Electronic Transactions on Numerical Analysis.
Volume 22, pp. 41-70, 2006.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

A NETWORK PROGRAMMING APPROACH IN SOLVING DARCY'S EQUATIONS BY MIXED FINITE-ELEMENT METHODS*

M. ARIOLI† AND G. MANZINI‡

Abstract. We use the null space algorithm approach to solve the augmented systems produced by the mixed finite-element approximation of Darcy's laws. Taking into account the properties of the graph representing the triangulation, we adapt the null space technique proposed in [5], where an iterative-direct hybrid method is described. In particular, we use network programming techniques to identify the renumbering of the triangles and the edges, which enables us to compute the null space without floating-point operations. Moreover, we extensively take advantage of the graph properties to build efficient preconditioners for the iterative algorithm. Finally, we present the results of several numerical tests.

Key words. augmented systems, sparse matrices, mixed finite-element, graph theory

AMS subject classifications. 65F05, 65F10, 64F25, 65F50, 65G05

1. Introduction.

The approximation of Darcy's Laws by Mixed Finite-Element techniques produces a finite-dimensional version of the continuous problem which is described by an augmented system. In this paper, we present an analysis of a null space method which uses a mixture of direct and iterative solvers applied to the solution of this special augmented system. The properties of this method, in the general case, have been studied in [5] where its backward stability is proved, when using finite-precision arithmetic, and where a review of the bibliography on the topic is also presented. Here, we will take advantage of network programming techniques for the design of a fast algorithm for the direct solver part and for the building of effective preconditioners. The relationship between the graph properties of the mesh and the augmented system has been pointed out in [2]. Several authors used similar data structures and network techniques in a rather different context or for different purposes. In [1, 10, 28] similar techniques have been suggested in the area of computational electromagnetics for gauging vector potential formulations. In the field of computational fluid dynamics, analogous methods have been applied to the finite-difference method for the solution of Navier-Stokes equations [3, 24]. Finally, in [6], a similar approach in the approximation of a 3-D Darcy's Law by Hybrid Finite-Element techniques is studied.

The null space algorithm is a popular approach for the solution of augmented systems in the field of numerical optimization but is not widely used in fluid dynamics. For a review of other existing methods for the solution of saddle point problems we advise to read the comprehensive survey [8]. Among the possible alternative methods, we indicate the direct approach where a sparse LDL^T decomposition of the symmetric augmented matrix is computed using preprocessing that will help to minimize the fill-in during the factorization combined with one by one and two by two numerical pivot strategies [18, 17]. In our numerical experiments we will compare our approach with one of these direct solvers.

We point out that our null space method is an algebraic approach to the computation of the finite-element approximation of $H(\mathrm{curl})$ which characterizes the subspace of the divergence-free vector fields in $H(\mathrm{div})$, [32, 7, 34, 35]. Nevertheless, we emphasize that

*Received January 12, 2005. Accepted for publication September 5, 2005. Recommended by M. Benzi. The work of the first author was supported in part by EPSRC grants GR/R46641/01 and GR/S42170. The work of second author was supported by EPSRC grant GR/R46427/01 and partially by the CNR Short-Term Mobility Programme, 2005.

†Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, OX11 0QX, UK (M.Arioli@rl.ac.uk).

‡Istituto di Matematica Applicata e Tecnologia Informatica C.N.R., via Ferrata 1, 27100 Pavia, Italy (marco.manzini@imati.cnr.it).

our method does not require the explicit storage of the null space and, therefore, of the related finite-element approximation of $H(\mathrm{curl})$. Our approach is applicable when RT_0 and BDM_1 finite elements [11] are used. Nevertheless, we do not consider this a limitation in practical situations: the RT_0 and BDM_1 finite elements are widely used in the 3D simulation of physical phenomena, where higher order approximations have an exceeding computational complexity and the indetermination in the evaluations of the physical parameters is high. For the sake of simplicity, we describe our approach only for the RT_0 finite elements.

In Section 2, we will briefly summarize the approximation process and describe the basic properties of the linear system and augmented matrix. In Section 3, the null space algorithm and its algebraic properties are presented. The direct solver is based on the LU factorization of the submatrix of the augmented system which approximates the divergence operator div. We will see in Section 4, how the basic structures of the matrices involved are described in terms of graph theory and how the LU decomposition can be performed by Network Programming classical algorithms. In particular, we will use the "Shortest Path Tree" (SPT) algorithms to achieve a reliable fast decomposition. Furthermore, the same graph properties allow us to describe the block structure of the projected Hessian matrix on which we will apply the conjugate gradient algorithm. This will be used in Section 5 to develop the preconditioners. Finally, in Section 6, we show the results of the numerical tests that we conducted on selected experiments, in Section 7, we describe the possible extension of our techniques to the three dimensional domain case, and we give our conclusions in Section 8.

2. The analytical problem and its approximation.

2.1. Darcy's Law. We consider a simply connected bounded polygonal domain Ω in $\mathbf{R}^2$ which is defined by a closed one-dimensional curve Γ. The boundary Γ is the union of the two distinct parts Γ_D and Γ_N, i.e. $\Gamma = \Gamma_D \cup \Gamma_N$, where either Dirichlet- or Neumann-type boundary conditions are imposed. In the domain Ω, we formulate the mathematical model that relates the pressure field p (the hydraulic head) and the velocity field $\mathbf{v}$ (the visible effect) in a fully saturated soil with an incompressible soil matrix. This relationship is given by the steady Darcy's model equations; the assumption of soil incompressibility implies that the soil matrix characteristics, e.g. density, texture, specific storage, etc, be independent of time, space, and pressure. The Darcy's model equations read as

$$\mathbf{v} = -\mathcal{K}\,\mathrm{grad}\ p, \qquad \text{in } \Omega \tag{2.1}$$

$$\mathrm{div}\,\mathbf{v} = f, \qquad \text{in } \Omega \tag{2.2}$$

Equation (2.1) relates the vector field $\mathbf{v}$ to the scalar field p throughout the permeability tensor $\mathcal{K}$, which accounts for the soil characteristics. Equation (2.2) relates the divergence of $\mathbf{v}$ to the right-hand side source-sink term f. These model equations are supplemented by the following set of boundary conditions for $\mathbf{v}$ and p:

$$p|_{\Gamma_D} = g_D, \qquad \text{on } \Gamma_D \tag{2.3}$$

$$\mathbf{v}\cdot\mathbf{n}|_{\Gamma_N} = g_N, \qquad \text{on } \Gamma_N \tag{2.4}$$

where $\mathbf{n}$ is the unit vector orthogonal to the boundary Γ and pointing out of Ω, g_D and g_N are two regular functions that take into account Dirichlet and Neumann conditions, respectively. For simplicity of exposition, g_N is taken equal to zero.

2.2. Mixed finite-element method for Darcy's law. In this section, we shortly review some basic ideas underlying the mixed finite-element approach that is used in this work to approximate the model equations (2.1)-(2.2). The mixed weak formulation is formally obtained

ETNA
Kent State University
etna@mcs.kent.edu

in a standard way by multiplying equation (2.1) by the test functions

$$\mathbf{w} \in \mathcal{V} = \left\{ \mathbf{q} \,|\, \mathbf{q} \in \left(L^2(\Omega)\right)^2 ,\ \operatorname{div} \mathbf{q} \in L^2(\Omega),\ \mathbf{q}\cdot\mathbf{n}|_{\Gamma_N} = 0 \right\} ,$$

and equation (2.2) by $\phi \in L^2(\Omega)$ and integrating by parts over the domain of computation. We refer to [11] for the definition and the properties of the functional space $\mathcal{V}$. The weak formulation of (2.1)-(2.2) reads as

find $\mathbf{v} \in \mathcal{V}$ *and* $p \in L^2(\Omega)$ *such that:*

$$\begin{cases} \displaystyle\int_\Omega \mathcal{K}^{-1}\mathbf{v}\cdot\mathbf{w}\,d\mathbf{x} - \int_\Omega p \operatorname{div}\mathbf{w}\,d\mathbf{x} = -\int_{\Gamma_D} g_D \mathbf{w}\cdot\mathbf{n}\,ds & \text{for every } \mathbf{w} \in \mathcal{V}, \\ \displaystyle\int_\Omega (\operatorname{div}\mathbf{v})\phi\,d\mathbf{x} = \int_\Omega f\phi\,d\mathbf{x} & \text{for every } \phi \in L^2(\Omega). \end{cases}$$

The discrete counterpart of the weak formulation that we consider in this work can be introduced in the following steps. First, we consider a family of conforming triangulations covering the computational domain Ω that are formed by the sets of disjoint triangles $\mathfrak{T}_\mathfrak{h} = \{T\}$. Every family of triangulations is *regular* in the sense of Ciarlet, [13, page 132], i.e. the triangles do not degenerate in the approximation process for $\mathfrak{h}$ tending to zero. Additionally, we require that no triangle in $\mathfrak{T}_\mathfrak{h}$ can have more than one edge on the boundary Γ nor that a triangle can exist with a vertex on Γ_D and any other vertex on Γ_N. The label $\mathfrak{h}$, which denotes a particular triangulation of this family, is the maximum diameter of the triangles in $\mathfrak{T}_\mathfrak{h}$, i.e. $\mathfrak{h} = \max_{T\in\mathfrak{T}_\mathfrak{h}} \operatorname{diam}(T)$. Then, we consider the functional spaces

$$V_\mathfrak{h} = \left\{ \mathbf{w}(\mathbf{x}) : \Omega \to \mathbf{R}^2,\ \mathbf{w}(\mathbf{x})|_T = \alpha\mathbf{x} + \mathbf{z},\ \alpha \in \mathbf{R},\ \forall T \in \mathfrak{T}_\mathfrak{h},\ \mathbf{w}\cdot\mathbf{n}|_{\Gamma_N} = 0 \right\} ,$$

i.e. the space of the lowest-order Raviart-Thomas vector fields defined on Ω by using $\mathfrak{T}_\mathfrak{h}$, and

$$Q_\mathfrak{h} = \left\{ \phi(\mathbf{x}) : \Omega \to \mathbf{R},\ \phi(\mathbf{x})|_T = \text{const},\ \forall T \in \mathfrak{T}_\mathfrak{h} \right\} ,$$

i.e. the space of the piecewise constant functions defined on Ω by using $\mathfrak{T}_\mathfrak{h}$. For any given $\mathfrak{h}$, the two functional spaces $V_\mathfrak{h}$ and $Q_\mathfrak{h}$ are finite-dimensional subspaces of $\mathcal{V}$ and $L^2(\Omega)$, respectively, and are dense within these latter spaces for $\mathfrak{h} \to 0$ [11]. The discrete weak formulation results from substituting the velocity and pressure field $\mathbf{v}$ and p by their discretized counterparts $\mathbf{v}_\mathfrak{h}$ and $p_\mathfrak{h}$ and reformulating the weak problem in the spaces $V_\mathfrak{h}$ and $Q_\mathfrak{h}$ [11].

The dimension of the functional space $V_\mathfrak{h}$ is equal to the total number of internal and Dirichlet boundary edges N_e. Any element of $V_\mathfrak{h}$ can be expressed as a linear combination of the basis functions $\{\mathbf{w}_{e_j}, \text{for } j = 1, \dots, N_e\}$, where e_j may be an internal edge or a boundary edge with a Dirichlet condition. The basis function $\mathbf{w}_{e_j}$, which is associated to the edge e_j, is uniquely defined by

$$\int_{e_i} \mathbf{w}_{e_j}\cdot\mathbf{n}_{e_i}\,ds = \begin{cases} 1, & \text{for } i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{2.5}$$

where $\mathbf{n}_{e_j}$ is the unit vector with direction orthogonal to e_j and arbitrarly chosen orientation. Likewise, the dimension of the functional space $Q_\mathfrak{h}$ is equal to the number of triangles N_T. In fact, any element of $Q_\mathfrak{h}$ can be expressed as a linear combination of the basis functions $\{\phi_j, \text{for } j = 1, \dots, N_T\}$. The basis function ϕ_j, which is associated to the triangle T_j, is such that $\phi_j = 1$ on T_j and $\phi_j = 0$ on $\Omega \setminus T_j$.

The solution fields $\mathbf{v}_\mathfrak{h}$ and $p_\mathfrak{h}$ of the discretized weak formulation are taken as approximation of the solution fields $\mathbf{v}$ and p of the original weak formulation. The solution fields $\mathbf{v}_\mathfrak{h}$

ETNA
Kent State University
etna@mcs.kent.edu

and $p_{\mathfrak{h}}$ are expressed as linear combinations of the basis functions introduced in the previous paragraph. These expansions are formally given by

$$p_{\mathfrak{h}} = \sum_{j=1}^{N_T} p_j \phi_j, \qquad \text{and} \qquad \mathbf{v}_{\mathfrak{h}} = \sum_{j=1}^{N_e} u_j \mathbf{w}_{e_j}. \tag{2.6}$$

The coefficient p_j of the basis function ϕ_j in (2.6) can be interpreted as the approximation of the cell average of the pressure field p on T_j. The coefficient u_j of the basis function $\mathbf{w}_{e_j}$ in (2.6) can be interpreted as the approximation of the flux, or the 0^{th}-order momentum, of the normal component of $\mathbf{v}$ on the edge e_j. We collect these coefficients in the two algebraic vectors $u = \{u_j\}$ and $p = \{p_j\}$. It turns out that these vectors are solving the augmented system of linear equations

$$\begin{bmatrix} M & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}, \tag{2.7}$$

where the elements of the matrices M and A of the augmented matrix of the left-hand side of (2.7) and the elements of the vectors q and b of the right-hand side of (2.7) are given by

$$(M)_{e_i e_k} = \int_\Omega \mathcal{K}^{-1} \mathbf{w}_{e_i} \cdot \mathbf{w}_{e_k} \, d\mathbf{x}\,, \tag{2.8}$$

$$(A)_{e_i j} = -\int_\Omega \operatorname{div} \mathbf{w}_{e_i} \phi_j \, d\mathbf{x}\,, \tag{2.9}$$

$$(q)_{e_i} = \int_\Gamma g_D \mathbf{w}_{e_i} \cdot \mathbf{n} \, ds\,, \tag{2.10}$$

$$(b)_j = -\int_\Omega f \phi_j \, d\mathbf{x}\,. \tag{2.11}$$

The augmented system (2.7) has a unique solution because it holds that

$$\operatorname{Ker}(A^T) \cap \operatorname{Ker}(M) = \{0\}.$$

This property is an immediate consequence of the *inf-sup* condition, which is satisfied by the functional operators represented by (2.8) and (2.9) on the basis function sets $\{\mathbf{w}e_j\}$ and $\{\phi_j\}$. These functional operators are used to define the discrete weak formulation in the functional spaces $V_{\mathfrak{h}}$ and $Q_{\mathfrak{h}}$ (see [11] for more details).

Finally, we describe some major properties of the matrices M and A that are related to the triangulation $\mathfrak{T}_{\mathfrak{h}}$. First, we assume that e_i be an internal edge, and denote its two adjacent triangles by $T'_{e_i} \subset \Omega$ and $T''_{e_i} \subset \Omega$. As the support of the basis function $\mathbf{w}_{e_i}$ is given by the union of T'_{e_i} and T''_{e_i}, every row $M_{e_i \bullet}$ of the matrix M has at most 5 nonzero entries. These nonzero entries are on the columns corresponding to the internal or Dirichlet boundary edges of the triangles T'_{e_i} and T''_{e_i} (the Neumann boundary edges are not considered). Similarly, the row $A_{e_i \bullet}$ of the matrix A must have two different nonzero entries on the columns corresponding to the two triangles T'_{e_i} and T''_{e_i}. Then, we assume that e_i be a Dirichlet boundary edge, and denote the unique trianglethat e_i belongs to, by T'_{e_i}. In this second case, the row $M_{e_i \bullet}$ has surely three nonzero entries on the columns corresponding to the three edges of T'_{e_i}. From our initial assumption on the triangulations, it follows that the triangle T'_{e_i} cannot have more than one edge on the boundary Γ; furthermore, we are actually assuming that this boundary edge is of Dirichlet type. Therefore, the row $A_{e_i \bullet}$ has one nonzero entry only on the column corresponding to T'_{e_i}. Moreover, the rectangular matrix A has maximum rank. By applying

ETNA
Kent State University
etna@mcs.kent.edu

the Gauss theorem to the integral in (2.9) restricted to T_j (where $\phi_j = 1$) and taking into consideration (2.5), we deduce that the nonzero entries of the matrix A must be either $+1$ or -1. The sign depends on the relative orientation of the triangle edges with respect to the edge orientation that was arbitrarily chosen in (2.5). Furthermore, for every internal edge e_i shared by the triangles T'_{e_i} and T''_{e_i}, the sum of the two nonzero entries on the matrix row $A_{e_i \bullet}$ is zero; formally, $A_{e_i T'_{e_i}} + A_{e_i T''_{e_i}} = 0$. Let us consider the matrix $A' = [A|r]$ that is obtained by augmenting A with the column vector r which is built as follows. The column vector r only has nonzero entries for the row indices corresponding to the edges $e_i \subset \Gamma_D$ and these non-zero entries are such that $r_{e_i} + A_{e_i\, T_{e_i}} = 0$. The matrix A' is the *incidence matrix* of the edge-triangle graph underlying the given triangulation, and, then [31, 12] the matrix A' is a *totally unimodular* matrix and has rank N_T. Therefore, the matrix A has rank N_T because every submatrix that is obtained by removing a column of a totally unimodular matrix has full rank [31, 12]. Furthermore, the pattern of M is equal to the pattern of $|A||A|^T$, because the nonzero entries of the row $M_{e_i \bullet}$ must match the nonzero entries of the edges of the triangles T'_{e_i} and T''_{e_i}, and from the unimodularity properties of A. By construction, the matrix M is symmetric and positive semidefinite.

3. Null space algorithms.

In this section, we take into account the classical null space algorithm for the minimization of linearly constrained quadratic forms, which is described for example in [21]. In particular, we choose the formulation of this algorithm that is based on the factorization of the matrix A.

In order to simplify the notations, we use the letter n instead of the symbol N_e to indicate the total number of internal edges and Dirichlet boundary edges of the mesh, and m instead of N_T to indicate the total number of triangles of the mesh. It holds that $n \geq m$. Finally, we denote by E_1 and E_2 the $n \times m$ and $n \times (n-m)$ matrices

$$E_1 = \begin{array}{c} \left[\begin{array}{c} I_m \\ 0 \end{array} \right] \end{array} \begin{array}{l} \}\, m \\ \}\, n-m \end{array}, \qquad \text{and} \qquad E_2 = \begin{array}{c} \left[\begin{array}{c} 0 \\ I_{n-m} \end{array} \right] \end{array} \begin{array}{l} \}\, m \\ \}\, n-m \end{array}$$

where, I_m and I_{n-m} are respectively the identity matrix of order m and the identity matrix of order $n-m$.

As already stated in Section 2.2, A is a full rank submatrix of a totally unimodular matrix of order $n \times (m+1)$, and its entries are either ± 1 or 0. It turns out the "LU" factorization of A is obtainable without any floating-point operations, throughout a couple of suitable permutation matrices P and Q (see [31, 12]). In Section 4, we will see how it is possible to determine efficiently P and Q by using network programming algorithms. The matrices P and Q allow us to write the permutation of the matrix A as

$$PAQ = \left[\begin{array}{c} L_1 \\ L_2 \end{array} \right] = \left[\begin{array}{cc} L_1 & 0 \\ L_2 & I_{n-m} \end{array} \right] E_1, \tag{3.1}$$

where L_1 is a nonsingular lower triangular matrix of order m. Without loss of generality, we assume that all the diagonal entries in L_1 are equal to 1 and the non-zero entries below the diagonal are equal to -1. This choice simply corresponds to a symmetric diagonal scaling of the permuted augmented system. By introducing the lower triangular matrix

$$L = \left[\begin{array}{cc} L_1 & 0 \\ L_2 & I_{n-m} \end{array} \right], \tag{3.2}$$

and recognizing that E_1 plays the role of the matrix "U", the "LU" factorization of A is given by

$$PAQ = LE_1.$$

ETNA
Kent State University
etna@mcs.kent.edu

In the rest of this section, we assume, for the sake of exposition simplicitythat the matrices M and A and the vectors q and b have already been consistently permuted and omit to indicate explicitly the permutation matrices P and Q. Thus, by identifying the matrix A with LE_1 after the row and column permutations, the augmented matrix in (2.7) can be factorized as follows

$$\begin{bmatrix} M & A \\ A^T & 0 \end{bmatrix} = \begin{bmatrix} L & 0 \\ 0 & I_m \end{bmatrix} \begin{bmatrix} L^{-1}ML^{-T} & E_1 \\ E_1^T & 0 \end{bmatrix} \begin{bmatrix} L^T & 0 \\ 0 & I_m \end{bmatrix}. \tag{3.3}$$

We indicate the matrix $L^{-1}ML^{-T}$ by the symbol $\tilde{M}$. The inverse matrix and the transposed inverse matrix of L are formally given by

$$L^{-1} = \begin{bmatrix} L_1^{-1} & 0 \\ L_2 L_1^{-1} & I_{n-m} \end{bmatrix} \quad \text{and} \quad L^{-T} = \begin{bmatrix} L_1^{-T} & -L_1^{-T} L_2^T \\ 0 & I_{n-m} \end{bmatrix}. \tag{3.4}$$

The block-partitioning of L^{-1} and L^{-T} induces on $\tilde{M}$ the block-partitioned form

$$\tilde{M} = \begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} \\ \tilde{M}_{12}^T & \tilde{M}_{22} \end{bmatrix}, \tag{3.5}$$

where, formally, $\tilde{M}_{ij} = E_i^T \tilde{M} E_j$, for $i, j = 1, 2$, and we exploited the symmetry of the matrix M (and, of course of $\tilde{M}$) to set $\tilde{M}_{21} = \tilde{M}_{12}^T$. Similarly, we introduce the block partition $(u_1, u_2)^T$ of the velocity vector u and denote, for consistency of notation, the pressure vector p by u_3. Thus, the algebraic vector $(u, p)^T = (u_1, u_2, u_3)^T$ denotes the solution of the (suitably permuted) linear problem (2.7) discussed in Section 2.2. We use the decomposition (3.3) and take into account the block-partitioned definitions (3.4) and (3.5) of the matrices L^{-1}, L^{-T} and $\tilde{M}$ to split the resolution of the linear problem (2.7) in the two following steps. First, we solve the linear system

$$\begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} & I_m \\ \tilde{M}_{12}^T & \tilde{M}_{22} & 0 \\ I_m & 0 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} L^{-1} & 0 \\ 0 & I_m \end{bmatrix} \begin{bmatrix} -q \\ b \end{bmatrix}, \tag{3.6}$$

for the auxiliary unknown vector $(z_1, z_2, z_3)^T$ and, then, we compute the unknown vector $(u_1, u_2, u_3)^T$ by solving

$$\begin{bmatrix} L_1^T & L_2^T & 0 \\ 0 & I_{n-m} & 0 \\ 0 & 0 & I_m \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}. \tag{3.7}$$

Note that the left-hand side matrix of (3.6) can be put in a block-triangular form by interchanging the first and the third block of equations. The final computational algorithm, which is solved by the null space strategy, is formulated by introducing the vector $h = -L^{-1}q$. The vector h is consistently partitioned in the two subvectors $h_1 = E_1^T h$, and $h_2 = E_2^T h$ according to the block-partitioning (3.5).

NULL SPACE ALGORITHM:

(i) *solve the block lower triangular system:*

$$\begin{bmatrix} I_m & 0 & 0 \\ \tilde{M}_{12}^T & \tilde{M}_{22} & 0 \\ \tilde{M}_{11} & \tilde{M}_{12} & I_m \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} b \\ h_2 \\ h_1 \end{bmatrix},$$

ETNA
Kent State University
etna@mcs.kent.edu

(ii) *and, then, let*

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = L^{-T} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} L_1^{-T}(b - L_2^T z_2) \\ z_2 \end{bmatrix},$$

$$u_3 = z_3.$$

Note that in step (ii) we have $u_2 = z_2$ in view of the second formula in (3.4).

The null space algorithm as formulated above requires the formal inversion of the matrix $\tilde{M}_{22}$, which is the projected Hessian matrix of the quadratic form associated to the augmented matrix of (2.7). In order to solve the linear algebraic problem $\tilde{M}_{22}u_2 = h_2 - \tilde{M}_{12}^T z_1$ for u_2, we may proceed in two different ways. If $n - m$ is small, i.e the number of constraints is very close to the number of unknowns, or $\tilde{M}_{22}$ is a sparse matrix, we may explicitly compute $\tilde{M}_{22}$ and then solve the linear system involving this matrix by using a Cholesky factorization. Nonetheless, the calculation of the product matrix $L^{-1}ML^{-T}$ might be difficult because of the high computational complexity, which is of order $\mathcal{O}(n^3)$, or because the final matrix itself would be fairly dense despite the sparsity of M. In such cases, we prefer to solve the linear system $\tilde{M}_{22}u_2 = h_2 - \tilde{M}_{12}^T z_1$ by using the pre-conditioned conjugate gradient algorithm [23]. The conjugate gradient algorithm does not require the explicit knowledge of the matrix $\tilde{M}_{22}$ but only the capability of performing the product of this matrix times a vector. The product of the block sub-matrix $\tilde{M}_{ij}$ of $\tilde{M}$ times a suitably sized vector y is effectively performed by implementing the formula

$$\tilde{M}_{ij}y = E_i^T L^{-1}(M(L^{-T}E_j y)), \qquad \text{for } i, j = 1, 2. \tag{3.8}$$

We point out that the formula (3.8) only requires the resolution of the two triangular systems with matrices L^{-1} and L^{-T}, which can be easily performed by, respectively, forward and backward substitutions, and the product of the sparse matrix M by an n-sized vector. Furthermore, we observe that the matrix-vector product given by (3.8) is backward stable [5].

If we use the conjugate gradient method to solve $\tilde{M}_{22}u_2 = h_2 - \tilde{M}_{12}^T z_1$, it is quite natural to adopt a stopping criterion which takes advantage of the minimization property of this algorithm. At every step j the conjugate gradient method minimizes the energy norm of the error $\delta u_2 = u_2 - \overline{u}_2^{(j)}$ on a Krylov space between the “exact” solution vector u_2 and the computed j-th iterative approximation $\overline{u}_2^{(j)}$. The energy norm of the vector $y \in \mathbf{R}^{n-m}$, which is induced by the matrix $\tilde{M}_{22}$, is defined by

$$\|y\|_{\tilde{M}_{22}} = (y^T \tilde{M}_{22} y)^{(1/2)}.$$

This norm induces the dual norm

$$\|y'\|_{\tilde{M}_{22}^{-1}} = (y^{',T} \tilde{M}_{22}^{-1} y')^{(1/2)}$$

for the dual space vectors $y' \in \mathbf{R}^{n-m}$. We terminate the conjugate gradient iterations by the following stopping criterion

$$\text{IF} \quad \|h_2 - \tilde{M}_{12}^T z_1 - \tilde{M}_{22}\overline{u}_2\|_{\tilde{M}_{22}^{-1}} \leq \eta \|h_2 - \tilde{M}_{12}^T z_1\|_{\tilde{M}_{22}^{-1}} \qquad \text{THEN STOP}, \tag{3.9}$$

where η is an *a priori* defined threshold with value less than 1. The choice of η clearly depends on the properties of the problem that we want to solve; however, in many practical cases, η can be taken much larger than the machine precision of the floating-point operations. In our experiments, we choose $\eta = \mathfrak{h}$ following the results in [4].

In order to use (3.9), we need some tool for estimating the value $e_{\tilde{M}_{22}} = \|h_2 - \tilde{M}_{12}^T z_1 - \tilde{M}_{22}\overline{u}_2\|^2_{\tilde{M}_{22}^{-1}}$. This goal can be achieved by using the Gauss quadrature rule proposed in [22] or the Hesteness and Stiefel rule [25, 4, 37, 38]. This latter produces a lower bound for the error $e_{\tilde{M}_{22}}$ using the quantities already computed during each step of the conjugate gradient algorithm with a negligible marginal cost. In [37, 38], its numerical stability in finite arithmetic has been proved. All these lower bound estimates can be made reasonably close to the value of $e_{\tilde{M}_{22}}$ at the price of d additional iterations of the conjugate gradient algorithm. In [4, 22], the choice $d = 10$ is indicated as a successful compromise between the computational costs of the additional iterations and the accuracy requirements; several numerical experiments support this conclusion ([4, 22, 5, 37, 38]). Finally, following Reference [5], we estimate

$$\|h_2 - \tilde{M}_{12}^T z_1\|^2_{\tilde{M}_{22}^{-1}}$$

by taking into account that

$$\|h_2 - \tilde{M}_{12}^T z_1\|_{\tilde{M}_{22}^{-1}} = \|\overline{u}_2\|_{Mt_{22}}.$$

and replacing $\overline{u}_2$ with its current evaluation $\overline{u}_2^j$ at the step j if $e_{\tilde{M}_{22}} < \eta^2 \|h_2 - \tilde{M}_{12}^T z_1\|_2^2$.

4. Graph and Network properties. In this section, we first review some basic definitions relative to graph theory (more detailed information can be found in [31, 39]). We also discuss how these definitions are relied to the graph structure underlying the triangulations of the mixed finite element formulation of Section 2.2. Then, we show how a strategy that relies on network programming can be used to determine the permutation matrices P and Q of the "LU" factorization of the matrix A that was introduced in the previous sections. In particular, we show how these two matrices can be constructed in a fast and efficient way by exploting the *Shortest-Path algorithm* (SPT).

A graph $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$ is made up of a set of nodes $\mathcal{N} = \{\tau_i\}_{i=1,\ldots,n_{\mathcal{N}}}$ and a set of arcs $\mathcal{A} = \{\alpha_j\}_{j=1,\ldots,n_{\mathcal{A}}}$. An arc α_j is identified by a pair of nodes τ_i and τ_k; $\{\tau_i, \tau_k\}$ denotes an unordered pair and the corresponding undirected arc, and by $[\tau_i, \tau_k]$ an ordered pair and the corresponding directed arc. Either $\mathcal{G}$ is an undirected graph, in which case all the arcs are undirected, or $\mathcal{G}$ is a directed graph, in which case all the arcs are directed. We can convert every directed graph $\mathcal{G}$ to an undirected one called the *undirected version* of $\mathcal{G}$ by replacing each $[\tau_i, \tau_k]$ by $\{\tau_i, \tau_k\}$ and removing the duplicate arcs. Conversely, we can build the *directed version* of an undirected graph $\mathcal{G}$ by replacing every $\{\tau_i, \tau_k\}$ by the pair of arcs $[\tau_i, \tau_k]$ and $[\tau_k, \tau_i]$. In order to avoid repeating definitions, (τ_k, τ_i) may denote either an undirected arc $\{\tau_i, \tau_k\}$ or a directed arc $[\tau_i, \tau_k]$ and the context resolves the ambiguity.

The nodes τ_i and τ_k in an undirected graph $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$ are *adjacent* if $\{\tau_i, \tau_k\} \in \mathcal{A}$, and we define the *adjacency* of τ_i by

$$Adj_{\tau_i} = \{\tau_j \,:\, \{\tau_i, \tau_j\} \in \mathcal{A}\}.$$

Analogously, in a directed graph $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$, we define

$$Adj_{\tau_i} = \{\tau_j \,:\, \text{if } [\tau_i, \tau_j] \in \mathcal{A} \text{ or } [\tau_j, \tau_i] \in \mathcal{A}\}.$$

We define the adjacency of an arc as follows:

$$Adj_{\{\tau_i,\tau_j\}} = \{\{\tau_i, \tau_k\} \in \mathcal{A}\} \cup \{\{\tau_j, \tau_k\} \in \mathcal{A}\}$$

ETNA
Kent State University
etna@mcs.kent.edu

for an undirected graph, and

$$Adj_{[\tau_i,\tau_j]} = \{[\tau_i,\tau_k] \in \mathcal{A}\,,\, [\tau_k,\tau_i] \in \mathcal{A}\} \cup \{[\tau_j,\tau_k] \in \mathcal{A}\,,\, [\tau_k,\tau_j] \in \mathcal{A}\}$$

for a directed graph.

A *path* in a graph from node τ_j to node τ_k is a list of nodes $[\tau_j = \tau_{j_1}, \tau_{j_2}, \ldots, \tau_{j_k} = \tau_k]$, such that $(\tau_{j_i}, \tau_{j_{i+1}})$ is an arc in the graph $\mathcal{G}$ for $i = 1, \ldots, k-1$. The path *does contain* the nodes τ_i for $i \in [1, \ldots, k]$ and arcs (τ_i, τ_{i+1}) for $i \in [1, \ldots, k-1]$ and *does not contain* any other nodes and arcs. The nodes τ_j and τ_k are the *ends* of the path. The path is *simple* if all of its nodes are distinct. The path is a *cycle* if $k > 1$ and $\tau_j = \tau_k$ and a *simple cycle* if all of its nodes are distinct. A graph without cycles is *acyclic*. If there is a path from node τ_w to node τ_v then τ_v is *reachable* from τ_w.

An undirected graph is *connected* if every node of its undirected version is reachable from every other node and *disconnected* otherwise. The maximal connected subgraphs of $\mathcal{G}$ are its *connected components*.

A *rooted tree* is an undirected graph that is connected and acyclic with a distinguished node τ_R, called *root*. A rooted tree with k nodes contains $k-1$ arcs and has a unique simple path from any node to any other one. When appropriate we shall regard the arcs of a rooted tree as directed. A *spanning rooted tree* $\mathcal{T}_R = \{\mathcal{N}, \mathcal{A}_{\mathcal{T}_R}\}$ in $\mathcal{G}$ is a rooted tree which is a subgraph of $\mathcal{G}$ with $n_{\mathcal{N}}$ nodes. If τ_v and τ_w are nodes in $\mathcal{T}_R$ and τ_v is in the path from τ_R to τ_w with $\tau_w \neq \tau_v$ then τ_v is an *ancestor* of τ_w and τ_w is a *descendant* of τ_v. Moreover, if τ_v and τ_w are adjacent then τ_v is the *parent* of τ_w and τ_w is a *child* of τ_v. Every node, except the root, has only one parent, which will be denoted by *parent*(τ_v), and, moreover, it has zero or more children. A node with zero children is a *leaf*. We will call the arcs in $\mathcal{A}_{\mathcal{T}_R}$ *in-tree* and the arcs in $\mathcal{A} \setminus \mathcal{A}_{\mathcal{T}_R}$ *out-of-tree*. A *forest* is a node-disjoint collection of trees.

The *depth*(τ_i) of the node τ_i in a rooted tree is the (integer) top-down distance from the root node that is recursively defined by

$$\mathit{depth}(\tau_i) = \begin{cases} 0, & \text{if } \tau_i = \tau_R, \\ \mathit{depth}(\mathit{parent}(\tau_i)) + 1, & \text{otherwise,} \end{cases}$$

and, similarly, the *height*(τ_i) is the (integer) down-top distance of the node τ_i from the deepest leafthat is recursively defined by

$$\mathit{height}(\tau_i) = \begin{cases} 0, & \text{if } \tau_i \text{ is a leaf,} \\ \max\{\mathit{height}(\tau_w) \,:\, \tau_w \text{ is a child of } \tau_i\} + 1, & \text{otherwise.} \end{cases}$$

The *subtree rooted at* node τ_i is the rooted tree consisting of the subgraph induced by the descendants of τ_i and having root in τ_i. The *nearest common ancestor* of two nodes is the deepest node that is an ancestor of both. A node τ_i is a *branching node* if the number of its children is greater than or equal to two. For each out-of-tree arc $[\tau_j, \tau_k]$, the *cycle of minimal length* or the *fundamental cycle* is the cycle composed by $[\tau_j, \tau_k]$ and the paths in $\mathcal{T}_R$ from τ_j and τ_k to their nearest common ancestor.

We can associate the graph $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$ to the triangulation $\mathfrak{T}_{\mathfrak{h}}$ as follows. First, we associate a distinct node of the graph to every triangle of the mesh, e.g. τ_i is the (unique) node corresponding to the triangle T_i, for $i = 1, \ldots, N_T$. Then, the (directed or undirected) arc (τ_i, τ_j) exists in the arc set $\mathcal{A}$ if and only if the triangles T_i and T_j share an edge. Furthermore, we add the root node τ_R, which represents the "exterior world" $\mathbf{R}^2 \setminus \Omega$, to the node set $\mathcal{N}$, and the arcs (τ_R, τ_k) for every node τ_k associated to a triangle T_k with a boundary edge of Dirichlet type to the arc set $\mathcal{A}$. The incidence matrix of the graph $\mathcal{G}$ is the $n \times m$ totally unimodular matrix A' that has been introduced at the end of Section 2.2; its rank is $m (= N_T)$.

If we remove the column corresponding to the root from A', we obtain the matrix A of problem (2.7), which also has rank m. Moreover, every spanning tree of $\mathcal{G}$ with root in τ_R induces a partition of the rows of A in in-tree rows and out-of-tree rows. If we renumber the in-tree arcs first and the out-of-tree arcs last, we can permute the in-tree arcs and the nodes such that the permuted matrix A has the form

$$PAQ = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix},$$

where $L_1 \in \mathbf{R}^{m\times m}$ is a lower triangular and non-singular matrix, see Reference [31, 12].

As the matrix A is obtained by simply removing the root column from the totally unimodular matrix A', then, the entries of the matrix L_1^{-1} must also be ± 1 or 0. The non-zeroes of the matrix $L_2 L_1^{-1}$ are also equal to ± 1 and its rows correspond to the out-of-tree arcs. The number of nonzeros of a row of this matrix is equal to the number of arcs in the cycle of minimal length that the corresponding out-of-tree arc forms with the in-tree arcs. We recall (see Section 3) that without loss of generality, all the diagonal entries in L_1 can be chosen equal to 1 and, therefore, the entries outside the diagonal are 0 or -1. This signifies selecting the directions of the arcs in $\mathcal{A}_{\mathcal{T}_R}$ as $[\tau_i, \mathit{parent}(\tau_i)],\ \forall \tau_i$. Given the out-of-tree arc $[\tau_j, \tau_k]$, the values of the nonzero entries in the corresponding row of $L_2 L_1^{-1}$ will be 1 if both the nodes of the in-tree arc corresponding to the nonzero entry are ancestors of τ_k, and will be -1 if both the nodes of the in-tree arc corresponding to the nonzero entry are ancestors of τ_j.

We now give some basic results the proof of which is straightforward.

LEMMA 4.1. *Let τ_v be a branching node with k children. The descendants of τ_v can be partitioned in k sets $\mathcal{N}_1, \dots, \mathcal{N}_k$ such that $\mathcal{N}_i \cap \mathcal{N}_j = \emptyset$, for $i \neq j$. Each $(\tau_i, \tau_j) \in \mathcal{A}$ with $\tau_i \in \mathcal{N}_i$ and $\tau_j \in \mathcal{N}_j$ is an out-of-tree arc.*

If we define the adjacency of a set of nodes $\mathcal{S}_{\mathcal{N}} \subset \mathcal{N}$ and the adjacency of a set of arcs $\mathcal{S}_{\mathcal{A}} \subset \mathcal{A}$ as follows:

$$\mathit{Adj}(\mathcal{S}_{\mathcal{N}}) = \bigcup_{\tau_i \in \mathcal{S}_{\mathcal{N}}} \mathit{Adj}_{\tau_i}, \quad \mathit{Adj}(\mathcal{S}_{\mathcal{A}}) = \bigcup_{(\tau_i,\tau_j) \in \mathcal{S}_{\mathcal{A}}} \mathit{Adj}_{(\tau_i,\tau_j)},$$

we have the following Corollary.

COROLLARY 4.1. *Let τ_v be a branching node with k children and $\mathcal{N}_1, \dots, \mathcal{N}_k$ such that $\mathcal{N}_i \cap \mathcal{N}_j = \emptyset$, for $i \neq j$ be the partitioning of the descendants of τ_v. Let $(\tau_q, \tau_j) \in \mathcal{A}$ and $(\tau_p, \tau_i) \in \mathcal{A}$ be out-of-tree arcs, with $\tau_p, \tau_i \in \mathcal{N}_{\ell_1}$ and $\tau_q, \tau_j \in \mathcal{N}_{\ell_2}$. The minimal length circuits $\mathcal{C}_{(\tau_q,\tau_j)} \subset \mathcal{A}_{\mathcal{T}_R} \cup \{(\tau_q, \tau_j)\}$ and $\mathcal{C}_{(\tau_p,\tau_i)} \subset \mathcal{A}_{\mathcal{T}_R} \cup \{(\tau_p, \tau_i)\}$ are disjoint:*

$$\mathcal{C}_{(\tau_p,\tau_i)} \cap \mathcal{C}_{(\tau_q,\tau_j)} = \emptyset,$$

and

$$\mathcal{C}_{(\tau_p,\tau_i)} \cap \mathit{Adj}(\mathcal{C}_{(\tau_q,\tau_j)}) = \emptyset, \qquad \mathit{Adj}(\mathcal{C}_{(\tau_p,\tau_i)}) \cap \mathcal{C}_{(\tau_q,\tau_j)} = \emptyset.$$

Proof. From Lemma 4.1 the descendants of the children of the branching node τ_v form disjoint subtrees. If the two circuits $\mathcal{C}_{(\tau_q,\tau_j)}$ and $\mathcal{C}_{(\tau_p,\tau_i)}$ had an arc in common, this arc should simultaneously be an in-tree arc of the two subtrees of nodes $\mathcal{N}_{\ell_1}$ and $\mathcal{N}_{\ell_2}$. Thus, this arc would close a a circuit on the ancestor τ_v, but this fact is in contradiction with the definition of a tree (which cannot contain closed circuits).

Similarly, if $\mathcal{C}_{(\tau_p,\tau_i)}$ and $\mathit{Adj}(\mathcal{C}_{(\tau_q,\tau_j)})$ had a common arc, this arc should be an in-tree arc because it would lie on $\mathcal{C}_{(\tau_p,\tau_i)} \cap \mathcal{A}_{\mathcal{T}_R}$. This fact is in contradiction with the results of Lemma 4.1 because this arc would connect two disjoint sets. $\square$

ETNA
Kent State University
etna@mcs.kent.edu

Finally, if the root τ_R of the spanning tree is a branching node with k children τ_i^D , $i = 1, \ldots, k$, the subtrees having τ_i^D as roots form a forest. Therefore, the matrix L_1 can be permuted in block diagonal form with triangular diagonal blocks.

We refer to [31, 39] for surveys of different algorithms for computing spanning trees. An optimal choice for the rooted spanning tree is the one minimizing the number of nonzero entries in the matrix $Z = L^{-T}E_2$ the columns of which span the null space of A^T. In [16], it is proved that the equivalent problem of finding the tree for which the sum of all the lengths of the fundamental circuits is minimal, is an $\mathcal{NP}$-complete problem. In [9, 14, 15, 16, 20, 27, 33] several algorithms have been proposed which select a rooted spanning tree reducing or minimizing the number of nonzero entries in Z.

In this paper, we propose two different approaches based on the introduction of a function *cost*$(\cdot)$ defined on each arc of the graph and describing some physical properties of the original problem.

From Corollary 4.1 it follows that a rooted spanning tree, having the largest possible number of branching nodes, normally has many disjoint circuits. The columns of Z corresponding to these disjoint circuits are structurally orthogonal, i.e. the scalar product of each pair is zero, independent of the values of the entries.

Moreover, we choose the function *cost* : $\mathcal{A} \to \mathbb{R}_+$ in the following way:

$$\forall \alpha \in \mathcal{A}\,, \qquad \begin{cases} \mathit{cost}(\alpha) = 0 & \text{if } \exists i \text{ such that } \alpha = (\tau_R, \tau_i) \\ \mathit{cost}(\alpha) = M_{\alpha\alpha} & \text{otherwise} \end{cases} \tag{4.1}$$

Using the *cost*$(\cdot)$ function, we can compute the spanning tree rooted in τ_R that is actually solving the SPT problem [19] on the graph. In particular, we have chosen to implement the heap version of the shortest path tree algorithm (see the SHEAP algorithm that is described in Reference [19]).

The resulting spanning tree has the interesting interpretation that the path "circumnavigates" low permeability regions in the sense specified below. In the presence of "islands" of very low permeability in Ω, i.e. regions with very large values for $\mathcal{K}^{-1}$, the paths from the root to a node corresponding to a triangle that lies outside the islands of low permeability will be made of nodes corresponding to triangles that are also outside the islands. In this sense, we can state that the shortest path tree "circumnavigates the islands". Therefore, the set of these paths reasonably identifies the lines where the flux is expected to be greater.

Owing to the fact that we assume a null cost for the arcs connected to the root node τ_R, both strategies provides a forest if the number of zero cost arcs is greater than one. We observe that we do not need to build the matrix A explicitly: the tree (or forest) can be computed by using the graph only. Moreover, the solution of the lower and upper triangular systems can be performed by taking advantage of the *parent* function alone. This strategy results in a very fast and potentially highly parallel algorithm. For a problem with only Dirichlet conditions and an isotropic mesh (i.e. the number of triangles along each direction is independent from the direction), we may have a forest with a number of trees proportional to the number of triangles with a boundary edge. This number is $\mathcal{O}(\mathfrak{h}^{-1})$, and, therefore, the matrix L_1 has $\mathcal{O}(\mathfrak{h}^{-1})$ diagonal blocks of average size $\mathcal{O}(\mathfrak{h}^{-1})$. Finally, we point out that most non-leaf nodes in the SPT have two children.

5. Preconditioning and quotient tree. In the presence of islands of very low permeability in Ω, the values of $\mathcal{K}$ can differ by many orders of magnitude in the different regions of the domain. In this case, the projected Hessian matrix $\tilde{M}_{22}$ has a condition number which is still very high. It is then necessary to speed up the convergence of the conjugate gradient by use of a preconditioner. Obviously, it is not usually practical to explicitly form $\tilde{M}_{22}$ to

ETNA
Kent State University
etna@mcs.kent.edu

compute or identify any preconditioner. In this section, we will show how we can compute the block structure of $\tilde{M}_{22}$ using only the tree and the graph information. We will then use the block structure to compute a preconditioner.

Denoting by H the matrix $L_2L_1^{-1}$, the projected Hessian matrix $\tilde{M}_{22}$ can be written as follows:

$$\tilde{M}_{22} = Z^T M Z = M_{22} + H M_{11} H^T - (M_{21} H^T + H M_{12}). \tag{5.1}$$

The row and column indices of the matrix $\tilde{M}_{22}$ correspond to the out-of-tree arcs. Moreover, we recall that the matrix H has entries $-1, 0, 1$, its row indices correspond to the out-of-tree arcs, and the number of nonzeros in one of its rows will be the number of arcs in the cycle of minimal length which the corresponding out-of-tree arc forms with the in-tree arcs.

LEMMA 5.1. *Let α and β be two out-of-tree arcs, and $\mathcal{C}_\alpha \subset \mathcal{A}_{\mathcal{T}_R} \cup \alpha$ and $\mathcal{C}_\beta \subset \mathcal{A}_{\mathcal{T}_R} \cup \beta$ be their corresponding circuits of minimal length, then*

$$M_{\alpha\beta} \neq 0 \Longleftrightarrow \alpha, \beta \in Adj_\alpha \cap Adj_\beta \tag{5.2}$$

$$\mathcal{C}_\alpha \cap \mathcal{C}_\beta \neq \emptyset \Longleftrightarrow \left\{ \begin{array}{l} \mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta) \neq \emptyset \\ \mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha) \neq \emptyset \end{array} \right\} \tag{5.3}$$

Proof. Because the orientation of the arcs in the graph is arbitrary and will be determined by the sign of the solution, we will use the undirected version of the graph. Let $\alpha = (\tau_v, \tau_w)$, $\beta = (\tau_x, \tau_y)$, where the graph nodes τ_v, τ_w, τ_x and τ_y respectively correspond to the mesh cells T_1, T_2, T_3 and T_4. The out-of-tree arc α uniquely corresponds to the common edge e_{12} between the triangles T_1 and T_2, and the out-of-tree arc β uniquely corresponds to the common edge e_{34} between the triangles T_3 and T_4. From (2.8), we have that $M_{\alpha\beta} \neq 0$ if and only if $supp(\mathbf{w}_{e_{12}}) \cap supp(\mathbf{w}_{e_{34}}) \neq \emptyset$. Since $supp(\mathbf{w}_{e_{12}}) = T_1 \cup T_2$ and $supp(\mathbf{w}_{e_{34}}) = T_3 \cup T_4$, then $M_{\alpha\beta} \neq 0$ if and only if the two supports share a mesh cell. Assuming, without loss of generalitythat $T_1 = T_3$ is the shared cell, (5.2) follows from the definition of Adj.

In (5.3) the $\Rightarrow$ is trivial. We give a proof *ab absurdo* for the $\Leftarrow$. First of all, we note that the cardinal number of Adj_{τ_j} is less or equal than 3, $\forall \tau_j \in \mathcal{A}$ (a triangle has three edges). If we assume that $\mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta) \neq \emptyset$, $\mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha) \neq \emptyset$ and $\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \emptyset$, there must exist a node τ_j such that τ_j belongs to both the paths in the tree linking τ_v to τ_w and τ_x to τ_y respectively. Because each node in the path has only one *parent* and one *child* , the Adj_{τ_j} contains the *parent* and the *child* of the first path and those of the second path. Because we assumed that $\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \emptyset$ the cardinal number of Adj_{τ_j} would be 4 which is in contradiction with the upper bound on the cardinal number. ◻

Thus, from (5.1) and the previous Lemma 5.1, we have the following Corollary.

COROLLARY 5.1. *Let α and β be two out-of-tree arcs, and $\mathcal{C}_\alpha \subset \mathcal{A}_{\mathcal{T}_R} \cup \alpha$ and $\mathcal{C}_\beta \subset \mathcal{A}_{\mathcal{T}_R} \cup \beta$ be their corresponding circuits of minimal length, then*

$$\begin{aligned} \tilde{M}_{\alpha\beta} = M_{\alpha\beta} + & \sum_{\gamma \in \mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta)} \ \sum_{\rho \in \mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha)} H_{\gamma\alpha} H_{\rho\beta} M_{\gamma\rho} \\ - & \sum_{\gamma \in \mathcal{C}_\alpha \cap Adj_\beta} H_{\gamma\alpha} M_{\gamma\beta} - \sum_{\gamma \in \mathcal{C}_\beta \cap Adj_\alpha} H_{\gamma\beta} M_{\beta\gamma} \end{aligned}$$

and

$$\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \emptyset \ \Rightarrow \ \tilde{M}_{\alpha\beta} = 0 \tag{5.4}$$

ETNA
Kent State University
etna@mcs.kent.edu

Proof. The first part of the Corollary follows directly from the expansion of (5.1). The implication (5.4) follows from Lemma 5.1 taking into account that $\mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta) \supseteq \mathcal{C}_\alpha \cap Adj_\beta, \mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha) \supseteq \mathcal{C}_\beta \cap Adj_\alpha$, and $Adj_\alpha \cap Adj_\beta \neq \emptyset \Rightarrow \mathcal{C}_\alpha \cap \mathcal{C}_\beta \neq \emptyset$. ◻

Corollary 5.1 gives the complete description of the pattern of $\tilde{M}_{22}$ in terms of the rooted spanning tree. However, we observe that the results (5.3) and (5.4) rely on the 2-D structure of the mesh and they cannot be generalized to a mesh in 3-D.

In the second part of this section, we build the block structure of $\tilde{M}_{22}$ using the Quotient Tree concept, without explicitly forming the matrix $\tilde{M}_{22}$. In the following, we process the root node separately from the other nodes. The root node will always be numbered by 0, and if it is a branching node with k children the tree will contain k subtrees directly linked to the root. Given a rooted spanning tree $\mathcal{T}_R = \{\mathcal{N}, \mathcal{A}_{\mathcal{T}_R}\}$, let $\mathcal{B} \subseteq \mathcal{N} \setminus \{0\}$ be the set of the branching nodes in $\mathcal{T}_R$, and let $\mathcal{L} \subseteq \mathcal{N} \setminus \{0\}$ be the set of the leaves in $\mathcal{T}_R$. We define the set

$$\mathcal{Y} = \mathcal{B} \cup \mathcal{L} = \{\tau_1, \dots, \tau_k\}\,.$$

If $\mathcal{Y} = \mathcal{N} \setminus \{0\}$, then $\mathcal{T}_R$ is a binary tree. Otherwise, we can compute the following paths:

$$\begin{aligned}&\forall \tau_i \in \mathcal{Y}\\ &\wp_{\tau_i} = \{\tau_i, parent(\tau_i), parent(parent(\tau_i)), \dots, parent^k(\tau_i)\} \text{ and } \wp_{\tau_i} \cap \mathcal{Y} = \{\tau_i\}.\end{aligned}$$

The path $\wp_{\tau_i}$ connects τ_i to all its ancestors which are not branching nodes, and it can contain τ_i alone.

The set $\mathfrak{P} = \{\wp_{\tau_1}, \dots, \wp_{\tau_\ell}\} \cup \{0\}$ is a partition of $\mathcal{N}$:

$$\wp_{\tau_j} \cap \wp_{\tau_i} = \emptyset, \quad \bigcup_{i=1}^{\ell} \wp_{\tau_i} = \mathcal{N}\,.$$

Therefore, we can build the quotient tree

$$\mathcal{T}/\mathfrak{P} = \{\mathfrak{P}, \mathfrak{E}_\mathcal{T}\}, \quad (\wp_{\tau_i}, \wp_{\tau_j}) \in \mathfrak{E}_\mathcal{T} \iff Adj\,|_\mathcal{T}\,(\wp_{\tau_i}) \cap \wp_{\tau_j} \neq \emptyset,$$

and the quotient graph

$$\mathcal{G}/\mathfrak{P} = \{\mathfrak{P}, \mathfrak{E}_\mathcal{G}\}, \quad (\wp_{\tau_i}, \wp_{\tau_j}) \in \mathfrak{E}_\mathcal{G} \iff Adj(\wp_{\tau_i}) \cap \wp_{\tau_j} \neq \emptyset,$$

where $Adj\,|_\mathcal{T}$ is the restriction of the Adj operator to the graph $\mathcal{T}$.

For the sake of clarity in the following, we will call the quotient tree nodes Q-nodes. The root of the quotient tree is still the node 0, and each subtree rooted at its children has a binary quotient tree.

5.1. Data structures and separators. In this subsection, we shortly review the basic data structures that we used to implement the null space method presented in the previous section. We also discuss some major details concerning the renumbering strategy that allows us to perform a nested dissection-like decomposition of the matrix A. This kind of permutation and the resulting sub-block decomposition makes it possible to build the block-Jacobi preconditioner mentioned in the next sub-section and considered in the numerical experiments of Section 6. More details on the data structure and algorithm implementation are reported in the final appendix.

The data structure that is used for the graph $\mathcal{G}$ is based on a double representation of the sparse matrix A. This double representation implements the collection of compressed row and column sparse data which is described in [17] and allows us to access simultaneously

ETNA
Kent State University
etna@mcs.kent.edu

to matrix rows and columns. We assume that rows and columns are consistently permutated when we renumber the graph node. This special design facilitates and speeds up the algorithms that are reported in the appendix.

Trees, which are used to perform row/column permutations, are represented by storing the following data for any node:

- the parent node,
- the node children list,
- the chain index,
- the depth.

In Figure 5.1, we give a simple example of a tree and, in Table 5.1, we list the labels of each node. It is relevant to observe that the reordering obtained by the depth first search of $\mathcal{T}$ renumbers the nodes such that the nodes forming a chain are consecutive. Therefore, we can directly access the list using vector arrays.

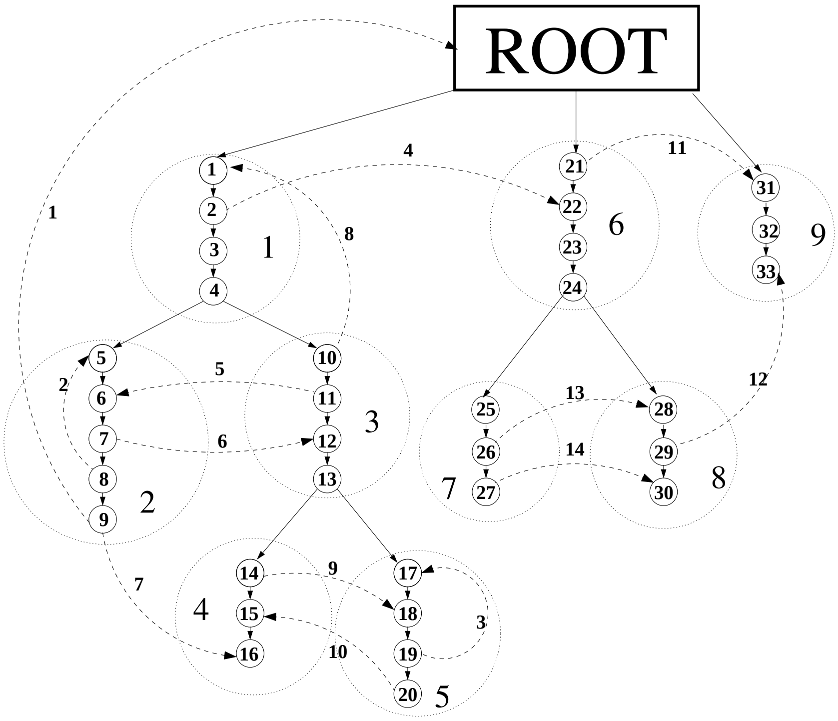


FIG. 5.1. *Example of a spanning tree.*

After the identification of the chains, we build the *quotient tree*, and, descending the quotient tree $\mathcal{T}/\mathfrak{P}$ with a "depth first search" algorithm, we renumber the nodes and build its data structure. In the new data structure, we associate with each Q-node the following objects:

- Q-parent in $\mathcal{T}/\mathfrak{P}$,
- first and last node in $\mathcal{T}$ of the chain corresponding to the Q-node,
- depth in $\mathcal{T}/\mathfrak{P}$,
- Q-last, the last Q-node of the subtree rooted in the current Q-node,
- Q-star, the list of the out-of-tree arcs in $\mathcal{G}$ which have one extreme belonging to the Q-node,
- Q-children, the list of the Q-nodes children of the Q-node.

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 5.1
Labels of nodes in the tree of Figure 5.1.

Node index	Parent	Children list	Chain Index	depth
0	0	1,21,31	0	0
1	0	2	1	1
2	1	3	1	2
3	2	4	1	3
4	3	5,10	1	4
5	4	6	2	5
6	5	7	2	6
7	6	8	2	7
8	7	9	2	8
9	8		2	9
10	4	11	3	5
11	10	12	3	6
12	11	13	3	7
13	12	14,17	3	8
14	13	15	4	9
15	14	16	4	10
16	15		4	11
17	13	18	5	9
18	17	19	5	10
19	18	20	5	11
20	19		5	12
21	0	22	6	1
22	21	23	6	2
23	22	24	6	3
24	23	25,28	6	4
25	24	26	7	5
26	25	27	7	6
27	26		7	7
28	24	29	8	5
29	28	30	8	6
30	29		8	7
31	0	32	9	1
32	31	33	9	2
33	32		9	3

In Figure 5.2, we show the quotient tree relative to the example of Figure 5.1, and in Table 5.2, we list the labels of each Q-node.

Taking advantage of the data structures described above, we can order the out-of-tree arcs in the following way. Firstly, we identify the Q-nodes which are children of the external root (node 0), and the subtrees rooted in each of these Q-nodes. Then, we separate each of the subtrees from the others marking the out-of-tree edges that connect it to the others. The out-of-tree arcs lying within one of these subtrees cannot have fundamental cycles with the out-of-tree arcs lying within one of the others, because of Corollary 4.1. This corresponds to a one-way dissection applied to $\tilde{M}_{22}$. Then, within each of the subtrees, we seek the out-of-tree arcs that separate the two subtrees rooted in the Q-nodes children of the Q-node

ETNA
Kent State University
etna@mcs.kent.edu

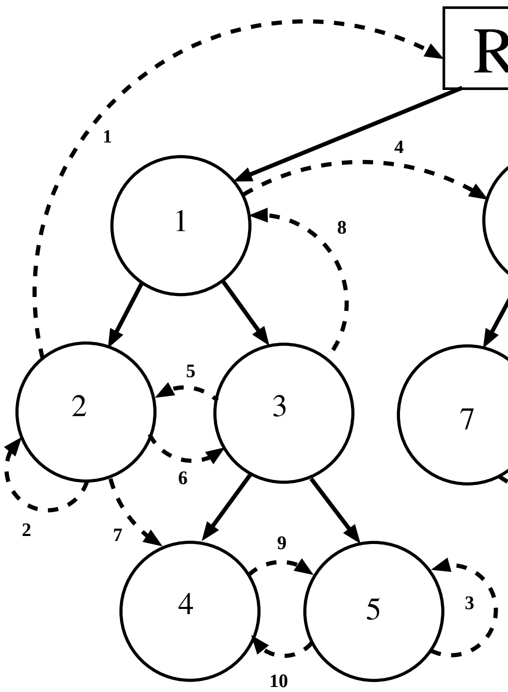


FIG. 5.2. *Quotient tree relative to the tree of Figure 5.1*

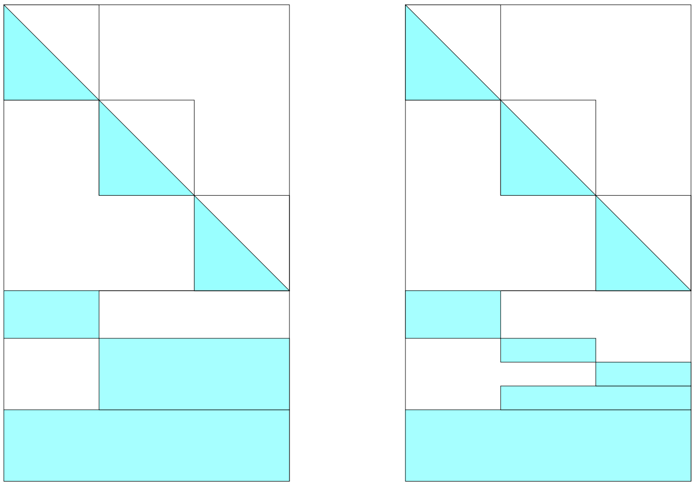
TABLE 5.2
Labels of the Q-nodes in Figure 5.2.

Q-node	Q-parent	First, last node of chain	Depth	Q-last	Q-star	Q-children
1	0	1,4	1	5	4,8	2,3
2	1	5,9	2	2	1,2,5,6,7	
3	1	10,13	2	5	5,6,8	4,5
4	3	14,16	3	4	7,9,10	
5	3	17,20	3	5	3,9,10	
6	0	21,24	1	8	4,11	7,8
7	6	25,27	2	7	13,14	
8	6	28,30	2	8	12,13,14	
9	0	31,33	1	9	11,12	

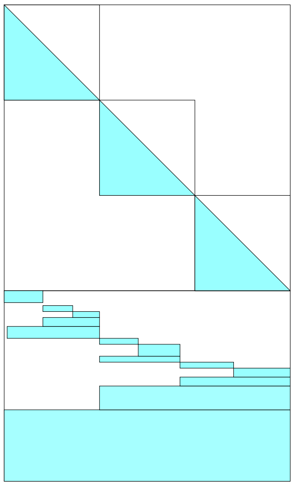
root of the subtree containing both of them. This is equivalent to a nested dissection strategy applied to one of the diagonal blocks resulting from the previous one-way dissection phase. In Figure 5.3 (a), we show the result of the one-way dissection on the matrix A when the root node has only three descendant subtrees. Note that each subtree is now disconnected from the others, is a binary tree and it can be identified by the Q-node on which it is rooted. If the nested dissection process is recursively re-applied on these matrix sub-blocks, we obtain the matrix structure shown in Figure 5.3 $(b - c)$.

5.2. Preconditioners. The *a priori* knowledge of the structure of $\tilde{M}_{22}$, allows us to decide what kind of preconditioner we can afford. If we are subjected to a strong limitation of the size of the memory in our computer, we can choose among several alternative precondi-

ETNA
Kent State University
etna@mcs.kent.edu



(a) (b)



(c)

FIG. 5.3. *Example of a one-way dissection (a, b), and of a nested dissection (c) on the matrix A*

tioners. We have a choice ranging from the diagonal matrix obtained by using the diagonal part of M_{22} and the block Jacobi preconditioner using the diagonal blocks corresponding to the separators. The possibility of using the simplest choice of the diagonal of M_{22} is sensible because the SPT algorithm places on this diagonal the biggest entries of the diagonal of M. In Section 6, we will give numerical evidence that this choice is very efficient for several test problems. Nevertheless, in the presence of strong discontinuities and anisotropies in the permeability function $\mathbb{K}$, we are obliged to use either the diagonal Jacobi preconditioner or a block diagonal Jacobi.

ETNA
Kent State University
etna@mcs.kent.edu

6. Numerical experiments.

6.1. Test problems. We generated the test problems using four different domains. The first two are square unit boxes and in the second one we have four rectangular regions where the tensor $\mathcal{K}$ assumes different values. In Figure 6.1, we plot the geometry of Domain 1 and the boundary conditions. In Figure 6.2, we plot the geometry of Domain 2 and the boundary conditions. The values of the tensor $\mathcal{K}$ are chosen as follow

$$\mathcal{K}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \Omega \setminus \{\Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4\}, \\ 0.5 & \mathbf{x} \in \Omega_1, \\ 10^{-4} & \mathbf{x} \in \Omega_2, \\ 10^{-6} & \mathbf{x} \in \Omega_3, \\ 10^{-8} & \mathbf{x} \in \Omega_4. \end{cases}$$

The two remaining domains have an L-shape geometry. In Figure 6.3, we plot the geometry and the boundary conditions of the Domain 3. In Figure 6.4, we plot the geometry of the fourth and last Domain 4 and the relative boundary conditions: within the domain, we have four rectangular regions where the tensor $\mathcal{K}$ takes the same values defined for the second domain in (6.1).

In (2.2), we take the right-hand side $f(\mathbf{x}) = 0$ in all our test problems. For the domains one and three, the tensor $\mathcal{K}$ in (2.1) is isotropic. For a given triangulation, its values are constant within each triangle and this value is computed following the law:

$$\mathcal{K}_{T_i} = 10^{-12 r_i^3}\ ,\ i = 1, \ldots, N_T$$

where $r_i \in [0, 1]$ are numbers computed using a random uniform distribution. For each domain, we generated 4 meshes using TRIANGLE [36]. In Tables 6.1 and 6.2, we report, for our domains, the number N_T of triangles, the number N_E of edges, the number N_V of vertices of each mesh, the number $M.nnz$ of nonzero entries in the matrix M, the number $A.nnz$ of nonzero entries in matrix A, and the corresponding value of $\mathfrak{h}$.

TABLE 6.1
Data relative to the meshes for domains 1 and 2.

	Mesh 1	Mesh 2	Mesh 3	Mesh 4
N_T	153	1567	15304	155746
N_E	246	2406	23130	234128
N_V	94	840	7827	78383
$M.nnz$	1164	11808	114954	1168604
$A.nnz$	429	4599	45601	466319
$\mathfrak{h}$	0.2090	0.0649	0.0225	0.0069

TABLE 6.2
Data relative to the meshes for domains 3 and 4.

	Mesh 1	Mesh 2	Mesh 3	Mesh 4
N_T	156	1494	15206	150033
N_E	251	2305	23102	225599
N_V	96	812	7843	75567
$M.nnz$	1187	12269	114662	1125797
$A.nnz$	442	4386	45462	449281
$\mathfrak{h}$	0.1625	0.0590	0.0186	0.0063

ETNA
Kent State University
etna@mcs.kent.edu

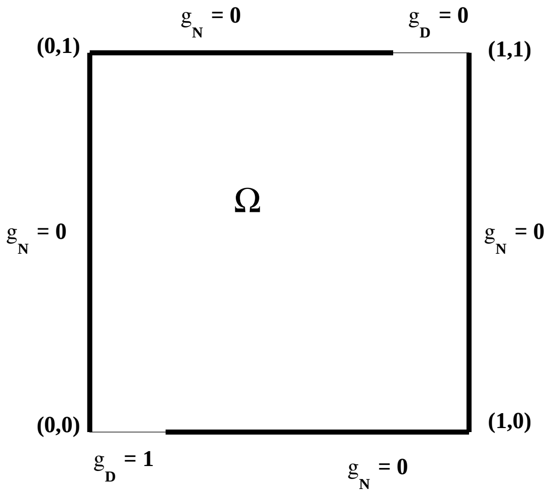


FIG. 6.1. *Geometry of the first domain* Ω.

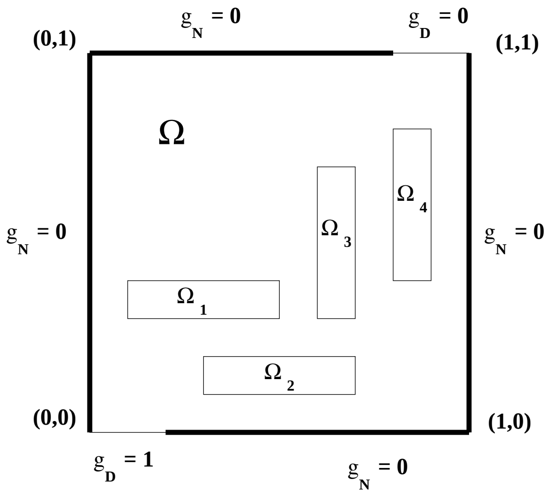


FIG. 6.2. *Geometry of the second domain* Ω.

6.2. Practicalities. We analysed the reliability of the stopping criterion when we change the parameter d. In Figures 6.5 and 6.6, we display the behaviour of the estimates of the true relative energy norm of the error for Mesh 3, Domain 3 and Domain 4: for the other cases the behaviour is similar. In all our test we choose $\eta = \mathfrak{h}$. The results show that the choice $d = 10$ is the best compromise between reliability and cost. When convergence is slow and there are regions where the slope changes rapidly, the choice $d = 5$ can be inaccurate. We reserve for future work the study of a self-adaptive technique which will change the value of d with the slope of the convergence curve. In Section 3, we discussed the opportunity of starting the estimate of the relative error only when $e_{\tilde{M}_{22}} < \eta^2 \|h_2 - \tilde{M}_{12}^T z_1\|_2^2$. This makes it possible a reduction of the number of additional matrix-vector products. Both figures show an initial phase where the estimates have not been computed because of the introduction of this check on the absolute value of the error.

ETNA
Kent State University
etna@mcs.kent.edu

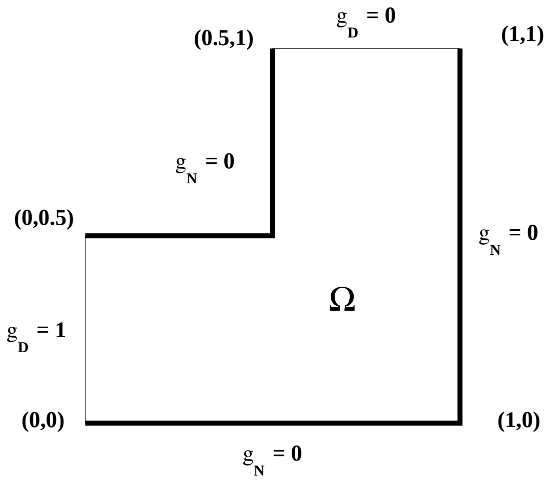


FIG. 6.3. *Geometry of the third domain* Ω.

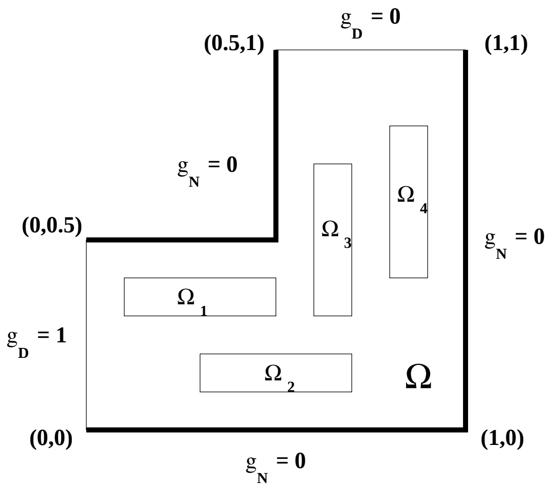


FIG. 6.4. *Geometry of the fourth domain* Ω.

Moreover, to avoid an excessive number of additional matrix-vector products in the stopping criterion, we choose to update the value of the denominator $\overline{u}_2^{(k)T}\tilde{M}_{22}\overline{u}_2^{(k)}$ every 10 steps of the conjugate gradient method. The energy norm of $\overline{u}_2^{(k)}$ converges quite quickly to the energy norm of the solution and this justifies our choice. In Figures 6.7 and 6.8, we see that after 25% of the iterations, the ratio $\frac{||u||_{\tilde{M}_{22}}}{||u^{(k)}||_{\tilde{M}_{22}}}$ is greater than 0.9.

6.3. Numerical results. We generated and ran all our test problems on a SPARC processor of a SUN ENTERPRISE 4500 (4CPU 400 MHertz, 2GByte RAM). In our test runs, we compare the performance of our approach with the performance of **MA47** of the **HSL2000** library [26]. The package **MA47** implements a version of the LDL^T decomposition for symmetric indefinite matrices that takes advantage of the structure of the augmented system [18]. The package is divided into three parts corresponding to the symbolic analysis where the reordering of the matrix is computed, the factorization phase, and the final solution using the

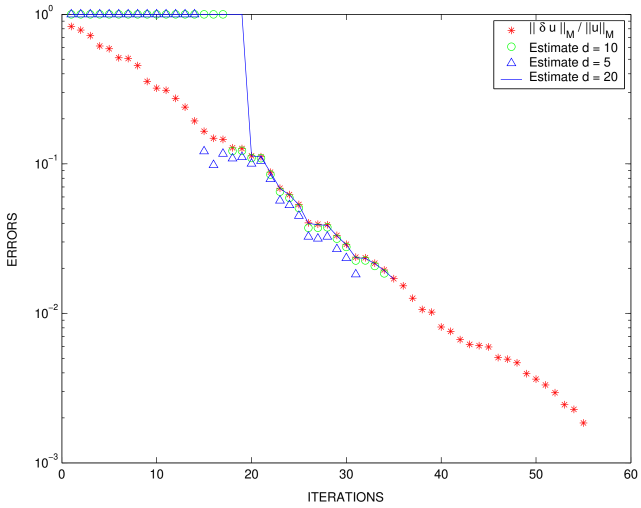


FIG. 6.5. *Error energy norm and its estimates for* $d = 5$*,* $d = 10$*, and* $d = 20$ *for Mesh 3 and domain 3.*

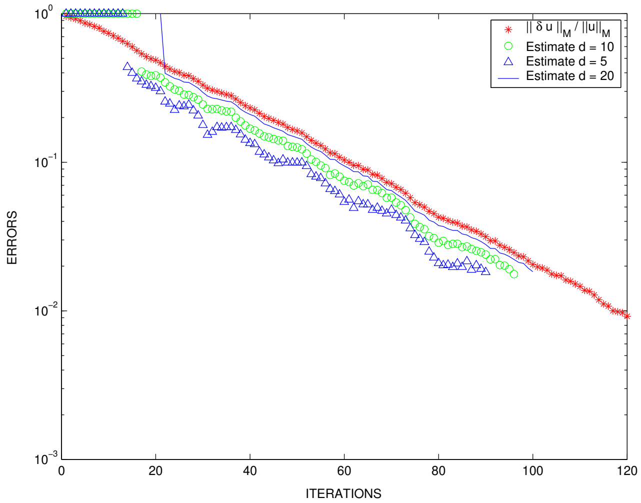


FIG. 6.6. *Error energy norm and its estimates for* $d = 5$*,* $d = 10$*, and* $d = 20$ *for Mesh 3 and domain 4.*

triangular matrices.

Similarly, the null space algorithm which we implemented, can be subdivided into three phases: a first symbolic phase where the shortest path tree and the quotient tree are computed, a second phase where the projected Hessian system is solved by the conjugate gradient algorithm, and a final third phase where we compute the pressure. This enables us to compare the direct solver **MA47** with the null space approach in each single phase.

Generally, in the test runs that we will present, we fix the parameter d in the stopping criterion to the value of 10. Nevertheless, we will show the influence of different choices on the parameter d on the stopping criterion using Mesh 3.

In Table 6.3, we give the CPU times (in seconds) and the storage (in MByte) required by **MA47** and the CPU times (in seconds) of the null space algorithm where we use the diagonal of M_{22} to precondition the projected Hessian matrix within the conjugate gradient algorithm.

ETNA
Kent State University
etna@mcs.kent.edu

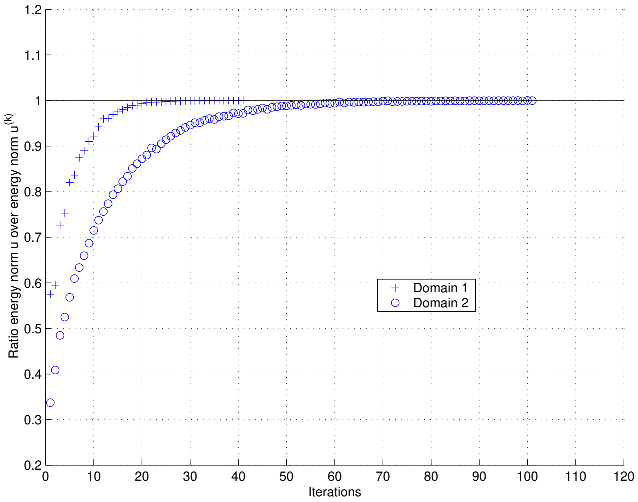


FIG. 6.7. *Convergence of* $\frac{||u||_{\tilde{M}_{22}}}{||u^{(k)}||_{\tilde{M}_{22}}}$ *for Mesh 3 and domains 1 and 2.*

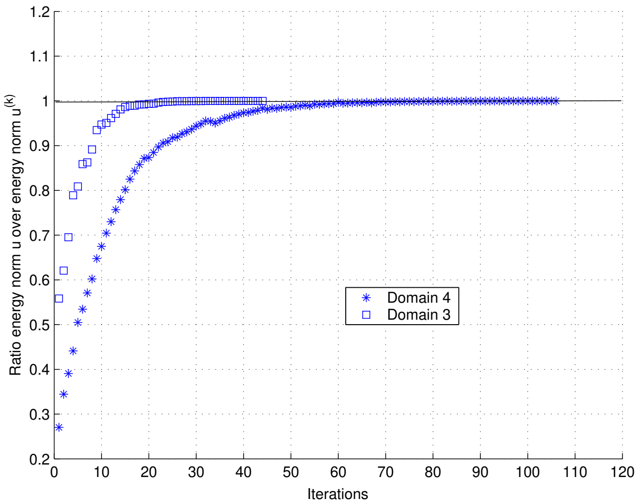


FIG. 6.8. *Convergence of* $\frac{||u||_{\tilde{M}_{22}}}{||u^{(k)}||_{\tilde{M}_{22}}}$ *for Mesh 3 and domains 3 and 4.*

From Table 6.3, we see that the null space algorithm performs better in the case of random permeability which can be a realistic simulation of an underground situation. Nevertheless, the global CPU time of the null space algorithm can be 10 times more than the CPU time of the direct solver. We point out that the **MA47** storage requirement for the L and D factors grows with the size of the problem whereas the null space algorithm needs only the storage of the matrices M and A. We forecast that this will become even more favourable to the null space algorithm when we want to solve 3D simulations: for these problems the **MA47** storage could become so large that we could be obliged to use an out-of-core implementation.

In Table 6.4, we display the behaviour of three different preconditioners on the conjugate gradient iteration number. We fixed the mesh (Mesh 3) and we use as a preconditioner one of the following matrices: $diag(M_{22})$, $diag(\tilde{M}_{22})$ (the classical Jacobi), and $blockdiag(\tilde{M}_{22})$

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.3
MA47 *vs null space algorithm: CPU times (in seconds) and storage (in MBytes).*

Mesh	Domain	**MA47**				null space algorithm		
		Symbolic	Factorization	Solver	Storage	Symbolic	CG(#Iterations)	Solve
1	1	0.008	0.013	0.001	0.048	0.002	0.013 (12)	0.002
1	2	0.008	0.008	0.001	0.035	0.002	0.013 (14)	0.002
1	3	0.008	0.010	0.001	0.032	0.001	0.013 (13)	0.001
1	4	0.007	0.007	0.001	0.040	0.002	0.016 (17)	0.001
2	1	0.088	0.173	0.006	0.634	0.017	0.145 (19)	0.018
2	2	0.088	0.101	0.005	0.458	0.007	0.275 (35)	0.019
2	3	0.079	0.152	0.006	0.556	0.011	0.145 (20)	0.018
2	4	0.084	0.095	0.005	0.418	0.009	0.265 (37)	0.017
3	1	1.058	3.028	0.114	9.15	0.911	4.681 (41)	0.290
3	2	1.070	1.577	0.084	6.06	0.087	11.63 (101)	0.265
3	3	1.035	2.393	0.111	8.14	0.290	4.945 (44)	0.280
3	4	1.041	1.310	0.081	5.72	0.088	12.06 (106)	0.268
4	1	14.63	264.6	1.543	132.32	7.8	210.6 (176)	2.941
4	2	14.55	49.03	1.091	81.89	1.179	463.0 (390)	2.855
4	3	13.58	84.33	1.481	118.9	3.887	298.7 (272)	2.748
4	4	13.54	34.49	1.067	77.62	1.056	445.6 (395)	2.733

(block Jacobi) which has been computed using the quotient tree of the shortest path tree (see Section 5). For each preconditioner and each domain, we display the number of conjugate gradient iterations, the CPU time (in seconds) for the building of the matrix, and the CPU time (in seconds) spent by the conjugate gradient algorithm to solve the projected Hessian linear system. From the results of Table 6.4, we conclude that the simplest preconditioner $diag(M_{22})$ is faster even if the conjugate gradient algorithm does more iterations than the conjugate gradient algorithm using the other preconditioners. The Jacobi and the block Jacobi preconditioner building cost is very high and overwhelms the good performance of the conjugate gradient algorithm.

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.4
Comparison between the preconditioners: CPU times and # Iterations of conjugate gradient algorithm.

Domain	Preconditioner	CG #Iterations	CPU Time (in seconds)	
			Building	CG solve
1	$diag(M_{22})$	41	0.026	4.681
1	$diag(\tilde{M}_{22})$	28	13.60	3.212
1	$blockdiag(\tilde{M}_{22})$	19	15.06	2.854
2	$diag(M_{22})$	101	0.025	11.63
2	$diag(\tilde{M}_{22})$	79	13.16	8.859
2	$blockdiag(\tilde{M}_{22})$	69	12.13	10.18
3	$diag(M_{22})$	44	0.025	5.049
3	$diag(\tilde{M}_{22})$	26	13.94	2.954
3	$blockdiag(\tilde{M}_{22})$	19	15.63	3.122
4	$diag(M_{22})$	106	0.025	12.06
4	$diag(\tilde{M}_{22})$	92	13.10	10.04
4	$blockdiag(\tilde{M}_{22})$	79	13.23	12.37

In Table 6.5, we report the CPU time (in seconds) spent by our implementation of the Algorithms A.1, A.2, and A.3. These algorithms build the nested dissection structure of $\tilde{M}_{22}$ and the null space matrix Z. In particular, the nested dissection structure can be useful in building a parallel version of the matrix-vector products involving the implicit form of the matrix $\tilde{M}_{22}$. Moreover, it is possible to use the structure for the evaluation of the complexity of the explicit computation of the null space matrix Z. We observe that comparing Table 6.5 and Table 6.3, the computational cost of Algorithms A.1, A.2, and A.3 is comparable with the cost of the **MA47** symbolic phase. Algorithms A.1, A.2, and A.3 are the basis on which it is possible to build a parallel version of the matrix-vector computation (3.8). In particular, it is possible to use Algorithm A.3 to generate a suitable data structure for the solution of the triangular systems involving the matrices L and L^T.

7. Generalization to the 3-D case. Almost all the results of the paper can be generalized to the case $\Omega \subset \mathbf{R}^3$. The only exceptions are Lemma 5.2 and Corollary 5.1. In practice, we cannot predict a priori the pattern of the projected Hessian $\tilde{M}_{22}$ when the graph $\mathcal{A}$ is not planar, as is the case of the 3D meshes. In particular, we can build the shortest path tree in the 3D case as described in Section 4. In the resulting tree (or forest) each parent has at most 3 children. Moreover, for a problem with only Dirichlet boundary conditions and an isotropic mesh the resulting forest has a number of trees proportional to the number of triangles meshing the boundary surfaces. This number is proportional to $\mathfrak{h}^{-2}$. Therefore, the potential parallelism in solving the lower and upper triangular matrices in the null space algorithm increases of one order of magnitude from the two dimensional to the three dimensional case. The extension to the three dimensional domain of the preconditioners presented in Section 6.3 is straightforward. Furthermore, the increased parallelism suggest the possibility of reducing considerably the cost of building the block versions based on the quotient tree such as $diag(\tilde{M}_{22})$ or $blockdiag(\tilde{M}_{22})$.

In particular, we want to highlight that the absence of fill-in is promising when we need to solve Darcy's equations in 3D domains. It is reasonable to expect that the fill-in and complexity of a direct solver, applied to the augmented systems related to the approximation of Darcy's equations in 3D domains, would grow as $\mathcal{O}(m^{4/3})$ and $\mathcal{O}(m^2)$ respectively [30, 29]. Instead, our algorithm does not have any fill-in and its complexity only depends on the

ETNA
Kent State University
etna@mcs.kent.edu

TABLE 6.5
CPU Time (in seconds) for the computation of nested dissection structure for null space matrix Z.

Mesh	Domain	CPU Time (in Seconds) Algorithms A.1,A.2,A.3
1	1	0.001
1	2	0.001
1	3	0.002
1	4	0.001
2	1	0.025
2	2	0.009
2	3	0.022
2	4	0.006
3	1	1.225
3	2	0.159
3	3	0.982
3	4	0.178
4	1	29.89
4	2	4.775
4	3	28.53
4	4	5.118

condition number of the scaled projected Hessian matrix $Z^T M Z$. We can reasonably assume that this condition number does not change with the dimension of the domain Ω, analogous to the behaviour of the classical finite-element method. Therefore, the stopping criterion will stop the conjugate gradient algorithm after a number of steps which is not dependent on the dimension of the domain Ω. The performance analysis of our algorithm when applied to three dimensional domains will be considered in future work.

8. Conclusions. We have analysed a variant of the null space algorithm that takes full advantage of the relation between the structure of the augmented system and the network programming structure of the mixed finite-element approximation of the Darcy's equations. We remark that this method can be applied to any diffusion equation. We compared the performance of our prototype version of the algorithm with a well established direct solver and we concluded that even if our implementation can be 10 times slower than the direct solver, the absence of fill-in makes our code competitive for large problems.

We did not implement a parallel version of our algorithm. Nevertheless, we are confident that a parallel version will speed up the performance. In particular, the matrix-vector product involving $Z^T M Z$ can largely benefit from the parallelization, where the block structure of L_1 can be exploited.

Finally, once the null basis has been computed, it can be used to solve a sequence of problems (2.1-2.2) with different permeability tensors, i.e., different soil characteristics, since in this case only the block M changes and A stays the same. In contrast, direct solvers, like **MA47**, computing a Gaussian factorization of the augmented system or the Schur-complement based approaches cannot take advantage of this. The same applies to the case of time-dependent or nonlinear problems.

This advantage is fairly important in the three dimensional cases.

Appendix. In this appendix, we describe the major details regarding the data structures and algorithm implementations of Section 5.1.

ETNA
Kent State University
etna@mcs.kent.edu

The matrix A is the incidence matrix of the graph representing the unstructured bidimensional mesh used in the computations. The the row/column compressed data structure mentioned in Section 5.1 [17] can thus be easily implemented as follows by using vector arrays. Each column of A has at most *three* nonzero entries; then, we store for each node (i.e. triangle) i the *three* arcs (i.e. edges) indices consecutively in position $i,\ i+1,\ i+2$. If the node corresponds to a triangle having an edge on the Neumann boundary, we explicitly store a 0 which means that the triangle has the root as a neighbour. Analogously, as each row j of A has only *two* nonzero entries, we store the *two* nodes (i.e. triangles) describing the arc j in position j and $j+1$ of a vector. Again, we need to explicitly store some 0 values for the Dirichlet edges. The overhead of the explicit storage of these zeroes is less than the one we would have if we stored the pointers within the classical row and column compressed forms because the number of the boundary edges is of the order of the square root of the number of triangles in the mesh.

Before starting the one-way dissection phase, we visit the tree and identify the out-of-tree arcs in each Q-node that are within the Q-node and the arcs directly linking the external root (node 0) to a Q-node. These arcs are stored in *sep.list* which is a queue data structure. The phase relative to the one-way dissection is implemented by the following algorithm:

ALGORITHM A.1.

procedure root.sep.count(sep.size)
  for each $i \in$ *children*(0) *do*
    $Q_i =$ *chain(i); sep.size*$(Q_i) = 0$ *;*
    for $Q_k = Q_i$*:Q-last*(Q_i)*, do*
     for each $\ell = (p,q) \in$ *Q-star*(Q_k) *with chain(p)=* Q_k *do*
       if $q >$ *last(Q-last*(Q_i)*) or* $q <$ *first*(Q_i) *then*
        sep.size(Q_i) = *sep.size*(Q_i) + *1;*
       end if;
     end for each;
    end for;
  end for each;
end procedure.

procedure sep.tree (sep.list,sep.size,Q_{root}*,mask)*
  for $Q_k = Q_{root}$*:Q-last*(Q_{root})*, do*
   for each $\ell = (p,q) \in$ *Q-star*(Q_k) *with chain(p)=* Q_k *do*
     if mask$(\ell) = 0$ *and* $\{q >$ *last(Q-last*(Q_i)*) or* $q <$ *first*$(Q_i)\}$ *then*
      insert ℓ *in sep.list ;*
      mask$(\ell) = 1$ *;*
      sep.size = sep.size + 1;
     end if;
   end for each;
  end for;
end procedure.

procedure one.way.dissection(sep.list,sep.size,mask)
  root.sep.count(sep.size)
  sort Q-children(Q_{root}) *in decreasing order of sep.size ;*
  for each child $\in$ *Q-children*(Q_{root}) *do*
    sep.tree (sep.list, sep.size(child), child, mask);

end for each;
end procedure.

Because each out-of-tree arc can be counted only twice, the complexity of Algorithm A.1 is $\mathcal{O}(2\xi)$, where ξ is the number of nonzero entries in the submatrix L_2 and is equal to twice the number of the out-of-tree arcs.

From Corollary 4.1, Corollary 5.1, and Lemma 5.1 it follows that each block i, of size $sep.size(i)$, contains out-of-tree arcs that have fundamental cycles intersecting each other. Therefore, each of these blocks corresponds to a full diagonal block in $\tilde{M}_{22}$. The order of the diagonal block is equal to the corresponding value of $sep.size$. The out-of-tree arcs in $sep.list$ form the external *separator*. The external separator identifies a curve on the mesh. Thus, because the graph is planar, the size χ of the external separator will be $\mathcal{O}(\sqrt{m})$ (m is the number of triangles in the mesh) [30, 29].

Finally, we renumber the out-of-tree arcs, such that the arcs in the external separator are the last ones and the arcs lying within a descendant subtree of root are consecutive. This is equivalent to permute the rows of the matrix L_2 so that we have a block diagonal submatrix followed by a block of rows connecting the previous diagonal blocks.

In the following Algorithm A.2, we denote by Q_{root} the root of one of the Q-subtrees obtained from the one-way dissection phase. The algorithm proceeds as follows. Basically, we visit each subtree and we reorder the Q-children list of each Q-node such that the first child is the root of the subtree with the least number of nodes, and the other children are sorted in increasing order with respect to the number of nodes in their respective subtrees. Moreover, we label each out-of-tree arc ℓ with the quotient tree ancestor *QT-ancestor(ℓ)*, the Q-node closing the fundamental cycle of ℓ in the spanning tree.

By means of *QT-ancestor* and by the data structure of the spanning tree, we can implicitly describe the structure of the null space matrix Z.

ALGORITHM A.2.
procedure nested.sep(sep.list, sep.size, Q_{root}, mask, QT-ancestor)
    if Q-children(Q_{root}) $\neq \emptyset$ then
        Q_1 head of Q-children(Q_{root}) list;
        Q_2 tail of Q-children(Q_{root}) list;
        sep.tree(sep.list, sep.size(Q_1), Q_1, mask);
        sep.size(Q_2) = 0;
        for each $\ell \in$ Q-star(Q_{root}) do
            if mask(ℓ) = 0 then
                insert ℓ in sep.list ;
                mask(ℓ) = 1 ;
                sep.size(Q_2) = sep.size(Q_2) +1;
                QT-ancestor(ℓ) = Q_{root};
            end if
        end for each;
        nested.sep(sep.list, sep.size, Q_1, mask);
        nested.sep(sep.list, sep.size, Q_2, mask);
    end if;
end procedure

Algorithm A.2 takes advantage of the binary structure of the quotient subtree and of the reordering of the Q-children list whose first entry has the least number of descendants. These two properties allow the possibility of separating the two subtrees rooted in the children of the current Q_{root} visiting only one of them and the Q-star of Q_{root}. Moreover, at each recursion we visit and separate a subtree which, in the worst case, has half of the nodes of the tree in it. Let N_Q be the number of nodes in one of the subtrees obtained after the one-way dissection phase. The Nested Dissection phase applied to this subtree will visit all the levels of the tree. For each level, algorithm A.2 will separate the subtrees with the least number of descendants. In the worst case, when the number of nodes in each subtree is half of the nodes of the previous subtree which contains it, the number of levels is $\mathcal{O}(\log N_Q)$. Thus, the worst complexity of algorithm A.2 is $\mathcal{O}(N_Q \log N_Q)$.

If the tree is unbalanced, i.e. only one of the two children of a Q-node is a branching Q-node, the complexity of algorithm A.2 is $\mathcal{O}(N_Q)$. Using the *sep.size* and *sep.list*, we can build the data structure *ND*, which is composed of *ND.list* and *ND.pointer*. *ND.list* contains the entries of *sep.list* in reverse order. *ND.pointer(Q_i)* contains the pointer to the first entry in *ND.list* of the separator of the tree rooted in Q_i. In the following algorithm, we denote by N_{out} the number of out-of-tree arcs. We point out that the number of entries in *sep.list* is equal to N_{out}.

ALGORITHM A.3.

procedure nested.dissection(sep.list, sep.size, ND, QT-ancestor)
    for ℓ *= 1:*N_{out} *do*
        ND.list(N_{out} *-* ℓ *+1) = sep.list(*ℓ*);*
        temp = QT-ancestor(N_{out} *-* ℓ *+1);*
        QT-ancestor(N_{out} *-* ℓ *+1) = QT-ancestor(*ℓ*);*
        QT-ancestor(ℓ*) = temp;*
    end for;
    size = 0;
    for Q_i *=* Q_{root} *: Q-last(*Q_{root}*) do*
        size = size +sep.size(Q_i*);*
        ND.pointer(Q_i*) =* N_{out} *- size + 1;*
    end for;
end procedure

The *ND.list* gives the permutation that will reorder rows and columns of $\tilde{M}_{22}$ in a nested dissection order. Finally, we can build the pattern of the upper triangular part of $\tilde{M}_{22}$ by using the following algorithm A.4, which takes advantage of the consecutive order of the Q-nodes forming a tree, obtained by applying a *depth first search* on $\mathcal{T}/\mathfrak{P}$, and of the nested ordering of the out-of-tree arcs in *ND*. We point out that for each Q_i, the rows and the columns with indices between *ND.pointer(*Q_i*)* and *ND.pointer(*Q_i*+1)* form a full diagonal block in $\tilde{M}_{22}$. This pattern of the $\tilde{M}_{22}$ is described by the usual compressed row collection data structure *(irow,jcol)* where for the row i of $\tilde{M}_{22}$ is stored in *jcol(irow(i):irow(i+1)-1)*.

ALGORITHM A.4.

procedure insert.columns(ND, QT, irow, jcol, jcount, count)
    for Q_k *= QT+1:Q-last(QT) do;*
        for each $\ell = (p, q) \in$ *Q-star(*Q_k*) with chain(p)=* Q_k *do*
            if $q >$ *last(Q-last(*Q_i*)) or* $q <$ *first(*Q_i*) then*
                jcount = jcount + 1;
                jcol(jcount) = ℓ*;*
                count = count + 1;

```
                end if;
            end for each;
        end for;
end procedure;
procedure M̃22-pattern(ND, QT-ancestor, irow, jcol)
        irow(1) = 1; jcount = 0;
        for i = 1:N_out do
            count = 0;
            Q1 head of Q-children(QT-ancestor(i)) list;
            Q2 tail of Q-children(QT-ancestor(i)) list;
            for k = i:ND.pointer(Q1+1) - 1 do
                jcount = jcount + 1;
                jcol(jcount) = ND.list(k);
                count = count + 1;
            end for;
            let (p_i, q_i) the ends of arc i;
            Qp_i = chain(p_i); Qq_i = chain(q_i);
            if QT-ancestor(i) ≠ Qp_i and QT-ancestor(i) ≠ Qq_i then
              insert.columns(ND, QT-ancestor(i), irow, jcol, jcount, count)
              irow(i+1) = irow(i) + count;
            else
              if q_i > last(Q-last(Q1)) or q_i < first(Q1) then
                insert.columns(ND, Q1, irow, jcol, jcount, count);
                irow(i+1) = irow(i) + count;
              else
                insert.columns(ND, Q2, irow, jcol, jcount, count);
                irow(i+1) = irow(i) + count;
              end if;
            end if;
        end for;
end procedure
```

Finally, we observe that the final value of *jcount* gives the total number of nonzero entries of the upper triangular part of $\tilde{M}_{22}$. Therefore, without the explicit computation of the real values of the nonzero entries in $\tilde{M}_{22}$ we can predict the total amount of memory we need for storing the matrix.

REFERENCES

[1] R. ALBANESE AND G. RUBINACCI, *Integral formulation for 3D eddy-current computation using edge elements*, IEEE Proc. A, 135 (1988), pp. 457–462.

[2] P. ALOTTO AND I. PERUGIA, *Mixed finite element methods and tree-cotree implicit condensation*, CALCOLO, 36 (1999), pp. 233–248.

[3] R. AMIT, C. A. HALL, AND T. A. PORSCHING, *An application of network theory to the solution of implicit Navier-Stokes difference equations*, J. Comp. Phys., 40 (1981), pp. 183–201.

[4] M. ARIOLI, *A stopping criterion for the conjugate gradient algorithm in a finite element method framework*, Numer. Math., 97 (2004), pp. 1–24.

[5] M. ARIOLI AND L. BALDINI, *A backward error analysis of a null space algorithm in sparse quadratic programming*, SIAM J. Matrix Anal. and Applics., 23 (2001), pp. 425–442.

[6] M. ARIOLI, J. MARYŠKA, M. ROZLOŽNÍK, AND M. TŮMA, *Dual variable methods for mixed-hybrid finite element approximation of the potential fluid flow problem in porous media*, to appear in Electron. Trans. Numer. Anal., 2005. Special Volume on Saddle Point Problems:

Numerical Solution and Applications. Electron. Trans. Numer. Anal., 22 (2006), pp. 17–40, http://etna.mcs.kent.edu/vol.22.2006/pp17-40.dir/pp17-40.html.

[7] D. N. ARNOLD, R. S. FALK, AND R. WINTHER, *Preconditioning in $H(div)$ and applications*, Math. Comp., 66 (1997), pp. 957–984.

[8] M. BENZI, G. H. GOLUB, AND J. LIESEN, *Numerical solution of saddle point problems*, Acta Numerica, 14 (2005), pp. 1–137.

[9] M. W. BERRY, M. T. HEATH, I. KANEKO, M. LAWO, R. J. PLEMMONS, AND R. C. WARD, *An algorithm to compute a sparse basis of the null space*, Numer. Math., 47 (1985), pp. 483–504.

[10] O. BIRÓ, K. PREIS, G. VRISK, K. R. RICHTER, AND I. TICAR, *Computation of 3D magnetostatic fields using a reduced scalar potential*, IEEE Trans. Magnetics, 29 (1993), pp. 1329–1332.

[11] F. BREZZI AND M. FORTIN, *Mixed and Hybrid Finite Element Methods*, vol. 15, Springer-Verlag, Berlin, 1991.

[12] R. BRUALDI AND H. RYSER, *Combinatorial Matrix Theory*, Cambridge University Press, 1991.

[13] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[14] T. F. COLEMAN AND A. POTHEN, *The null space problem I. Complexity*, Algebraic Discrete Math., 7 (1986), pp. 527–537.

[15] ———, *The null space problem II. Algorithms.*, Algebraic Discrete Math., 8 (1987), pp. 544–563.

[16] N. DEO, G. M. PRABHU, AND M. S. KRISHNAMOORTHY, *Algorithms for generating fundamental cycles in a graph*, ACM, Trans. Math. Softw.,, 8 (1982), pp. 27–42.

[17] I. S. DUFF, A. M. ERISMAN, AND J. REID, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, UK, 1989.

[18] I. S. DUFF, N. I. M. GOULD, J. K. REID, J. A. SCOTT, AND K. TURNER, *The factorization of sparse symmetric indefinite equations*, J. Numer. Anal., 11 (1991), pp. 181–204.

[19] G. GALLO AND S. PALLOTTINO, *Shortest path methods: a unifying approach*, Mathematical Programming Study, 26 (1986), pp. 38–64.

[20] J. R. GILBERT AND M. T. HEATH, *Computing a sparse basis for the null space*, Algebraic Discrete Math., 8 (1987), pp. 446–459.

[21] P. H. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, London, UK, 1981.

[22] G. H. GOLUB AND G. MEURANT, *Matrices, moments and quadrature II; how to compute the norm of the error in iterative methods*, BIT, 37 (1997), pp. 687–705.

[23] A. GREENBAUM, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, PA, 1997.

[24] C. A. HALL, *Numerical solution of Navier-Stokes problems by the dual variable method*, SIAM, J. Alg. Disc. Meth., 6 (1985), pp. 220–236.

[25] M. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[26] HSL, *Harwell Software Library. A collection of Fortran codes for large scale scientific computation*. http://www.cse.clrc.ac.uk/Activity/HSL, 2000.

[27] A. ITAI AND M. RODEH, *Finding a minimum circuit in a graph*, SIAM, J. Comput., 7 (1978), pp. 413–423i.

[28] L. KETTUNEN, K. FORSMAN, AND A. BOSSAVIT, *Gauging in Whitney spaces*, IEEE Trans. Magnetics, 35 (1999), pp. 1466–1469.

[29] G. L. MILLER, S.-H. TENG, W. THURSTON, AND S. A. VAVASIS, *Geometric separators for finite-element meshes*, SIAM J. Sci. Comput., 19 (1998), pp. 364–386.

[30] G. L. MILLER AND W. THURSTON, *Separators in two and three dimensions*, in STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing, New York, NY, USA, 1990, ACM Press, pp. 300–309.

[31] K. G. MURTHY, *Network Programming*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[32] J. C. NEDELEC, *Mixed finite elements in $\boldsymbol{R}^3$*, Numer. Math., 35 (1980), pp. 315–341.

[33] A. POTHEN, *Sparse null basis computations in structural optimization*, Numer. Math., 55 (1989), pp. 501–519.

[34] R. SCHEICHL, *A decoupled iterative method for mixed problems using divergence-free finite element*, Tech. Report maths0011, University of Bath, 2000.

[35] ———, *Iterative Solution od Saddle Point Problems Using Divergence -free Finite Elements with Applications to Groundwater Flow*, PhD thesis, University of Bath, 2000.

[36] J. R. SHEWCHUCK, *A two-dimensional quality mesh generator and Delaunay triangulator*. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/quake/public/www/triangle.html, 1996.

[37] Z. STRAKOŠ AND P. TICHÝ, *On error estimation by conjugate gradient method and why it works in finite precision computations*, Electron. Trans. Numer. Anal., 13 (2002), pp. 56–80, http://etna.mcs.kent.edu/vol.13.2002/pp56-80.dir/pp56-80.html.

[38] ———, *Error estimation in preconditioned conjugate gradients*, BIT, (to appear).

[39] R. E. TARJAN, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.