tools, enabled us to combine the three components of node movement, face/edge swapping and local h-refinement to generalise our new multilevel hybrid algorithm to three dimensions and to obtain provisional computational results.

5.1 Node movement

In three dimensions the strategy and the theory behind node movement is the same as presented for two dimensions in Section 2.2. The derivatives of the energy with respect to the nodal positions may still be computed using (2.10) (the index d now runs from 1 to 3) with a single loop over the elements of the mesh. If desired, this list may then be sorted and, beginning with the largest values of $|\frac{\partial E}{\partial \underline{v}_j}|$, the nodes may be moved in turn. In each case moving the node, j say, in the polyhedron Ω_j (formed by the union of all elements which contain node j) requires the solution of a local one-dimensional energy minimisation problem in the direction of steepest

descent (given by (2.4)). Once the updated location of node j has been found it is a simple matter to modify the corresponding solution value by solving local problem (2.5) on the patch of elements, Ω_j, surrounding the node j.

In three dimensions nodes on the surface of the domain are allowed to move in any direction subject to the constraint that the domain remains unchanged. Points which lie on a planar part of boundary are allowed to move only on that planar boundary section. Points which lie on the line of intersection of two planes or at a line of intersection of two different types of boundary condition are allowed to move only on that line. Although we have not considered any problem domains having a curved boundary in this chapter, the same rules apply as given in Section 2.2.1; the points at the corners or lying on the curved boundaries are not allowed to move. It could be possible to allow constrained movement of some points along curved boundary sections but this is not considered in this work.

As with the two dimensional case, we also introduce artificial constraints on the node movement to prevent the possibility of mesh tangling. The minimum area limit A_{min} used in Section 3.2.1 is replaced by V_{min}, a minimum volume limit of an element in the mesh. We have taken $V_{min} = 10^{-7}$ for all of the test problems presented in Section 5.4 but the exact choice is not critical. The maximum admissible limit (as defined in Section 2.2) up to which the node j can be moved in the polyhedron Ω_j is defined in a more cautious way than in two dimensions, as there are more chances of tangling

in three dimensions. We find the smallest edge length L_{min} in Ω_j and allow the node j to move in the direction of steepest descent up to a maximum distance of $w \times L_{min}$, where w is maximum admissible limit factor as defined in Section 3.2.4.

5.2 Face/Edge swapping

The aim of undertaking face or edge swapping in three dimensions is the same as discussed in Section 2.4 for two dimensions, i.e. to reduce the overall solution energy by reconnecting the mesh locally. We have generalised the two dimensional minimum energy triangulation algorithm of Ripa and Schiff [91] to three dimensions. This is achieved by applying face or edge swapping on the interpolant of the current solution, thus eliminating the need to solve the global problem repeatedly. Unfortunately, to achieve an optimal mesh with respect to connectivity of the vertices in three dimensions, based upon the two-dimensional definition given in Section 2.4, may not be possible: this is explained later in this section. We begin however with a brief description of the two classes of local mesh reconfiguration method that are available to use in three dimensions:

- face swapping,
- edge swapping.

These two techniques are discussed in detail below.

5.2.1 Face swapping

Face swapping is a three dimensional version of the common two dimensional diagonal edge swap. It is based on the possible configurations of sets of five distinct non-coplanar three dimensional points [67, 73]. Each interior face in a tetrahedral mesh separates two tetrahedra which contain a total of five points between them. However these five vertices may be reconnected to form two, three, or four new tetrahedra as shown in Figure 5.1. The five different configurations of five non-coplanar points A,B,C,D and E are shown in Figure 5.1 and Figure 5.2. The most common configuration to arise is configuration 1 in Figure 5.1, but the others can all occur depending on the geometry of the points $ABCDO$.

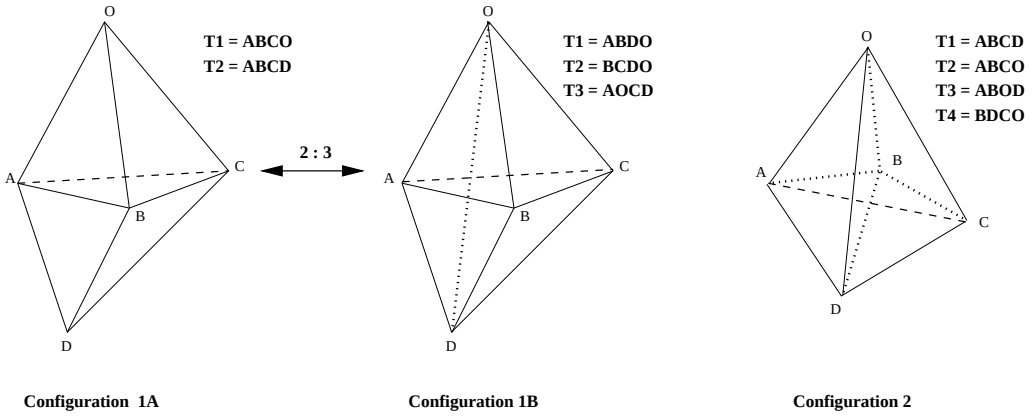

Figure 5.1: Possible configurations of five points where no four of the five points are coplanar.

In the two configurations shown in Figure 5.1, no four of the five points are coplanar. For all of the figures in this chapter solid lines are used to show the front view of the diagram, lines with dashes show the back of the diagram and dotted lines are used in the interior of the convex hull of the

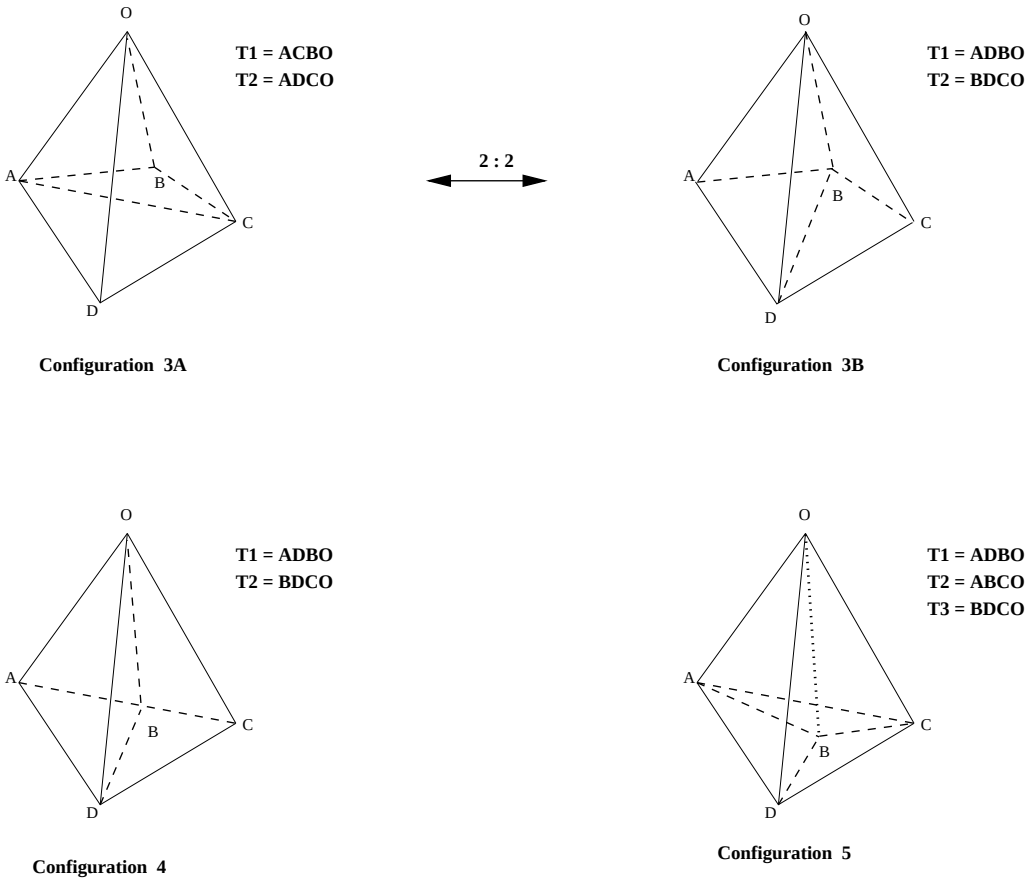

Figure 5.2: Possible configurations of five points where four of the five points are coplanar.

points. In configuration 2 the point B is in the interior of the convex hull formed by the points A, C, D and O. Face swapping is possible only if the five points are in configuration 1 where the triangulation changes from the A to the B version or vice versa. In the three configurations shown in Figure 5.2, the points A,B,C and D are coplanar, forming the possible configurations for these planar points. In configuration 4 points A, B and C are collinear. Face swapping is possible only if the five points are in configuration 3 where configurations $3A$ and $3B$ are interchangeable.

For face swapping, each reconfigurable case has only two valid config-

urations [73], hence a simple and quick comparison to find the one with the lower energy is possible. For this discussion of swappable and non-swappable configurations of these five points we follow the notation used by Joe [68]. Face swapping can be implemented by making a loop over all internal faces in the mesh and assigning a type S_{ba} or N_{ba}, where S stands for a swappable and N stands for a non-swappable face, b is for the number of tetrahedra before the swap and a is the number of tetrahedra after swap. The possible types of faces are S_{23}, S_{32}, S_{22}, S_{44}, N_{32}, N_{20}, N_{30}, N_{40} and N_{44}. These are in all nine types and we explain each of them in turn.

For each of these face types we consider the configuration of the five points in the two tetrahedra which share this face. For the case S_{23}, consider the configuration $1A$ in Figure 5.1. Face ABC will be assigned a type S_{23}, as by introducing an edge in this five point configuration we can get configuration 1B with three tetrahedra. For S_{32} consider the common face BOD of tetrahedra $ABDO$ and $BCDO$ in configuration 1B of Figure 5.1. The edge OD is surrounded by three tetrahedra and can be removed from this five point configuration giving configuration $1A$ of two tetrahedra. If tetrahedron $AOCD$ is not present there (i.e. edge OD is surrounded by more than three tetrahedra), then face BOD will be assigned a type N_{32}, as the three to two re-configuration is not possible for these five points. For the case S_{22} consider the configuration $3A$ and the face OAC in Figure 5.2. The four coplanar points $ABCD$ are on the boundary of two tetrahedra in $3A$ and the face OAC can be swapped to face OBD to yield two new

tetrahedra as in $3B$. Hence the face OAC in $3A$ will be of type S_{22}. This type of face is possible only if the two tetrahedra which share the face in consideration have one face on the boundary.

For the case N_{40}, consider configuration 2 in Figure 5.1. Face OBD is of type N_{40}. For the case N_{20}, consider the configuration 4 in Figure 5.2. Face OBD is of type N_{20}. For the case N_{30}, consider the configuration 5 in Figure 5.2. Face OBD is of type N_{30}. All of these three configurations are non convex and are the only possible configurations for these given five points.

There are two special cases, S_{44} and N_{44}, considered in assigning a type to a face. These special cases arise when for a face, four of the five points forming the two tetrahedra (which shares the face in consideration) are coplanar and these coplanar points are not on the boundary. Consider the face OBC in Figure 5.3, where points A, B, C and D are coplanar. If these coplanar points are backed by exactly two tetrahedra, as is the case in Figure 5.3, then face OBC will be of type S_{44}, indicating that it is possible to swap face OBC with OAD. If these coplanar points are not backed by exactly two tetrahedra on each side then face OBC will be of type N_{44}, as the face OBC can not be swapped with OAD.

Face swapping has two significant advantages. First, because there are at most two valid reconfigurations determining the optimum of these configurations is computationally inexpensive. Second, implementation of face swapping is relatively straightforward compared to implementation of edge

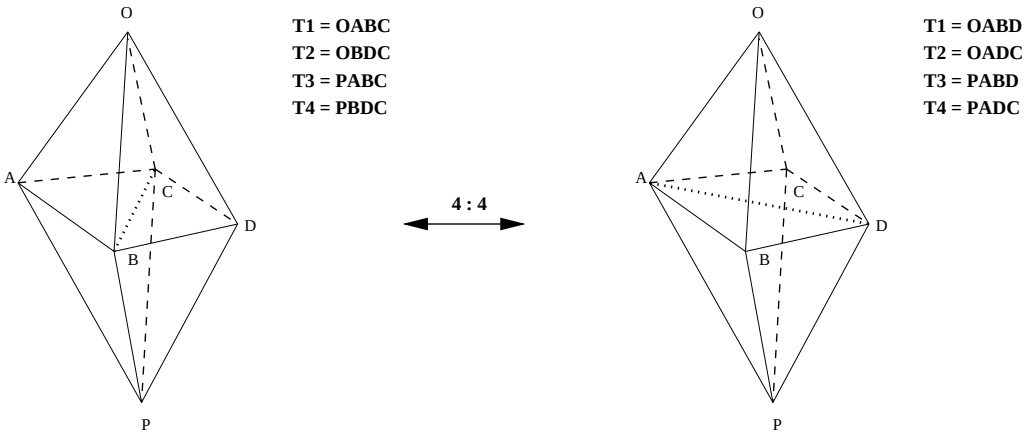

Figure 5.3: Face swapping for two interior coplanar faces.

swapping, as described below.

5.2.2 Edge swapping

Edge swapping in three dimensions is not really a swap but a removal of an edge followed by its replacement by one, two or many edges depending upon how many elements surround that edge (see Figure 5.4 for example). Edge swapping reconfigures the E tetrahedra incident on an edge of the mesh by removing that edge and replacing these E tetrahedra by $2E-4$ new tetrahedra. As an example, consider an initial configuration with five tetrahedra incident to an edge. The left side of Figure 5.4 shows five tetrahedra incident to an edge OP and the right side shows one possible reconfiguration of this submesh into six tetrahedra. This new configuration is specified by defining three "equatorial triangles", i.e. which are not incident on either of vertices O and P. In Figure 5.4 these triangles are $\triangle 124$, $\triangle 234$ and $\triangle 145$. There are four other possible configurations for this case (each corresponding to a different set of equatorial triangles), which can be obtained by rotating

the interior triangle in Figure 5.4. As edge swapping replaces E original tetrahedra into $2E-4$ tetrahedra, when $E > 4$ more elements are created than are removed.

In addition, the number of possible ways that elements can be reconnected after deleting an edge increases with increasing E and is given by

$$C_r = \frac{(2E-4)!}{(E-1)!(E-2)!} \tag{5.1}$$

(see [51]). When $E = 5$ this gives the five possibilities noted in the previous paragraph. However, as E grows the number of possible configurations grows very rapidly and so, following [48], we only consider edges with $E < 8$ as candidates for edge swapping. The possible configurations for $4 \leq E \leq 7$ are shown diagrammatically in Figure 5.5, where equatorial triangles are shown along with the number of unique rotations for each configuration. An optimisation method therefore has to search through a large number of connectivity permutations for large E in order to determine which reconfiguration of the original E tetrahedra has the lowest energy. For this we have to compute the energy for each tetrahedron in each configuration. Fortunately, when E is large, the number of unique tetrahedra is much smaller than the number of configurations times the number of tetrahedra since many tetrahedra appear in more than one configuration. This is shown in Table 5.1 (see [48]) and means that the cost of performing a local mesh optimisation is not quite as high as (5.1) initially suggests.

Edge swapping is normally used as a supplement to face swapping. As we saw in the previous subsection there are many situations where face

Tets before	Tets after	Configurations	Tets × configs	Unique tets
4	4	2	8	8
5	6	5	30	20
6	8	14	112	40
7	10	42	420	70

Table 5.1: Number of unique tetrahedra and possible configurations for edge swapping.

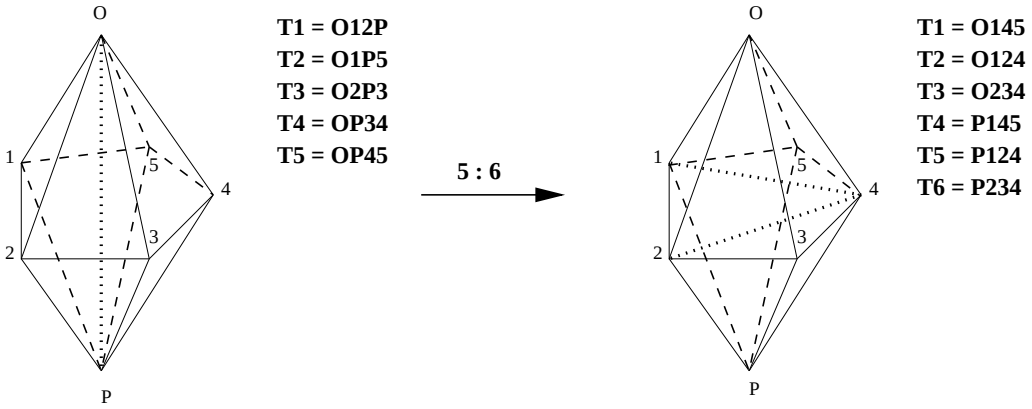

Figure 5.4: Edge swapping for 5 tetrahedra to 6, where edge OP is surrounded by 5 tetrahedra.

swapping is not possible hence edge swapping is then invoked to determine whether removing an edge is advantageous or not. There are two main disadvantages to the edge swapping technique. First, for each original configuration, determining the optimum configuration requires comparing multiple configurations which can be a computationally expensive process when E is moderate or large. Second, the practical implementation of edge swapping is relatively difficult.

Now we explain why an optimal mesh with respect to connectivity of vertices (according to the definition given in Section 2.4) may not be possible. In two dimensions exactly two elements share an internal edge, while in three dimensions an internal edge is shared by three or more elements.

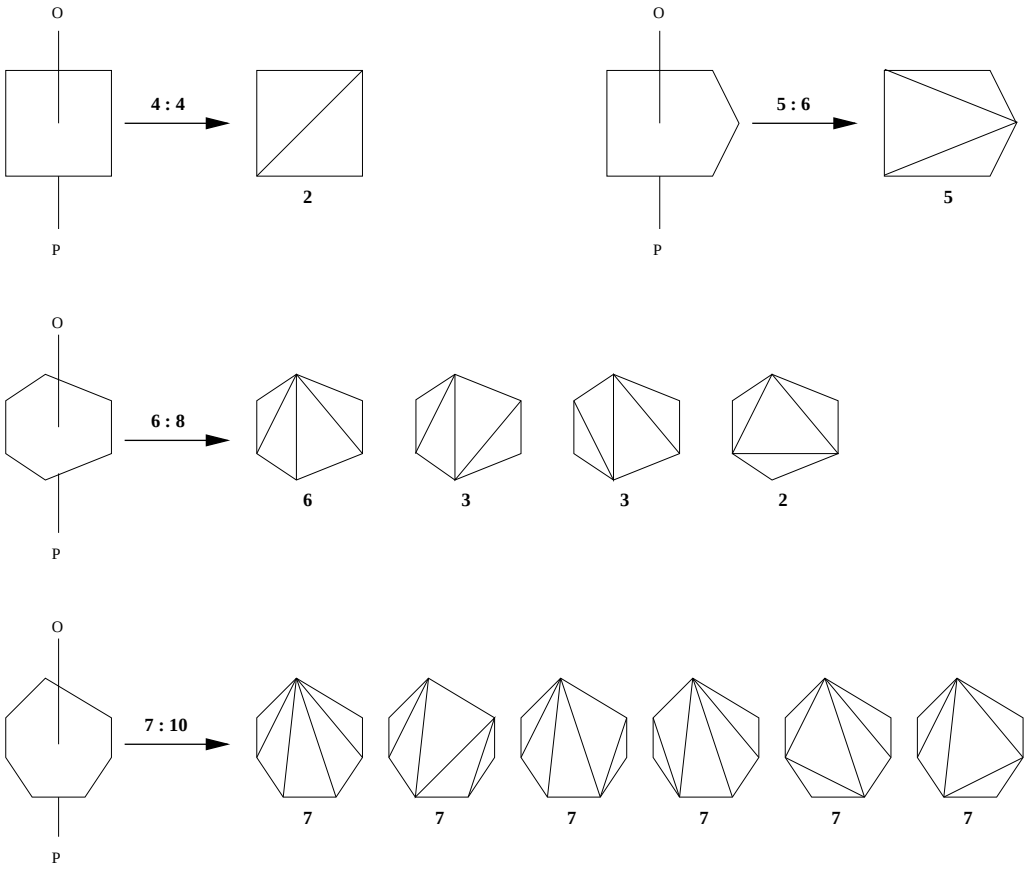

Figure 5.5: Equatorial triangles after swapping edge OP, surrounded by 4,5,6 and 7 tetrahedra, including the number of unique rotations for each configuration shown.

Thus local mesh reconfiguration in three dimensions involves changing the connectivity of two (for a simple face swap), three or many elements, with the possibility of introducing new elements and edges. It is clearly far from straightforward to guarantee the termination of an energy minimisation algorithm based upon this approach. Moreover, from a programmatic point of view, it is quite impossible to consider all edge swaps, as in principle a single edge swap (when E is very large) may involve changing the connectivity of more than a dozen elements with thousands of possible configurations for these elements (see (5.1)), to check for one optimal local configuration.

Despite this fact we have observed that for each of our test problems (given in Section 5.4), when GRUMMP (modified to minimise energy rather than geometric quality), which does not consider any edge swap for $E > 7$, was applied it always terminated successfully. Whilst the final mesh may not therefore be locally optimal in a strict sense our expectation is that it will still be of a high quality.

5.2.3 GRUMMP

GRUMMP [52] stands for Generation and Refinement of Unstructured Mixed-Element Meshes in Parallel. It is a set of libraries, written in C++, for manipulating unstructured meshes, and a set of executables built on those libraries. GRUMMP is available on the WWW from the GRUMMP home page **http://tetra.mech.ubc.ca**. Source code of GRUMMP is available for licence without fee, for educational and non-profit research purposes, to use it with or without modifications. It can be configured on almost all operating systems.

The software is still in the process of development and some of its features such as parallelism, mixed-element meshing and generation of meshes for non-flat surfaces, are still unimplemented. However GRUMMP is capable of generating and optimising meshes, using smoothing and swapping, for quite complex multi-domains in two and three space dimensions.

We briefly describe how GRUMMP works and then we provide details of the alterations that we have made to it. GRUMMP uses geometric quality criteria to optimise the mesh. It tries to improve the geometric

quality of each poor quality tetrahedron by face/edge swapping, smoothing or by adding points at strategic positions. It also removes the bad geometric quality tetrahedra which can not be improved by any strategy. Its face/edge swapping routines use three geometric quality measures to determine whether to locally reconnect a tetrahedral mesh.

- The maxmin sine of dihedral angle criterion chooses the configuration that maximises the minimum sine of the dihedral angles of the tetrahedra in a patch.

- The minmax dihedral angle criterion minimises the maximum dihedral angle of the tetrahedra.

- The in-sphere criterion, appropriate only for face swapping, selects the configuration in which no tetrahedra in the five point local submesh contains the other point in its circumsphere.

Our aim of optimisation is to reduce the energy of the solution rather than any of the above geometric criteria. In fact our optimal meshes may include some poor geometric quality elements since, as discussed in [90] for example, long and thin elements can form part of good mesh for an anisotropic solution. Hence, the first task of modifying GRUMMP was to isolate the face/edge swapping routines from the other mesh improvement routines.

The second task in modifying GRUMMP was to introduce a field for the solution value for each node. As GRUMMP is totally based on improving

the geometric quality of the mesh it does not need, or support, any solution value within it. We introduced solution values by modifying the GRUMMP data structure 'node to vertices'. This was achieved by increasing the dimension of the vertices for the three dimension routines from three (i.e. x,y,z) to also include a solution value for each node. Accomplishment of this task was not entirely straightforward however. GRUMMP routines use a C++ function 'assert' which checks the dimension of vertices at hundreds of places in the software, and if this dimension does not match with the previous ones, an error is generated. Also, if one routine is passing this dimension to the other routines, they must match. It was necessary therefore to take great care to change this dimension wherever it appeared in the software. In a second version, when we applied GRUMMP to systems of equations, we changed this dimension to accept three solution values for each node.

The third task in modifying GRUMMP was to replace its geometric quality criteria with our energy quality criterion. This involved replacing the GRUMMP routines which compute the geometric quality of each tetrahedron with our own routine which computes the energy over each tetrahedron in the original and possible configurations. We put a minimum volume limit, V_{min} ($= 10^{-7}$ say), for each tetrahedron in the original and possible configurations. If any of the elements has volume less than V_{min} we assign that tetrahedron a very large value of energy, which rules out its inclusion in the final configuration. It should be noted that GRUMMP reconfigures

the submesh formed by E tetrahedra if every new tetrahedron in a possible reconfiguration has a better quality than the worst of the original E tetrahedra. This is in contrast to our energy quality criterion. Our reconfiguration of this submesh should be based on the cumulative energy of each tetrahedron in the submesh. Hence a final modification to GRUMMP was necessary such that a reconfiguration will be performed if the cumulative energy of the submesh formed after face/edge swapping is lower than the cumulative energy of the original configuration of the submesh.

5.3 Local h-refinement

Our local h-refinement is again based on reducing the solution energy, by increasing the number of elements in specific regions. Refinement in three dimensions is much more complicated than in two dimensions, due to extensive programming efforts and visualisation difficulties. To discuss refinement in three dimensions we use the terms (as discussed in Section 2.5), *regular/red refinement* for refining the elements and *irregular/green refinement* for treating the hanging nodes, which appear due to local regular refinement in the neighbouring elements.

There are three common types of refinement for three dimensional tetrahedral elements. In explaining these types of refinement we use the terms (as used in [98]), *parent elements* for the elements which are refined and *child elements* for the elements created by refining the parent element. The most popular type of refinement in the literature appears to be the one

which divides the parent element into 8 child elements by bisecting all of the edges of the parent element. This type of refinement is also named as *regular refinement* and is presented in, e.g. [76, 83, 98]. Figure 5.6 shows a tetrahedron divided into 8 tetrahedra by introducing nodes at the mid points of each edge. Each new node is then connected to the other two new nodes lying on each face. This cuts off four subtetrahedra at the corners of the parent tetrahedron. Finally, joining any two new nodes on unconnected parent edges by an internal diagonal dissects the remaining central octahedron into another four child elements. The choice of which internal diagonal to insert is important: the approach in [76, 98] is to choose the longest one. Other approaches in choosing this diagonal are also possible [83].

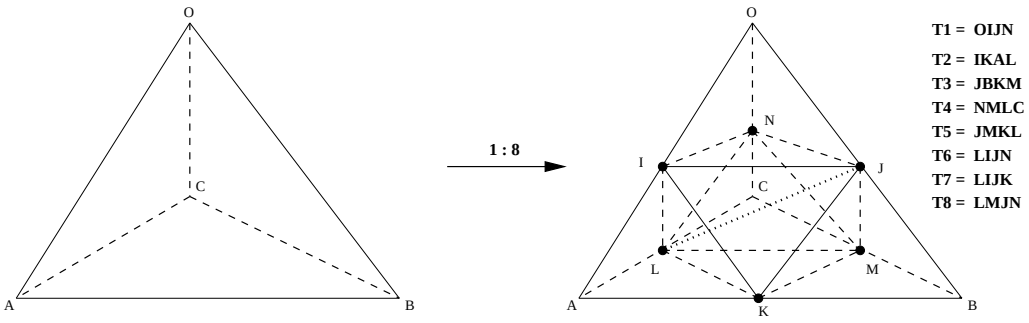

Figure 5.6: Regular refinement of a tetrahedron into 8 child tetrahedra, by bisecting all of the edges.

The second type of regular refinement is *directional regular refinement*, which is considered in [98]. This type of refinement is important to resolve certain flow features such as boundary layers in viscous problems [31]. It is based upon a directional dissection of the parent tetrahedra. Figure 5.7 shows a tetrahedron divided into 4 child elements using directional refine-

ment. Three edges on one face are bisected. The new nodes are joined to each other and to the opposite vertex of the tetrahedron. The choice of which face to subdivide depends on the direction in which refinement is required.

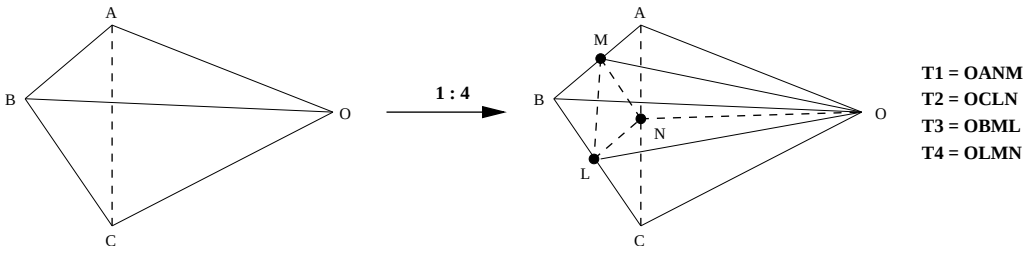

Figure 5.7: Regular directional refinement of a tetrahedron into 4 child tetrahedra by bisecting edges on one face.

The third type of regular refinement is *bisection*, which bisects the parent tetrahedron. The bisection method in three dimensions is presented by [26, 79] for example. Figure 5.8 shows a tetrahedron divided into 2 child elements using the bisection method. In this method any of the edges is bisected and the newly created node is joined to the opposite vertices as shown in Figure 5.8. The edge to be bisected can be chosen on some geometric quality criteria (e.g. to bisect the longest edge [92]) or some directional refinement criteria.

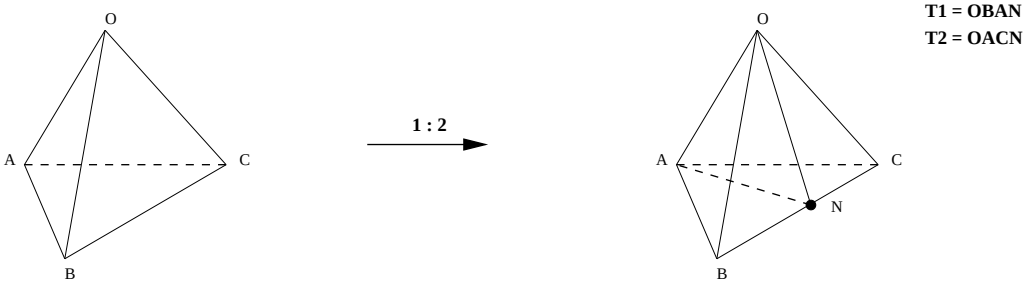

Figure 5.8: Bisection of a tetrahedron into 2 child tetrahedra by bisecting one edge.

The consequence of a local regular refinement is the existence of tetrahedra with hanging/green nodes. These are tetrahedra which were not refined themselves but at least one of their edges is common with a regularly refined element. For the conformality of the mesh these hanging nodes must be removed, which is generally achieved through the use of a transitional refinement layer. There are two common types of irregular refinement; one which does not insert a node at the centre of the parent element, e.g. [7, 30], and the other which does introduce a node into the parent element, e.g. [71, 98]. For both of these types six refinement possibilities can arise (i.e. 1 to 6 edges of a tetrahedron have hanging nodes). However for each type, if all of the six edges have hanging nodes the best strategy is to refine the parent element regularly as shown in Figure 5.6, see e.g. [30] and [98]. Both irregular refinement types and the remaining five possibilities are discussed below.

For the green refinement which does not introduce nodes into the parent element only four cases (shown in Figures 5.9−5.12) have been considered in [7, 30]. For the rest of the cases, where three or more edges have hanging nodes not belonging to a common face, regular refinement is used as shown in Figure 5.6. For this red refinement closure new nodes on the rest of the edges are introduced and it may be that some of them lie at edges of green tetrahedra. Hence the refinement process must be treated as being iterative. For the completeness of our discussion of this type of irregular refinement, we present the rest of the cases as shown in Figures 5.13−5.17. In all of

these cases the hanging nodes are connected without the introduction of any new nodes to the mesh.

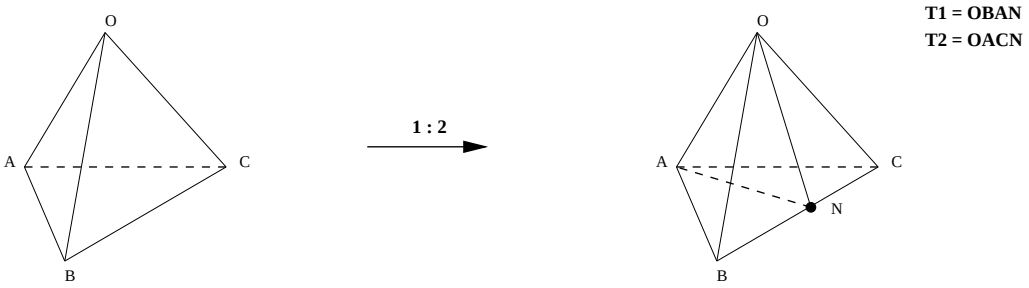

Figure 5.9: Division of a tetrahedron with 1 green node into 2 subtetrahedra.

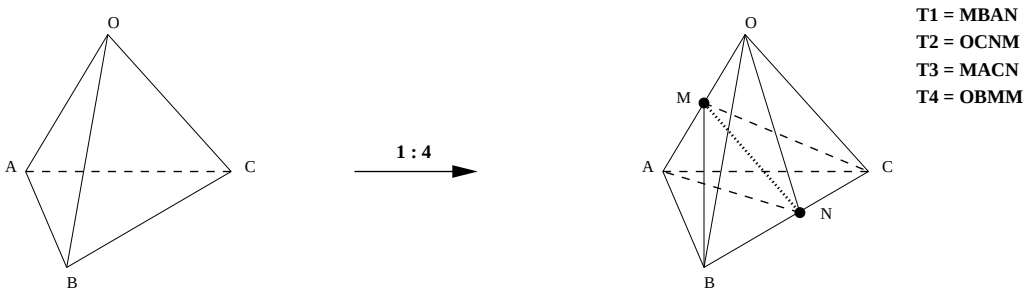

Figure 5.10: Division of a tetrahedron with 2 green nodes on opposite edges, i.e. on two different faces, into 4 subtetrahedra.

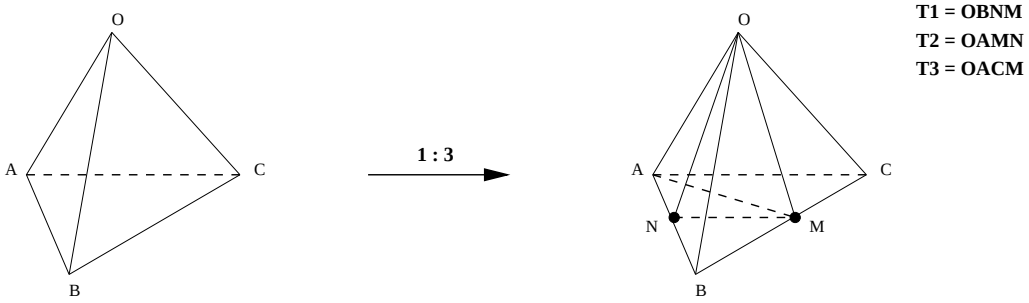

Figure 5.11: Division of a tetrahedron with 2 green nodes on adjacent edges, i.e. on one face, into 3 subtetrahedra.

For all of the cases shown in Figures 5.9–5.17, if two or more green nodes lie on one face they are joined together (i.e. in the same face). When exactly two green nodes lie on one face then one of them must be joined

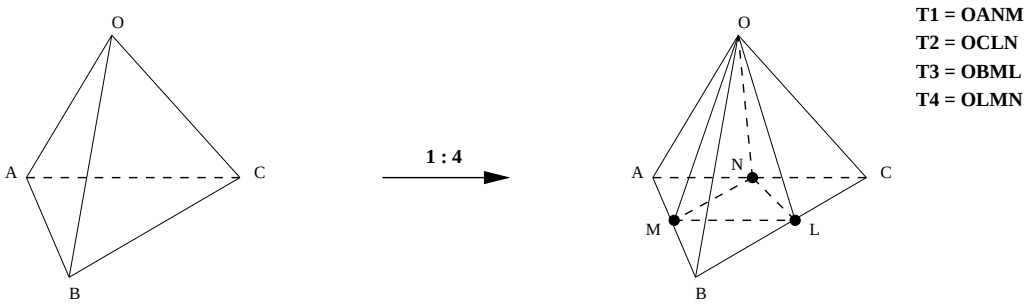

Figure 5.12: Division of a tetrahedron with 3 green nodes on a single face into 4 subtetrahedra.

to the opposite vertex in the same face. Special care must be taken for the other element sharing this face, i.e. to consider the same orientation of the new introduced edges. For the cases where it is necessary to join the two nodes lying on two different faces by an internal diagonal, only these two should be joined to the opposite vertex in their own face as shown in Figures 5.15 and 5.16.

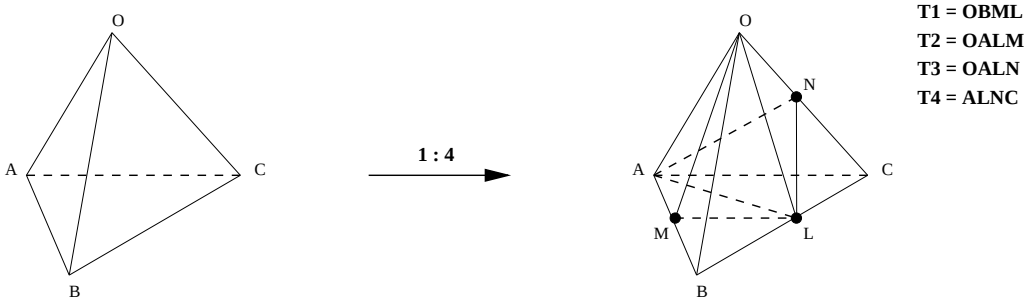

Figure 5.13: Division of a tetrahedron with 3 green nodes on two different faces into 4 subtetrahedra.

This type of green refinement is advantageous in the sense that the transitional layer formed by the green refinement has fewer elements than the red refined element. For all of the possibilities the number of green child elements is never greater than 7. However significant additional programming effort is needed for this type of refinement because many different

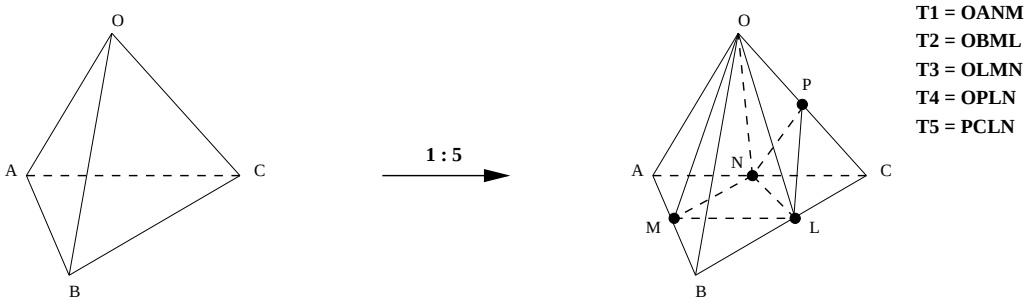

Figure 5.14: Division of a tetrahedron with 4 green nodes (three on one face and one on a different face) into 5 subtetrahedra.

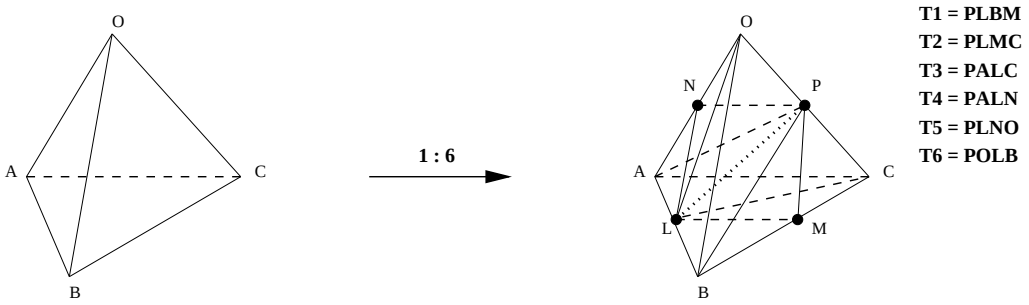

Figure 5.15: Division of a tetrahedron with 4 green nodes (two on each face) into 6 subtetrahedra.

cases arise, with different numbers of child elements, when the pattern of same number of green nodes changes (see the cases shown in Figures 5.14 and 5.15 for example).

The second type of green refinement that we describe here introduces an extra node into the parent tetrahedron, which is subsequently connected to

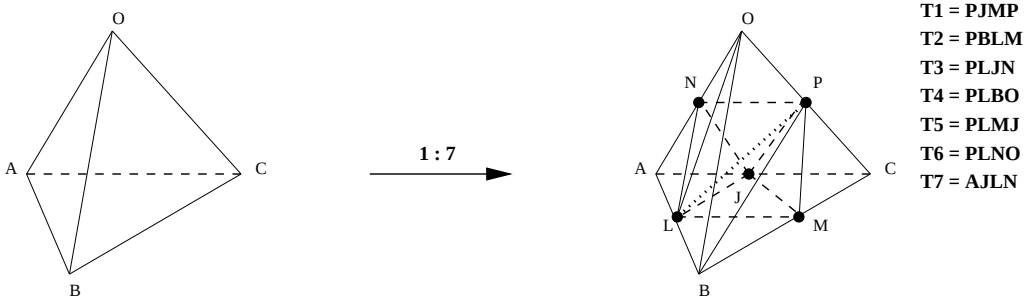

Figure 5.16: Division of a tetrahedron with 5 green nodes into 7 subtetrahedra.

all of the parent vertices and any additional nodes which bisect the parent edges. Three cases of this type of refinement are shown in Figures 5.17–5.19. For all of the cases of this type of green refinement, if two or more green nodes are lying on one face they must be joined together (in the same face). The rest of the possibilities, which are not shown here, can be drawn according to similar patterns to those shown in Figures 5.17–5.19.

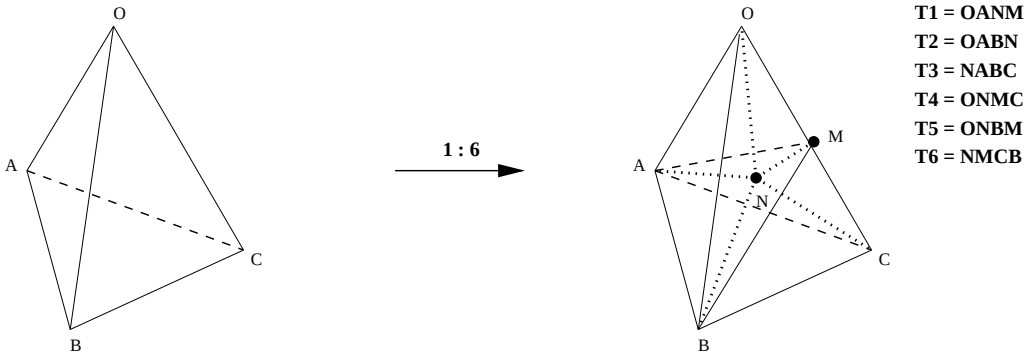

Figure 5.17: Division of a tetrahedron with 1 green node into 6 subtetrahedra by introducing a node into the parent tetrahedron.

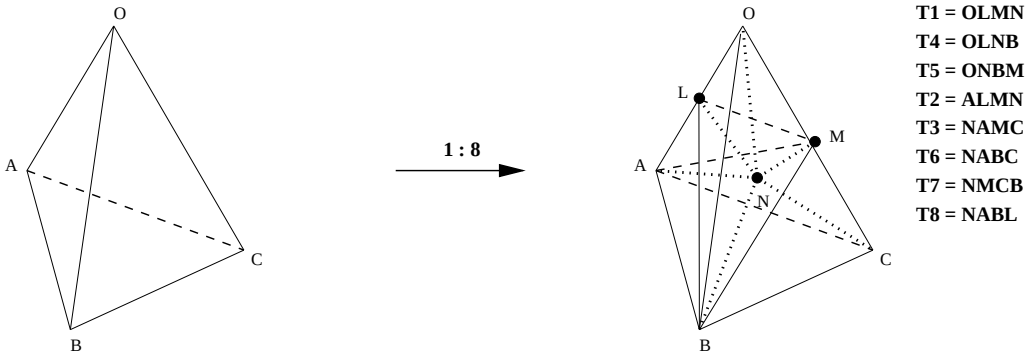

Figure 5.18: Division of a tetrahedron with 2 green nodes on one face into 8 subtetrahedra by introducing a node into the parent tetrahedron.

The six green node possibilities give rise to between 6 and 14 green child elements. The good things about this type of refinement are that the same number of green child elements are created for a given number

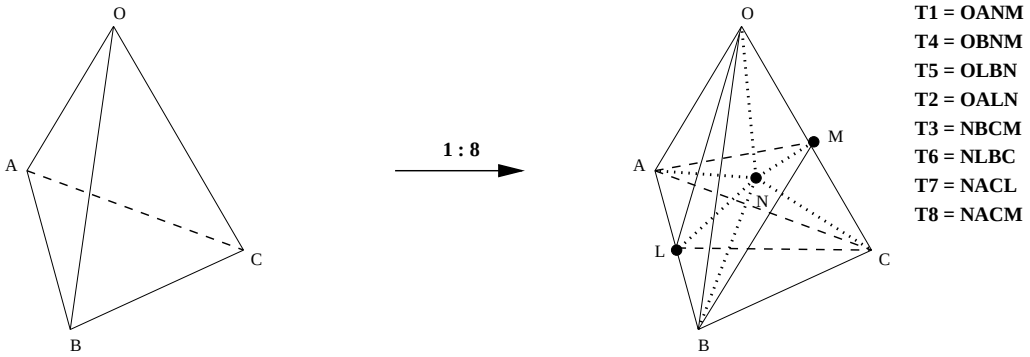

Figure 5.19: Division of a tetrahedron with 2 green nodes on opposite edges into 8 subtetrahedra by introducing a node into the parent tetrahedron.

of green nodes, whatever the pattern of green nodes, and two green nodes lying on two different faces are never joined together. This provides an easy way of dealing with any pattern of parent edge refinement [98]. However the drawback in this type of refinement is that it introduces more elements than needed in the transitional layer, of which many may be of poor geometric quality.

5.3.1 TETRAD

TETRAD stands for TETRahedral ADaptivity [99]. It was developed within School of Computing, Leeds. It is capable of refinement and derefinement of tetrahedral meshes according to user defined criteria provided to it by an interface. We first provide a brief introduction to TETRAD and then we describe its interface.

The approach taken for adaptivity in TETRAD is hierarchical in nature. The mesh adaption strategy assumes that there exists a good quality initial unstructured tetrahedral meshing of the computational domain, which is taken to be the invariant base mesh of the region. TETRAD uses two

types of refinement, regular refinement and green refinement. For regular refinement it subdivides each parent element into 8 child elements as shown in Figure 5.6. For green refinement it subdivides parent elements by introducing a node into the parent element as shown in Figures 5.17−5.19. The five green refinement possibilities (if all of the edges are refined it refines the element regularly) give rise to between 6 and 14 interior child elements. Due to the implementation of the refinement strategy in TETRAD green refinement of this type (which introduces some poor geometric quality elements) does not cause significant difficulties with mesh quality however. This is because the refinement hierarchy is implemented in such a way that a green element may not be further refined. That is, if a green element has an edge targeted for refinement, then the green refinement of the parent element is replaced by a set of regularly refined child elements which may themselves be further refined if necessary. Consequently green elements always signal a change between mesh levels.

TETRAD works on the principle that some (or all) of the mesh elements are flagged for adaption. This results in each mesh edge being targeted either for refinement, derefinement or no action. To construct the list of edges and elements to be refined, each element adjacent to an edge targeted for refinement is examined. If the element is green, then its parent element is placed on the list of elements whose child elements are to be refined. To see how this works, consider Figure 5.20. The refinement of edge 'a' implies that the green Tetrahedron T1 must be processed. Since further refinement

is ruled out, T1 and its sibling tetrahedra must be replaced by a regular set. This in turn requires the refinement of edge 'b', with the same argument now applying to tetrahedron T2 and so on.

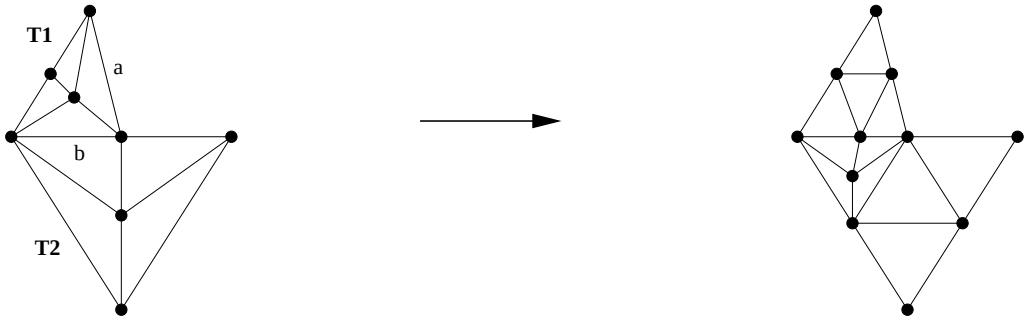

Figure 5.20: Refinement of adjacent green elements

The main problem we faced in using TETRAD for the mesh refinement component of our hybrid algorithm is its strategy of green refinement and derefinement. When node positions are changed it may not be possible for TETRAD to derefine the mesh. The TETRAD refinement and derefinement strategy works for parent edges and after moving nodes parent edges are generally no longer valid for derefinement (as shown in Figure 5.21 for example). The main effect of this on our hybrid algorithm is that we have to disable the derefinement of green elements before they are further refined. If we were just applying h-refinement alone this would be potentially very damaging to the quality of some of the elements in our mesh. Fortunately however, the use of edge swapping and node movement within the hybrid algorithm appears to prevent this from becoming a problem.

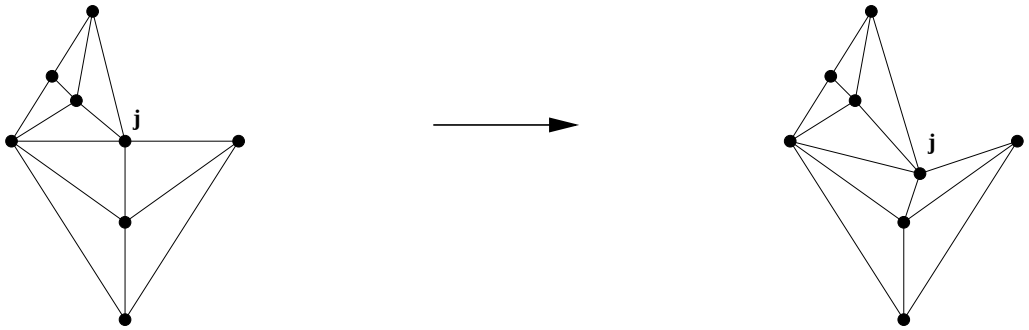

Figure 5.21: Due to movement of green node j, the parent edge containing j no longer remains valid for derefinement.

5.4 Numerical results

In this section we study two representative test problems in order to assess the quality of our new three-dimensional multilevel hybrid algorithm. For the results presented in this section we do not apply the hybrid algorithm with full convergence of the node movement and face/edge swapping algorithm, as presented in our full multilevel hybrid algorithm in Section 2.6. There are three reasons for this. The first is that, as noted above, we have no theoretical guarantee of the convergence of the edge swapping algorithm. The second reason is the expense of the node movement and edge swapping steps in three dimensions. Hence, since the vast majority of the energy reduction always appears to occur in the first couple of sweeps, we simply apply node movement (up to approximate convergence) then face/edge swapping (up to convergence within GRUMMP) and then node movement again (up to convergence), for the meshes at each level. The final reason for not iterating the hybrid algorithm to full convergence is due to the green refinement strategy adopted by TETRAD. During the local

h-refinement many poor shaped elements are introduced, hence when many steps of node movement and face/edge swapping are applied it is more likely that some elements will reach the minimum volume limit. In the next step of local h-refinement these elements then give rise to child elements having volume less than this limit.

The overall strategy that we have adopted appears to work well despite the fact that the meshes produced are not precisely optimal. Furthermore, it is quite efficient since we are not converging the process of node movement and face/edge swapping in combination.

5.4.1 Problem one

For an initial test problem we consider the following generalisation of the first two dimensional problem solved in Section 3.3.1.

$$-\Delta u + \frac{1}{\varepsilon^2} u = 0 \ , \qquad \underline{x} \in \Omega = (0,1) \times (0,1) \times (0,1) \ , \tag{5.2}$$

subject to the Dirichlet boundary conditions

$$u = e^{-x_1/\varepsilon} \tag{5.3}$$

throughout $\partial\Omega$. As with the two dimensional example, (5.3) is the true solution of (5.2) over all of Ω, and the corresponding energy functional ((3.4) but with this modified Ω and summation of the repeated suffices from 1 to 3) has a minimum value given by (3.5). We again choose $\varepsilon = 0.01$ to yield a thin boundary layer near $x_1 = 0$ and an optimal energy $E = 50.0000$.

Following the approach used in Section 3.3, for testing the two-dimensional algorithm, we begin by assessing the performance of three-dimensional mul-

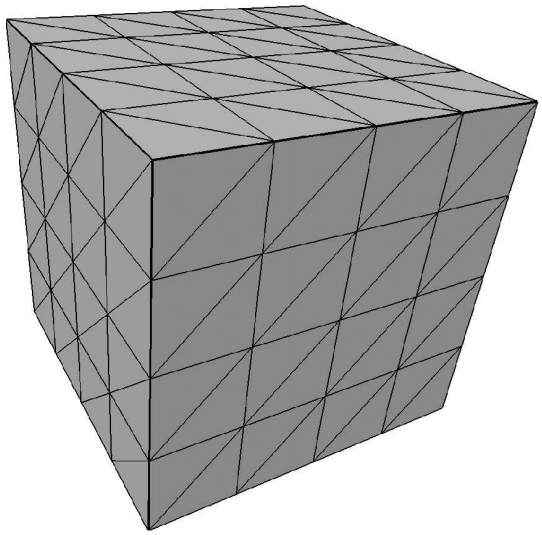

Figure 5.22: An illustration of an initial uniform mesh containing 384 tetrahedral elements.

tilevel mesh optimisation when combined with global h-refinement. Initially the test problem is solved on a regular coarse grid of 384 tetrahedral elements, as illustrated in Figure 5.22. This mesh is then optimised locally using node movement and face/edge swapping and the total energy of the solution reduces from 378.62763 to 62.113265. However the number of elements increases from 384 to 407 due to the application of face/edge swapping. Three levels of uniform refinement, each followed by optimisation, then yield solutions with energies of 51.223148, 50.200687 and 50.048211 on meshes of 3330, 27346 and 220769 elements respectively. For each of these three levels the number of elements increased by slightly more than a factor of eight due to the face/edge-swapping. The sequence of meshes obtained in this way is shown in Figure 5.23.

To see that this final mesh is superior to one obtained without multilevel optimisation the problem is then solved on a three level uniform refinement

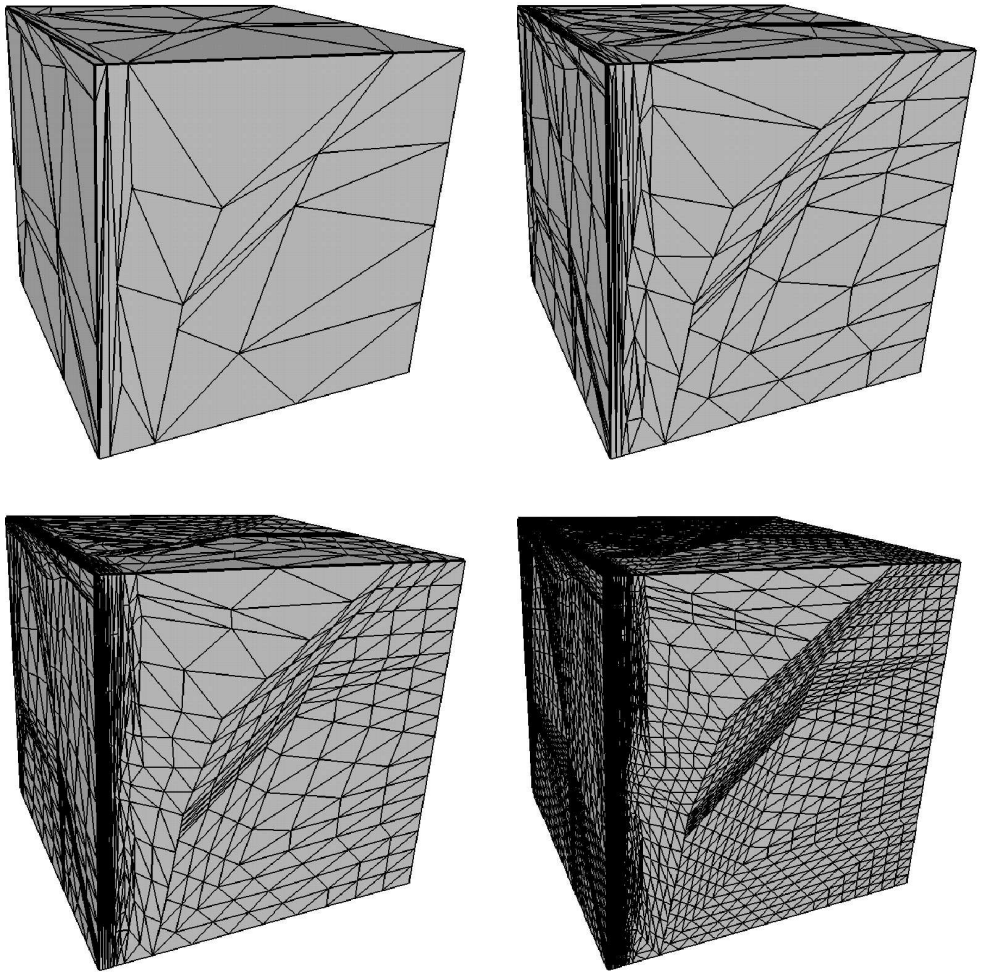

Figure 5.23: An initial locally optimised mesh (top left) followed by a sequence of meshes obtained by combinations of global h-refinement with r-refinement.

of the initial mesh, (with 196608 elements therefore), to yield a solution with energy 67.278957. When this mesh is optimised, however the energy only decreases to a value of 52.338504 with an increase in the number of elements to 197070 due to face/edge swapping.

We now demonstrate that the potential advantages of using local refinement with the multilevel optimisation also appear to apply in three dimensions. Starting with the locally optimal 384 element grid, a sequence

of three further meshes is obtained through local h-refinement (by refining those elements whose local energy exceeded $X = 60\%$ of the maximum local energy on any element) followed by local optimisation. These meshes contain 2931, 18741 and 110170 tetrahedral elements and the corresponding solutions have energies of 51.226773, 50.200292 and 50.043149 respectively. The sequence of meshes obtained in this way is shown in Figure 5.24. The first mesh is the same initial locally optimised mesh shown in Figure 5.23.

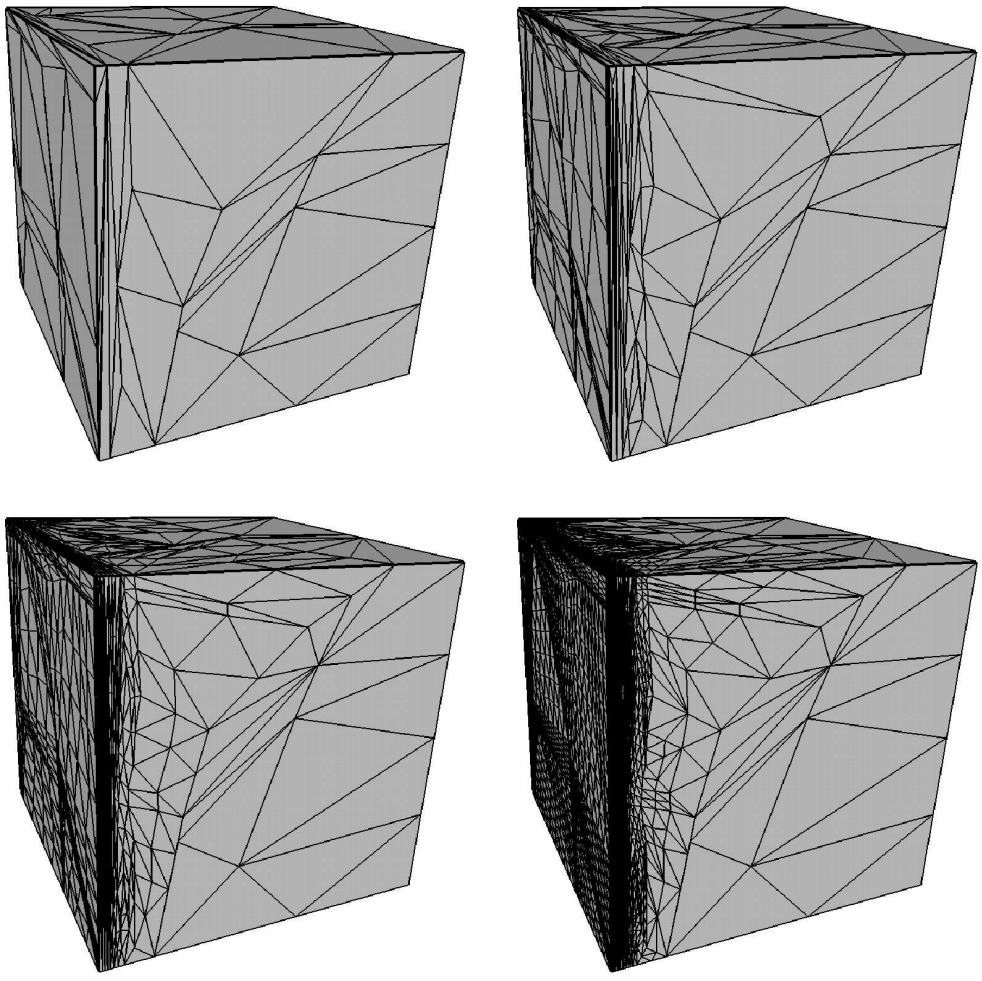

Figure 5.24: An initial locally optimised mesh (top left) followed by a sequence of meshes obtained by combinations of local h-refinement with r-refinement.

Finally, we demonstrate the superiority of this final mesh over one obtained using only local h-refinement followed by local optimisation at the end. This comes from the observation that a grid of 232140 elements obtained using only local h-refinement yields a solution energy of 54.813215 and, when this is optimised, the solution energy only reduces to 51.443760. A summary of all of these computational results is provided in Table 5.2.

It is worth noting at this point just how important the edge swapping part of the hybrid algorithm really is. In [65] we solved this same problem before the edge swapping algorithm was implemented and the results from this are shown in Table 5.3. It is clear that these are significantly worse than the corresponding results shown in Table 5.2.

Elements	Energy	Description
384	378.62763	Energy on initial mesh.
407	62.113265	Multilevel optimisation and
3330	51.223148	global h-refinement.
27346	50.200687	
220769	50.048211	
196608	67.278957	Global h-refinement followed
197070	52.338504	by optimisation.
407	62.113263	
2931	51.226773	Multilevel optimisation and
18741	50.200292	local h-refinement.
110170	50.043149	
232140	54.813215	Local h-refinement followed
233506	51.443760	by optimisation.

Table 5.2: Summary of the results obtained for the first test problem (the global energy minimum is 50.0000).

It can be observed from the Figures 5.23 and 5.24 and the results presented in Table 5.2, that the final mesh obtained by our suggested multilevel

Elements	Energy	Description
384	378.62763	Energy on initial mesh.
384	104.85725	
3072	59.907732	Multilevel optimisation and
24576	52.398871	global h-refinement.
196608	50.755212	
196608	67.279033	Global h-refinement followed
196608	52.434265	by optimisation.
384	104.85704	
2655	59.902412	Multilevel optimisation and
16933	52.381223	local h-refinement.
100866	50.746025	
573834	54.885230	Local h-refinement followed
573834	51.332477	by optimisation.

Table 5.3: Summary of the results obtained for the first test problem without edge swapping (the global energy minimum is 50.0000).

hybrid algorithm has the lowest energy, with least number of elements. The final mesh in Figure 5.24 certainly seems to possess the required qualities of being both fine in the direction perpendicular to the boundary layer (near the face $x_1 = 0$) and quite coarse in the directions parallel with the layer.

5.4.2 Problem two

The second problem that we consider is a three dimensional generalisation of the second problem presented in Section 3.3, which is a calculation of the displacement field for a three dimensional linear elastic model of an overhanging cantilever beam. The domain Ω is

$$\Omega = \{(x, y, z) : 0 < x < 4, 0 < y < 1, 0 < z < 1\}.$$

The bottom half of the beam is fixed as illustrated by the shaded region in Figure 5.25.

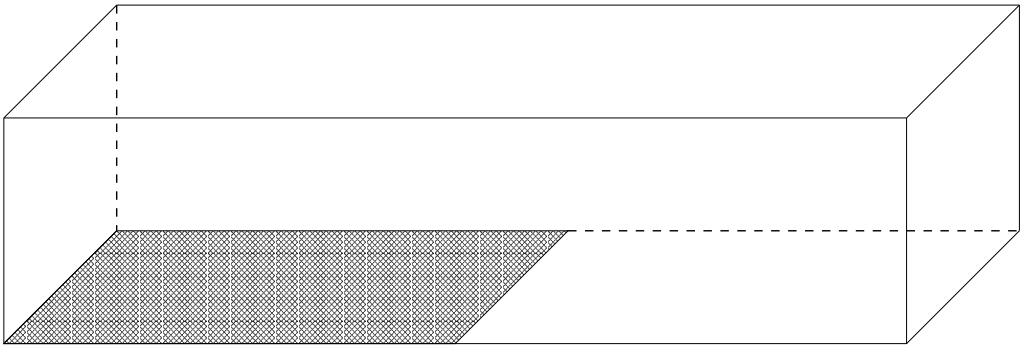

Figure 5.25: An illustration of the overhanging cantilever beam

The energy functional is given by,

$$E = \frac{1}{2}\int_{\Omega} \frac{\partial u_i}{\partial x_j} C_{ijk\ell} \frac{\partial u_k}{\partial x_\ell}\, d\underline{x} - \int_{\Omega} \rho b_i u_i \, d\underline{x} \,. \tag{5.4}$$

Here, all repeated suffices are summed from 1 to 3, $\mathbf{C}$ is the usual fourth order elasticity tensor, chosen to correspond to an isotropic material with a non dimensionalised Young's modulus $E = 100$ and a Poisson ratio $\nu = 0.001$, $\rho\underline{b}$ provides the external body forces due to gravity. The small value of Poisson's ratio is chosen to make the material sufficiently elastic to ensure that it bends under its own weight. This makes the problem a suitable problem for mesh adaptivity.

As before we begin by solving the problem on a uniform coarse mesh, this time containing 192 elements. This mesh is then optimised using the r-refinement algorithm to reduce the total energy from -0.168295 to -0.208546. For this particular mesh the face/edge swapping keeps the number of elements same. This optimal mesh is now uniformly refined to produce 1536 elements which are also optimised, leading to a solution with a total stored energy of -0.262773, with an increased number of elements, 1548, due to the face/edge swapping. A further global refinement gives

12384 elements which is then optimised leading to a solution with total stored energy of -0.280849 on a mesh of 12415 elements. We further refine this mesh globally to get a mesh of 99320 elements. This mesh is then optimised leading to a solution with a total stored energy of -0.285704 over 99349 elements. The sequence of meshes obtained in this manner is shown in Figure 5.26.

We consider two further meshes of 98304 elements and 98370. The first of these is obtained by global refinement of the initial uniform mesh and the second by optimising this mesh directly. The energies of the solutions on these meshes are -0.272196 and -0.283207 respectively and so we again observe the superiority of the hierarchical approach when r-refinement is combined with global h-refinement.

As for the previous example, our goal is to assess the hybrid algorithm that combines r-refinement with local h-refinement hence we consider a sequence of meshes obtained in this manner. The first mesh is the same optimised mesh, containing 192 elements, shown in Figure 5.26. The energy of the solution on this mesh is -0.208546. The second mesh is then obtained by refining those elements whose local energy exceeded $X = 60\%$ of the maximum local energy on any element. This local refinement produces a mesh having 947 elements. It is then optimised which reduces the total stored energy to -0.252279, and the number of elements is increased to 958. To get the mesh at the next level we further refine this mesh locally. Which produces a mesh of 4499 elements, after optimising the solution energy

reduces to -0.267699, and the number of elements increases to 4529, due to face/edge swapping. Another level is then obtained by refining the mesh at previous level locally. This produces a mesh of 15082 elements and after optimising we get a mesh of 15315 elements with the total stored energy reduced to -0.281052. This mesh is then further refined locally to produce a mesh of 48184 elements. When this mesh is optimised we get a mesh of 48403 elements with the total stored energy of the solution reduced to -0.286102. The sequence of meshes obtained in this manner is shown in Figure 5.27.

We again conclude our example by illustrating the advantage of applying the hybrid approach hierarchically by contrasting it with the use of local h-refinement alone, possibly followed by a single application of r-refinement. We refine locally the initial mesh of 192 elements in five levels to achieve a mesh of 132698 elements (again using a threshold of $X = 60\%$ for the local refinement). The total energy of the solution on this mesh is -0.278015. The mesh is then optimised to reduce the total stored energy to -0.284321, with increased number of elements, 132958, due to face/edge swapping. As before it is clear that the quality of the locally optimal meshes obtained in this manner is inferior to that of meshes obtained using the hierarchical approach. A summary of all of the computations made for this test problem is provided in Table 5.4.

Again, from the results presented in Table 5.4, it is clear that our multi-level hybrid algorithm is performing better than all of the other approaches

Elements	Energy	Description
192	-0.208546	
1548	-0.26773	Multilevel optimisation and
12415	-0.280849	global h-refinement.
99349	-0.285704	
98304	-0.272196	Global h-refinement followed
98370	-0.283207	by optimisation.
958	-0.252279	
4529	-0.267699	Multilevel optimisation and
15315	-0.281052	local h-refinement.
48403	-0.286102	
132698	-0.278015	Local h-refinement followed
132958	-0.284321	by optimisation.

Table 5.4: Summary of the results obtained for Problem Two (the global energy minimum is unknown).

considered here. One remark about the figures presented for local refinement; the problem is symmetric about the line $y = 0.5$, but the meshes shown in Figure 5.27 appear non symmetric about this line. This is due to the fact that the initial mesh is unsymmetric about the line $y = 0.5$.

5.5 Summary

This chapter not only provides the implementation details of our multilevel hybrid algorithm but also includes details of face/edge swapping and local h-refinement in three dimensions. A brief introduction to the tools GRUMMP and TETRAD, which have been used for face/edge swapping and local h-refinement respectively, is also provided, including the details of modifications made to them. For face/edge swapping the modified version of GRUMMP worked well to minimise the solution energy, and also avoided the necessity of implementing this from scratch.

There were two reasons for using TETRAD as our local h-refinement tool. First, it is developed within the School and its source code is available as tested and robust software. Second, it was possible to integrate our node movement algorithm within the interface for TETRAD so as to simplify the implementation and improve the efficiency. The coupling with GRUMMP was not integrated however.

Results have been presented for two three-dimensional test problems. For these test problems the hierarchical optimisation provides a better final mesh than is obtained by only optimising the mesh at the highest level of refinement. The use of local h-refinement was clearly superior to the use of global h-refinement. However further improvement may be possible when a consistent green node removal strategy (which does not introduce nodes into the parent elements as discussed in Section 5.3) is used within our hybrid algorithm.

Overall the results in this section are very encouraging. Furthermore, the inclusion of face/edge swapping clearly demonstrated to yield an improved quality of solution over corresponding solutions obtained on meshes using node movement and local refinement alone (see Tables 5.2 and 5.3). So far we have not considered the efficiency of our implementation in three dimensions and so no timings are provided. Nevertheless we believe that there is scope for a sufficiently efficient implementation to make these ideas worthwhile in practice.

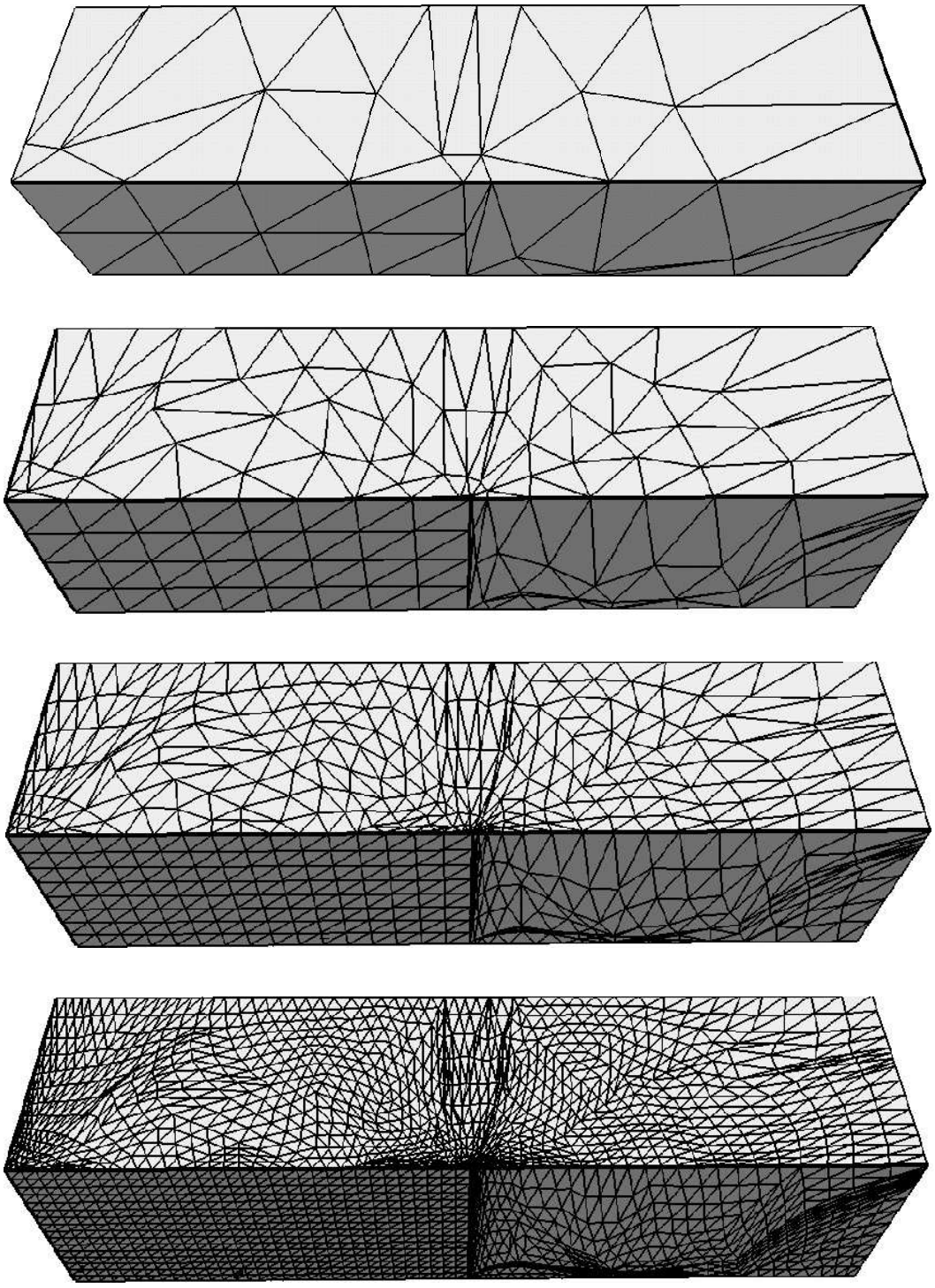

Figure 5.26: A sequence of meshes obtained by r-refinement and then combinations of global h-refinement with r-refinement.

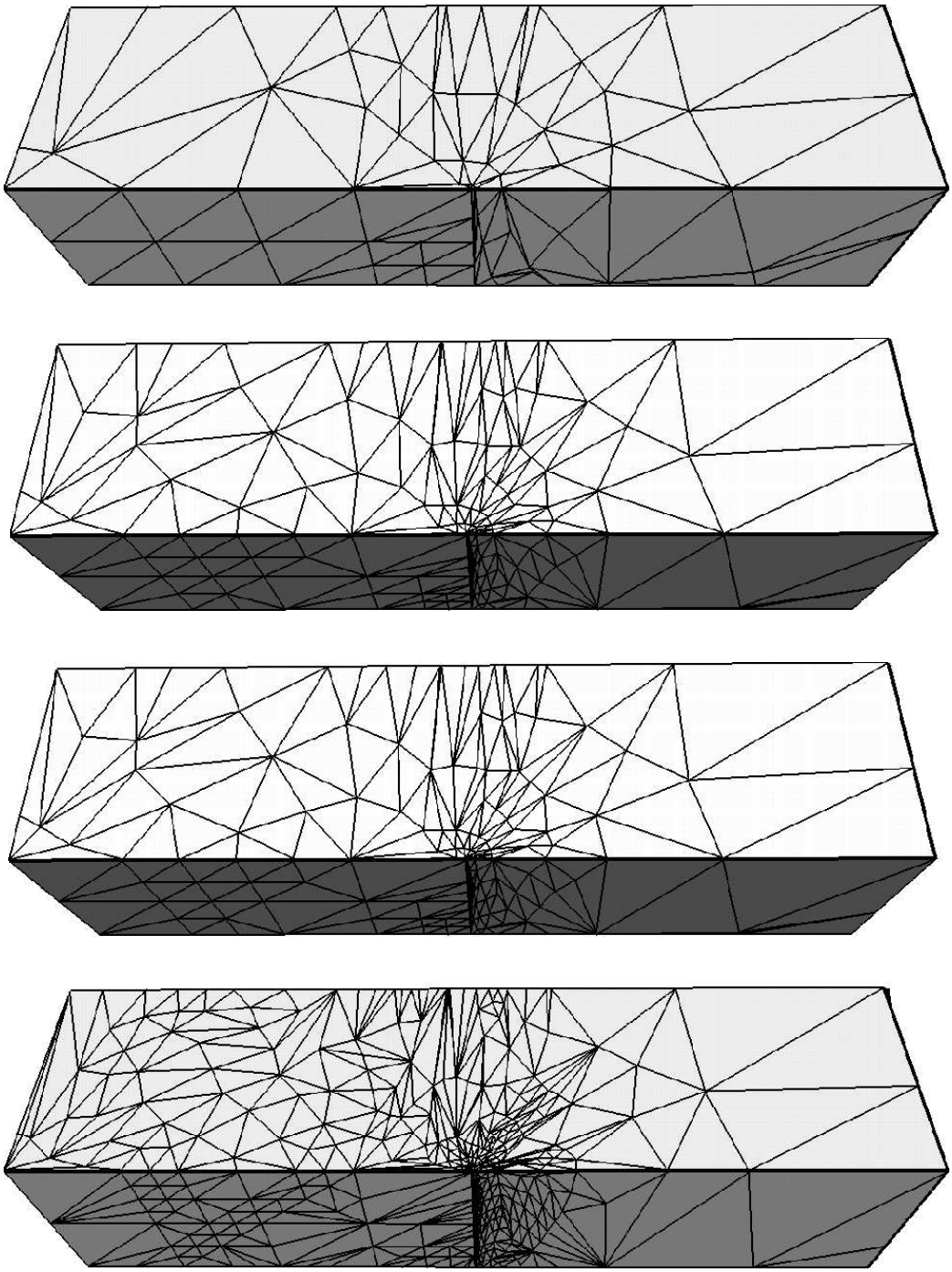

Figure 5.27: A sequence of meshes obtained by combinations of local h-refinement with r-refinement.

Chapter 6

Conclusions and Future Work

6.1 Summary of work undertaken

In this thesis we have considered mesh adaptivity for the finite element solution of variational problems which accept the energy formulation. The main contribution of this thesis is the development, implementation and testing of a new multilevel hybrid algorithm in two and three space dimensions for unstructured triangular and tetrahedral meshes. We have applied this algorithm to a variety of test problems including linear, nonlinear and systems of equations. It is evident from the results presented in previous chapters that our hybrid algorithm is very effective in achieving an excellent combination of small mesh sizes and low energies for the corresponding finite element solutions. Here we provide a short summary of the work presented in this thesis.

In Chapter 1 we have presented a general overview of the variational problems and the finite element method. We also discuss some related topics regarding the solution of the discrete finite element equations. The philosophy behind mesh adaptivity is introduced and then numerous error

estimators and indicators are discussed for driving the adaptivity. A general introduction to the most common adaptive schemes is provided. We then discuss edge swapping and moving mesh algorithms in some detail, and we present a number of possible moving mesh schemes.

Our main work begins from Chapter 2. We first present the formulation of the problem and the notation to be used in the thesis. We explain the strategy of node movement and then we present the derivation of the gradient of the energy functional with respect to the position of nodes (2.10) and the local problem (2.5). We then present our proposed algorithm for node movement. Edge swapping and local h-refinement in two space dimensions are also discussed and possible algorithms for these are provided. Finally our new multilevel hybrid algorithm is presented by putting together the three components of node movement, edge swapping and local h-refinement.

In Chapter 3 we provide some implementation details of our hybrid algorithm. Algorithms for line minimisation considering exact and inexact line searches are provided. The algorithm for an inexact line search was originally considered in [48] to improve the geometric quality of the mesh. We have modified it to minimise the energy functional. Results presented in Chapter 4 indicate that the inexact line search algorithm has great potential to improve the efficiency of our node movement algorithm. Some more options which can affect the performance of the hybrid algorithm are also discussed in Chapters 3 and 4. In the former, parameters which arise during the implementation of the hybrid algorithm are considered and their

possible range of values is provided; backed by empirical results. The final section of Chapter 3 introduces four test problems which have been considered. The results produced by our hybrid algorithm are compared with corresponding results produced by some other adaptive strategies for each of the test problems. These show that the mesh quality obtained is generally very good although we do not consider the computational costs of obtaining these optimal meshes.

In Chapter 4 we discuss some of the most significant of the options available within our hybrid algorithm by considering one particular representative test problem with respect to both the computational costs and the quality of the meshes achieved. Some recommendations have been made on the basis of these numerical results. Some possible options to improve the efficiency of the hybrid algorithm are also provided with some preliminary numerical results.

In Chapter 5 we discuss in detail edge swapping and local h-refinement in three dimensions. Here we provide an overview of GRUMMP [52] and TETRAD [99]. The modifications we have made in GRUMMP and TETRAD are discussed in order that we have been able to use them as tools for our applications of edge swapping and local h-refinement respectively in three dimensions. Finally we presented some preliminary test problems and the results produced by our hybrid algorithm are compared with the results produced by some other adaptive strategies for each of the test problems. These results are extremely encouraging overall. To our knowledge the algorithms

presented for node movement and face/edge swapping have never been implemented before in three dimensions in order to reduce the energy of the solution. Results presented in Chapter 5 show that edge swapping in three dimensions is an extremely important component of the overall adaptive algorithm. When considered in isolation node movement and edge swapping are hardly competitive with other adaptive strategies, however when these are combined with local h-refinement in either two or three space dimensions they produce an extremely effective algorithm, when assessed in terms of the quality of the meshes produced.

6.2 Possible future work

The results presented in the previous chapters demonstrate that the quality of meshes achieved by our hybrid algorithm is generally better than obtained from the other adaptive approaches which we have considered for comparison. Further enhancements that need to be investigated relate mainly to the efficiency of the hybrid algorithm.

The results presented in Chapter 4 demonstrate that the most time consuming part of our hybrid algorithm is the node movement process. In order to make our hybrid algorithm a practical tool the node movement part of our hybrid algorithm definitely needs some mechanism which make its convergence faster. One possibility, as suggested in Chapter 4, is that it is not necessary to converge the node movement process. Significant amounts of computational time can be saved by stopping the node movement process at

an appropriate point. This needs some analysis or experimentation in order to determine the minimum number of node movement iterations that are required before stopping the process. Moreover, how much this will affect the overall performance of the hybrid algorithm needs to be understood.

Although our hybrid algorithm does not need any global finite element solves at intermediate points, the preliminary results presented in Chapter 4 indicate that this option is a better choice. However the cost-benefit of the intermediate global solves may not be so substantial for problems where the computation of a global solution is expensive. We have proved in Section 2.2.3 that solving a sequence of local problems (2.5) iteratively is equivalent to a global solve of the problem. We suggest therefore that before applying the node movement algorithm, the nodal solution values should be updated approximately by doing some iterations of solving the sequence of local problems. This needs some research in order to better understand how approximately the nodal solution values should be updated and in order to best improve the efficiency of the algorithm. It may be the case however that this provides the best compromise between undertaking a global solve after each h-refinement or just using the interpolated nodal solution values as the starting point for the node movement algorithm.

It is also suggested in Chapter 4 that some efficiency can be gained by not applying the edge swapping algorithm up to convergence. It needs some additional work to ascertain when it is best to stop the edge swapping algorithm, and how much the performance of our hybrid algorithm will be

affected by stopping the edge swapping algorithm at such an appropriate point.

In our current implementations of the local h-refinement (in both two dimensions and three dimensions) we are inserting the new nodes at the centre of each edge marked for refinement. It is claimed in [84] that 'it is possible to simultaneously optimise the new node position as well as split the edge'. We believe that this can be incorporated within our hybrid algorithm setting. This idea is worthwhile to try in order to see if it will be beneficial or not.

For local h-refinement we have seen that many new unnecessary elements are introduced in order to remove the hanging nodes that are produced by our refinement algorithms (this situation is especially noticeable in three dimensions). It is possible to consider another type of refinement which does not introduce any hanging nodes. In two dimensions any element marked for refinement can be refined in to three elements as shown in Figure 6.1. In three dimensions an element can be refined into four elements as shown in Figure 6.2. In each case a new node is added at the centroid of the element.

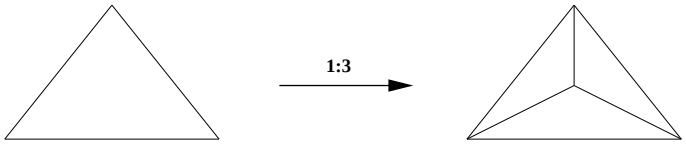

Figure 6.1: Refinement of a triangle in to three triangles.

There are a number of disadvantages to this type of refinement that tend to make it unpopular in practice. Firstly, the geometric quality of the re-

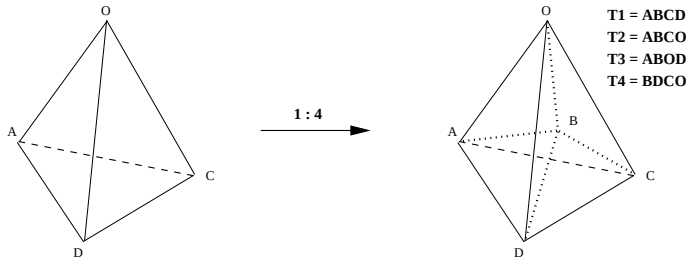

Figure 6.2: Refinement of a tetrahedron in to four tetrahedra.

fined elements can become very poor. Secondly, the boundary of the domain can not be refined. When used as part of our hybrid algorithm however the geometric quality of the refined elements will be improved by the combination of node movement and face/edge swapping. To overcome the problem of boundary refinement we can introduce a mixed element refinement strategy: every interior element (having no face/edge on the boundary) may be refined as discussed above, and each boundary element marked for refinement may be bisected so that the new node inserted should be on boundary. It would be interesting to test how this idea works practically.

A less radical alteration to the h-refinement procedure used in this work would be to implement an element bisection algorithm [26] in three dimensions (as opposed to the refinement into 8 children currently used). The only potential problem with this type of refinement is that poor geometric quality elements can appear. However as part of our hybrid algorithm the combination of node movement and face/edge swapping will improve this. We have implemented this type of h-refinement in two dimensions. The results presented in Section 3.3 show that within the setup of our hybrid algorithm this type of refinement can produce better results than the one-

to-four element refinement scheme. Also there is a need to carry out more tests in three dimensions. These include implementing a suitable strategy for green node removal (as discussed in Section 5.3), considering some more test problems and producing an integrated version of the face/edge swapping within the hybrid algorithm.

Finally, it is possible in principle to implement a parallel version of this hybrid algorithm. There has been some advancement in implementing GRUMMP [52] in parallel, which uses edge swapping and smoothing to improve the geometric quality of the mesh. There has also been work on a parallel version of TETRAD, [96], for parallel local mesh refinement. With an appropriate colouring of the nodes of the mesh it should also be possible to implement the node movement algorithm in parallel, provided no two neighbouring nodes are updated simultaneously. If these steps could be combined as part of a completely parallel hybrid adaptive algorithm then the outlook for an efficient high quality parallel adaptive procedure would be good.

Bibliography

[1] S Adjerid, B Belguendouz, and J E Flaherty. *A Posteriori* finite element error estimation for diffusion problems. *Siam Journal on Scientific Computing*, 21(2):728–746, 2000.

[2] M Ainsworth and J T Oden. *A posteriori* error estimators for second order elliptic systems part 2: An optimal order process for calculating self equilibrating fluxes. *Comput. Math Appl.*, 26:75–87, 1993.

[3] M Ainsworth and J T Oden. A unified approach to *a posteriori* error estimation using element residual methods. *Numerische Mathematik*, 65:23–50, 1993.

[4] M Ainsworth and J T Oden. *A posteriori error estimation in finite element analysis*. John Willey and sons, 2000.

[5] T Apel. An adaptive finite element technique with *a priori* mesh grading. Technical Report 9, Bicom Institute of Computational Mathematics, 1993.

[6] T Apel, S Grosman, P K Jimack, and A Meyer. A new methodology for anisotropic mesh refinement based upon error gradients. *Submitted to Applied Numerical Mathematics*, 2001.

[7] T Apel and G Lube. Anisotropic mesh refinement in stabilized galerkin methods. *Numer. Math.*, 74:261–282, 1996.

[8] T Apel and S Nicaise. The finite element method with anisotropic mesh grading for elliptic problems in domains with corners and edges. *Math. Methods Appl. Sci.*, 21:519–549, 1998.

[9] I Babuska and A Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13:214–226, 1976.

[10] I Babuska and B Q Guo. Approximation properties of the hp version of the finite element method. *Comp. Meth. Appl. Mech. Eng.*, 133:319–349, 1996.

[11] I Babuska and W C Rheinboldt. Error estimates for adaptive finite element computations. *Siam J Num. Anal*, 15:736–754, 1978.

[12] I Babuska and W C Rheinboldt. *A posteriori* error estimates for the finite element method. *Journal Numer. Method Eng.*, 12:1597–1615, 1978.

[13] I Babuska and W C Rheinboldt. *A posteriori* error analysis of finite element solutions for one dimensional problems. *Siam J Numer. Anal*, 18:565–589, 1981.

[14] I Babuska, T Strouboulis, and K S Gangaraj. *A posteriori* estimation of the error in the recovered derivatives of the finite element solution. *Computer Methods in Appl. Mech. Eng.*, 150:369–396, 1997.

[15] I Babuska, T Strouboulis, and C S Upadhyay. A model study of the quality of *a posteriori* error estimators for linear elliptic problems. error estimation in the interior of patch wise uniform grids of triangles. *Comput. Methods Appl. Mech Eng.*, 114:307–378, 1994.

[16] I Babuska, B A Szabo, and I N Katz. The p-version of the finite element method. *Siam Journal on Numerical Analysis*, 18:515–545, 1981.

[17] M J Baines. Grid adaption via node movement. *Appl Numer. Math*, 26:77–96, 1998.

[18] M J Baines. Moving finite element, least squares, and finite volume approximations of steady and time-dependent pdes in multidimensions. *Journal Of Computational And Applied Mathematics*, 128:363–381, 2001.

[19] J M Ball, P K Jimack, and T Qi. Elastostatics in the presence of a temperature distribution or inhomogeneity. *Zeitschrift Fur Angewandte Mathematic Und Physik*, 43:943–973, 1992.

[20] R E Bank. *PLTMG Users' Guide 7.0.* Siam, Philadelphia, 1994.

[21] R E Bank and L R Scott. On the conditioning of finite element equations with highly refined meshes. *Siam J. Numer. Anal.*, 26(6):1383–1394, 1989.

[22] R E Bank, A H Sherman, and A Weiser. Refinement algorithms and data structures for regular local mesh refinement. *Scientific Computing*, 1983.

[23] R E Bank and R K Smith. Mesh smoothing using *a posteriori* error estimates. *Siam J Numer Anal*, 34:979–997, 1997.

[24] R E Bank and A Weiser. Some *a posteriori* error estimators for elliptic partial differential equations. *Math Comp*, 44:283–301, 1985.

[25] E Bänsch. An adaptive finite element strategy for the 3-dimensional time-dependent navier-stokes equations. *Journal of Computational and Applied Mathematics*, 36:3–28, 1991.

[26] E Bänsch. Local mesh refinement in 2 and 3 dimensions. *Impact of Comuting in Science and Engineering*, 3:181–191, 1991.

[27] G Beckett, J A Mackenzie, A Ramage, and D M Sloan. On the numerical solution of one-dimensional pdes using adaptive methods based on equidistribution. *Journal of Computational Physics*, 167(2):372–392, 2001.

[28] E Bertoti and B Szabo. Adaptive selection of polynomial degrees of a finite element mesh. *International Journal for Numerical Methods in Engineering*, 42:561–578, 1998.

[29] M Berzins. A solution-based triangular and tetrahedral mesh quality indicator. *Siam Journal On Scientific Computing*, 19:2051–2060, 1998.

[30] J Bey. Tetrahedral grid refinement. *Computing*, 55:355–378, 1995.

[31] R Biswas and R C Strawn. Mesh quality control for multiply refined tetrahedral grids. *Appl. Numerical Methods*, 13, 1994.

[32] C D Boor. *A practical guide to splines.* Springer-Verlag, New York, 1978.

[33] C D Boor and J R Rice. An adaptive algorithm for multivariate approximation giving optimal convergence rates. *J. Approx. Theory*, 25:337–359, 1979.

[34] Liu By. The analysis of a finite element method with streamline diffusion for the compressible navier-stokes equations. *SIAM Journal on Numerical Analysis*, 38(1):1–16, 2000.

[35] W M Cao, W Z Huang, and R D Russell. An r-adaptive finite element method based upon moving mesh pdes. *Journal of Computational Physics*, 149:221–244, 1999.

[36] P J Capon and P K Jimack. On the adaptive finite element solution of partial differential equations using hr-refinement. Research report 96.13, School of computer studies, University of Leeds, 1996.

[37] N N Carlson and K Miller. Design and application of a gradient-weighted moving finite element code II: in two dimensions. *SIAM Journal on Scientific Computing*, 19, 1998.

[38] J H Cheng. Adaptive grid optimization for structural analysis geometry based approach. *Computer Methods in Applied Mechanics and Engineering*, 107:221–243, 1993.

[39] C K Chui and D Hong. Swapping edges of arbitrary triangulations to acheive the optimal order of approximations. *Siam J. Numer. Anal.*, 34:1472–1482, 1997.

[40] P G Ciarlet. *The finite element method for elliptic problems.* North Holland, Amesterdam, 1978.

[41] G R Cowper. Gaussian quadrature formulas for triangles. *Internat. J. Numer. Methods Engrg*, 7:405–410, 1973.

[42] M Delfour, G Payre, and J P Zolesio. An optimal triangulation for second order elliptic problems. *Computer Methods in Applied Mechanics and Engineering*, 50:231–261, 1985.

[43] M R Dorr. The approximation of solutions of elliptic-boundary value problems via the p-version of the finite element method. *Siam J of Numer. Anal.*, 23:58–77, 1986.

[44] I S DUFF, A M Erisman, and J K Reid. *Direct methods for sparse matrices.* Oxford University Press, London, 1986.

[45] K Erikson and C Johnson. Adaptive finite element methods for parabolic problems i: A linear model problem. *Siam J. Num. Anal.*, 28:43–77, 1991.

[46] K Eriksson, D Estep, P Hansbo, and C Johnson. Introduction to adaptive methods for differential equations. *Acta Numerica (A. Iserles, ed.)*, pages 105–158, 1995.

[47] M J Forray. *Variational calculus in science and engineering.* McGraw-Hill, New York, 1968.

[48] L A Freitag and C Ollivier Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40:3979–4002, 1997.

[49] I Fried. Condition of finite element matrices generated form nonuniform meshes. *AIAA*, 10:219–221, 1972.

[50] R J Gelinas, S K Doss, and K Miller. The moving finite element method: application to general partial differential equations with multiple large gradients". *J. Comput. Phys.*, 40:202–249, 1981.

[51] P L George. *Delaunay triangulation and meshing: Application to Finite elements.* Editions HERMES, Paris, 1998.

[52] C O Gooch. *GRUMMP Users' Guide 7.0.* University of British Columbia, 1999.

[53] W Gui and I Babuska. The h,p and h-p version of the finite element method in 1 dimension. *Numerische Math*, 48:557–683, 1986.

[54] M D Gunzberger, G J Fix, and R A Nicolaides. On finite element methods of the least squares type. *Comp. and Maths. with Appls.*, 5:87–98, 1979.

[55] W G Habashi, J Dompierre, Y Bourgault, D A A Yahia, M Fortin, and M Vallet. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent cfd. part i: general principles. *International Journal for Numerical Methods in Fluids*, 32(6):725–744, 2000.

[56] M Heath. *Parallel direct methods for sparse linear systems*, chapter Parallel Numerical Algorithms, pages 55–90. Academic Publishers, 1997.

[57] W Huang, Y Ren, and R D Russell. Moving mesh partial differential equations (mmpdes) based on the equidistribution principle. *Siam J. Numerical Analysis*, 31:709–730, 1994.

[58] W Huang and R D Russell. A high dimensional moving mesh strategy. *Appl. Numer. Math.*, 26:63–76, 1998.

[59] W Huang and R D Russell. Moving mesh strategy based on a gradient flow equation for two-dimensional problems. *Siam journal on scientific computing*, 20:998–1015, 1999.

[60] T Iliescu. A 3d flow-aligning algorithm for convection-diffusion problems. *Applied Mathematics Letters*, 12(4):67–70, 1999.

[61] P K Jimack. A best approximation property of the moving finite element method. *SIAM Journal on Numerical Analysis*, 33:2206–2232, 1996.

[62] P K Jimack. An optimal finite element mesh for elastostatic structural analysis problems. *Computers and Structures*, 64:197–208, 1997.

[63] P K Jimack. Local minimisation of errors and residuals using the moving finite element method. Technical report, School of computer studies, University of Leeds, 1998.

[64] P K Jimack and R Mahmood. A multilevel approach for obtaining locally optimal finite element meshes. *In Developments in Engineering Computational Technology, ed. B.H.V. Topping (Civil-Comp Press)*, pages 191–197, 2000.

[65] P K Jimack, R Mahmood, M Walkley, and M Berzins. A multilevel approach for obtaining locally optimal finite element meshes (in 2 and 3 dimensions). *Submitted to Advances in Engineering Software*, 2000.

[66] P K Jimack and A J Wathen. Temporal derivatives in the finite element method on continuously deforming grids. *SIAM Journal on Numerical Analysis*, 28:990–1003, 1991.

[67] Barry Joe. Three-dimensional triangulations from local transformations. *SIAM Journal on Scientific and Statistical Computing*, 10:718–741, 1989.

[68] Barry Joe. Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, 16:1292–1307, 1995.

[69] A Johnson. *Meshme*. University of Minnesota, http://www .arc.umn.edu/ johnson/meshme.html.

[70] C Johnson. *Numerical solution of partial differentia equations by the finite element method*. Cambridge University Press, 1987.

[71] Y Kallinderis, V Parthasarathy, and J Wu. A new euler scheme and adaptive refinement/coarsening algorithm for tetrahedral grids. *AIAA Paper 92-0446*, 1992.

[72] C L Lawson. Software for c^1 surface interpolation. *In J R Rice, editor, Mathematical software III, Academic Press, New York*, pages 161–194, 1977.

[73] C L Lawson. properties of n-dimensional triangulations. *Computer Aided Geometric Design*, 3:231–246, 1986.

[74] L Y Li, P Bettess, and J W Bull. Mesh refinement formulations in adaptive finite element methods. *Journal of Mechanical Engineering Science*, 210:353–361, 1996.

[75] R Lohner. An adaptive finite element scheme for transient problems in cfd. *Computer Methods in Applied Mechanics and Engineering*, 61:267–281, 1987.

[76] R Lohner and J D Baum. Adaptive h-refinement on 3d unstructured grids for transient problems. *Int. J. Numer. Methods in Fluids*, 14:1407–1419, 1992.

[77] J A Mackenzie. The efficient generation of simple two dimensional adaptive grids. *Siam J. Sci. Comput.*, 19:1340–1365, 1998.

[78] J A Mackenzie and M L Robertson. The numerical solution of one-dimensional phase change problems using an adaptive moving mesh method. *Journal of Computational Physics*, 161:537–557, 2000.

[79] J L Maubach. Local bisection refinement for n-simplicial grids generated by reflection. *Siam J. Sci. Comput.*, 16:210–227, 1995.

[80] K Miller and R N Miller. Moving finite elements i. *SIAM Journal of Numerical Analysis*, 18:1019–1032, 1981.

[81] Keith Miller. Moving finite elements ii. *Siam Journal of Numerical Analysis*, 18:1033–1057, 1981.

[82] J T Oden and J N Reddy. *Variational methods in theoretical mechanics.* Springer-Verlag, 2nd edition, 1983.

[83] M E Ong. Uniform refinement of tetrahedron. *Siam J. Sci. Comput.*, 15, 1994.

[84] C C Pain, A P Umpleby, C R E deOliveira, and A J H Goddard. Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Computer Methods in Applied Mechanics and Eng.*, pages 3771–3796, 2001.

[85] S V Pennington, P K Jimack, and K McFarlane. Adaptive numerical simulation of the remediation of soil contamination by in-situ gas venting. *Computational Geosciences*, 3:135–160, 1999.

[86] J Peraire, M Vahdati, K Morgan, and O C Zienkiewicz. *Adaptive remeshing for compressible flow computations.* Academic press, INc., 1991.

[87] Y Qiu, D M Sloan, and T Tang. Numerical solution of a singularly perturbed two-point boundary value problem using equidistribution: analysis of convergence. *J. Comput. Appl. Math.*, 116:121–143, 2000.

[88] J N Reddy. *An Introduction to the Finite Element Method.* McGraw-Hill, 1993.

[89] J Riccius, K Schweizerhof, and M Baumann. Combination of adaptivity and mesh smoothing for the finite element analysis of shells with intersections. *International Journal for Numerical Methods in Engineering*, 40:2459–2474, 1997.

[90] S Ripa. Long and thin triangles can be good for linear interpolation. *Siam Journal of Numerical Analysis*, 29:257–270, 1992.

[91] S Ripa and B Schiff. Minimum energy triangulations for elliptic problems. *Computer Methods in Applied Mechanics and Engineering*, 84:257–254, 1990.

[92] M C Rivara. Design and data structure of a fully adaptive multigrid finite element software. *ACM Trans. on Math Software*, 10:242–264, 1984.

[93] P Roe. Compounded of many simplices. reflections on the role of model problems in cfd. In *Barriers and Challenges in Computational Fluid Dynamics*, pages 241–258. Kluwer Academic, 1998.

[94] Y Saad. A basic tool kit for sparse matrix computation, version 2. Technical report, Centre for Supercomputing Research and Development, University of Illinois, Urbama, 1994.

[95] Y Saad. *Iterative Methods for Sparse Linear Systems.* PWS, Boston, 1996.

[96] P M Selwood and M Berzins. Portable parallel adaption of unstructured tetrahedral meshes. *J. Parallel Distrib. Comput.*, 52:150–177, 1998.

[97] R B Simpson. Anisotropic mesh transformation and optimal error control. *Applied Numerical Math*, 14:183–198, 1994.

[98] W Speares and M Berzins. A 3-d unstructured mesh adaptation algorithm for time-dependent shock dominated problems. *International Journal for Numerical Methods in Fluids*, 25:81–104, 1997.

[99] W Spears. *TETRAD Users' Guide V103.* School of Computing, University of Leeds, 1995.

[100] J Stoer and R Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, Berlin, Heidelberg, New York, 2nd edition, 1993.

[101] G Strang and G J Fix. *An analysis of the finite element method.* Prentice-Hall, 1973.

[102] J Szmelter, J Marchant, and M J Evans. 2-dimensional navier stokes equations with adaptivity on structured meshes. *Computer Methods in Applied Mechanics and Engineering*, 101:355–368, 1992.

[103] BH V Topping and A I Khan. *Parallel Finite Element Computations.* Saxe-Coburg Publications Edinburgh, 1996.

[104] Y Tourigny and F Hulsemann. A new moving mesh algorithm for the finite element solution of variational problems. *SIAM Journal on Numerical Analysis*, 35:1416–1438, 1998.

[105] University of Trieste,Italy, http://www-dinma.univ.trieste.it/nirftc/research/easymesh. *Easymesh*, version 1.4 edition, 1999.

[106] L Vardapetyan and L Demkowicz. hp-adaptive finite elements in eletromagnetics. *Comp. Meth. Appl. Mesh. Eng.*, 169:331–344, 1999.

[107] R Verfürth. *A review of a posteriori error estimation and adaptive mesh refinement techniques.* Wiley-Teubner, Chichester; Stuttgart, 1996.

[108] M Walkley, P K Jimack, and M Berzins. Mesh quality for three-dimensional finite element solutions on anisotropic meshes. In *Proceedings of FEM3D (to appear) , Univ. of Jyvaskyla, Finland*, 2000.

[109] N P Weatherill. Grid adaptation using a distribution of sources applied to inviscid compressible flow simulations. *International Journal for Numerical Methods in Fluids*, 19:739–764, 1994.

[110] H P William. *Numerical recipes in C : the art of scientific computing.* Cambridge University Press, 2nd edition, 1992.

[111] X Xu, C C Pain, A J H Goddard, and C R E deOliveira. An automatic adaptive meshing technique for delaunay triangulations. *Computer Methods in Applied Mechanics and Engineering*, 161:297–303, 1998.

[112] O C Zienkiewicz and R L Taylor. *The finite element method*, volume 1. Butterworth-Heinemann, 5th edition, 2000.

[113] O C Zienkiewicz and J Z Zhu. A simple error estimator and adaptive procedure for practical engineering analysis. *International Journal of Num. Meth. Eng.*, 24:337–357, 1987.

[114] O C Zienkiewicz and J Z Zhu. The superconvergent patch recovery (spr) and adaptive finite element refinement. *Comput. Methods Appl. Mech. Eng.*, 101(1):207–224, 1992.

[115] O C Zienkiewicz, J Z Zhu, and N G Gong. Effective and practical hp version adaptive analysis procedures for the finite element method. *International Journal of Num. Meth. Eng.*, 28:879–891, 1989.