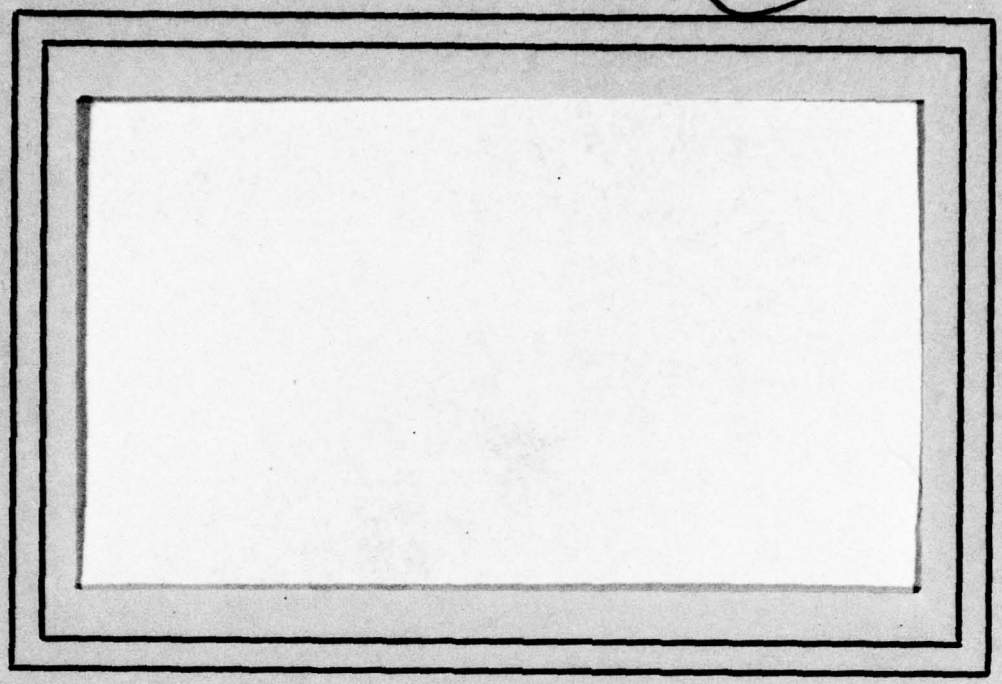
AD-A055 625 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 9/2
ON A DATA STRUCTURE FOR ADAPTIVE FINITE ELEMENT MESH REFINEMENT--ETC(U)
JUN 78 W C RHEINBOLDT, C K MESZTENYI N00014-77-C-0623
UNCLASSIFIED TR-660 NL

| OF |
AD
A055 625

END
DATE
FILMED
8-78
DDC



COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(1)

Technical Report TR-660
ONR-N00014-77-C-0623-660

June 1978

On a Data Structure for Adaptive Finite Element Mesh Refinements[1)]

by

Werner C. Rheinboldt[2)] and Charles K. Mesztenyi[2)]

Abstract

A general labelled tree structure is introduced for a class of non-uniform two-dimensional finite element meshes. The theoretical basis of the structure and the fundamental access algorithms on the tree are presented in a manner which lends itself to extensions to higher dimensions. For use in finite element computations, the tree is truncated considerably and then the principal, relevant algorithms are discussed, including the refinement of the mesh, the computation of the elemental stiffness matrices, and the assembly and decomposition of the global stiffness matrices based on nested dissection techniques. An outlook to various possible extensions of the structure is also given.

1) This work was in part supported under NSF Grant MCS-72-03721A06 and ONR Contract N00014-77-C-0623.

2) Computer Science Center, University of Maryland, College Park MD 20742.

1. Introduction

Currently an experimental software system is under development at the University of Maryland which has the following design properties:

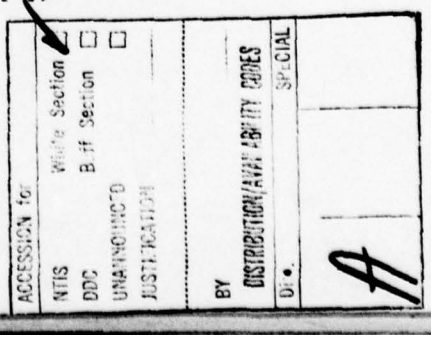
(i) The system constitutes an applications-independent finite element solver for a certain class of two-dimensional, linear, elliptic boundary value problems defined by a weak mathematical formulation.

(1.1) (ii) Adaptive approaches are employed extensively. The a-posteriori estimates developed in [1]-[5] are used to control the adaptive processes and to provide a solution with near optimal error within a prescribed cost range.

(iii) In the systems design advantage was taken of the natural parallelism and modularity of the finite element method.

The general design of the system has been described in [6]. Since it represents an experimental prototype rather than a production system, extensive provisions for evaluating the performance are incorporated.

A principal feature of (1.1)(ii) is an adaptive mesh refinement algorithm. Briefly, after a solution has been obtained on some mesh error indicators are evaluated on the individual elements and from these a very reliable estimate of the error in the energy norm is composed. The error is (asymptotically)optimal for the degrees of freedom used if all indicators are essentially equal. This provides the basis for the refinement algorithm which in essence divides certain elements so as to achieve a more equal distribution of the error indicators (see, e.g., [5]).



The efficiency of such a refinement strategy depends critically on the design of the data structure for the meshes. This is the topic of the present paper. More specifically, we present here a tree structure for the class of meshes considered in [6]. Chapter 2 outlines the general definition of the meshes and introduces the theoretical basis of the tree structure for them and the fundamental access algorithms on the tree. This structure is not restricted to two-dimensional meshes, but the extension shall not be considered here. For the finite element computations under discussion the tree can be truncated considerably. This is discussed in Chapter 3 together with the principal algorithms needed, namely, the refinement of the mesh, the computation of the elemental stiffness matrices, and the assembly and decomposition of the global stiffness matrix. Finally, in Chapter 4 we indicate some possible extensions of the structure.

2. The Basic Data Structure

2.1 Domain and Mesh Definition

The parallelism of the design property (1.1)(iii) is on the procedural level rather than the instructional level. It is specified in terms of processes which are autonomous units with their own programs and data. These processes run in parallel and communicate asynchronously in a limited and highly structured manner.

A natural parallel process structure for the system derives from the familiar substructure analysis in engineering design. The domain $\bar{\Omega} \subset R^2$ is defined as the union $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2 \cup \ldots \cup \bar{\Omega}_N$ of finitely many closed, bounded subsets $\bar{\Omega}_i \subset R^2$ which have nonempty interiors Ω_i such that $\Omega_i \cap \Omega_j = \emptyset$, $i \neq j$. On each subdomain $\bar{\Omega}_i$ a finite element mesh is introduced and, to a considerable extent, the computations on the different subdomains are performed in parallel (see [6]).

For the design of a reasonably efficient mesh refinement algorithm some restrictions on the choice of the subdomains are desirable. It is assumed that each $\bar{\Omega}_i$ is a diffeomorphic image of some fundamental figure F in R^2 on which a simple hierarchy of subdivisions can be defined. On the intersections of the subdomains these diffeomorphisms have to satisfy appropriate compatibility conditions. The finite element meshes on each $\bar{\Omega}_i$ consist of curvilinear elements which are first defined on F and then mapped into $\bar{\Omega}_i$. Thus the mesh construction takes place in F and for the discussion of the data structures it suffices to restrict the attention to F.

From a practical viewpoint there are essentially only two types of fundamental figures F that should be considered here, namely, a square or an equilateral triangle. In order to keep the data structures manageable, it is advantageous to require that the subdivisions of the chosen figure F consist solely of the same type. For instance, if F is the closure $\bar{Q}_0$ of the open unit square

$$Q_0 = \{x \in R^2 \mid 0 < x_i < 1,\ i = 1,2\}\ , \tag{2.1}$$

then admissible meshes on $\bar{Q}_0$ may be defined as collections M of closed squares in $\bar{Q}_0$ which are generated by recursive application of the two rules:

(2.2)
(i) The mesh M consisting only of $\bar{Q}_0$ itself is admissible

(ii) If M is an admissible mesh on $\bar{Q}_0$, then the mesh M' is admissible that is obtained from M by subdividing any one closed square $\bar{Q}$ of M into four congruent squares of half the side length of Q.

A typical mesh on $\bar{Q}_0$ generated in this way is shown in Figure 1. Clearly, the refinement introduces "irregular" points --marked by small circles--which are not corners of all the squares incident with them. If conforming elements are used, the solution at these points is specified by continuity conditions; this results in

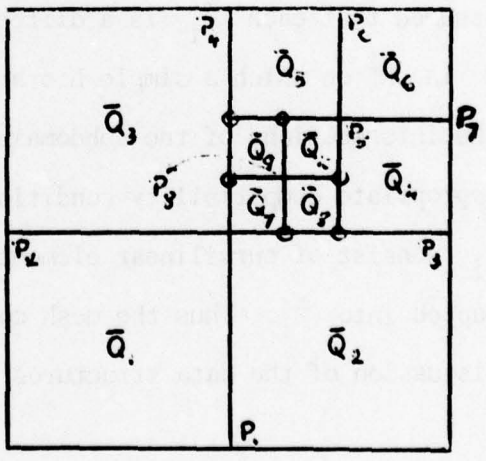


Figure 1

certain complications in the solution process.

If instead of $\bar{Q}_0$ an equilateral triangle is used, then the analogous algorithm (2.2) leads to meshes of the form shown in Figure 2. Here irregular points appear even more frequently. But we might "regularize" them by introducing "irregular" lines--marked by dashes. Then the triangles are no longer similar to each other and, worse yet, the refinement algorithm must be modified considerably to avoid a proliferation of triangles of various shapes. In particular "lengthy" triangles with small angles are numerically very undesirable.

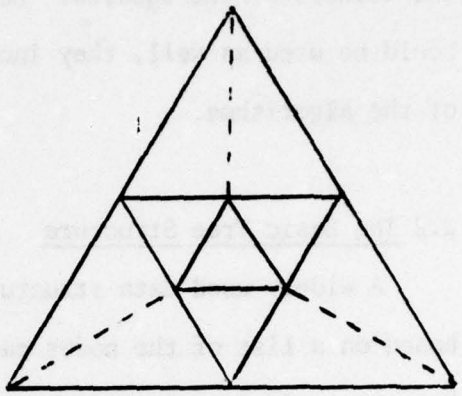


Figure 2

There are other schemes that can be considered each with its particular advantages and disadvantages. At the same time, the types of meshes produced by the various schemes are not at all equivalent when it comes to our design objective of generating near optimal meshes. No halving procedure such as (2.2) can be expected to equalize the error indicators. But it appears that, in general, for the meshes generated by (2.2) on Q_0 the indicators tend to be closer together than for meshes generated by other schemes, e.g., the "regularized" meshes of Figure 2.

For this reason, in our systems design we chose the mesh generation scheme (2.2) on the unit square Q_0 as fundamental figure F. Suitable finite elements are used which have the squares of the admissible meshes

as carriers. For simplicity of the presentation we restrict ourselves to Hermitian elements for which all degrees of freedom are concentrated in the corners of the squares. Lagrangian elements requiring additional nodes could be used as well; they increase only slightly the level of complexity of the algorithms.

2.2 The Basic Tree Structure

A widely used data structure for finite element computations is based on a list of the nodes each pointing to the elements to which it belongs, and of a list of the elements which in turn point to the nodes incident with them. This essentially static structure is not very efficient when mesh refinements are introduced. Instead, the recursive definition (2.2) of the admissible meshes suggests the use of a tree structure that corresponds to the refinement process and has several obvious advantages.

Evidently, the subdivision of a square in some mesh requires only a simple extension of the tree while in the node/element list structure various changes are needed in widely dispersed places. When the tree becomes too large, it is easily partitioned into logically coherent parts for storage on secondary devices. Since the tree structure reflects the refinement process, it provides for an efficient decomposition of the global stiffness matrix corresponding to the well-known nested dissection technique (see, e.g., [7]). The tree structure also allows for a rather efficient treatment of the irregular points discussed above in connection

with Figure 1. In particular, it turns out that it suffices to represent only the regular points on the tree.

In this section we introduce a formal definition of the basic tree structure which will then be simplified in Chapter 3.

Let $B = \{e^1, e^2\}$ be the set of the standard basic vectors $e^1 = (1,0)^T$ and $e^2 = (0,1)^T$ of R^2. For any one of the four subsets $E \subset B$ the cardinality is denoted by $|E|$. In the $|E|$-dimensional plane u+span E the open "square" Q with center u and side length $h > 0$ is defined by

$$(2.3) \quad Q = sq(u,h,E) = \{x \in R^2 \mid \|x-u\|_\infty < \tfrac{1}{2} h,\ x^T e^i = u_i,\ \forall\ e \notin E\} \ .$$

We write $\dim Q = |E|$. In particular, $sq(\frac{1}{2}(e^1+e^2),1,B)$ is the open unit square (2.1) and for any $u \in R^2$ and $h > 0$, $sq(u,h,\emptyset)$ is the point u.

Now let A be the set of nine vectors

$$(2.4) \qquad A = \{a \in R^2 \mid a_i = -1,0,+1;\ i = 1,2\}$$

and set

$$(2.5) \qquad A_E = \{a \in A \mid a_i = 0 \text{ if } e^i \notin E\},\ \forall\ E \subset B \ ,$$

$$(2.6) \qquad E[a] = \{e^i \in E \mid a_i = 0\},\ \forall\ a \in A \ .$$

Then the faces of any open square $Q = sq(u,h,E)$, $E \subset B$, are the $3^{\dim Q}$ squares

$$(2.7) \qquad P = sq(u + \tfrac{1}{2} ha,\ h,\ E[a]),\ \forall\ a \in A_E \ .$$

Note that Q itself is the unique $|E|$-dimensional face of itself.

In line with the refinement rule (2.2)(ii), we introduce a subdivision operator σ which associates with any open square $Q = sq(u,h,E)$, $E \subset B$, the set $\sigma(Q)$ consisting of the $3^{\dim Q}$ squares

$$sq(u + \tfrac{1}{4} ha, \tfrac{1}{2} h, E \sim E[a]), \forall\, a \in A_E \,. \tag{2.8}$$

Note that $\sigma(Q)$ contains exactly one point, namely, $sq(u, \frac{1}{2} h, \emptyset)$. For $E = \emptyset$ this is the point Q itself; otherwise, it is the center of Q.

In order to formalize the refinement algorithm (2.2) on the closed unit square $\bar{Q}_0$, we imbed $\bar{Q}_0$ in the larger open square $Q_{-2} = sq(e^1+e^2, 4, B)$. Now we establish a fixed rudimentary tree T_0 as follows:

(i) The root of T_0 represents the square Q_{-2}.

(ii) The successors of the root are four nodes corresponding to the squares $sq((e^1+e^2)+a, 2, B\sim E[a])$ of $\sigma(Q_{-2})$ with $a \in A_B$, $a \leq 0$.*) (2.9)

(iii) Any node constructed in (ii) representing, say, $Q = sq(u,2,E)$, has $2^{\dim E}$ successor nodes representing the squares of $\sigma(Q)$ with $a \in A_E$, $a \geq 0$.

It is easily seen that the resulting rudimentary tree T_0 has exactly nine terminal nodes representing the open unit square Q_0 and its eight one- and zero-dimensional faces.

Now all admissible subdivision trees T are obtained recursively by application of the two rules:

---

*We use here the standard partial ordering on R^2 defined by $x \geq 0$ whenever $x_i \geq 0$, $i = 1,2$.

(2.10)

(i) T_0 is an admissible tree.

(ii) Let T be an admissible tree and consider any terminal node of T corresponding to a two-dimensional square Q. Then a new admissible tree T' is obtained if we attach to each terminal node of T that represents a face P of Q exactly $3^{\dim P}$ successor nodes corresponding to the squares (2.8) of $\sigma(P)$.

For any such tree T the terminal nodes representing two-dimensional squares correspond exactly to the interiors of the undivided, closed squares of an admissible mesh defined by (2.2).

Clearly, these admissible trees grow rapidly very large; hence, for the implementation they have to be reduced. This will be discussed in Chapter 3 below where also some illustrative examples are given.

2.3 Labels

Let T be an admissible tree as defined by (2.10). We label the nodes of T as follows:

(2.11)

(i) The root of T is labelled $\lambda(\text{root}) = (1,1)^T$.

(ii) For any nonterminal node of T representing the square $Q = sq(u,h,E)$ each successor node p corresponds to a square $P = sq(u + \frac{1}{4} ha, \frac{1}{2} h, E \sim E[a])$, $a \in A_E$, of $\sigma(Q)$. This node p is labelled $\lambda(p) = a$.

These labels allow for an easy reconstruction of the square $Q = sq(u,h,E)$ corresponding to a node q of T. In fact let

$$\pi = \text{path}(q,\text{root}) = \langle p^1, p^2, \ldots, p^n \rangle,\ p^1 = q,\ p^n = \text{root},\ n \geq 1\ , \tag{2.12}$$

be the unique path from the node q to the root. Suppose that p^k corresponds to the square $P_k = sq(u^k,h_k,E_k)$, $k = 1,\ldots,n$. Then we have by construction

$$h_n = 4,\quad u^n = \lambda(p^n) = e^1 + e^2$$

$$h_{k-1} = \frac{1}{2} h_k,\quad u^{k-1} = u^k + \frac{1}{4} h_k \lambda(p^k),\ k = n,n-1,\ldots,2$$

and hence

$$h_k = 2^{2+k-n},\quad u^k = \lambda(p^n) + \sum_{j=k}^{n-1} \lambda(p^j) 2^{j-(n-1)},\quad k = n,n-1,\ldots,1\ . \tag{2.13}$$

Moreover, by (2.8) we have

$$E_k = E \sim E[\lambda(p^k)],\quad k = n,n-1,\ldots,1\ . \tag{2.14}$$

The labelled tree T allows also a simple reconstruction of the faces and neighboring squares of a given square. For this it is useful to introduce some notation.

Note first that the set A of (2.4) is a multiplicative, commutative semigroup under the product

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \otimes \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ a_2 b_2 \end{pmatrix}\ . \tag{2.15}$$

With the members of A we form strings of finite length

$$\alpha = \alpha[1]\alpha[2]\ldots\alpha[n], \quad \alpha[i] \in A, \quad i = 1,\ldots,n . \tag{2.16}$$

For such strings the following three operations are well defined:

$$\begin{aligned} &\text{substring:} && \alpha[i:j] = \alpha[i]\alpha[i+1]\ldots\alpha[j], \quad 1 \le i \le j \le n \\ &\text{concatenation:} && \alpha \cdot \beta = \alpha[1]\ldots\alpha[n]\beta[1]\ldots\beta[m] \\ &\text{multiplication:} && a \otimes \alpha = (a\otimes\alpha[1])(a\otimes\alpha[2])\ldots(a\otimes\alpha[n]), \ \forall\, a \in A . \end{aligned} \tag{2.17}$$

Now suppose that on the subdivision tree T the node q represents the two-dimensional square $Q = sq(u,2^{3-n},B)$ with side length $h = 2^{3-n}$, $n \ge 3$. We wish to find the nodes of T corresponding to the corner points P_{ij}, $i,j = \pm 1$, the sides T_{ij}, $i = \pm 1$, $j = 0$, or $i = 0$, $j = \pm 1$, and the neighboring squares of the same size S_{ij}, $i = \pm 1$, $j = 0$, or $i = 0$, $j = \pm 1$, if they exist (see Figure 3.)

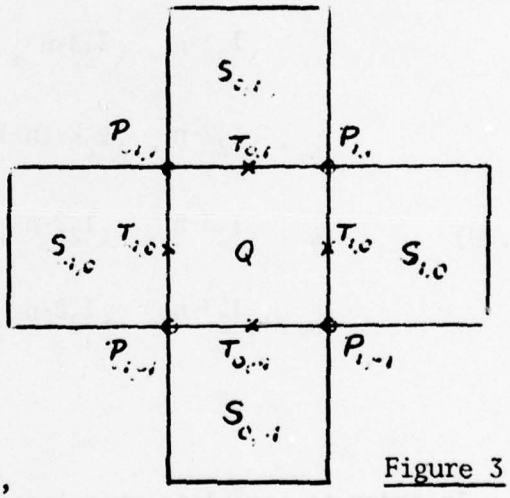


Figure 3

Since $u = u(Q)$ is the center of Q, it is obvious that the centers of these various "squares" are given by

$$\begin{aligned} u(P_{ij}) &= u(Q) + 2^{2-n}\binom{i}{j}, && i,j = \pm 1 \\ u(T_{ij}) &= u(Q) + 2^{2-n}\binom{i}{j} \Big\} && i = \pm 1,\ j = 0 \\ &\qquad \text{and} \\ u(S_{ij}) &= u(Q) + 2^{3-n}\binom{i}{j} \Big\} && i = 0,\ j = \pm 1 \end{aligned} \tag{2.18}$$

Let π be the path (2.12) from q to the root and $\alpha(q) = \text{string}(\pi)$ the corresponding label string with $\alpha[k] = \lambda(p^k) = (\lambda_1^k, \lambda_2^k)^T$, $k = 1,\ldots,n$. For ease of notation we denote by ω any one of the components of $u(Q)$. By (2.13) we then have

$$\omega = \lambda^1 2^{2-n} + \lambda^2 2^{3-n} + \ldots + 2^{-1}\lambda^{n-2} + \lambda^{n-1} + \lambda^n .$$

An addition or subtraction of 2^{2-n} evidently generates a carry. More specifically, assume that

$$\lambda^1 = \lambda^2 = \ldots = \lambda^{k-2}, \quad \lambda^{k-1} = -\lambda^{k-2}, \quad k \geq 2 ,$$

then it follows readily that

$$\omega - \lambda^1 2^{2-n} = \lambda^2 2^{3-n} + \ldots + \lambda^n$$

$$\omega + \lambda^1 2^{2-n} = \lambda^k 2^{k-(n-1)} + \ldots + \lambda^n$$

(2.19)
$$\omega - \lambda^1 2^{3-n} = -\lambda^1 2^{2-n} + \lambda^2 2^{3-n} + \ldots + \lambda^n$$

$$\omega + \lambda^1 2^{3-n} = -\lambda^1 2^{2-n} - \ldots - \lambda^{k-2} 2^{k-n-1} - \lambda^{k-1} 2^{k-n} + \lambda^k 2^{k-n+1} + \ldots + \lambda^n$$

In order to translate this into label strings we introduce for any node q with label string $\alpha = \text{string}(\pi)$ the indices

(2.20)
$$k_i = k_i(q) = 1 + \min(k | \lambda_i^k = -\lambda_i^1), \; i = 1,2, \; \alpha[j] = \begin{pmatrix} \lambda_1^j \\ \\ \lambda_2^j \end{pmatrix} .$$

Since always $\lambda_i^n = 1$, $\lambda_i^{n-1} = -1$, $\lambda_i^{n-2} = 1$, $i = 1,2$, the existence of these indices k_i is guaranteed for any node q for which $\pi = path(q,root)$ has at least length $n \geq 2$. Evidently the nodes with $n = 0,1$ are nonterminal nodes of the rudimentary tree and have no direct bearing on our subdivision process.

Now let p_{ij}, t_{ij}, and s_{ij} denote the nodes of T (if they exist) corresponding to the "squares" P_{ij}, T_{ij}, and S_{ij}, respectively. Moreover, assume that $\lambda(q) = (\lambda_1,\lambda_2)^T$. Then we obtain from (2.19) the following formulas for the label strings $\beta(p_{ij}) = string(path(p_{ij},root))$:

$$\beta(p_{-\lambda_1,-\lambda_2}) = \alpha[2:n]$$

$$\beta(p_{-\lambda_1,\lambda_2}) = \{e^1 \otimes \alpha[2:k_2-1]\} \cdot \alpha[k_2:n]$$

$$\beta(p_{\lambda_1,-\lambda_2}) = \{e^2 \otimes \alpha[2:k_1-1]\} \cdot \alpha[k_1:n] \tag{2.21}$$

$$\beta(p_{\lambda_1,\lambda_2}) = \begin{cases} \{e^1 \otimes \alpha[k_1:k_2-1]\} \cdot \alpha[k_2:n] & \text{if } k_1 < k_2 \\ \alpha[k_2:n] & \text{if } k_1 = k_2 \\ \{e^2 \otimes \alpha[k_2:k_1-1]\} \cdot \alpha[k_1:n] & \text{if } k_1 > k_2 \end{cases}$$

Similarly it follows for $\beta(t_{ij}) = string\ (path(t_{ij},root))$ that

$$\beta(t_{0,-\lambda_2}) = \lambda_1 e^1 \cdot \alpha[2:n]$$

$$\beta(t_{-\lambda_1,0}) = \lambda_2 e^2 \cdot \alpha[2:n] \tag{2.22}$$

$$\beta(t_{0,\lambda_2}) = \{e^1 \otimes \alpha[1:k_2-1]\} \cdot \alpha[k_2:n]$$

$$\beta(t_{\lambda_1,0}) = \{e^2 \otimes \alpha[1:k_1-1]\} \cdot \alpha[k_1:n]$$

and for $\beta(s_{ij}) = \text{string}(\text{path}(s_{ij},\text{root}))$ that

$$\beta(s_{0,-\lambda_2}) = \binom{\lambda_1}{-\lambda_2} \cdot \alpha[2:n]$$

$$\beta(s_{-\lambda_1,\lambda_2}) = \binom{-\lambda_1}{\lambda_2} \cdot \alpha[2:n]$$

(2.23)

$$\beta(s_{0,\lambda_2}) = \{\binom{1}{-1}\times\alpha[1:k_2-1]\} \cdot \alpha[k_2:n]$$

$$\beta(s_{\lambda_1,0}) = \{\binom{-1}{1}\times\alpha[1:k_1-1]\} \cdot \alpha[k_1:n]$$

Figure 4 shows the nodes p_{ij}, t_{ij}, and s_{ij} on the tree and the paths represented by the various label strings. In order to find the different nodes, we need only follow the indicated paths in T given by the label sequences written besides them.

The following algorithm retrieves the nodes and their labels on the path $\pi = \text{path}(q,\text{root})$ in the arrays p and α, respectively. It also gives the length n of π and the two indices $k_i = k_i(q)$, $i = 1,2$.

```
Algorithm Up-Path
Input (q)
n := 1; p[1] := q
k_i := 0; α[1]_i := λ(q)_i, for i = 1,2
r := father(q)
While r ≠ nil do
  begin n := n+1; p[n] := r
    For i = 1,2 do begin α[n]_i := λ(r)_i;
      if (k_i=0) and (α[n]_i≠λ(r)_i) then k_i := n+1 end
    r := father(r)
  end
Output (n,p,α,k)
```

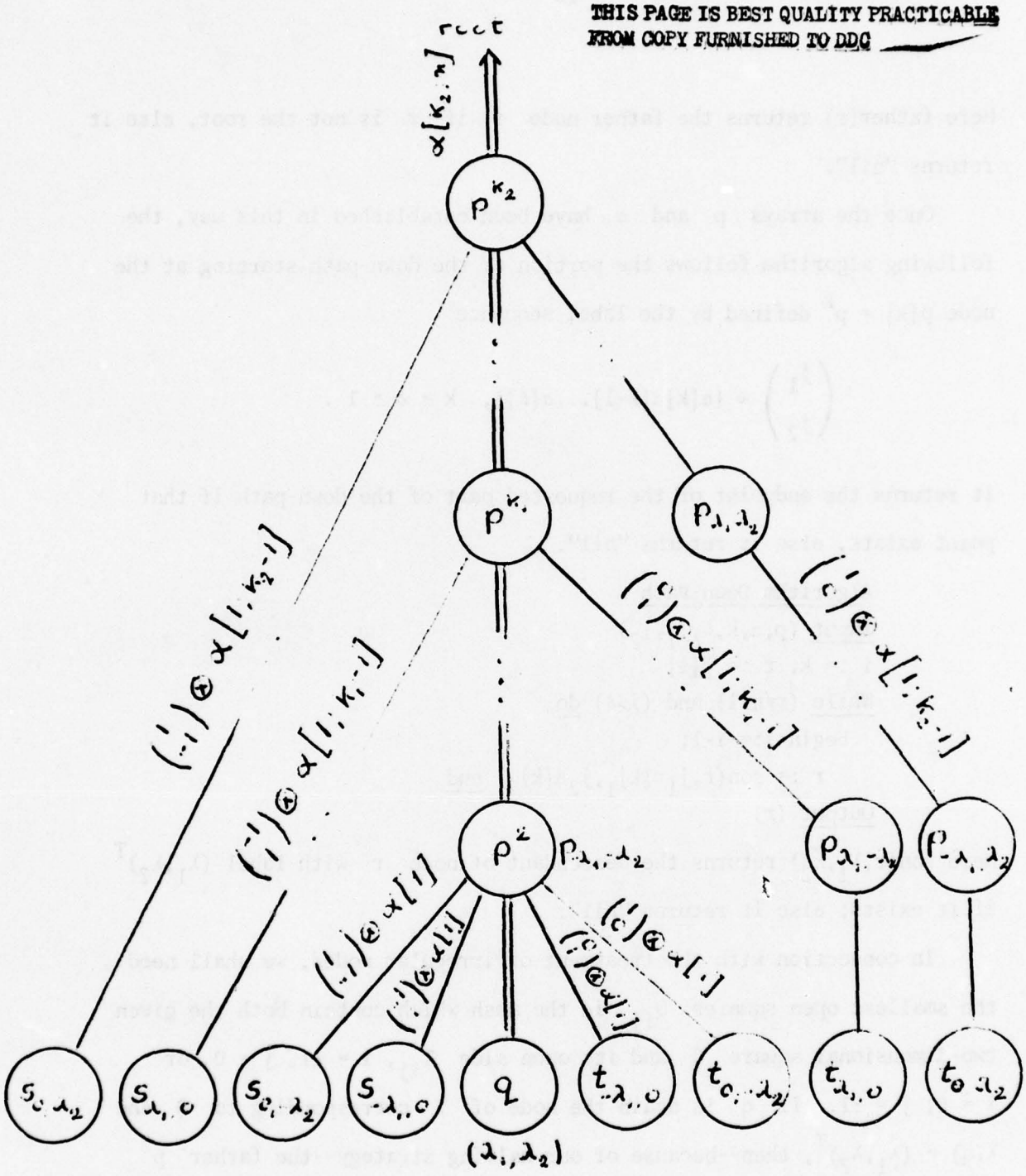


Figure 4 ($k_1 < k_2$)

Here father(r) returns the father node r if r is not the root, else it returns "nil".

Once the arrays p and α have been established in this way, the following algorithm follows the portion of the down-path starting at the node p[k] = p^k defined by the label sequence

$$\binom{j_1}{j_2} \times \{\alpha[k]\alpha[k-1]\ldots\alpha[\ell]\}, \quad k \geq \ell \geq 1 .$$

It returns the endpoint of the requested part of the down-path if that point exists, else it returns "nil".

```
Algorithm Down-Path
Input (p,α,k,ℓ,j1,j2)
i := k; r := p[i]
While (r≠nil) and (i>ℓ) do
  begin i:= i-1;
    r := son(r,j1α[k]1,j2α[k]2) end
Output (r)
```

Here $son(r,\lambda_1,\lambda_2)$ returns the descendant of node r with label $(\lambda_1,\lambda_2)^T$ if it exists; else it returns "nil".

In connection with the treatment of irregular nodes, we shall need the smallest open squares Q_{ij} in the mesh which contain both the given two-dimensional square Q and its open side T_{ij}, i = ±1, j = 0, or i = 0, j = ±1. If q is again the node of T corresponding to Q and $\lambda(q) = (\lambda_1,\lambda_2)^T$, then--because of our halving strategy--the father p^2 of q always represents the squares $Q_{-\lambda_1,0} = Q_{0,-\lambda_2}$. Moreover, it is geometrically obvious that p^{k_1} and p^{k_2} correspond to $Q_{\lambda_1,0}$ and Q_{0,λ_2}, respectively. Thus, these four cases are a simple byproduct of our algorithm.

3. Implementation Aspects

3.1 The Truncated Subdivision Tree

As indicated earlier, for the finite element computations considered here it is unnecessary to implement the full subdivision tree T discussed in Chapter 2. In fact, we may truncate T considerably by deleting branches carrying information not needed in the calculations. This truncation is achieved in several steps.

When an open square $Q = sq(u,2h,B)$ is subdivided, a node p representing the center point $u = sq(u,h,\emptyset)$ is introduced in T. Thereafter, if one of the squares incident with u is subdivided, p receives one descendant node representing again the point $u = sq(u,h/2,\emptyset)$. This repeats itself; that is, a string of nodes with single descendants is generated, all of which represent the center u of Q. A first truncation of T therefore consists in deleting all nodes that represent a point $u = sq(u,h,\emptyset)$, $h > 0$.

This leaves us only with nodes on T that represent squares of dimension one or two. Once such a square has been subdivided, it no longer plays a direct role in the computation. Hence from now on any nonterminal node of T corresponding to a subdivided square $sq(u,h,E)$, $|E| = 1,2$, will be considered to represent its center point u.

In general, undivided line segments $S = sq(u,h,E)$, $|E| = 1$, do not carry independent information. Thus, a second truncation of T consists in the deletion of all terminal nodes that correspond to such line segments S. With this, all terminal nodes of T represent undivided, two-dimensional squares or points, and all nonterminal nodes correspond to points.

Recall that in the admissible mesh defined by the given tree T, a point in the open unit square Q_0 is irregular if it is not a corner point of all two-dimensional squares incident with it (the circled points of Figure 1). If conforming elements are used, as discussed in Section 2.1, then these irregular points carry no independent information. Obviously, an irregular point P can only occur somewhere on a divided line segment $S = sq(v,h,E)$, $|E| = 1$, which is the side of some undivided square $sq(u,h,B)$. In other words, any other point on S must also be irregular. Thus, a node of T corresponding to an irregular point can never have a descendant node that represents a regular point. We may therefore truncate T a third time by deleting all nodes that correspond to an irregular point. Clearly, this third truncation cannot be applied when nonconforming elements are used.

By definition, all points on the boundary ∂Q_0 of the unit square are regular. This reflects the fact that on ∂Q_0 other conditions have to be taken into consideration. A side S of $\bar{Q}_0$ is either the image of the intersection $\bar{\Omega}_i \cap \bar{\Omega}_j$ of two of the subdomains mentioned in Section 2.1 or of a part of the boundary $\partial\bar{\Omega}$ of $\bar{\Omega}$ where some boundary condition may or may not be specified. In the first case the points on S are represented on the subdivision trees of either one of the subdomains. Here it appears to be advantageous not to delete the corresponding nodes from these trees even if they are irregular on the union of the subdomains. The second case depends on the type of boundary condition. For instance, if the solution is specified on the part of $\partial\bar{Q}_0$ corresponding to S then there is certainly no need to represent the points of S on the tree. We shall not enter into the details of the various other possibilities.

At any irregular point the solution is obtained by interpolation. Hence special consideration must be given to the computation of the local stiffness matrix of an element that has as carrier a square with some irregular corners. This will be discussed in Section 3.3. To simplify this computation, a regularity tag is assigned to each node of T representing a divided or undivided square $Q = sq(u,h,B)$. It consists of a triple (ρ_0,ρ_1,ρ_2) of single bit numbers ρ_i which indicate the regularity of three of the corners of Q. If the corner is regular, the bit ρ_i is one; otherwise, it is zero. The assignment of the tag proceeds recursively as follows:

(3.1)

(i) The root of T has the tag (1,1,1).

(ii) If a nonterminal node represents the center u of a divided square and q is a descendant node corresponding to the square Q, then ρ_0 indicates the regularity of the corner of Q opposite to u and ρ_i that of its corners in the x_i-direction, $i = 1,2$, from u.

The scheme is illustrated in Figure 5.

In Figure 6 we give an example of the truncated subdivision tree for the mesh shown there. The nodes corresponding to divided or undivided squares are marked by rectangular boxes. All others,

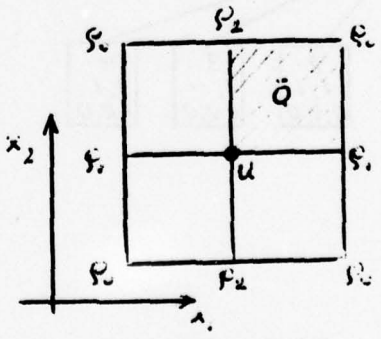


Figure 5

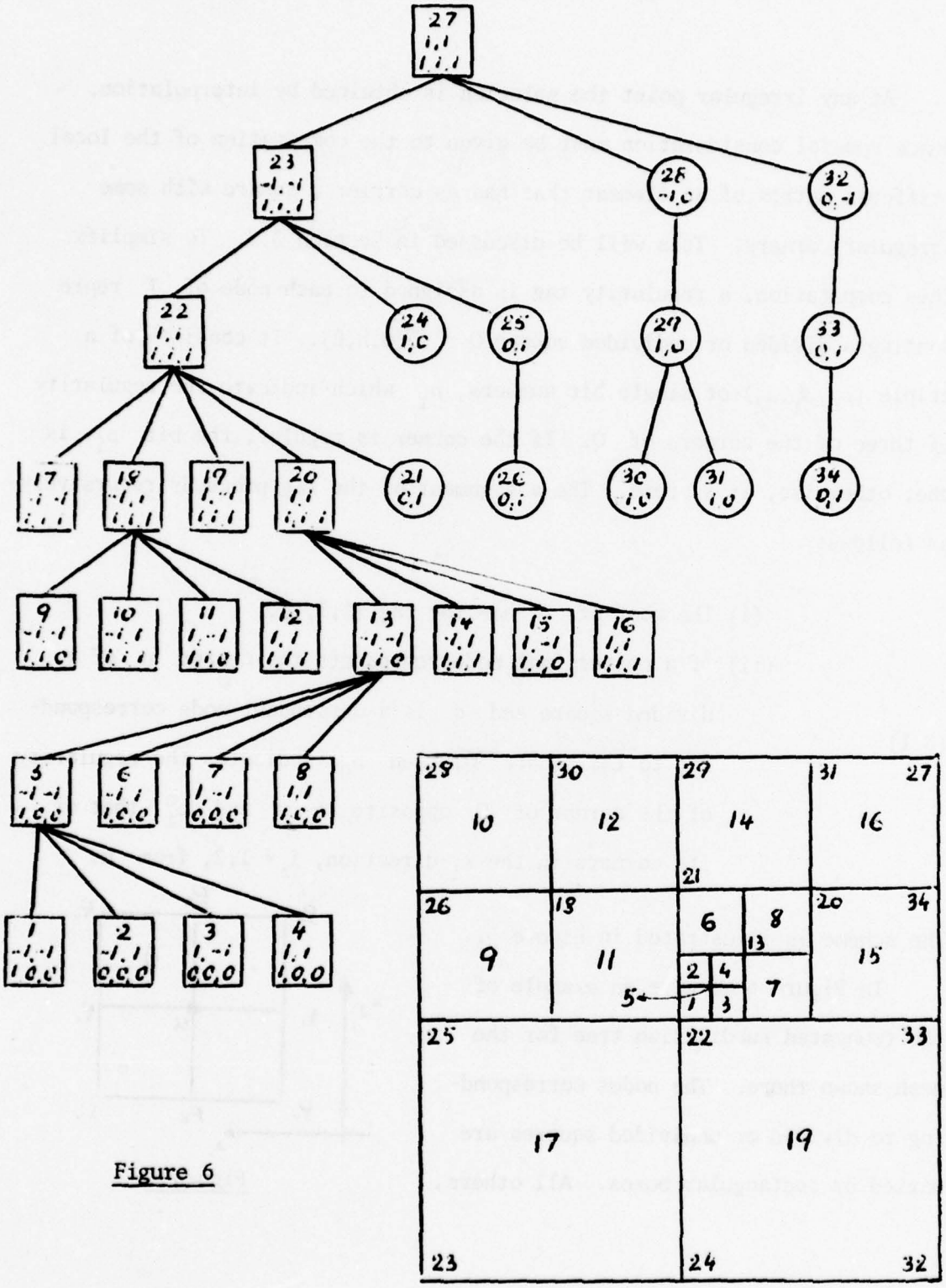

28
30
29
31
27
10
12
14
16
21
26
18
6
8
20
34
9
11
13
15
2
4
5
1
3
7
25
22
33
17
19
23
24
32

Figure 6

of course, carry no regularity tags. In all nodes the first number is the identifier, the pair of numbers below it the label, and--where applicable--the triple below that the regularity tag. It may be noted that the full tree for the same mesh would have more than a hundred nodes.

3.2 Subdivision

Let q be a terminal node of T corresponding to an (undivided) square $Q = sq(u,h,B)$. When Q is subdivided the following steps have to be taken:

(1) New nodes have to be created on the tree corresponding to the new squares created from Q and to any points on the sides of Q that become regular.

(2) Regularity tags have to be assigned to the newly created nodes where needed; and if some points on the sides of Q have become regular the regularity tags of all nodes have to be modified that represent squares outside Q incident with these points.

The first part of the resulting algorithm creates the nodes for the new squares and assigns their ρ_0 values:

```
for i,j = ±1 do
begin
  create nodes q_ij as son of q;
    λ(q_ij) := (i,j);
    ρ1(q_ij) := ρ2(q_ij) := 0;
```

```
    if (i,j) = (-λ1,-λ2) then ρ0(qij) := 1;
    if (i,j) = (λ1,λ2) then ρ0(qij) := ρ0(q);
    if (i,j) = (λ1,-λ2) then ρ0(qij) := ρ1(q);
    if (i,j) = (-λ1,λ2) then ρ0(qij) := ρ2(q);
end;
```

Here (λ_1,λ_2) represents the label of q.

The second part of the algorithm needs the output of the up-path algorithm of Section 2.3. Using it we can check whether the neighboring square $s = s_{\pm1,0}$ or $s = s_{0,\pm1}$ exists and is divided in which case the center of that side of Q becomes regular and a corresponding node $t = t_{\pm1,0}$ or $t = t_{0,\pm1}$ has to be created. Let ν be the index of the coordinate direction x_ν, $\nu = 1,2$, and set $\mu = 1$ if the neighboring square is in p^2 and $\mu = 2$ if it is in p^{k_ν}. Then the down-path for a particular s or t starts from the node p^n where $n = m_{\mu\nu}$ with $m_{1\nu} = 2$, $m_{2\nu} = k_\nu$, $\nu = 1,2$. Here k_1, k_2, of course, are the indices (2.20) obtained by the up-path algorithm. We also introduce the following label functions $\eta_\pm$ of the labels λ_1,λ_2 of q and the indices μ,ν:

μ	ν	$\eta_{\pm1}(\lambda_1,\lambda_2;\mu,\nu)$
1	1	$(-\lambda_1,\pm1)$
1	2	$(\pm1,-\lambda_2)$
2	1	$(\lambda_1,\pm1)$
2	2	$(\pm1,\lambda_2)$

Then the next part of the subdivision algorithm has the generic form:

```
(n,p,α,k) := up-path (q)
for μ,ν = 1,2 do
begin m̄ := m_μν
  if m̄ < n-1 then s := down-path (p,n,m̄,1,2ν-3,3-2ν)
  if (m̄=n-1) or (s≠terminal) then
  begin for ℓ = ±1 do begin (i,j) := η_ℓ(λ_1,λ_2;μ,ν);
                             ρ_ν(q_ij) := 1 end;
    r := down-path(p,n,m̄,2,ν-1,2-ν);
    create node t as son of r;
    λ(t) := ((ν-1)λ_1,(2-ν)λ_2);
    if m̄ < n-1 then set regularity tags for the squares in s
                                  incident with t;
  end;
end;
```

Here the condition $\bar{m} \geq n-1$ indicates that the particular side of Q is on the boundary $\partial\bar{Q}_0$ of $\bar{Q}_0$. Hence there is no neighboring square. Moreover, since all points on $\partial\bar{Q}_0$ are treated here as regular points, corresponding nodes are always created on the tree.

The statement requiring the setting of the regularity tags for the squares in s incident with t has the form of the following algorithm:

```
for j = ±1 do
begin for ℓ = ±1 do
  begin r := son of s with label η_ℓ(-λ_1,-λ_2;μ,ν);
    ρ_ν(r) := 1;
    while (r≠terminal) do
    begin r := son of r with label η_-ℓ(-λ_1,-λ_2;μ,ν);
          ρ_0(r) := 1;
    end;
  end;
end;
```

The while-loop covers the case when there is a nested set of squares with corners at t.

3.3 Elementary Stiffness Matrices

As mentioned in Section 2.1, the appearance of irregular points complicates the calculation of the elementary stiffness matrices when conforming elements are used. We sketch here the algorithm for the simple case of conforming, bilinear, square elements. It should be evident how the approach extends to more general cases.

Let $Q = sq(u,h,B)$ be a given square with corners $P_1,\ldots,P_4$, and $w_j = w_j(x_1,x_2;h)$ the usual bilinear shape functions with $w_j(P_i) = \delta_{ij}$, $i,j = 1,\ldots,4$. Then the desired solution y on Q has the form

$$y(x_1,x_2) = \sum_{j=1}^{4} y_j w_j(x_1,x_2;h),\ x \in Q\ , \tag{3.2}$$

and--if all corners are regular--the elementary stiffness matrix for Q is

$$M = (\mathcal{B}(w_i,w_j),i,j = 1,\ldots,4) \tag{3.3}$$

where $\mathcal{B} = \mathcal{B}(.,.)$ denotes the given bilinear form. In the case of irregularity some of the values $y_j = y(P_j)$ are given as linear combinations of some other values, say,

$$y_j = \sum_{\ell=1}^{m} c_{j\ell}\tilde{y}_\ell,\ j = 1,\ldots,4\ . \tag{3.4}$$

Then

$$y = \sum_{\ell=1}^{m} \left(\sum_{j=1}^{4} c_{j\ell} w_j \right) \tilde{y}_\ell$$

calls for the introduction of the new functions

$$\tilde{w}_\ell = \tilde{w}_\ell(x_1,x_2;h) = \sum_{j=1}^{4} c_{j\ell} w_j(x_1,x_2;h), \quad \ell = 1,\ldots,4 ,$$

and the corresponding elementary stiffness matrix of Q is given by

$$\tilde{M} = (B(\tilde{w}_i,\tilde{w}_j),\ i,j = 1,\ldots,\tilde{m}) = C^T M C \tag{3.5}$$

where $C = (c_{j\ell},\ j = 1,\ldots,4,\ \ell = 1,\ldots,\tilde{m})$ is the interpolation matrix in (3.4).

The $\tilde{y}_\ell$ are the solution values at certain regular points and our problem is to find these points and the interpolation coefficients $c_{j\ell}$. As an example, consider the square 4 of Figure 6. Here the $\tilde{y}_\ell$ are values at the points 5, 13, 21, 22, and 33 and, if the corners of the square are numbered counter-clockwise starting from 5, the interpolation matrix has the form

$$C = \begin{array}{c} \begin{matrix} (5) & (13) & (21) & (22) & (33) \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 3/8 & 1/8 \\ 0 & 1 & 0 & 0 & 0 \\ 1/2 & 0 & 1/4 & 1/4 & 0 \end{pmatrix} \end{array} \tag{3.6}$$

Basically the desired algorithm is a modified version of the down-path procedure of Section 2.3. Let (n,p,α,k) be the output of the up-path algorithm starting from q. If (λ_1,λ_2) is the label of q, then the corner

$P_{-\lambda_1,-\lambda_2}$ (in the notation of Figure 3) must be regular, and the other potentially irregular corners of Q are found on the down-paths from $p[k_\nu]$, $\nu = 1,2$, with label sequences given by (2.21). These three label sequences have the general form

$$\{e^\nu \infty \alpha[\kappa_2:\kappa_1-1]\} \cdot \alpha[\kappa_1:n] \tag{3.7}$$

where, for instance, in the case of $P_{-\lambda_1,\lambda_2}$ we have $\nu = 1$, $\kappa_1 = k_2$, $\kappa_2 = 2$.

Let R be one of these corners and suppose that R is irregular. Then it must be on a line S created by the subdivision of the square Q_ζ corresponding to $p[\kappa_1]$. Then any regular point on S is on the subtree rooted at $p[\kappa_1]$ with labels that have a zero ν-th component. Figure 7 shows a typical situation.

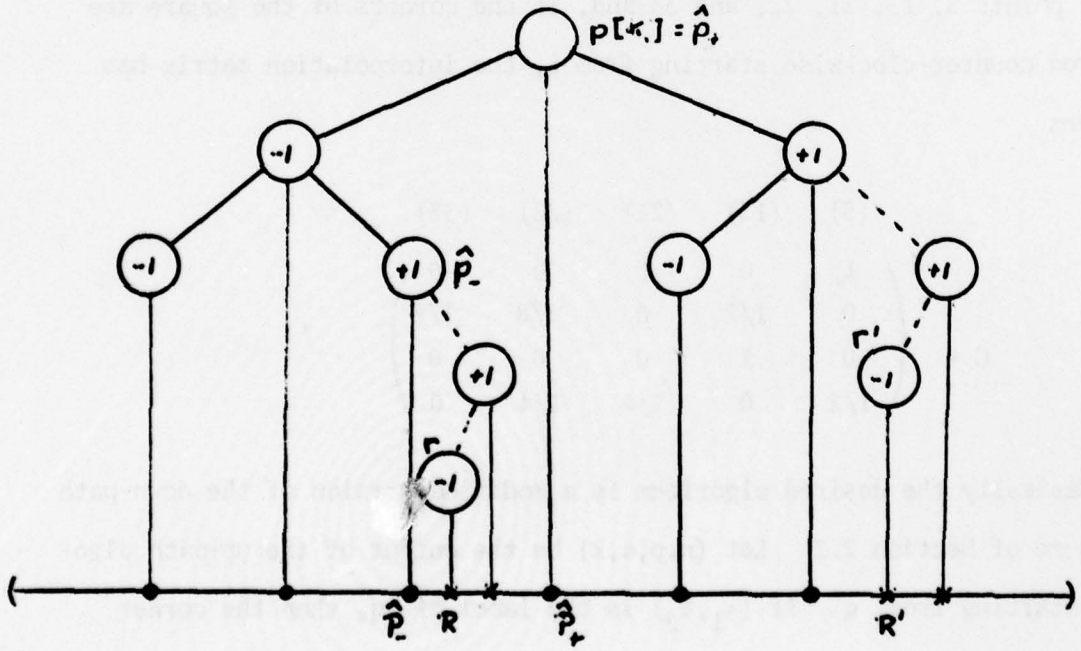


Figure 7

The branches deleted from the full subdivision tree are marked by dashed lines. Only the nonzero component of the label is shown at each node. Briefly, we wish to find the nodes $\hat{p}_{+1}$ and $\hat{p}_{-1}$ on the truncated tree which correspond to the nearest regular points $\hat{P}_{+1}$ and $\hat{P}_{-1}$ bracketing P.

The following algorithm for finding these nodes (if they exist) should be self-explanatory. It starts from the node $p[i_1]$, where initially $i_1 = \kappa_1$, and proceeds along the prescribed down-path. If the requested endpoint r does not exist, then the nodes $\hat{p}_{+1}, \hat{p}_{-1}$ are returned with the relative distances $\hat{d}_{-1}, \hat{d}_{+1}$, respectively, between the corresponding points and P. One of these nodes may not exist--as for the point R' in Figure 7--in which case the particular output node is nil.

```
Algorithm: Down-path interval
Input (p,α,i1,i2,ν)
for j = ±1 do begin p̂j := nil; dj = 0 end
j = i1; r := p[i1]
while (i>i2) and (r≠nil) do
begin i := i-1
      η := α[i]ν
      p̂η := r
      r := son of r with label (2-ν choose ν-1) ⊗ α[i]
end
if r = nil then
begin γ := 1/2; d := 0
  while j ≥ i2 do begin d := d +γα[i]ν; j := i-1; γ := (1/2)γ end
  d-η = abs(d); dη = 1 - d-η
end
Output (r, p̂±1,d±1)
```

The distance calculation uses the fact that by (2.13)

$$\text{dist}(R,\hat{P}_\nu) = \left| \sum_{j=i_2}^{i} \alpha[j]\, 2^{j-N+1} \right|$$

where j is the last index used in the while loop. Since R must be in one half of the interval centered at $\hat{P}_\nu$ we have

$$\text{dist}\,(\hat{P}_{+1},\hat{P}_{-1}) = 2^{2+i-n},$$

and hence

$$d = \frac{\text{dist}(R,\hat{P}_\nu)}{\text{dist}(\hat{P}_{+1},\hat{P}_{-1})} = \frac{1}{2}\left| \sum_{j=i_2}^{i} \alpha[j]_\nu 2^{j-i} \right|.$$

There are three cases for the output of this algorithm, namely, (i) $r \neq$ nil, (ii) $r =$ nil, and no $\hat{p}_\nu$ is nil, (iii) $r =$ nil, and one $\hat{p}_\nu$ is nil. In the first case the corresponding row of the interpolation matrix has a one in the column corresponding to r and zeroes elsewhere. In the second case, there are the values $d_{\mp 1}$ in the columns for $\hat{p}_{\pm 1}$ and zeroes elsewhere. In the third case, we have the situation of the point R' in Figure 7; that is, in the search for one of the points $\hat{P}_\nu$ we reached one of the endpoints of S. Then we have to apply our algorithm to the line perpendicular to S centered at that endpoint. The following algorithm covers these three cases. The output consists of the number m of nonzero elements in the particular row of C. These elements are identified by pairs $(r[j],c[j])$, $j = 1,\ldots,m$, where $c[j]$ is the element of C' in that row and the column identified by the node $r[j]$.

```
Algorithm: Interpolation Row
Input (p,α,κ1,κ2,ν)
m := 1; r[1] := nil; c[1] := 1
ν0 := ν; i1 := κ1; i2 := κ2;
while r[m] = nil do
begin (r0,p̂±1,d±1) := down-path interval (p,α,i1,i2,ν0)
  if r ≠ nil then r[m] := r0
    else begin
      if p̂-1 = nil then j := 1 else j := -1;
      r[m] := pj, r[m+1] := p-j;
      c[m+1] := c[m]d-j; c[m] := c[m]dj;
      m := m+1
      if r[m] = nil then
        begin i2 := i1; ν0 := 3-ν0; m := α[i1]ν0; i1 := i1+2;
          while α[i1-1]ν0 = m do i1 := i1+1;
        end
      end
  end
Output (m,r,c)
```

In the case of r[m] = nil we must continue the search on the line perpendicular to the current search direction. This is reflected by the statement $\nu_0 := 3 - \nu_0$. To find the needed endpoint of S is equivalent to determining the centerpoint T of one of the sides of Q_S (see Figure 3). Thus, we need to construct on the up-path from q_s corresponding to Q_S the node from where the new line containing the desired endpoint is branching off. This is the content of the loop "while $\alpha[i_1-1]_{\nu_0} = m$ do".

With these algorithms the interpolation matrix C of (3.4) is constructed row by row. The calculation of the transformed elementary stiffness matrix $\hat{M}$ of (3.5) is then straightforward.

3.4 Matrix Assembly and Decomposition

The assembled global stiffness matrix has the bordered block-diagonal structure shown in Figure 8. The matrices A_i correspond to the finite element nodes in the open subdomains Ω_i, $i = 1,\ldots,N$, and the border represents the points on the intersections $\bar{\Omega}_i \cap \bar{\Omega}_j$, $i,j = 1,\ldots,N$. In general, the size of B is much smaller than that of the other diagonal blocks; moreover, B is often not very sparse, especially when N is small.

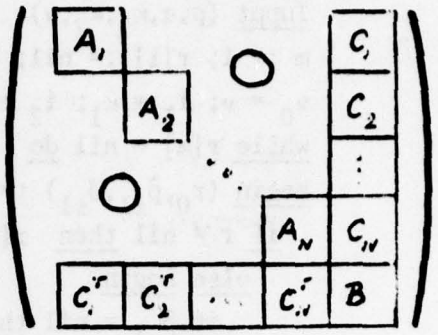


Figure 8

The form of the matrix calls for the use of block decomposition. Since the A_i-blocks are independent of each other, it suffices to consider only one block at a time. The triangular factorization

$$A_i = U_i^T D_i U_i \tag{3.8}$$

of one of these blocks introduces the following partial decomposition of the overall matrix

$$\begin{pmatrix} A_i & C_i \\ C_i^T & B \end{pmatrix} = \begin{pmatrix} U_i^T & 0 \\ \tilde{C}_i^T & 0 \end{pmatrix} \begin{pmatrix} D_i & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} U_i & \tilde{C}_i \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & \tilde{B} \end{pmatrix} \tag{3.9)(i}$$

where

$$\tilde{B} = B - \tilde{C}_i^T D_i \tilde{C}_i, \quad C_i = U_i^T D_i \tilde{C}_i \,. \tag{3.9)(ii}$$

Accordingly, for each subdomain Ω_i our procedure consists of the four steps:

(i) Generate the matrices A_i and C_i;

(ii) Generate the contributions to the matrix B and send them to a separate solution process;

(3.10) (iii) Compute the decomposition (3.8) of A_i and the corresponding modified matrix $\tilde{C}_i$;

(iv) Compute the modification $\tilde{C}_i D_i \tilde{C}_i$ and send it to the solver for B.

When these steps have been completed for all subdomains, the solver for B can complete the decomposition of that matrix.

The structure of the solution routine for B is standard and needs no discussion. Instead we consider only the implementation of the steps (3.10) in the setting of our particular storage structure. As mentioned before, the tree structure allows for a natural use of the nested dissection technique [7] in the execution of step (iii). It also provides for an efficient segmentation of the block-row (A_i, C_i) into groups of rows for storage on secondary devices. The steps (i) and (ii) are combined with step (iii); that is, the generation of the coefficients of A_i, C_i (and B) from the corresponding elementary stiffness matrices is incorporated into the decomposition process. In other words, an elementary stiffness matrix is generated (see Section 3.3) only when it is needed in the decomposition step (iii).

Any subdomain Ω_i is the image of the unit square Q_0 and the mesh on Ω_i is defined by the mesh on Q_0. Let T be the (truncated) subdivision tree corresponding to this mesh. With any node q of T we associate the first nonterminal node r_q with regularity tag (1,1,1) on the path from q to the root. For any nonterminal node q of T with ρ-tag (1,1,1) we may then define the node set $q^* = \{p \mid p \text{ node of } T \text{ with } r_p = q\}$. By construction, this set specifies a subtree of T which shall be denoted by $T(q^*)$. Moreover, in the usual manner we derive from T the contracted tree T^* with the sets q^* as nodes. Figure 9 shows T^* and one of the subtrees $T(q^*)$ for the tree of Figure 6.

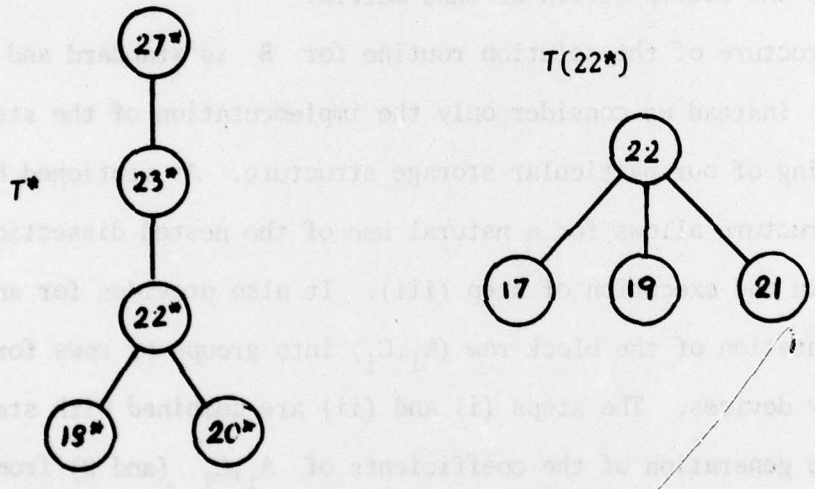


Figure 9

Note that in T the three nodes q_{-2},q_{-1},q_0 corresponding to the squares $Q_{-2} = sq(e^1+e^2,4,B)$, $Q_{-1} = sq(0,2,B)$, and Q_0 have ρ-tag (1,1,1). Hence in each case T^* begins with the succession of the three nodes q^*_{-2},q^*_{-1},q^*_0 before multiple branching sets in. Of course, $T(q^*_{-2})$ and $T(q^*_{-1})$ contain the nodes corresponding to the faces of Q_0.

Let N^* be the node set of T^* and $N^*_0 = N^* \sim \{q^*_{-2},q^*_{-1}\}$. For any $q^* \in N^*$ let $P(q^*)$ and $E(q^*)$ be the sets of nodes of $T(q^*)$ which in the mesh correspond to points or undivided squares, respectively. Because of $q \in P(q^*)$ the set $P(q)$ is never empty; on the other hand, $E(q^*)$ may well be empty as, for instance, for $q^* = q^*_{-2},q^*_{-1}$. The nodes of the union $U\{P(q^*)|q^* \in N^*_0\}$ correspond exactly to the rows and columns of the matrix block A_i for our particular subdomain Ω_i and those of $P(q^*_{-2}) \cup P(q^*_{-1})$ to those columns of C_i which are related to Ω_i.

In order to define the pivoting sequence on A_i reflecting nested dissection, we introduce bottom-up, left-to-right orderings (end-order traversals, see [8, p. 334]) on all nodes of T^* and on the nodes of $T(q^*)$ belonging to $P(q^*)$:

$$\eta^*: N^*_0 \to \{1,2,\ldots,|N^*_0|\}$$

$$\eta^*_q: P(q^*) \to \{1,2,\ldots,|P(q^*)|\},\ \forall\ q^* \in N^*_0 \tag{3.11}$$

In the example of Figure 10, η^* and η^*_{22} give the orderings

$$18^*,20^*,22^*,23^*,27^* \text{ and } 17,19,21,22\ .$$

Note that for the root r of $T(q^*)$ we always have $\eta_{q^*}(r) = |P(q^*)|$.

The pivotal sequence for A_i is now defined by the lexicographic ordering of the pairs

$$\eta(p) = (\eta^*(p), \eta_{q^*}(p)), \ \forall\ p \in P(q^*),\ q^* \in N_0^* .$$

We segment the rows of the block row (A_i, C_i) into groups of rows with the same η^*-value

$$(A_i, C_i) = \begin{pmatrix} A(q_1^*) & C(q_1^*) \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ A(q_n^*) & C(q_n^*) \end{pmatrix}, \quad n = |N_0^*| .$$

For clarity the index i was suppressed on the right side.

For any $q^* \in N_0^*$ the corresponding node q_η of T represents an open square Q_η with boundary lines $T_{\pm 1,0}, T_{0,\pm 1}$ and corners $P_{\pm 1,\pm 1}$, as shown in Figure 3 for $Q = Q_\eta$. Let $t_{\pm 1,0}$, $t_{0,\pm 1}$ and $p_{\pm 1,\pm 1}$ be the corresponding nodes of T (if they exist). We denote by $\vartheta(t_{\pm 1,0})$ and $\vartheta(t_{0,\pm 1})$ the sets of nodes of the subtrees of T rooted at the indicated nodes. If the particular node does not exist, the corresponding set is empty. All (potentially) nonzero entries in the upper triangular part of the block row $(A(q_\eta^*), C(q_\eta^*))$ then consist of a $|P(q_\eta^*)|$-dimensional upper triangular matrix and off-diagonal blocks as shown in Figure 10.

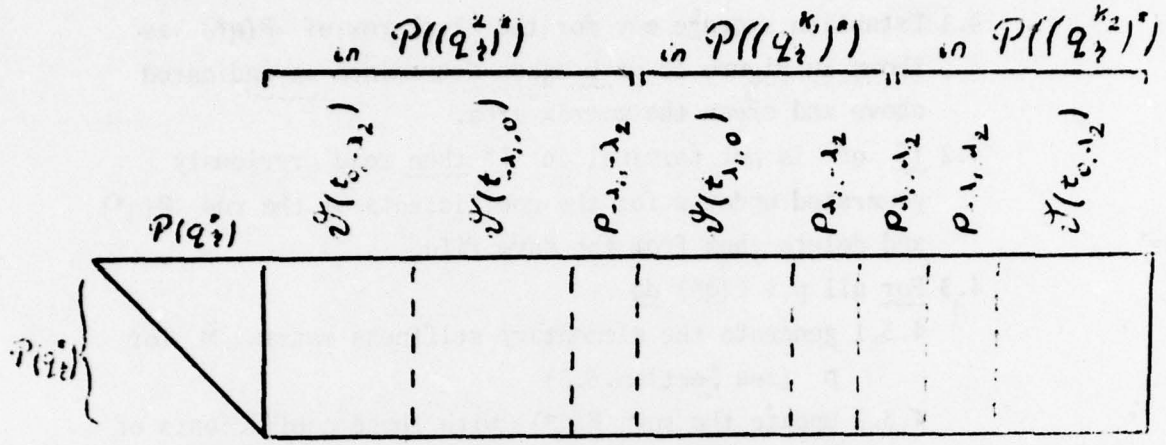


Figure 10

Here (λ_1,λ_2) is the label of q_η, and the nodes p^2, p^{k_1}, and p^{k_2} are defined by the up-path algorithm for q_η (see Figure 4). The node p_{λ_1,λ_2} belongs to the set $P((q_\eta^\kappa)^*)$ for $\kappa = \min(k_1,k_2)$. Under the ordering (3.11) the nodes of any set $\vartheta(t) \neq \emptyset$ are always consecutively numbered from some index ℓ_0 to $\ell_1 = \ell_0 - 1 + |\vartheta(t)|$. Here ℓ_0 is carried by the left-most, lowest descendant of t and ℓ_1 by t itself. This allows for a simple determination of the size of the particular matrix block.

The algorithm for assembling and decomposing the block row (A_i,C_i) now has the following schematic form:

1. Clear save file.
2. Establish the nodes of T^* and their ordering η^*.
3. For each $q^* \in N_0^*$ establish the numbering $\eta_{q^*}(p)$ of the nodes of $P(q^*)$.

4. For $q^* \in N_0^*$ in order of η^* do

   4.1 Establish storage map for the block row of $P(q^*)$ as shown in Figure 10 with sizes determined as indicated above and clear the matrix area.

   4.2 If q^* is not terminal in T^* then read previously generated updates for the coefficients of the row $P(q^*)$ and delete them from the save file.

   4.3 For all $p \in E(q^*)$ do

      4.3.1 generate the elementary stiffness matrix $\tilde{M}$ for p (see Section 3.3)

      4.3.2 update the row $P(q^*)$ with those coefficients of $\tilde{M}$ for which at least one of the two index nodes belongs to $P(q^*)$

      4.3.3 add the coefficients of $\tilde{M}$ to the save file for which both index nodes are in $P((p^{k_2})^*) \cup P((p^{k_1})^*) \cup P((p^{k_2})^*)$

   4.4 Decompose matrix row of $P(q^*)$ and during the decomposition add updates to the save file corresponding to coefficients in the rows of $P((p^{k_2})^*)$, $P((p^{k_1})^*)$, and $P((p^{k_2})^*)$.

   4.5 Output decomposed row for $P(q^*)$.

In the save file (in secondary memory) we retain triplets consisting of a value and two index nodes which represent needed updates for matrix rows to be decomposed later. At the end the save file contains the updates for the matrix B of (3.9)(ii). In step 3 the nodes $P(q^*_{-2}), P(q^*_{-1})$ are not numbered. We assume here that this numbering is done outside of this algorithm; it depends on the given domain Ω and the boundary conditions. The storage map for the row of $P(q^*)$ establishes the (one-to-one) correspondence between the index nodes of the rows and columns indicated in Figure 10 and the customary indexing of the matrix storage used here.

The above algorithm uses the save file for the full segmentation given in (3.13). In practice, several of the block rows may be taken together using subtrees of T^* for the control of the loop of step 4.

4. Outlook

The general tree structure introduced here is not restricted to the particular meshes under discussion. For example, an extension to analogous meshes on the closure $\bar{Q}_0$ of the d-dimensional open unit cube

$$Q_0 = \{x \in R^d \mid 0 < x_i < 1,\ i = 1,\ldots,d\},\ d \geq 1$$

is rather straightforward. Here we consider subsets $E \subset B = \{e^1,\ldots,e^d\}$ of the set of standard basis vectors of R^d and define the $|E|$-dimensional open square Q with center u and side length $h > 0$ by

$$Q = sq(u,h,E) = \{x \in R^d \mid \|x-u\|_\infty < \tfrac{1}{2}h,\ x^T e^i = u_i,\ \forall\ e^i \notin E\}\ .$$

If now the definition (2.4) of the "label" set A is changed to

$$A = \{a \in R^d \mid a_i = -1,0,+1,\ i = 1,\ldots,d\},$$

then with the same notations (2.5),(2.6) as before the formulas (2.7) and (2.8) for the faces of Q and for the subdivision operator σ, respectively, remain exactly the same. With this, the tree can be defined as before and the labelling rule carries over verbatim. Thus also the up-path and down-path algorithms extend to this case.

With this generalization it becomes possible to combine trees for meshes of different dimensionality. For the practical implementation of finite element computations, the trees may again be truncated. However, in higher dimensions the irregularity problem becomes more complex since not only points but also faces may be irregular. Thus, the corresponding

truncation of the tree and the interpolation algorithms need further analysis. The matrix assembly and decomposition algorithm, however, does not change in principle.

The tree structure can also be defined for meshes on equilateral triangles of the form of Figure 2 (without the irregular lines). Basically we need to change the basis vectors in the set B and take into account that now the center point and side length only define the triangle up to a rotation of size π. Clearly, with this an extension to meshes on higher dimensional simplices is also possible.

5. References

1. I. Babuška, W. Rheinboldt, Error estimates for adaptive finite element computations, University of Maryland, Institute for Physical Science & Technology, Tech. Note BN-854, May 1977; SIAM J. Num. Anal. 15, 4, August 1978 (*in press*).

2. I. Babuška, W. Rheinboldt, Computational aspects of finite element analysis, in Mathematical Software III, ed. J. R. Rice, Academic Press, New York, 1977, 225-255.

3. I. Babuška, W. Rheinboldt, A-posteriori error estimates for the finite element method, University of Maryland, Computer Science Technical Report TR-581, Sept. 1977; Int. J. Numer. Meth. Eng., 1978 (in press).

4. I. Babuška, W. Rheinboldt, Analysis of optimal finite element meshes in R^1, University of Maryland, Institute for Physical Science & Technology, Tech. Note BN-869, March 1978.

5. I. Babuška, W. Rheinboldt, On the reliability and optimality of the finite element method, University of Maryland, Computer Science Technical Report TR-643, April 1978.

6. P. Zave, W. Rheinboldt, Design of an adaptive parallel finite element system, University of Maryland, Computer Science Technical Report TR-593, Nov. 1977; Trans. Math. Software (to appear).

7. A. George, Nested dissection of a regular finite element mesh, SIAM J. Num. Anal. 10, 1973, 345-363.

8. D. E. Knuth, The Art of Computer Programming, Vol. 1/Fundamental Algorithms, Addison-Wesley Publ. Co., 1968.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: ONR-N00014-77-C-0623-660	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): ON A DATA STRUCTURE FOR ADAPTIVE FINITE ELEMENT MESH REFINEMENTS	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
6. PERFORMING ORG. REPORT NUMBER: TR-660	
7. AUTHOR(s): Werner C. Rheinboldt, Charles K. Mesztenyi	8. CONTRACT OR GRANT NUMBER(s): N00014-77-C-0623, NSF-MCS72-03721
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Center, University of Maryland, College Park, MD 20742	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Mathematics Branch, Office of Naval Research, Arlington, VA 22217	12. REPORT DATE: June 1978
	13. NUMBER OF PAGES: 41 (12) 42 p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES Abstract (continued)

...dissection techniques. An outlook to various possible extensions of the structure is also given.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

finite elements
mesh refinement
tree structure
access algorithms

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A general labelled tree structure is introduced for a class of nonuniform two-dimensional finite element meshes. The theoretical basis of the structure and the fundamental access algorithms on the tree are presented in a manner which lends itself to extensions to higher dimensions. For use in finite element computations, the tree is truncated considerably and then the principal, relevant algorithms are discussed, including the refinement of the mesh, the computation of the elemental stiffness matrices, and the assembly and decomposition of the global stiffness matrices based on nested...

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

403 018

CL