SUPPORT GRAPH PRECONDITIONERS FOR SPARSE LINEAR SYSTEMS

A Thesis

by

RADHIKA GUPTA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2004

Major Subject: Computer Science

SUPPORT GRAPH PRECONDITIONERS FOR SPARSE LINEAR SYSTEMS

A Thesis

by

RADHIKA GUPTA

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

December 2004

Major Subject: Computer Science

ABSTRACT

Support Graph Preconditioners for Sparse Linear Systems. (December 2004)

Radhika Gupta, B.E., Indian Institute of Technology, Bombay;

M.S., Georgia Institute of Technology, Atlanta

Chair of Advisory Committee: Dr. Vivek Sarin

Elliptic partial differential equations that are used to model physical phenomena give rise to large sparse linear systems. Such systems can be symmetric positive definite and can be solved by the preconditioned conjugate gradients method. In this thesis, we develop support graph preconditioners for symmetric positive definite matrices that arise from the finite element discretization of elliptic partial differential equations. An object oriented code is developed for the construction, integration and application of these preconditioners. Experimental results show that the advantages of support graph preconditioners are retained in the proposed extension to the finite element matrices.

To my parents

ACKNOWLEDGMENTS

I would like to express sincere thanks to my advisor, Dr. Vivek Sarin, for his invaluable guidance, motivation, support, and patience. He has played a central role in guiding my thesis efforts. His constant encouragement and insightful discussions were instrumental in making this work successful. I appreciate the freedom he has given me while working with him.

I would also like to thank my committee members, Dr. Paul Nelson and Dr. N. K. Anand, for their help and support. I thank Dr. Bart Childs for consenting to review the thesis.

I would like to thank Bheem, Hemant, Juttu, Thomas, Kasturi and Xue for research discussions. I would like to thank my friends, Sangeeta, Vaddi, Anup, Aravind, and Vivek for making my stay at College Station an enjoyable and fun experience.

Lastly, I take this opportunity to thank my parents and brothers for their unending love and support. I would also like to thank my husband Hemant for keeping alive the motivational factor, his constructive criticism and aggressive encouragement. He has always been my constant companion in those wee hours.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Physical phenomena are modeled by equations that relate several partial derivatives of physical quantities, such as velocity, momentum, force, energy, temperature, etc. Such partial differential equations are one of the biggest source of sparse linear systems. We are especially interested in large linear systems that arise from the discretization of elliptic partial differential equations by finite element method. These systems are sparse and symmetric positive definite (SPD) in nature. They arise in computational fluid dynamics (CFD) applications while simulating the fluid flow. Examples include diffusion, incompressible and irrotational fluid flows, flow through porous media, heat conduction, pressurized membranes and circulation of fluid flow.

Iterative methods are very popular and are preferred for solving such large sparse linear systems. They often require less memory and computational effort, and are more parallelizable compared to direct methods. For large three dimensional problems they are necessary due to the prohibitive computational complexity of direct methods. Conjugate Gradients (CG) is a Krylov subspace based iterative method that is used to solve SPD systems. Preconditioning is further used to increase the rate of convergence of CG. In general, preconditioners are application dependent and the reliability of results for a preconditioned system depends a lot on the quality of the preconditioner used. This makes preconditioning a challenging task. Some of the most common preconditioners used for CG are diagonal scaling and incomplete Cholesky factorization. A relatively new and underdeveloped class of preconditioners is the class of support graph preconditioners.

The journal model is SIAM Journal on Matrix Analysis and Applications.

In support graph preconditioning, the graph of the coefficient matrix is used to develop the graph of the preconditioner. The coefficient matrix is denoted by A and the preconditioner by M. Edges in M are chosen such that they are a subset of the actual graph, with the nodes still maintaining connectivity. In this thesis, we have developed a type of support graph preconditioner for the linear systems arising from elliptic partial differential equations.

Support theory began more than a decade ago with the work of Pravin Vaidya, in which he proposed and analyzed maximum weight spanning tree preconditioners for Laplacian matrices [17, 5]. Later, Gremban [11] extended this work for generalized Laplacian matrices and devised new parallel hierarchical support tree preconditioners. Bern, et al [2] used support graph preconditioning to analyze two classes of existing preconditioners, namely modified incomplete-Cholesky and multilevel diagonal scaling preconditioners. Boman, et al [3] provided a framework for bounding the extreme eigenvalues and condition numbers for symmetric positive semidefinite (SPSD) matrices instead of only M-matrices. This analysis was further used by Boman, et al [4, 5] to construct and implement maximum weight basis (MWB) preconditioners for diagonally dominant symmetric matrices.

Most of the earlier work is applicable to M-matrices and diagonally dominant matrices only. The approach presented in this thesis is applicable to SPD matrices arising from finite element discretization of partial differential equations. Such matrices are not M-matrices due to positive off diagonal terms and lack of diagonal dominance. Finite element methods are good for problems with complex geometries or with strongly varying internal properties or when there is a need to track internal boundaries. Finite elements were also chosen for the accuracy that can be obtained from their low-order approximations.

The thesis is organized in following manner. Chapter II introduces the necessary mathematical background on iterative methods, support graph theory and finite element method. It also summarizes the prior known work done in the field of support graph theory. Chapter III outlines the proposed scheme, its implementation, and its analysis. Chapter IV describes the software design of the object oriented code. A set of experimental results are presented in Chapter V to study the effectiveness of the proposed scheme. Chapter VI provides a summary of this research work.

CHAPTER II

MATHEMATICAL BACKGROUND

One of the most common examples of elliptic partial differential equations encountered in various areas of engineering is the Poisson's equation. The two dimensional Poisson's equation with Dirichlet boundary conditions is given by

$$\begin{aligned} -\nabla \cdot (p(x,y)\nabla u) &= f(x,y) \quad \text{in} \quad \Omega, \\ u &= g \qquad\quad\ \ \text{on} \quad \partial\Omega, \end{aligned} \tag{2.1}$$

where u is the physical quantity, $p(x,y)$ determines the isotropy of the problem, f is external forcing function, Ω is an arbitrary domain, g is the boundary data, and $\partial\Omega$ is the boundary of domain Ω. The Laplace equation is obtained when $f(x,y) = 0$. A general approach to solve these equations is to discretize them, i.e., approximate them by a set equations that involve a finite number of unknowns, and then use iterative methods to solve the resulting linear system.

A. Iterative Methods for the Solution of Linear Systems

Consider solving

$$Ax = b, \tag{2.2}$$

for the unknown x. An iterative method starts with an initial estimate x_0 for the solution and successively improves on it at each iteration. Typically, iterations are terminated when the estimate at the i^{th} step, x_i, is close enough to the solution, i.e., the relative residual norm is smaller than a specified tolerance ϵ:

$$\frac{||b - Ax_i||}{||b||} \leq \epsilon.$$

Classical iterative methods include Jacobi, Gauss Seidel, Successive Over Relaxation, etc. The details of these methods can be found in many textbooks on iterative methods [9, 14]. Our interest is in Krylov subspace based methods for symmetric positive definite (SPD) systems such as the method of conjugate gradients (CG).

1. Conjugate Gradients Method

The conjugate gradients method for solving a linear system of equations

$$Ax = b$$

is an algorithm for finding the local minimum of a quadratic function [1],

$$\Phi = \frac{1}{2}x^T Ax - x^T b.$$

When A is SPD, it can be shown that minimization of Φ is equivalent to solving the linear system $Ax = b$, provided the gradient of Φ can be computed. The minimization takes place over a certain vector space called the Krylov subspace, $\mathcal{K}$ defined by

$$\mathcal{K}_k(A, b) = \text{span}\{b, Ab, A^2b, \ldots, A^{k-1}b\}. \tag{2.3}$$

At the k^{th} iteration, the search direction p_k is selected such that Φ is minimized along the direction d_k where

$$d_k = x_k + \left(\frac{r_k^T \cdot r_k}{p_k^T \cdot Ap_k}\right) p_k.$$

The algorithm uses search directions that are conjugate (or A-orthogonal) to all the previous search vectors:

$$p_k^T \cdot Ap_j = 0.$$

It can also be shown that the residuals are orthogonal:

$$r_k^T \cdot r_j = 0, \quad \forall \;\; j \leq k.$$

The Krylov subspace will eventually cover the whole space (or the space spanned by the eigenvectors of A), and the method will give the exact solution after n steps, where n is the order of A. In presence of rounding errors, the generated vectors will not be exactly orthogonal. If A has m distinct eigenvalues, then CG requires m iterations to converge to a solution. Thus, clustering of the eigenvalues of A can improve the convergence of CG. The CG algorithm is summarized in Algorithm 1.

Algorithm 1 Conjugate Gradients method for solving $Ax = b$.

```
Require: $x_0$, $\epsilon$ and maxiter
 1: $k = 0$
 2: $r_0 = b - Ax_0$, $p_0 = r_0$
 3: while $||r_k||/||r_0|| > \epsilon$ or $k \leq$ maxiter do
 4:     $\alpha_k = r_k^T \cdot r_k / p_k^T \cdot Ap_k$          $\Longrightarrow$ Step length
 5:     $x_{k+1} = x_k + \alpha_k p_k$                      $\Longrightarrow$ Update Solution
 6:     $r_{k+1} = r_k - \alpha_k Ap_k$                     $\Longrightarrow$ Update residual
 7:     $\beta_k = r_{k+1}^T r_{k+1} / r_k^T r_k$           $\Longrightarrow$ Improvement step
 8:     $p_{k+1} = r_{k+1} + \beta_k p_k$                   $\Longrightarrow$ Search direction
 9:     $k = k + 1$
10: end while
11: $x = x_k$
```

The rate of convergence of CG is given by

$$\frac{||e_k||_A}{||e_0||_A} \leq 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k \tag{2.4}$$

where e_k is the error at the k^{th} iteration and $\kappa(A)$ is the condition number of A [9].

The condition number of an SPD matrix is defined as

$$\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}. \tag{2.5}$$

To reduce the relative A-norm of the error below the tolerance ϵ the method requires the following number of iterations

$$i \leq \left\lceil \frac{1}{2}\,\sqrt{\kappa(A)}\,\ln\,\left(\frac{2}{\epsilon}\right)\right\rceil. \tag{2.6}$$

From the above equation, one can see that number of iterations required by CG is $O(\sqrt{\kappa(A)})$.

One can improve the convergence of CG by reducing the condition number $\kappa(A)$ through the use of preconditioning. With M as the preconditioner, the preconditioned system becomes

$$M^{-1}Ax = M^{-1}b. \tag{2.7}$$

Thus, the convergence rate of the preconditioned conjugate gradients method (PCG) is

$$\frac{||e_k||_{M^{-1}A}}{||e_0||_{M^{-1}A}} \leq 2\left(\frac{\sqrt{\kappa(A,M)}-1}{\sqrt{\kappa(A,M)}+1}\right)^k$$

where $\kappa(A, M)$ is called the generalized condition number of the ordered pair of matrices (A, M) and is defined as

$$\begin{aligned}\kappa(M^{-1}A) &= \frac{\lambda_{max}(M^{-1}A)}{\lambda_{min}(M^{-1}A)}\\ &= \lambda_{max}(M^{-1}A)\ \lambda_{max}((M^{-1}A)^{-1})\\ &= \lambda_{max}(M^{-1}A)\ \lambda_{max}(A^{-1}M)\end{aligned} \tag{2.8}$$

From the above equation, one can see that the upper bound on $\lambda_{max}(A^{-1}M)$ will give a lower bound on $\lambda_{min}(M^{-1}A)$. Thus, if one can develop techniques to compute the upper bound on the largest eigenvalue of $M^{-1}A$, then by exchanging the roles of A

and M the lower bound can be obtained as well.

B. Support Graph Theory

The main idea behind support graph theory is to use a subgraph of the graph of A as a preconditioner. These graphs are connected graphs, with the preconditioner being a subset of the actual graph edges. It is interpreted as if the preconditioner graph edges *support* the edges in the actual graph.

Let us denote $G(A)$ as the graph of the coefficient matrix A and $G(M)$ as the graph of the preconditioner matrix M. One such example is shown in Fig. 1. $G(A)$ is represented by union of dotted and solid edges. $G(M)$ is represented by the solid edges only. The two edges e and f are such that $e \in G(M)$ and $f \in G(A \backslash M)$. $G(A \backslash M)$ is a shorthand for $G(A) \backslash G(M)$. For an edge $f \in G(A \backslash M)$ between nodes i and j, one can specify a *support path* in $G(M)$ from i to j.

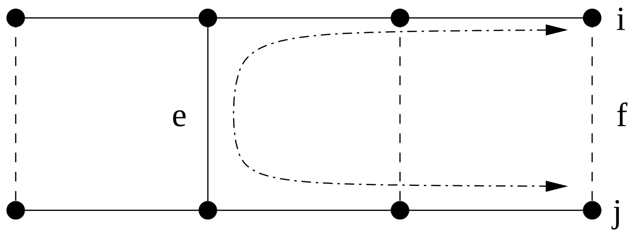

Fig. 1. Simple support graph.

Basic definitions and concepts used in support graph theory are defined in different forms by several authors in [2, 3, 10, 12]. Given below is a collection of some common definitions with consistent notation.

- Dilation: The *dilation* of an edge $f \in G(A \backslash M)$, denoted by $d(f)$ is the number of edges in its support path. In our example of Fig. 1, dilation for f is 5 for the path shown by the dotted line with arrows. Dilation of the entire graph, $d(G(A\backslash M))$ is the largest dilation over all the edges in $G(A\backslash M)$.

- Congestion: The *congestion* of an edge $e \in G(M)$, denoted by $c(e)$ is the number of support paths that include e, i.e., the number of edges of $G(A\backslash M)$ supported by e. In the above example, congestion for e is 3. The congestion for the entire graph $c(G(M))$ is defined as the maximum edge congestion taken over all the edges in $G(M)$. In a weighted graph, congestion is defined as the ratio of sum of the weights of the support paths that include e to the weight of e.

- Condition number: The condition number of the preconditioned matrix is bounded by the product of congestion and dilation of the graph i.e.

$$\kappa(M^{-1}A) \leq c(G(M))\ d(G(A\backslash M)) \tag{2.9}$$

  This is also known as the congestion-dilation lemma [3, 10, 12]. A detailed proof is included in Chapter III.

- Generalized eigenvalue: λ is said to be a finite generalized eigenvalue of the ordered pair of matrices (A, M) if there exists a vector x such that $M^{-1}Ax = \lambda x$, and $Mx \neq 0$ [10]. The set of generalized eigenvalues is denoted by $\lambda(A, M)$.

- Support lemma [10]: Suppose A and M are SPD matrices, and τ is a real number. If $\tau M - A$ is positive semidefinite matrix, then $\lambda_{max}(M^{-1}A) \leq \tau$.

- Support: The support $\sigma(A, M)$, of matrix M for A is the largest lower bound over all the τ, satisfying the support lemma [11, 4], i.e,

$$\sigma = \min\{\tau : \tau M - A \text{ is positive semidefinite}\}.$$

  If there is no τ, then $\sigma(A, M) = \infty$.

- Splitting lemma [2]: If $Q = Q_1 + Q_2 + \cdots + Q_m$, where $Q_1, Q_2, \ldots, Q_m$ are all positive semidefinite, then Q is positive semidefinite. Further it can proved that,
$$\sigma(A, M) \leq \max\{\sigma(A_i, M_i)\}$$
where $A = \sum_{i=1}^{m} A_i$ and $M = \sum_{i=1}^{m} M_i$, satisfy the splitting lemma.

1. Related Work

Support theory began with the work of Vaidya [17] in early nineties. He proposed several families of preconditioners. The first one was based on maximum spanning tree (MST) of the underlying graph of the matrix. The second one augmented the MST with extra edges. The third one was based on a maximum weight basis (MWB) of the matriod associated with the graph of the matrix. The first two families were applicable only to M-matrices[1], and the third one was applicable to diagonally-dominant[1] symmetric matrices. Chen et al [2, 5, 6] provided an extensive implementation and evaluation of Vaidya's preconditioners. Although Vaidya did not formally published any of his work, it has led to research in several directions.

Gremban et al [10, 11] extended Vaidya's techniques for generalized Laplacian matrices and devised parallel support tree preconditioners. These Laplacian matrices are SPD, diagonally dominant and have non-positive off-diagonal elements. The preconditioner is constructed as a tree in a space of higher dimension than the original matrix. If one tries to visualize a planar mesh, then the preconditioner will be *sticking out* in the third dimension. It was named support tree because the mesh appears to be supported by this tree. In general these preconditioners have more nodes than the original mesh but fewer edges. The support tree is constructed by recursive graph

[1]Defined in Appendix A.

partitioning of the actual graph until only singleton sets of nodes are left, which form leaves of the support tree. A graph partitioning approach deletes a subset of edges of the graph such that the resulting subgraphs contain roughly the same number of nodes. For a one dimensional problem with a mesh with n nodes, the tree will have a depth of $\log n$ with $n-1$ internal nodes and n leaves. The leaf nodes are numbered upto n, coinciding with the n mesh nodes, and the internal nodes are numbered from $n+1$ to $2n-1$. The graph partitioning scheme results in a preconditioner of larger size, $(2n-1)^2$ as compared to n^2. Thus the original system is augmented with zeros to solve the problem and then the extra variables are thrown away. The tree structure allows all the nodes at a given level to be evaluated in parallel. A two dimensional mesh of size $n \times n$ requires $2\lceil \log\ n \rceil$ parallel steps with an average of $n^2/\log\ n$ nodes evaluated at each step. When support tree preconditioners were compared with preconditioners arising from diagonal scaling and incomplete Cholesky (IC) decomposition, it was found that support tree preconditioners were faster and more parallelizable than diagonal scaling and IC preconditioners. Gremban, et al [11] also introduced linear algebra tools for bounding eigenvalues and finding theoretical bounds on the convergence rate of PCG. They defined the concept of *support* as defined earlier in this section and showed that

$$\kappa(M^{-1}A) \leq \sigma(A, M)\ \sigma(M, A).$$

They further proved that support tree preconditioners have a generalized condition number bound of $O((\text{dim})^2 n \log n)$ for a dim dimensional regular mesh of n^{dim} nodes.

Later Bern et al [2] extended the basic linear algebra tools for analyzing support graph preconditioners and used them to analyze modified incomplete Cholesky and multilevel diagonal scaling preconditioners. They also provided theoretical bounds on Vaidya's preconditioners. As already stated, Vaidya's first family of preconditioners was based on MST of the associated graph. Suppose the number of nonzeros in a $n \times n$ matrix is denoted by nnz. The cost of constructing the preconditioner is $O(\text{nnz} + n\log\, n)$, if implemented by an efficient maximum spanning tree algorithm. Its factorization has $O(\text{nnz})$ cost and produces no fill. The condition number is bounded by $\kappa = O(\text{nnz} \cdot n) \approx O(n^2)$. The second family of the augmented spanning tree achieves a better condition number, but it is more expensive to compute and factor. The preconditioner can be constructed by first forming the MST and then splitting it into t connected components of roughly the same size, where t is an integer parameter. The heaviest edge between every pair of subtrees is added to the preconditioner. Nothing is added if there are no edges present or if the heaviest edge is already in the tree. This gives the condition number bound of $O(n^2/t^2)$. The factorization cost of the preconditioner is $O(n + t^6)$ with $O(n + t^4)$ non zeros. These preconditioners are not parallelizable because the long diameter of the tree creates long chains of dependencies in the triangular factors. It was shown in [5, 6] that, within the class of symmetric diagonally dominant matrices, Vaidya's preconditioners are sensitive only to the nonzero structure of the coefficient matrix A and not to the values of its entries. Unlike IC, they converge at a constant rate on a variety of two dimensional problems and are almost unaffected by boundary conditions and the direction of anisotropy in anisotropic problems. For some three dimensional problems, they deliver poor performance compared to IC [6].

Boman and Hendrickson [4] provided a detailed set of linear algebra tools and techniques for bounding the extreme eigenvalues and the condition number for ordered pairs of matrices (A, M). They considered SPSD matrices instead of just SPD matrices. From the prior definitions, we can see that $\sigma(A, M)$ is an upper bound on the largest finite generalized eigenvalue of (A, M) i.e. $\lambda \leq \sigma(A, M)$. They called σ as the support number and used it instead of finite generalized eigenvalues. They proved that support numbers are well defined under rank deficiency and are more robust than generalized eigenvalues in that sense. These techniques were further used by Boman et al [4] to analyze Vaidya's third family of preconditioners called maximum weight basis (MWB) preconditioners. In MWB preconditioners, $A = \sum u_i u_i^T$ is represented as a sum of rank-1 matrices, where each small matrix corresponds to one edge of the underlying graph $G(A)$. The columns of u can be used to define a structure called *matroid*. The preconditioner $M = VV^T$ is constructed by considering a matrix V as a basis of u_i's that maximizes the trace of V^TV. This corresponds to finding a maximal independent set in a *matriod*. Vaidya suggested choosing a set of vectors that are linearly independent and have the largest possible norm. This corresponds to a maximum weight basis, which is a maximal independent set in a weighted matroid. If A is an M-matrix, the maximum weight basis is simply a maximum spanning tree. The condition number of MWB preconditioners is bounded by $O(4 \text{ nnz } n)$. The nnz corresponds to the number of non zero entries of the strictly upper triangular part of the matrix A.

To the best of our knowledge, the technique of support graph preconditioning is limited to the class of M-matrices and diagonally dominant matrices. These ideas are not applicable to SPD matrices arising from finite element discretization of elliptic partial differential equations. This thesis develops support graph preconditioning techniques for finite element matrices.

C. Finite Element Method

Numerical solutions are obtained by discrete approximations to continuous problems. Differences in the choice of discretization lead to different solution schemes. We use finite element method to discretize Eq. (2.1) to obtain the linear system in

Eq. (2.2). In this method, the region of interest is divided into a finite number of subregions called *elements*. We choose an unstructured mesh formed of triangular *elements* on a unit square ($\Omega \in (0,1) \times (0,1)$) to define our model problem. An example is shown in Fig. 2.

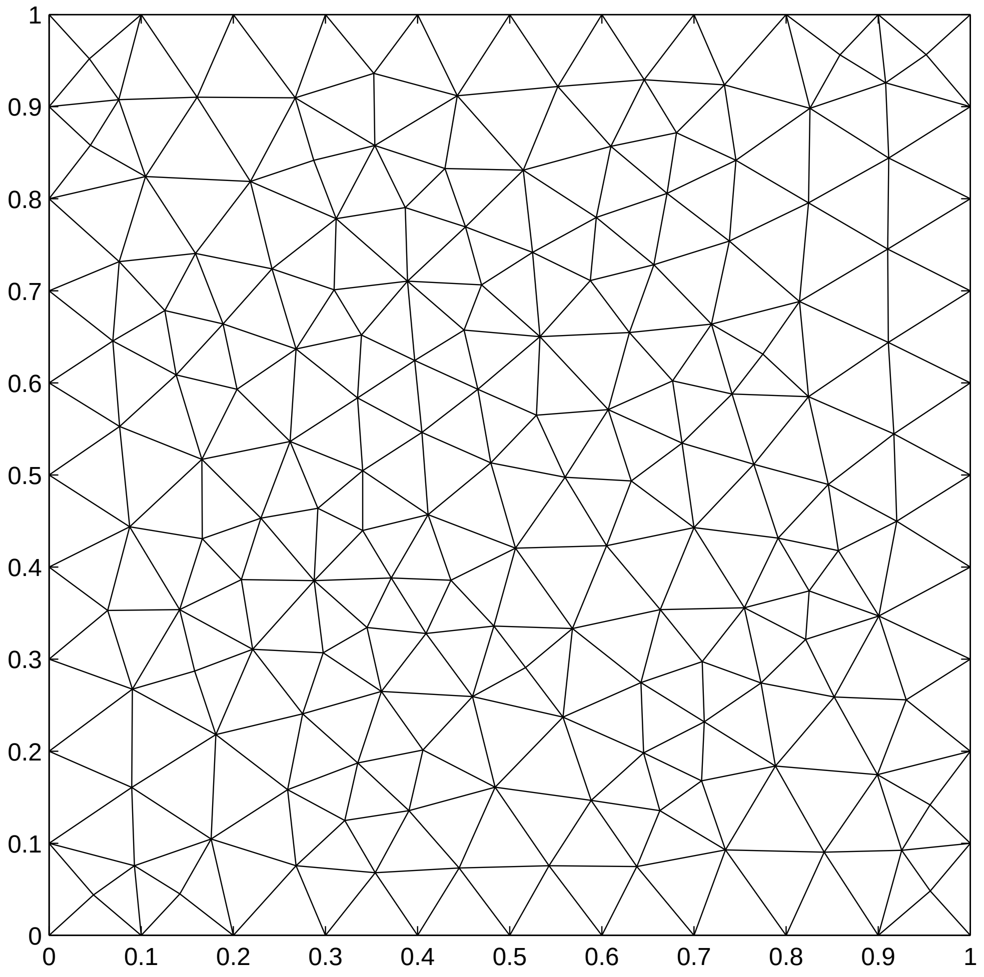

Fig. 2. An unstructured mesh with 185 nodes and 328 triangles is used to discretize a unit square domain.

The discrete system of equations is derived by multiplying the original PDE over each element by a test function v, and integrating over the physical domain.

The resulting equation is known as the variational formulation. It is also known as the weak form because performing integration on the diffusive term (Δ) reduces the continuity requirements of the system. For Eq. (2.1) the weak form is given by

$$\iint\limits_{\Omega} p(x,y)\ \nabla u \cdot \nabla v\ dx\ dy = \iint\limits_{\Omega} f(x,y)\ v\ dx\ dy + \int\limits_{\partial\Omega} p(x,y)\ v\frac{\partial u}{\partial \mathbf{n}} ds, \tag{2.10}$$

where $\dfrac{\partial u}{\partial \mathbf{n}} = \nabla u \cdot \mathbf{n}$, $\mathbf{n}$ is the unit normal direction pointing outward of the boundary, and ds is an arc length of an infinitesimal element along the boundary. For isotropic problems $p(x,y)$ is the identity matrix.

The next step is to define the basis functions over the triangulation and approximate u by

$$u = \sum_{j=1}^{n} u_j \phi_j$$

where $u_j = u(x_j, y_j)$ and n is the number of nodes in the mesh. The basis function ϕ is defined as

$$\phi_i(x_j, y_j) = \delta_{ij} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j. \end{cases}$$

By substituting $v = \phi_i$ in Eq. (2.10), the weak form can be written as a set of n algebraic equations

$$\sum_{j=1}^{n} \iint\limits_{\Omega} p(x,y)\ \nabla \phi_j \cdot \nabla \phi_i\ u_j\ dx\ dy = \iint\limits_{\Omega} f(x,y)\ \phi_i\ dx\ dy \quad + \int\limits_{\partial\Omega} p(x,y)\ \phi_i \frac{\partial u}{\partial \mathbf{n}} ds, \qquad i = 1, 2, \ldots, n \tag{2.11}$$

For a triangular element e, these equation are given by

$$\sum_{j=1}^{3} K_{ij}^{e}\ u_j^{e} = F_i^{e}, \tag{2.12}$$

where K^e_{ij} is an entry of stiffness matrix over an element e, and is defined by

$$K^e_{ij} = \iint\limits_e p(x,y)\nabla\phi_i \nabla\phi_j \; dx \; dy, \quad 1 \le i,j \le 3. \tag{2.13}$$

The forcing function F^e_i is defined by

$$F^e_i = \iint\limits_e f(x,y) \; \phi_i \; dx \; dy + \int \frac{\partial u}{\partial \mathbf{n}} \phi_i \; ds. \tag{2.14}$$

A straight forward choice of ϕ are the piecewise linear basis functions. The resultant element stiffness matrix K^e will be of size 3×3. As seen from Eq. (2.13) the entries of the stiffness matrix require calculation of derivatives of the basis functions. It is easier to find these derivatives in a local coordinate system (ξ, η) on a fixed master element [13] as shown by Fig. 3. The geometry is well behaved and one can write expressions for linear functions such that they are unity at one node and vanish at others,

$$\begin{aligned} L_1 &= 1 - \xi - \eta \\ L_2 &= \xi \\ L_3 &= \eta \end{aligned} \tag{2.15}$$

After transforming an element e to a master element, we need to integrate the equation in local coordinates. Change of integration variables can be done by

$$\iint\limits_e z(x,y) \; dx \; dy = \iint\limits_{master} z(x(\xi,\eta), y(\xi,\eta))|J| d\xi d\eta$$

where $z(x,y)$ is an arbitrary function. J is the Jacobian matrix given by

$$|J| = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{vmatrix} = \frac{\text{Area of actual element}}{\text{Area of master element}} = \frac{|Area|}{1/2} = 2|Area| \tag{2.16}$$

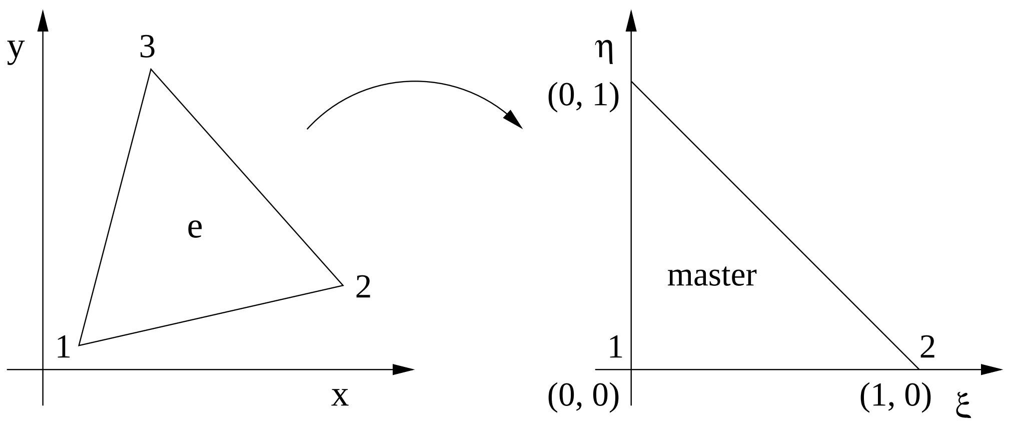

Fig. 3. A piecewise linear finite element transformation from global (x, y) to local (ξ, η) coordinates.

where $|Area|$ is the area of the actual element.

The derivatives of basis functions in global coordinates and local coordinates are related by

$$\begin{pmatrix} \frac{\partial \phi_i}{\partial x} \\ \frac{\partial \phi_i}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{pmatrix} \begin{pmatrix} \frac{\partial L_i}{\partial \xi} \\ \frac{\partial L_i}{\partial \eta} \end{pmatrix}. \tag{2.17}$$

The vector of derivatives of basis functions on the right hand side, can be calculated using Eqs. (2.15). To calculate the vector of derivatives of basis functions on the left hand side, one needs to calculate the transformation matrix. This transformation matrix is the inverse of the Jacobian matrix, J, that is used for the global to local coordinate transformation.

After calculation of the local basis functions, the global basis function ϕ_i is constructed by *piecing* together local basis functions on each element sharing the node i. Fig. 4 shows one such combination for node i. The global function at i is formed by the union of the triangles surrounding the node. The shape looks like a *closed tent.* For simple linear basis functions one could calculate the solution in global space itself, but as we move to higher order elements, e.g., piecewise quadratic basis

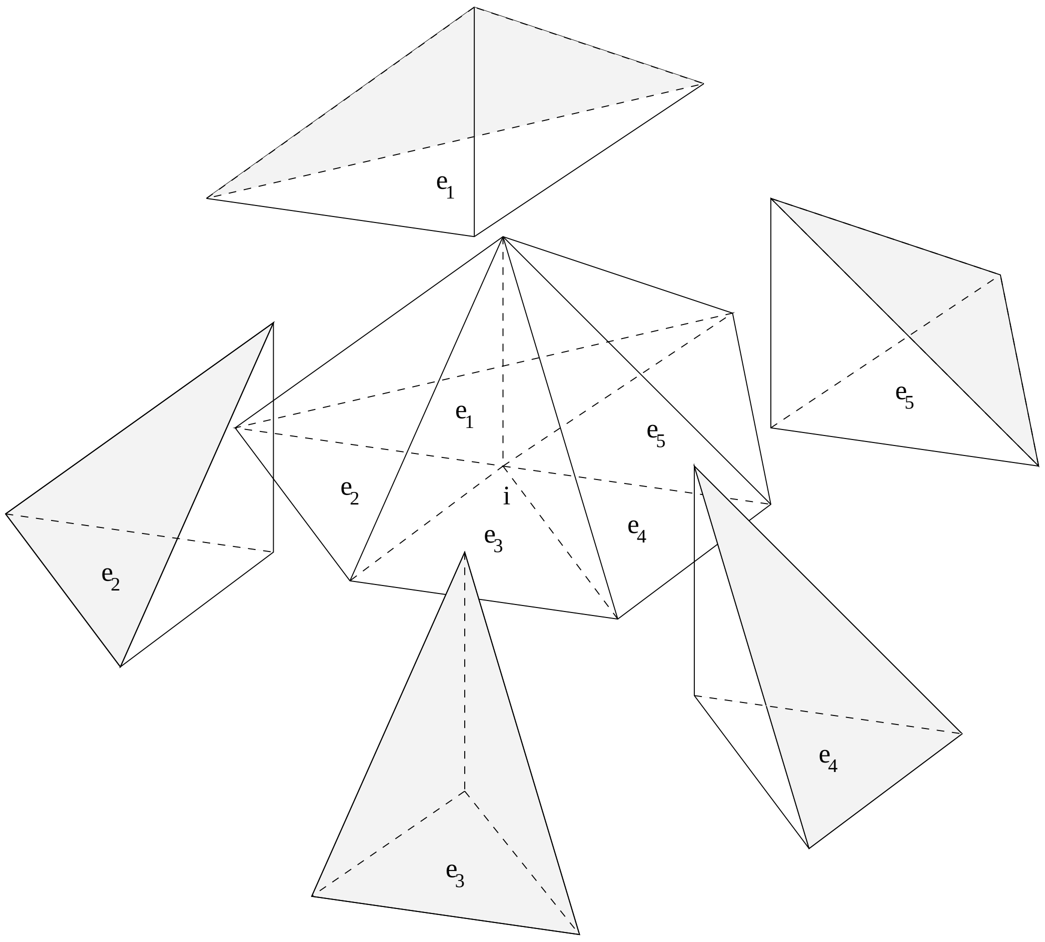

Fig. 4. Combining local basis functions to obtain a global basis function at i^{th} node. The shaded area represents the basis function ϕ_i on each element.

functions, it becomes necessary to use the master element approach.

Finally the global stiffness matrix A is obtained by

$$A = \sum_{e \in T} K^e, \quad b = \sum_{e \in T} F^e \tag{2.18}$$

where T is the set of all triangular elements.

CHAPTER III

THE PROPOSED SCHEME

In this chapter, the construction and analysis of the proposed support graph preconditioners are presented. The global stiffness matrix in Eq. (2.18) may not be an M-matrix. To transform A into an M-matrix, we propose a pre-processing step. We can express the system in Eq. (2.18) as follows

$$\begin{bmatrix} -I & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}, \tag{3.1}$$

such that $A = B^T B$. The matrix B is called the gradient matrix.

A. Pre-processing of the Stiffness Matrix

If one chooses piecewise linear basis functions, then one would get individual element stiffness matrices K^e of size 3×3. These matrices can also be written in terms of the element gradient matrix B^e:

$$K^e = B^{eT}\ B^e, \tag{3.2}$$

where

$$B^e = \begin{pmatrix} \frac{\partial \phi_1}{\partial x} & \frac{\partial \phi_2}{\partial x} & \frac{\partial \phi_3}{\partial x} \\ \frac{\partial \phi_1}{\partial y} & \frac{\partial \phi_2}{\partial y} & \frac{\partial \phi_3}{\partial y} \end{pmatrix}. \tag{3.3}$$

The global gradient matrix B is obtained by stacking up B^e for all the elements of the mesh. The combined expression is given by,

$$B = \begin{pmatrix} B^{e_1} \\ B^{e_2} \\ \vdots \\ B^{e_t} \end{pmatrix}. \tag{3.4}$$

Each element of the mesh contributes to two rows of B, with no particular ordering of the elements. This results in a B of size $2\ t \times n$, where t is the number of elements and n is the number of nodes in the mesh. The resulting global stiffness matrix K is of size $n \times n$.

For each element, a coordinate axis transformation is done from (x, y) to (τ_1, τ_2), where τ_1 and τ_2 are unit vectors parallel to the two edges of the triangle (see, e.g., Fig. 5).

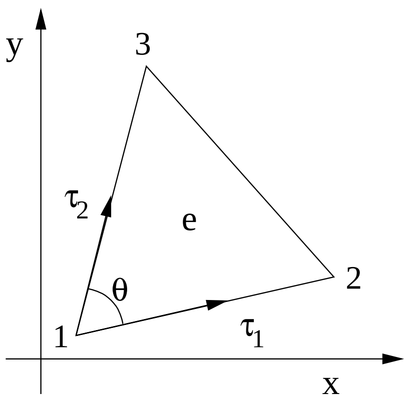

Fig. 5. A piecewise linear finite element transformation from (x, y) to (τ_1, τ_2) coordinate system.

Gradients of ϕ_i along τ_1 and τ_2 are obtained by the transformation

$$\begin{pmatrix} \frac{\partial \phi_i}{\partial \tau_1} \\ \frac{\partial \phi_i}{\partial \tau_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \tau_1} & \frac{\partial y}{\partial \tau_1} \\ \frac{\partial x}{\partial \tau_2} & \frac{\partial y}{\partial \tau_2} \end{pmatrix} \begin{pmatrix} \frac{\partial \phi_i}{\partial x} \\ \frac{\partial \phi_i}{\partial y} \end{pmatrix}. \tag{3.5}$$

The matrix transformation is of the form,

$$B' = G\ B \tag{3.6}$$

where G is a block diagonal matrix, with 2×2 sized blocks corresponding to the transformation for each element.

The two edges for the transformation are chosen such that the condition number of the entire system is minimized. Taking into account anisotropy as well, we chose the set of edges such that the condition number of matrix $G^e P^{e-1} G^{eT}$ is minimized. Here, P^e is the element's anisotropy matrix derived from $p(x, y)$. After this transformation, our linear system of Eq. (3.1) is represented as,

$$\begin{bmatrix} -GP^{-1}G^T & B' \\ B'^T & 0 \end{bmatrix} \begin{bmatrix} G^{-T}y \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}, \tag{3.7}$$

where P is a block diagonal matrix with blocks P^e. For isotropic problems, $GP^{-1}G^T$ reduces to GG^T, and it is beneficial to choose two edges of a triangle such that the angle between them is closest to 90°, i.e., the edges are nearly perpendicular to each other. Diagonal scaling can be used to improve the effectiveness of the resulting preconditioner. The matrix in Eq. (3.7) is scaled as follows:

$$\begin{bmatrix} -DGP^{-1}G^T D & B' \\ B'^T & 0 \end{bmatrix} \begin{bmatrix} D^{-1}G^{-T}y \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}, \tag{3.8}$$

where D is a block diagonal matrix of scaling matrices for each element. The matrix $DGP^{-1}G^T D$ is a block diagonal matrix with blocks of size 2×2 that correspond to individual elements. The element scaling matrices D^e are chosen such that they minimize the condition number of the diagonal blocks of $DGP^{-1}G^TD$, i.e., $D^eG^eP^{e-1}G^{eT}D^e$.

As an illustration, the resulting transformation of an element gradient matrix B^e is given below

$$\begin{aligned} D^eG^eB^e &= \\ & D^e \times \underbrace{\begin{pmatrix} \frac{1}{l_{12}} & 0 \\ 0 & \frac{1}{l_{13}} \end{pmatrix} \begin{pmatrix} \mathrm{x}_2-\mathrm{x}_1 & \mathrm{y}_2-\mathrm{y}_1 \\ \mathrm{x}_3-\mathrm{x}_1 & \mathrm{y}_3-\mathrm{y}_1 \end{pmatrix}}_{G^e} \times \\ & \underbrace{\frac{1}{2|Area|} \begin{pmatrix} \mathrm{y}_2-\mathrm{y}_3 & \mathrm{y}_3-\mathrm{y}_1 & \mathrm{y}_1-\mathrm{y}_2 \\ \mathrm{x}_3-\mathrm{x}_2 & \mathrm{x}_1-\mathrm{x}_3 & \mathrm{x}_2-\mathrm{x}_1 \end{pmatrix}}_{B^e}, \end{aligned} \tag{3.9}$$

where l_{12} and l_{13} are the lengths of edges $(1,2)$ and $(1,3)$ in Fig. 5, respectively, and D^e is a diagonal matrix. The transformed element gradient matrix B^e is

$$\begin{aligned} B^{e'} &= \begin{pmatrix} w_1 & -w_1 & 0 \\ w_2 & 0 & -w_2 \end{pmatrix}, \qquad (3.10) \\ w_1 &= \text{const } (\mathrm{x}_1(\mathrm{y}_3-\mathrm{y}_2)+\mathrm{x}_2(\mathrm{y}_1-\mathrm{y}_3)+\mathrm{x}_3(\mathrm{y}_2-\mathrm{y}_1)) \\ w_2 &= \text{const } (\mathrm{x}_1(\mathrm{y}_3-\mathrm{y}_2)+\mathrm{x}_2(\mathrm{y}_1-\mathrm{y}_3)+\mathrm{x}_3(\mathrm{y}_2-\mathrm{y}_1)) \end{aligned}$$

where w_1 and w_2 can be viewed as the modified weights of the edges along τ_1 and τ_2. Note that

$$B^{e'T}B^{e'} = \begin{pmatrix} w_1^2+w_2^2 & -w_1^2 & -w_2^2 \\ -w_1^2 & w_1^2 & 0 \\ -w_2^2 & 0 & w_2^2 \end{pmatrix} \tag{3.11}$$

is an M-matrix.

Consider the system

$$\begin{bmatrix} -I & B' \\ B'^T & 0 \end{bmatrix} \begin{bmatrix} y' \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}. \tag{3.12}$$

The stiffness matrix for this system is given as

$$A' = \sum_{e=1}^{t} B^{e'T} B^{e'} \tag{3.13}$$

Note that

$$|a'_{ii}| \geq \sum_{j \neq i} |a'_{ij}|, \quad a'_{ij} \leq 0, \text{ for } i \neq j, \quad \text{and} \quad a'_{ii} > 0, \text{ for } i = 1, \ldots, n \tag{3.14}$$

Thus, A becomes a diagonally dominant matrix.

B. Overview of the Scheme

The procedure for construction and implementation of support graph preconditioners (SGP) is outlined in Algorithm 2. Over the domain Ω, an unstructured triangulated mesh is generated with d subdomains. Fig. 6 shows an example of a unit square domain partitioned into four subdomains. The advantage of choosing an unstructured triangular mesh for discretization is the flexibility in representing complex or uneven geometry. Once the mesh is ready, we can compute the element gradient matrix B and pre-process it to produce B'. In the graph corresponding to B, all the nodes and edges of the original mesh are present, whereas the graph corresponding to B' contains all the nodes but only two edges per element of the original mesh.

1. Preconditioner: MST Inside Each Subdomain

To improve the convergence of iterative method, preconditioners are used. A good preconditioner, should be inexpensive to compute, should approximate the coefficient

Algorithm 2 Construction and implementation of SGP, and solution of the system.

```
1: Generate the mesh by partitioning the entire domain into few subdomains $d$.
2: Compute $B$ using FEM.
3: Apply matrix transformation to compute $B'$.
4: Use Prim's algorithm to get MST in each subdomain $d$, add edges on the interface
   between two subdomains, and assemble the preconditioner $M$.
5: Apply boundary conditions.
6: Perform minimum degree ordering of $M$ to minimize fill.
7: Factorize $M$ using Cholesky factorization.
8: Solve the system using preconditioned conjugate gradients.
```

matrix closely and should be easy to factorize. Maximum spanning tree (MST) edges are used to support the modified graph and act as the support graph preconditioner. An efficient spanning tree algorithm can be used to find MST. We use Prim's algorithm [7] and modify it to find the maximum weight spanning tree. The modified algorithm is given by Algorithm 3. For a graph $G(A)$ corresponding to matrix A, $V(G(A))$ and $E(G(A))$ represent the node and edge sets, respectively.

MST is obtained for each subdomain d_i. Subdomain boundary edges are forced to be part of the support graph. The advantage of using MST is that it does not introduce any fill during factorization. In the given case, MSTs are present inside each subdomain. This will introduce fill only due to the skeletal structure of the subdomains resulting from the boundary edges of each of the subdomain d. Since the calculation of MSTs can be done concurrently, one can develop a parallel algorithm with ease.

For Algorithm 3, the *while* loop in Step 7 will run $|V|$ times and the for loop in Step 9 will run $O(E)$ times. Here, V and E are the node and edge sets for each

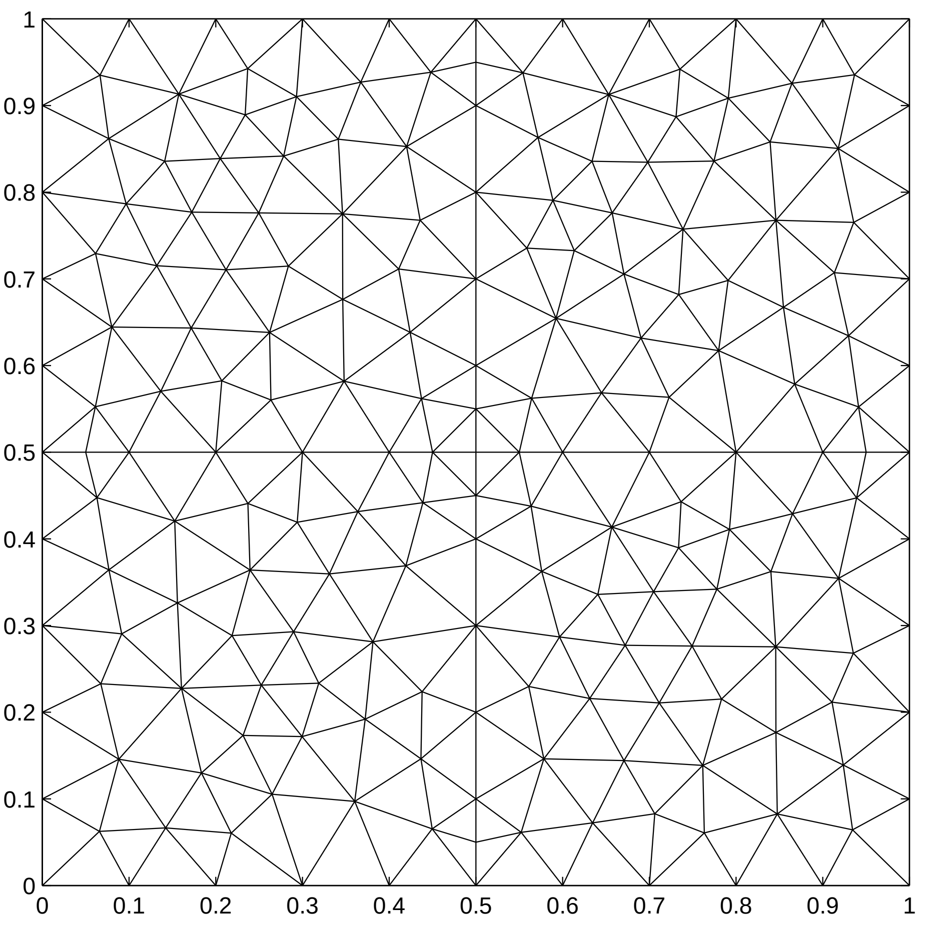

Fig. 6. An unstructured mesh with 199 nodes, 356 triangles, divided into 4 subdomains. Each subdomain has nearly 57 nodes and 7 edges on the boundary with other subdomains.

subdomain, respectively. Thus, the computational complexity of the algorithm is

$$O(d\ V\ \log V + d\ E\ \log V) \approx O(d\ E\ \log V) \approx O\left(n\ \log\left(\frac{n}{d}\right)\right)$$

where, V and E are approximately $O(n/d)$.

The preconditioner M is available after executing Algorithm 3. The corresponding graph $G(M)$ will have n vertices, with an edge present between vertex i and j if the corresponding entry $M_{ij} \neq 0$.

Algorithm 3 Prim's algorithm modified for MST.

```
1: for $u \in V(G(B'^T\ B'))$ of subdomain $d$ do
2:     $key[u] \leftarrow -\infty$
3:     $\pi[u] \leftarrow$ NIL
4: end for
5: $key[root] \leftarrow 0$
6: $Q \leftarrow V(G(B'^T\ B'))$ of subdomain $d$          $\Longrightarrow$ Steps 1-6: $O(V)$ complexity
7: while $Q$ is not empty do
8:     $u \leftarrow extract\_max(Q)$          $\Longrightarrow O(\log V)$ complexity
9:     for $v \in Adj[u]$ do
10:        if $v \in Q$ and $weight(u, v) > key[v]$ then
11:            $\pi[v] \leftarrow u$
12:            $key[v] \leftarrow weight(u, v)$          $\Longrightarrow O(\log V)$ complexity
13:        end if
14:    end for
15: end while
```

2. Boundary Conditions

The mathematical model is complete once the boundary conditions are defined. Consider a function u that satisfies Eq. (2.1) in domain Ω with smooth boundary $\partial\Omega$. The following types of linear boundary conditions are possible.

1. Dirichlet boundary condition: The function u is known on the boundary:

$$u(x, y) = g(x, y), \quad \text{on} \quad \partial\Omega$$

   where g is known. For Laplace equation, this boundary condition will always ensure unique solution.

2. Neumann boundary condition: The normal derivative of u is known on the boundary:

   $$\frac{\partial u}{\partial \mathbf{n}} = h(x,y), \quad \text{on} \quad \partial\Omega$$

   where h is known. The problem is ill-posed, until the compatibility condition is satisfied. For Laplace equation we have,

   $$\int_{\partial\Omega} \frac{\partial u}{\partial \mathbf{n}} ds = 0. \tag{3.15}$$

   The above condition should be satisfied at equilibrium. It does not guarantees the uniqueness of the solution and the solution can differ by an arbitrary constant. In practice, one usually specifies a simple reference value of u at a point on Ω so as to ensure a unique solution.

3. Dirichlet and Neumann boundary conditions: One can specify Dirichlet boundary condition on some parts of the boundary and Neumann on the remainder of the boundary.

In the present scheme, we include the boundary values as unknowns and modify the assembled system to incorporate the boundary values.

3. Ordering and Factorization

The preconditioning step includes factorization of the sparse matrix M. Since the support graph preconditioner is SPD, Cholesky factorization can be used:

$$M = LL^T, \tag{3.16}$$

where L is a lower triangular matrix. At each step of the factorization, a vertex is eliminated from the graph. Neighbors of the eliminated vertex form a *clique*, which can create nonzero entries in M. These new non zeros are called *fill* of the matrix. To minimize fill, the rows and columns of the matrix can be reordered before factorization. A commonly used technique called minimum degree ordering, eliminates nodes in order of increasing neighbors. Once the ordering is decided, the location of all the fill entries in L can be determined prior to numerical factorization. For SPD matrices, only the location of non zero entries is important. Since pivoting is not required for numerical stability, one does not need to know the numerical value at this time. This process of predetermining the non zero structure of the factors is called *symbolic* factorization. Symbolic factorization helps in setting up an efficient static data structure prior to numerical factorization. The process of factorization is summarized in Algorithm 4.

Algorithm 4 Sparse Cholesky factorization.

```
1: Ordering: Perform minimum degree ordering.
2: Symbolic factorization: Determine the nonzero structure of Cholesky factor $L$.
3: Numerical factorization: Compute the actual numerical values in $L$.
```

The computational complexity of Cholesky factorization for dense $n \times n$ matrix is $O(n^3/3)$ flops. Sparse Cholesky factorization can be done in $O([\text{nnz}(L)]^{3/2})$ flops.

Preconditioners can also be constructed via incomplete Cholesky factorization in which fill is ignored selectively. Examples include incomplete Cholesky with no fill (IC(0)) and modified Cholesky factorization (MIC).

4. Preconditioned Conjugate Gradients Method (PCG)

The linear system in Eq. (3.12) is solved using the PCG method that is summarized in Algorithm 5. Each iteration of PCG requires one matrix-vector product, one preconditioning step, and three vector operations. The two steps that dominate computation are matrix-vector product and the preconditioning step. For dense matrices, both matrix-vector product and the preconditioning step would require $O(n^2)$ flops, resulting in overall cost of $O(n^2)$ flops per iteration. For sparse matrices, however, matrix-vector product requires $O(\text{nnz}(A))$ flops and preconditioning step requires $O(n + 2\text{nnz}(L))$ flops, resulting in overall cost of $O(\text{nnz}(L))$ flops per iteration. From Eq. (2.6), we can see that the total number of iterations required by PCG to solve the system is bounded by $O(\sqrt{\kappa(M^{-1}A)})$. Thus, the total cost of solving the system via PCG can be approximated as

$$\text{PCG cost} \approx \underbrace{O(\sqrt{\kappa(M^{-1}A)})}_{\text{iterations}} \; \underbrace{[O(\text{nnz}(L)) + O(\text{nnz}(A))]}_{\text{work/iteration}}. \tag{3.17}$$

Algorithm 5 The PCG method for solving $Ax = b$ using preconditioner M.

```
Require: $x_0$, $\epsilon$ and maxiter
 1: $k = 0$
 2: $r_0 = b - Ax_0$, Solve $Mz_0 = r_0$, $p_0 = z_0$
 3: while $||r_k||/||r_0|| > \epsilon$ or $k \leq$ maxiter do
 4:     $\alpha_k = r_k^T \cdot z_k / p_k^T \cdot Ap_k$      $\Longrightarrow$ Sparse matrix-vector product $(A \cdot p_k)$
 5:     $x_{k+1} = x_k + \alpha_k p_k$                 $\Longrightarrow$ Vector operation
 6:     $r_{k+1} = r_k - \alpha_k Ap_k$                $\Longrightarrow$ Vector operation
 7:     Solve $Mz_{k+1} = r_{k+1}$                     $\Longrightarrow$ Preconditioning step $(M = LL^T)$
 8:     $\beta_k = r_{k+1}^T z_{k+1} / r_k^T z_k$
 9:     $p_{k+1} = z_{k+1} + \beta_k p_k$              $\Longrightarrow$ Vector operation
10:     $k = k + 1$
11: end while
12: $x = x_k$
```

C. Analysis of the Preconditioning Scheme

The effectiveness of a preconditioners may be judged by the following properties:

- The condition number of the preconditioned matrix should be significantly lower than the original matrix, i.e., $O(\sqrt{\kappa(M^{-1}A)}) \ll O(\sqrt{\kappa(A)})$. A lower condition number improves the rate of convergence of the PCG method.
- The preconditioner should be easy to construct.
- The preconditioner should be easy to factorize, with minimum fill. This would keep the computational cost and storage requirement modest.

1. Effect of Preprocessing the Matrix

As described in Section A of this chapter, the stiffness matrix $A = B^TB$, is preprocessed to obtain a transformed system $A' = B'^TB'$. The transformation is obtained by changing the coordinate axes from $(x, y) \rightarrow (\tau_1, \tau_2)$ in each element, where τ_1 and τ_2 are parallel to two edges of the triangle. This transformation converts the coefficient matrix A to a diagonally dominant matrix A'.

The support graph preconditioner is computed for the transformed system. One has to pay a penalty for using the transformed system to construct the preconditioner. If A' is used as a preconditioner, then

$$\kappa(A'^{-1}A) \quad \leq \quad \kappa(GG^T) = \max_e \kappa(G^eG^{eT}), \tag{3.18}$$

where

$$\kappa(G^eG^{eT}) \quad = \quad \left(\frac{1 + |\cos\theta^e|}{1 - |\cos\theta^e|}\right), \tag{3.19}$$

where θ^e is the angle between the edges selected in the element e. Fig. 7 plots

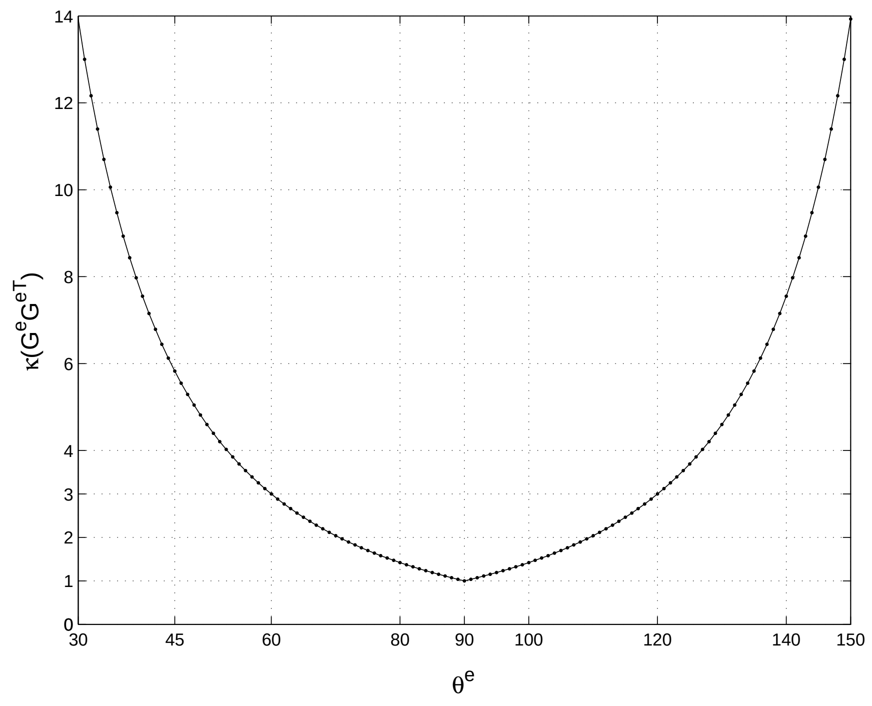

Fig. 7. Variation of $\kappa(G^e G^{eT})$ with θ^e.

$\kappa(G^e G^{eT})$ as a function of θ^e. It is easy to show that for $60° \leq \theta^e \leq 120°$, $\kappa(G^e G^{eT}) \leq 3$.

We use a software package called *Triangle* [15] to generate the mesh. *Triangle* attempts to construct a good quality mesh with a minimum angle of 33.8° [15, 16]. In most cases, the condition number of the transformed matrix A' is within a factor of 3 from that of A. For the anisotropic case, this factor may be higher since it will depend on $\kappa(DGP^{-1}G^T D)$ instead.

2. Condition Number Estimate

The rate of convergence of PCG depends on the condition number of the preconditioned system

$$\kappa(M^{-1}A) = \kappa(M^{-1}A')\ \kappa(A'^{-1}A) \tag{3.20}$$

Let us try to estimate the term $\kappa(M^{-1}A')$

$$\begin{aligned}\kappa(M^{-1}A') &= \frac{\lambda_{max}(M^{-1}A')}{\lambda_{min}(M^{-1}A')} \\ &= \frac{\max\limits_x \left(\dfrac{x^TA'x}{x^TMx}\right)}{\min\limits_x \left(\dfrac{x^TA'x}{x^TMx}\right)} \end{aligned} \tag{3.21}$$

If one interprets the Laplacian as a graph $G(A')$, with w_{ij} being the weight of the edge between nodes (i, j), and $\Delta x_e = x_i - x_j$ for edge e between nodes (i, j), then for all x,

$$x^TA'x = \sum_{e\in A'} w_e \Delta x_e^2. \tag{3.22}$$

A lower bound on the condition number can be obtained from the above equation,

$$\frac{x^TA'x}{x^TMx} = \frac{\sum\limits_{f\in A'} w_f\Delta x_f^2}{\sum\limits_{e\in M} w_e\Delta x_e^2} \tag{3.23}$$

Therefore,

$$\begin{aligned}\frac{x^TA'x}{x^TMx} &= \frac{\sum\limits_{f\in A'\setminus M} w_f\Delta x_f^2}{\sum\limits_{e\in M} w_e\Delta x_e^2} + 1 \\ &\geq 1, \end{aligned} \tag{3.24}$$

where w_e and w_f are weights of edges e and f respectively. To determine an upper bound, observe that

$$\begin{aligned}\frac{x^TA'x}{x^TMx} &= \frac{\sum\limits_{f\in A'\setminus M} w_f\Delta x_f^2}{\sum\limits_{e\in M} w_e\Delta x_e^2} + 1 \\ &= \frac{\sum\limits_{f\in A'\setminus M} w_f(\Delta x_{e_1} + \Delta x_{e_2} + \cdots + \Delta x_{e_p})^2}{\sum\limits_{e\in M} w_e\Delta x_e^2} + 1, \end{aligned} \tag{3.25}$$

where e_1, e_2, …, e_p are consecutive edges on the support path of f in M. Now,

$$(\Delta x_{e_1} + \Delta x_{e_2} + \cdots + \Delta x_{e_p})^2 \leq L_f(\Delta x_{e_1}^2 + \Delta x_{e_2}^2 + \cdots + \Delta x_{e_p}^2), \tag{3.26}$$

where L_f is the number of edges in f's support path. Thus, we can see that

$$\begin{aligned}
\frac{x^T A' x}{x^T M x} &\leq \frac{\sum\limits_{f \in A' \setminus M} w_f L_f(\Delta x_{e_1}^2 + \Delta x_{e_2}^2 + \cdots + \Delta x_{e_p}^2)}{\sum\limits_{e \in M} w_e \Delta x_e^2} + 1 \\
&= \frac{\sum\limits_{e \in M} \Delta x_e^2 \left(\sum\limits_{f \text{ supported by } e} w_f L_f \right)}{\sum\limits_{e \in M} w_e \Delta x_e^2} + 1 \\
&\leq \max_{e \in M} \left(\frac{\sum\limits_{f \text{ supported by } e} w_f L_f}{w_e} \right) + 1. \qquad (3.27)
\end{aligned}$$

From Eq. (3.24) and Eq. (3.27), we obtain the bound

$$\begin{aligned}
\kappa(M^{-1}A') &\leq 1 + \max_{e \in M} \left(\frac{\sum\limits_{f \text{ supported by } e} w_f L_f}{w_e} \right) \\
&= 1 + c(G(M))\ d(G(A' \setminus M)) \qquad (3.28)
\end{aligned}$$

where $c(G(M))$ and $d(G(A' \setminus M))$ are the congestion and dilation of $G(M)$ and $G(A' \setminus M)$ respectively. Further, the condition number of the preconditioned system is given by

$$\begin{aligned}
\kappa(M^{-1}A) &\leq [1 + c(G(M))\ d(G(A' \setminus M))]\ \kappa(GG^T) && \text{[Isotropic]} \\
\kappa(M^{-1}A) &\leq [1 + c(G(M))\ d(G(A' \setminus M))]\ \kappa(DGP^{-1}G^T D) && \text{[Anisotropic]}
\end{aligned}$$

Consider the two dimensional unit square shown in Fig. 6. If d is the number of subdomains with $m \times m$ nodes per subdomain then $m \approx \sqrt{(n/d)}$. With MSTs present inside each subdomain, the maximum dilation for this system is $O(m^2)$ and maximum congestion is $O(m^2)$. It was shown earlier that for meshes with $\theta^e \in [60°, 120°]$,

$\kappa(GG^T) \leq 3$. Hence, $\kappa(M^{-1}A)$ will be bounded by $O(m^4)$. When $d \approx \sqrt{n}$, this bound is $O(n)$. For anisotropic problems defined on the same mesh, the product of congestion and dilation will remain nearly the same, however, $\kappa(DGP^{-1}G^TD)$ may be higher depending upon the anisotropy matrix $p(x, y)$.

Note that, for the isotropic case, the bound on the condition number of the system depends only on the size of subdomain instead of the size of the entire domain or the numerical values of the entries of the coefficient matrix. Reducing the size of each subdomain will reduce the condition number. However, this is accompanied by an increase in the fill during factorization.

3. Numerical Factorization

The support graph preconditioner consists of maximum spanning trees inside each subdomain. During Cholesky factorization, the nodes in the MST can be eliminated without introducing any fill. Fill will occur only during the removal of nodes from the skeletal mesh that is formed by the subdomain boundary edges. After removal of MST nodes, the remaining nodes in the graph will have degree two or more. First the nodes on the subdomain boundary will be removed, introducing a fill of $O(d\ m)$, leaving a much smaller skeletal mesh of size $(d+1) \times (d+1)$. The skeletal mesh, in the worst case, would result in a dense matrix with $O(d^2)$ nonzeros. Thus, the amount of fill after factorization is $O(n + dm + d^2)$. With fewer, larger subdomains, one can expect less fill. However, at the same time the condition number of the preconditioned system will increase due to an increase in the congestion and the dilation.

Thus, the total cost of PCG solver can be approximated as

$$\begin{aligned} \text{PCG work} &= O(\sqrt{\kappa(M^{-1}A)})\ [O(\text{nnz}(L)) + O(\text{nnz}(A))] \\ &= O(m^2\ (n + dm + d^2)). \end{aligned}$$

It follows that the goal of finding an effective support graph preconditioner with small κ conflicts with the goal of finding an inexpensive one.

CHAPTER IV

SOFTWARE DESIGN

The proposed support graph preconditioners have been implemented using an object oriented code developed in C++. C++ coupled with modern compilers provides several mechanisms of high performance.

Since the systems in consideration are large and sparse a separate sparse matrix class has been written, as described in the following section.

A. Sparsity

Sparsity can be used to reduce both the storage and the computational complexity. A sparse storage scheme stores only the nonzero entries of the matrix. There are several storage schemes available in literature [14], viz. coordinate format, compressed sparse row (CSR) format and compressed sparse column (CSC) format. The software implementation uses CSC format to represent sparse matrices. This format is also known as the Harwell-Boeing sparse matrix format [8]. The implementation of CSC format consists of a sparse matrix class, viz. `CompColSparseMatrix`, with three `vectors` viz. `colptr`, `rowi`, and `value`. For a sparse matrix with m rows and n columns, `colptr` represents an integer `vector` of size $n + 1$ that contains the index of the start of each row in `rowi` and `value` `vectors`. Information about the non zero entries in i^{th} column are stored contiguously in `rowi` and `value` from index $\texttt{colptr}[\texttt{i}]$ to $\texttt{colptr}[\texttt{i}+1]-1$. For programming convenience, the last entry of `colptr` is set to the number of nonzeros denoted by nnz. The integer `vector` `rowi` is of size nnz and contains the row indices of the matrix entries. For example, $\texttt{rowi}[\texttt{colptr}[\texttt{i}]]$ gives the row index of the first nonzero entry in the i^{th} column. The `vector` `value` is a `vector` of `double` with size nnz and it contains the numerical values of the matrix entries. For example, `value`[`colptr`[`i`]] gives the value of the first nonzero entry in the i^{th} column. An example of a matrix stored in the CSC format is shown in Fig. 8. Various functions implemented for the sparse matrix class, `CompColSparseMatrix` are shown in Table I.

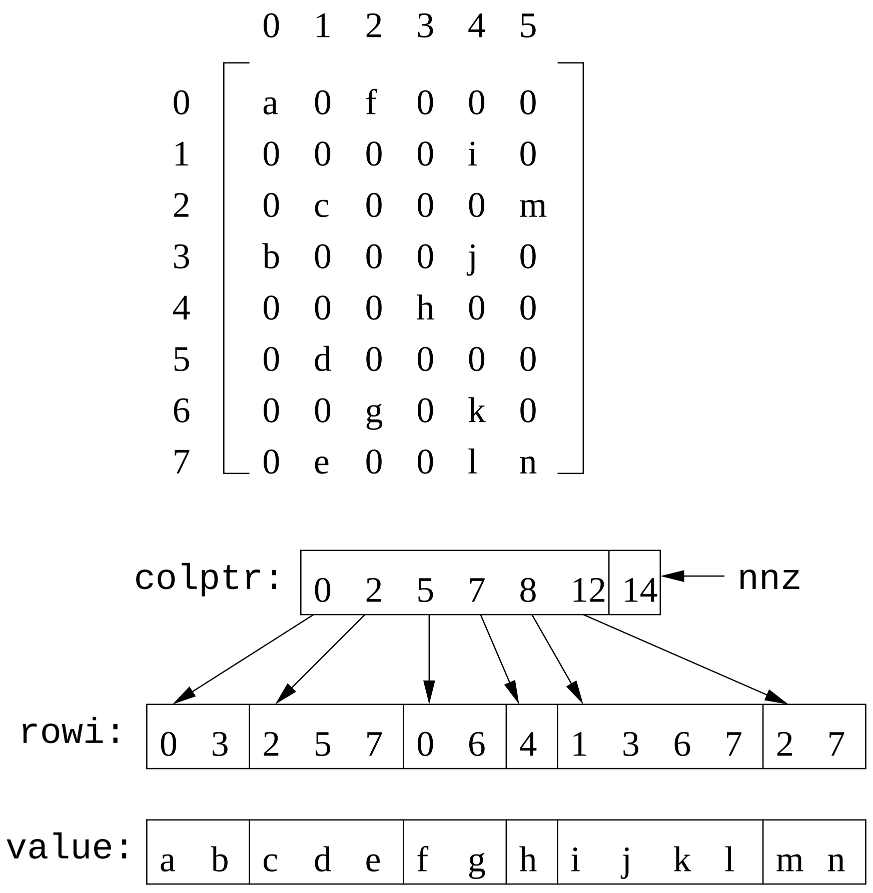

Fig. 8. Example of a matrix stored in the compressed sparse column (CSC) format.

B. Modules

The code can be broadly divided into three modules. These modules are for mesh generation, support graph preconditioner construction, and numerical methods, as

Table I. Sparse matrix class operations.

Class: `CompColSparseMatrix` (say matrix A)	
Function name	Operation
`n = numcols()`	$n \leftarrow$ number of columns of A
`nz = nnz()`	$nz \leftarrow$ number of non zeros in A
`same_lower_triangular(L)`	$L \leftarrow$ lower triangular entries of A nnz(L) = nnz(lower triangular matrix of A)
`diff_lower_triangular(L)`	$L \leftarrow$ lower triangular entries of A nnz(L) is decided by symbolic factorization nnz(L) $\geq$ nnz(lower triangular matrix of A)
`y = A.matvec(x)`	$y \leftarrow Ax$
`y = A.matvec_transpose(x)`	$y \leftarrow A^T x$
`extract(Anew, v)`	$A_{new} \leftarrow A(v, v)$
`extract_cols(Anew, vin, vout)`	$A_{new} \leftarrow A(:, v_{in})$
`reorder(Areorder, v)`	$A_{reorder} \leftarrow A(v, v)$
`reorder_cols(Areorder, v)`	$A_{reorder} \leftarrow A(:, v)$
`mat_mattrans(Anew)`	$A_{new} \leftarrow A^T A$

shown in Fig. 9. The following sections describe some important software aspects of these modules.

1. Mesh Generation Module

Mesh generation was done using *Triangle* [15]. *Triangle* uses a Delaunay triangulation algorithm to generate an unstructured mesh for given boundary data. The mesh information from *Triangle* is stored in static files that are later read by the software.

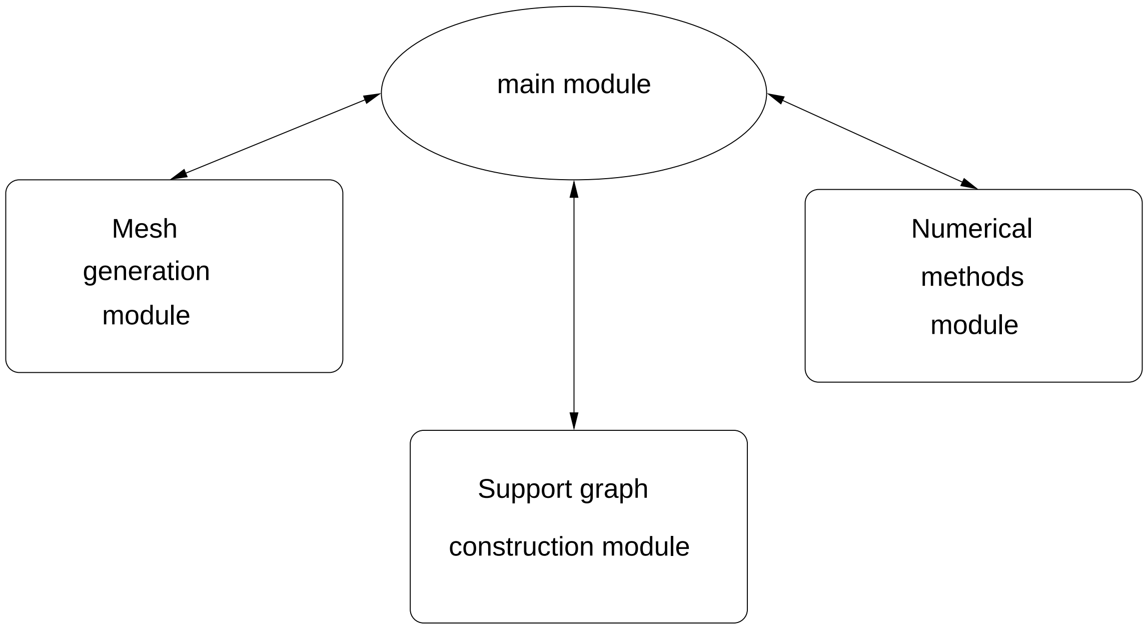

Fig. 9. Main modules.

The mesh data is stored in three data structures, given by the three classes, viz. `Node`, `Edge` and `Triangle`. The `Node` class contains information about mesh nodes. The `Edge` class contains information on the boundary edges and the `Triangle` class contains information about the triangles in a mesh. The variables of these three classes are described in Table II.

2. Support Graph Preconditioner Construction Module

The flow of information in this module is described in Fig. 10, and the related classes are outlined in Table III. The process starts by building a global data structure using a `vector` of `Domain` class. The size of this `vector` is equal to the number of nodes in the mesh. Each node stores information about its adjacent nodes in a `vector` of `Adjacency` class. The weights of the edges are obtained by considering elements in the mesh and the matrix B is implicitly built during this process via finite element

Table II. Mesh information classes.

Class	Variable name	Description
`Node`	`xcord`	x coordinate
	`ycord`	y coordinate
`Edge`	`starten`	Starting node number
	`enden`	Ending node number
	`rightsub`	Right subdomain number
	`leftsub`	Left subdomain number
`Triangle`	`vert`	Vector of vertices
	`subnum`	Subdomain number

method. Pre-processing of B is done simultaneously and the adjacency list of each node is filled only with the required edges.

Once the global data structure is built, selective information of a node is used to create a local data structure for each subdomain using a `vector` of `SubDomain` class. The size of this `vector` is the same as the number of nodes in a subdomain. A heap is built using a `vector` of `Heap` class to execute the maximum spanning tree algorithm. The output of this is a list of nodes belonging to the maximum spanning

Table III. SGP construction related classes.

Class	Description
`Adjacency`	contains a nodes adjacency list
`Domain`	contains node information
`SubDomain`	contains node information that is considered only in a subdomain
`Heap`	contains heap structure for use by MST algorithm

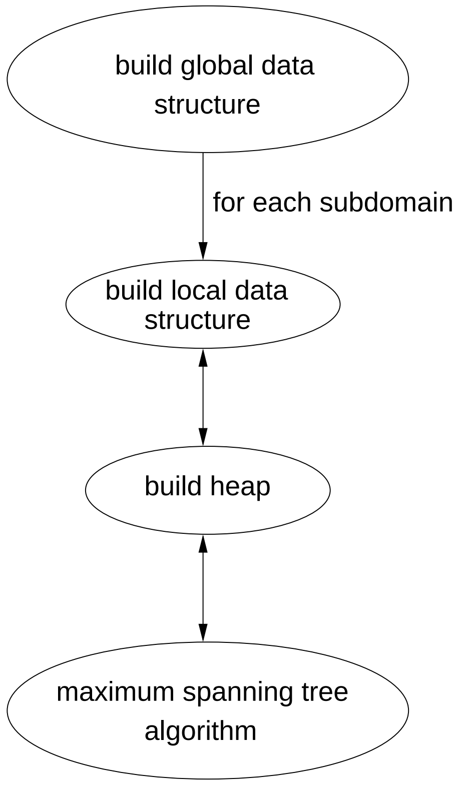
build global data structure
for each subdomain
build local data structure
build heap
maximum spanning tree algorithm

Fig. 10. The support graph preconditioner module.

tree. Although the `vector` of `Subdomain` class could be used as a heap, it would drastically increase the number of operations in the MST algorithm. Thus, with a slight increase in storage and data exchange, a separate heap was used.

3. Numerical Method Functions Module

This section briefly describes the module for implementing various numerical methods. Some of these methods and their helper functions are described in Table IV. The methods in this module are written as independent functions, utilizing the sparse matrix class and `vector` class. This list contains only the important functions and classes to reflect the software design and is not a comprehensive list.

Table IV. Numerical methods.

Main functions	
Function name	Operation
`sparseCompleteCholesky()`	$L \leftarrow A$, complete Cholesky factorization
`sparseIncompCholesky()`	$L \leftarrow A$, incomplete Cholesky factorization with no fill
`precg()`	Preconditioned Conjugate Gradients method
`mmd()`	Minimum degree ordering of a matrix
Helper functions	
Function name	Operation
`symbolic_factorization()`	$L \leftarrow A$, symbolic factorization
`choleskyFactor()`	$L \leftarrow A$, numeric factorization
`s = dot_product(x, y)`	$s \leftarrow x \cdot y$
`daxpy(x, y, a)`	$y = ax + y$
`dxpay(x, y, a)`	$y = x + ay$

CHAPTER V

NUMERICAL RESULTS

This chapter presents the results of numerical experiments to study the effectiveness of the proposed support graph preconditioners. As shown earlier in Chapter III, the condition number of the preconditioned system is bounded by the size of subdomain rather then the size of entire domain. These experiments were designed to illustrate the behavior of the approach for fixed subdomain size and varying number of subdomains in both x and y directions. In the first set of experiments, we consider the isotropic case where p is identity in Eq. (2.1). The second set of experiments considers anisotropic case. We compare the performance of our preconditioner with incomplete Cholesky factorization with no fill.

The experiments were conducted on Intel workstation with a processor speed of 2.4 GHz and 512 MB of RAM using Red Hat Linux 9. We report the number of iterations required by each solver as well as the time, in seconds, spent in the preconditioner construction phase, the Cholesky factorization step, and the iterative solver. Preconditioned conjugate gradients is used as the iterative solver for both support graph preconditioner (SGP) and incomplete Cholesky factorization with no fill, i.e., (IC(0)). All the experiments are run with a tolerance of 10^{-6} on the relative residual norm.

A. Isotropic Domain

The first set of experiments is for the Laplace equation. It can be obtained by substituting $p(x, y)$ with an identity matrix and taking $f(x, y) = 0$ in Eq. (2.1). Within this set of experiments, we have two types of support graph preconditioners. The first type of preconditioner denoted by SGP-I, uses the maximum spanning trees

(MST) inside each subdomain. The second type of preconditioner denoted by SGP-II, augments the MST with the heaviest k edges that were not part of SGP-I.

The pre-processing of stiffness matrix described in Chapter III increases the condition number of the preconditioner by a factor of $\kappa(GG^T)$. Fig. 11 shows the trend of $\kappa(G^eG^{eT})$ for element e in the mesh shown in Fig. 6. This condition number is plotted with respect to the angle θ^e between the two chosen edges during the pre-processing step. This trend is representative of the meshes under consideration. It can be seen that the value of $\kappa(G^eG^{eT})$ is below 3 for a majority of elements and is always below 6 for the entire mesh.

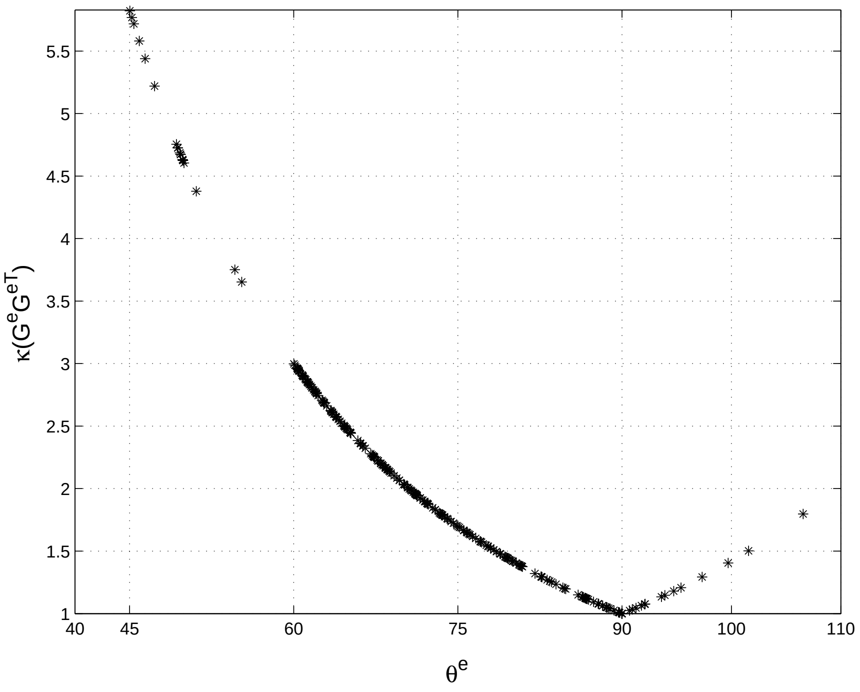

Fig. 11. The values of $\kappa(G^eG^{eT})$ for elements in the mesh shown in Fig. 6, with $d = 4$.

Fig. 12 shows the support graph preconditioner for the mesh shown in Fig. 6. Solid lines represent the edges of the MST within each subdomain. An union of solid

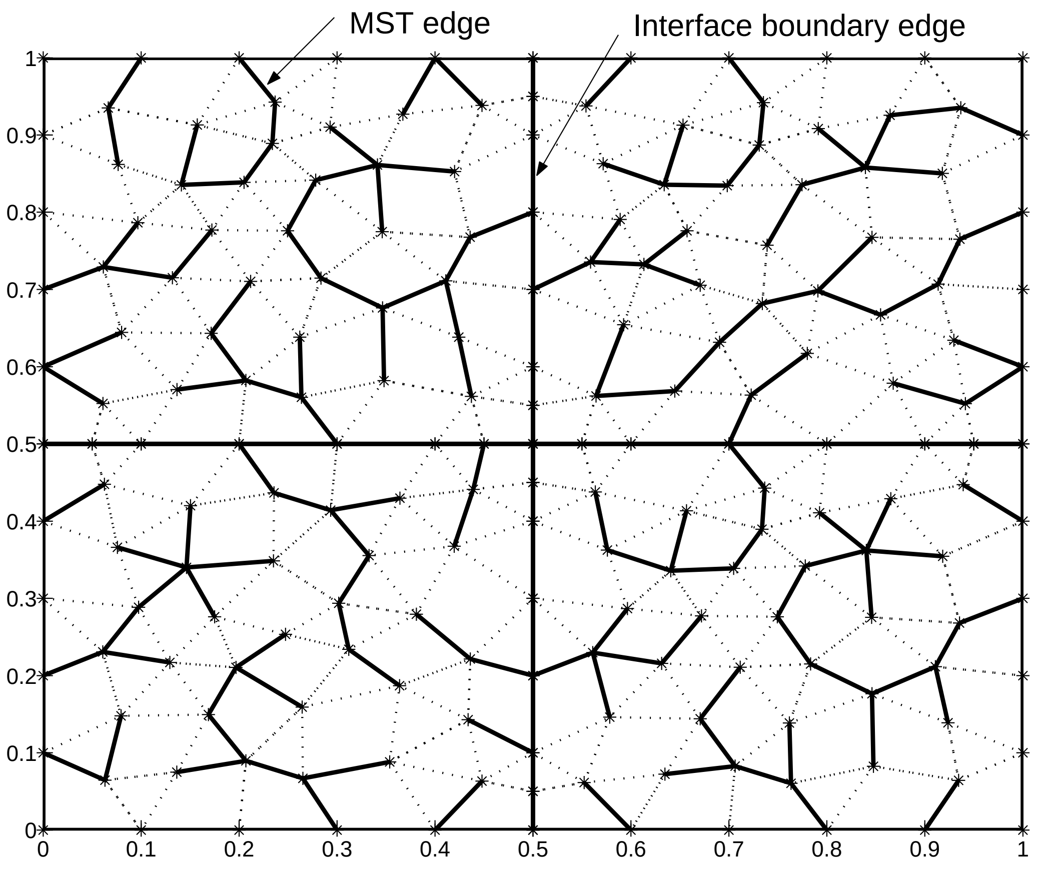

Fig. 12. The edges of the support graph preconditioner for a mesh with $d = 4$.

and dotted lines represent all the edges remaining after the pre-processing step. An edge was removed from the triangulation during the pre-processing step if the elements on both side did not include the edge during coordinate axis transformation. It can be seen how spanning trees are formed inside each subdomain. The preconditioner construction time is shown in Table V. This time includes the formation of the matrix B, the preprocessing of B and the construction of M. The cost of forming the preconditioner is amortized over the number of iterations required to solve the system.

Table V. Time for construction of the support graph preconditioner.

Mesh			SGP, M
d	Nodes	Triangles	Time(s)
2×2	177	304	0.01
4×4	688	1274	0.03
8×8	2638	5085	0.15
16×16	10382	20383	0.88
32×32	41117	81481	7.61

In the following experiments the number of subdomains are increased in each direction by a factor of two, while keeping the size of the subdomain fixed. This results in quadrupling the number of nodes at each step. We conducted experiments with all the three boundary conditions described earlier in Subsection III.B.2 viz. Dirichlet, Neumann and Mixed boundary condition. For Mixed boundary condition, Dirichlet boundary condition is specified on the top boundary. Tables VI, VII, and VIII show the results obtained. Fig. 13 shows the iterations with respect to increasing domain sizes corresponding to Tables VI VII, and VIII. It can be seen that the number of iterations for SGP-I saturates after a while, whereas for incomplete Cholesky they are monotonically increasing. This can be attributed to the fact that the condition number of SGP-I is bounded by the size of subdomains, which is kept fixed. Saturation of SGP-I iterations is observed independent of the type of boundary conditions. Moreover, the number of nonzeros for the factors of the SGP-I preconditioner is less as compared to the number of nonzeros in incomplete Cholesky factors, resulting in lesser time for factorization.

Table VI. Laplace problem on a unit square with Dirichlet boundary conditions. Time given in seconds.

Mesh		SGP				IC(0)			
		Factorization		Solver		Factorization		Solver	
d	Nodes	nnz(L)	Time	Iter	Time	nnz(L)	Time	Iter	Time
4	177	238	0.0	39	0.0	475	0.0	13	0.0
16	688	1231	0.01	67	0.01	2259	0.03	22	0.01
64	2638	5508	0.09	91	0.14	9610	0.46	42	0.04
256	10382	23851	2.22	102	0.62	39627	6.70	78	0.53
1024	41117	102115	39.19	110	3.98	160690	99.07	141	6.74

Table VII. Laplace problem on a unit square with Dirichlet boundary conditions only on the top boundary. Time given in seconds.

Mesh		SGP				IC(0)			
		Factorization		Solver		Factorization		Solver	
d	Nodes	nnz(L)	Time	Iter	Time	nnz(L)	Time	Iter	Time
4	177	367	0.01	46	0.00	611	0.00	19	0.00
16	688	1539	0.02	71	0.02	2548	0.04	39	0.01
64	2638	6214	0.16	93	0.09	10181	0.49	73	0.09
256	10382	25513	2.44	108	0.65	40750	6.96	132	1.00
1024	41117	106347	43.44	108	4.09	162948	106.06	251	12.5

Table VIII. Laplace problem on a unit square with Neumann boundary conditions. Time given in seconds.

Mesh		SGP				IC(0)			
		Factorization		Solver		Factorization		Solver	
d	Nodes	nnz(L)	Time	Iter	Time	nnz(L)	Time	Iter	Time
4	177	412	0.0	51	0.0	653	0.01	23	0.0
16	688	1653	0.01	77	0.02	2645	0.05	47	0.01
64	2638	6443	0.16	97	0.10	10356	0.56	91	0.12
256	10382	26138	2.66	98	0.62	41142	7.39	184	1.46
1024	41117	106948	43.33	112	4.26	163710	105.55	353	17.37

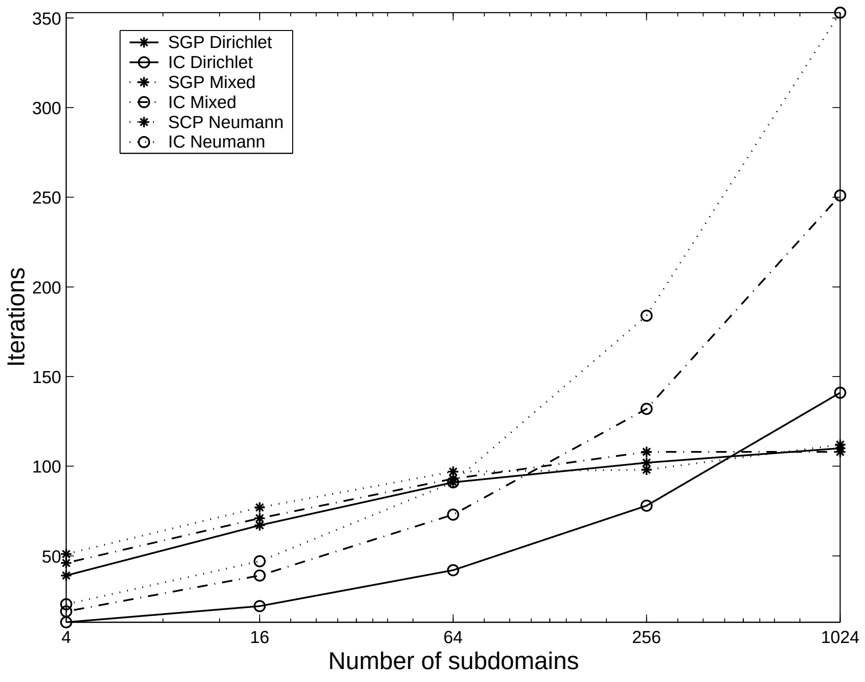

Fig. 13. The growth in the number of iterations with mesh size for SGP-I and IC(0) preconditioners.

To show the effect of subdomain size on convergence, we conducted set of experiments in which the subdomain size was decreased by a factor of four at each step while keeping the number of nodes in the mesh nearly fixed. It can be seen from Table IX that the number of iterations decreased by factor of 1.6 approximately. This is attributed to the fact that the condition number decreases by a factor of four. Since the total iterations in PCG are bounded by the square root of condition number, they reduce by a factor of two. At the same time, there is increase in fill due to the increase in the number of interface boundary nodes in the mesh. Overall, the algorithm takes less time for larger number of subdomains.

Table IX. Laplace problem on a unit square with Dirichlet boundary conditions (r: order of mesh refinement).

Mesh			SGP				
			Factorization		Solver		Total
d	Nodes	r	nnz(L)	Time(s)	Iter	Time(s)	Time(s)
4	39901	4	76932	25.9	917	32.2	58.1
16	41049	3	80874	28.5	782	28.0	56.5
64	41453	2	83672	30.0	450	16.7	46.7
256	41660	1	88678	32.6	250	9.53	42.1
1024	41117	0	102115	39.2	110	3.98	43.2

Another set of experiments was conducted in which the maximum spanning trees were augmented with a few of the maximum weight edges inside each subdomain that were not part of SGP-I. To provide a fair comparison with IC(0) preconditioner, edges were added so as to restrict the fill in SGP-II to the number of nonzeros in IC(0) factors. Fig. 14 shows the change in iterations and time taken to solve a problem with 1024 subdomains. The arrow indicates the point at which SGP-II and IC(0) preconditioners have the same amount of fill. Table X shows the iterations and time required by SGP-II to solve the problem when the number of subdomains is increased. Even though the time of Cholesky factorization increases, the total time spent in factoring SGP-II and solving the problem still stays less than IC(0).

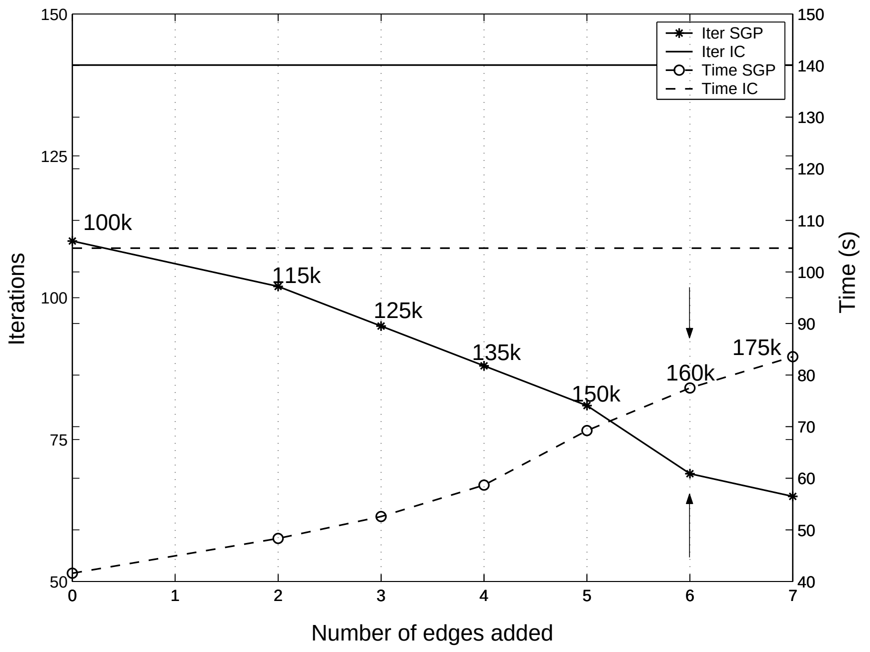

Fig. 14. The effect of augmenting SGP-I with additional edges on iterations and time, $d = 1024$.

B. Anisotropic Domain

The second set of experiments involved the anisotropic Laplace equation on anisotropic domains. The matrix $p(x, y)$ was chosen to be

$$p = \begin{pmatrix} 1 & 0 \\ 0 & \sigma^2 \end{pmatrix},$$

Table X. Laplace problem on a unit square with Dirichlet boundary condition using SGP-II.

Mesh		SGP			IC(0)		
d	Nodes	nnz(L)	Iter	Time(s)	nnz(L)	Iter	Time(s)
4	177	432	19	0.01	475	13	0.0
16	688	2204	27	0.02	2259	22	0.04
64	2638	9613	40	0.30	9610	42	0.48
256	10382	38277	58	4.4	39627	78	6.94
1024	41117	166021	69	77.5	160690	141	104.77

in Eq. (2.1) with σ as 10 and 0.1.

Figs. 15 and 16 show the support graph preconditioner for the mesh shown in Fig. 6 with $\sigma = 10$ and $\sigma = 0.1$, respectively. Solid lines represent the edges of the MST within each subdomain. The union of solid and dotted lines represent all the edges remaining after the pre-processing step. It can be seen from these figures that the maximum spanning tree edges are changed in presence of anisotropy.

Tables XI and XII show the performance of SGP-I for anisotropic problem with Dirichlet boundary conditions. The number of subdomains are increased whereas the domain size stays fixed. The results are almost the same for these two instances of σ. However, the number of iterations required to converge to the solution are higher than the isotropic case (Table VI). This is due to the increase in the factor $\kappa(DGP^{-1}G^TD)$, associated with pre-processing.

The pre-processing of stiffness matrix described in Chapter III increases the condition number of the preconditioner by a factor of $\kappa(GG^T)$. Fig. 11 shows the trend of $\kappa(G^eG^{eT})$ for element e in the mesh shown in Fig. 6. This condition number

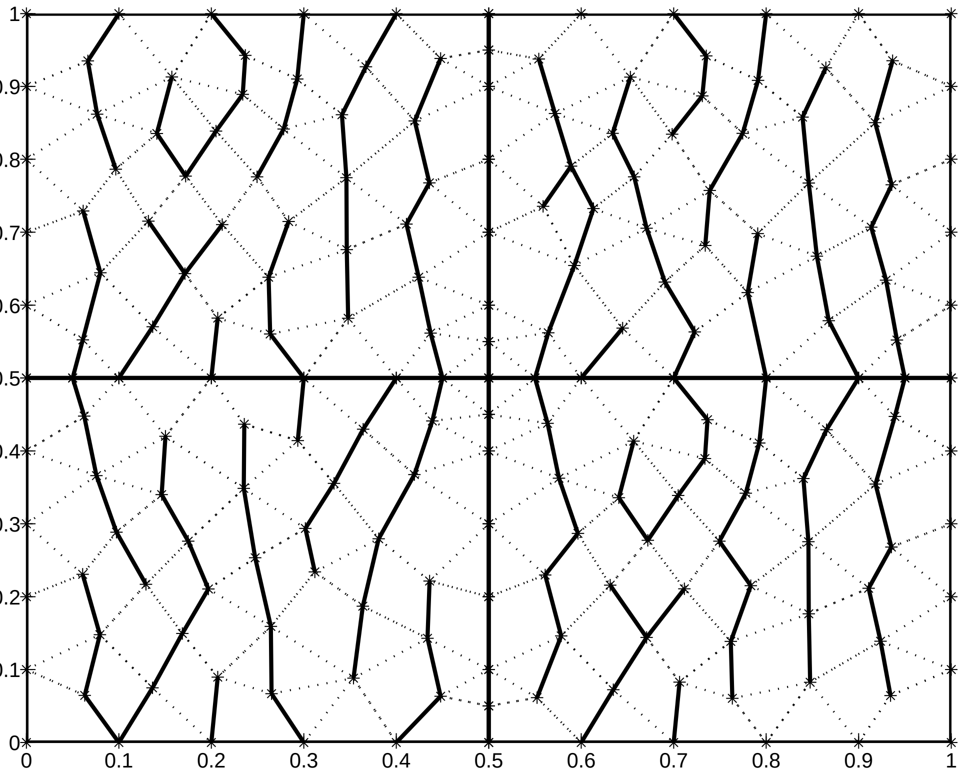

Fig. 15. SGP preconditioner edges for an anisotropic problem ($\sigma = 10$).

Table XI. Anisotropic Laplace problem on a unit square with Dirichlet boundary condition using SGP-I ($\sigma = 10$). Time given in seconds.

Mesh		SGP				IC(0)			
		Factorization		Solver		Factorization		Solver	
d	Nodes	nnz(L)	Time	Iter	Time	nnz(L)	Time	Iter	Time
4	177	249	0.0	81	0.0	475	0.0	18	0.0
16	688	1241	0.01	166	0.03	2259	0.03	33	0.01
64	2638	5583	0.14	256	0.23	9610	0.43	63	0.07
256	10382	23933	2.18	308	1.83	39627	6.34	116	0.79
1024	41117	102029	37.8	323	11.58	160690	96.4	222	10.42

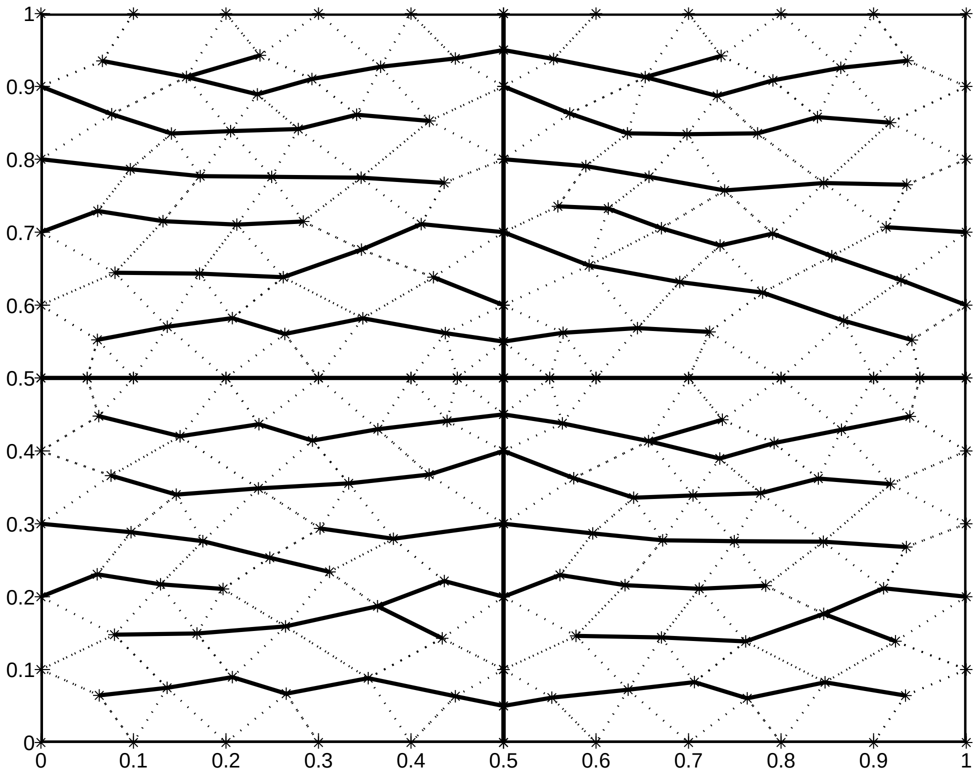

Fig. 16. SGP preconditioner edges for an anisotropic problem ($\sigma = 0.1$).

Table XII. Anisotropic Laplace problem on a unit square with Dirichlet boundary condition using SGP-I ($\sigma = 0.1$). Time given in seconds.

Mesh		SGP				IC(0)			
		Factorization		Solver		Factorization		Solver	
d	Nodes	nnz(L)	Time	Iter	Time	nnz(L)	Time	Iter	Time
4	177	243	0.0	92	0.01	475	0.0	21	0.01
16	688	1233	0.0	193	0.04	2259	0.03	42	0.01
64	2638	5530	0.14	306	0.27	9610	0.46	80	0.08
256	10382	23966	2.18	359	2.1	39627	6.35	147	1.03
1024	41117	102633	38.05	384	13.8	160690	96.65	285	13.34

is plotted with respect to the angle θ^e between the two chosen edges during the pre-processing step. This trend is representative of the meshes under consideration. It can be seen that the value of $\kappa(G^e G^{eT})$ is below 3 for a majority of elements and is always below 6 for the entire mesh.

Figs. 17 and 18 show the trend in $\kappa(D^e G^e P^{e-1} G^{eT} D^e)$ for element e in the mesh shown in Fig. 6. This condition number is plotted with respect to the angle θ^e between the two chosen edges during the pre-processing step. It can be seen that most of the values of $\kappa(D^e G^e P^{e-1} G^{eT} D^e)$ are below 100, for a majority of elements and is always below 400 for the entire mesh. This factor stays constant with increasing number of subdomains though it is much higher than the isotropic case.

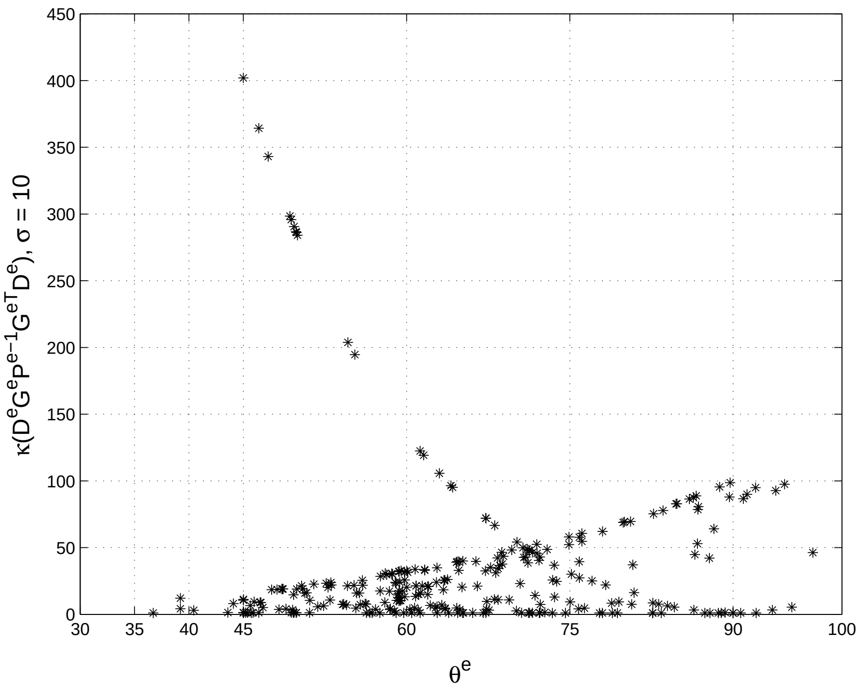

Fig. 17. The values of $\kappa(DGP^{-1}G^TD)$ for elements in the mesh shown in Fig. 6, with $d = 4$ and $\sigma = 10$.

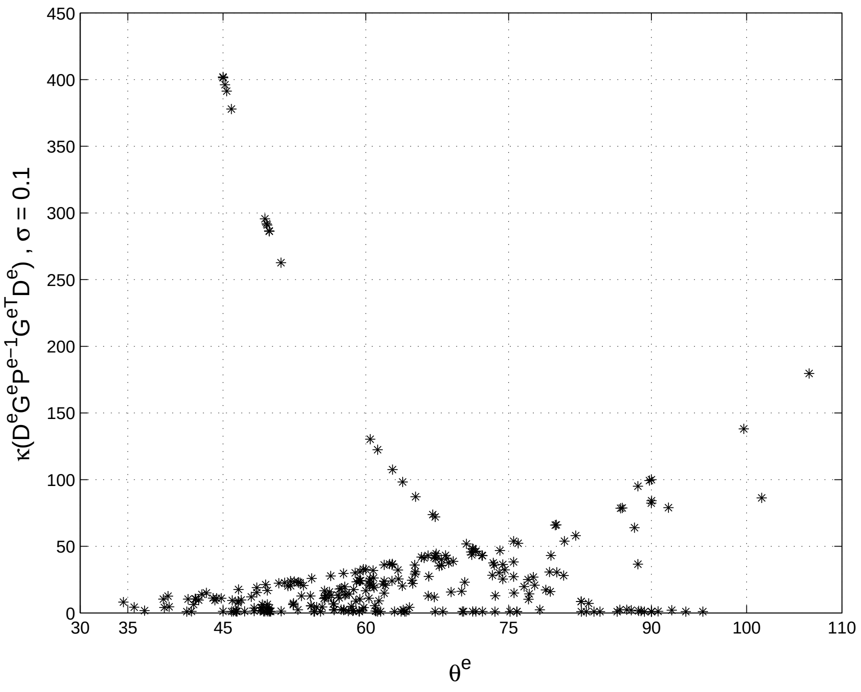

Fig. 18. The values of $\kappa(DGP^{-1}G^TD)$ for elements in the mesh shown in Fig. 6, with $d = 4$ and $\sigma = 0.1$.

The performance of the SGP-I preconditioner can be improved for anisotropic case by augmenting the spanning tree with maximum weight edges. Figs. 19 and 20 show the the change in iterations and time taken to solve a problem with 1024 subdomains using SGP-II. The rate of convergence of PCG improves as we add more edges inside each subdomain, though with a slight increase in time in Cholesky factorization. The arrow indicates the point at which SGP-II and IC(0) preconditioners have the same amount of fill.

Tables XIII and XIV show the iterations and time required by SGP-II to solve the problem when the number of subdomains is increased. Even though the time of Cholesky factorization increases, the total time spent in factoring SGP-II and solving the problem still stays less than IC(0).

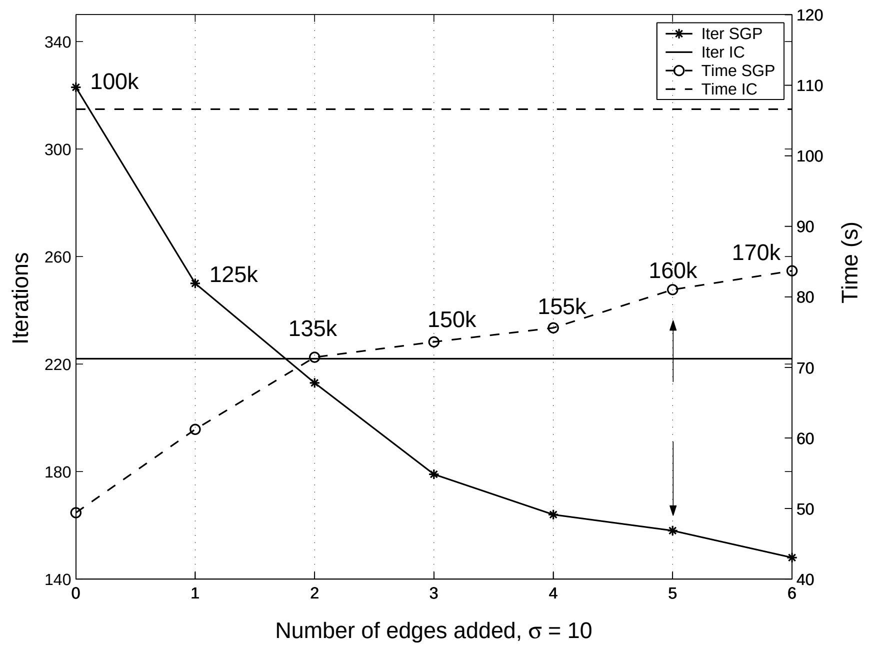

Fig. 19. The effect of augmenting SGP-I with additional edges on iterations and time, $\sigma = 10$, and $d = 1024$.

Table XIII. Anisotropic problem on a unit square with Dirichlet boundary condition using SGP-II ($\sigma = 10$).

Mesh		SGP			IC(0)		
d	Nodes	nnz(L)	Iter	Time(s)	nnz(L)	Iter	Time(s)
4	177	410	36	0.0	475	18	0.01
16	688	2058	53	0.02	2259	33	0.04
64	2638	9278	87	0.33	9610	63	0.49
256	10382	39209	125	4.98	39627	116	7.08
1024	41117	161809	159	78.57	160690	222	106.64

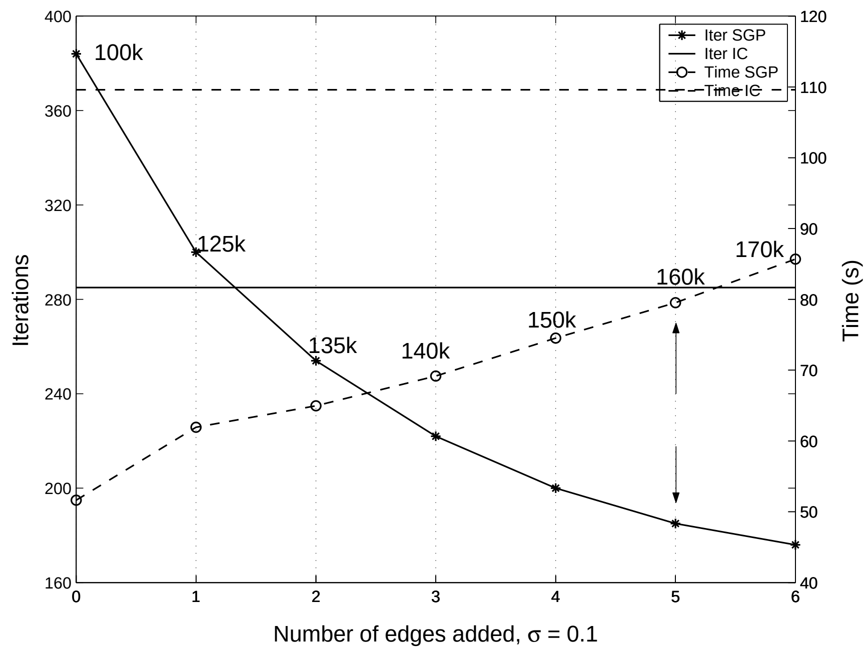

Fig. 20. The effect of augmenting SGP-I with additional edges on iterations and time, $\sigma = 0.1$, and $d = 1024$.

Table XIV. Anisotropic problem on a unit square with Dirichlet boundary condition using SGP-II ($\sigma = 0.1$).

Mesh		SGP			IC(0)		
d	Nodes	nnz(L)	Iter	Time(s)	nnz(L)	Iter	Time(s)
4	177	383	42	0.0	475	21	0.01
16	688	2086	68	0.02	2259	42	0.04
64	2638	9199	109	0.36	9610	80	0.51
256	10382	38887	152	5.14	39627	147	7.29
1024	41117	162059	185	79.52	160690	285	109.6

CHAPTER VI

CONCLUSIONS

In this thesis we developed an extension of the support graph preconditioning technique for symmetric positive definite matrices arising from the finite element discretization of partial differential equations. To illustrate the effectiveness of the approach, the Laplace equation was solved on a unit square unstructured mesh with different type of boundary conditions. This thesis investigates the effectiveness of a support graph preconditioner consisting of edges on the interface boundary between d subdomains along with edges on maximum spanning tree inside each subdomain. It was shown that

- The rate of convergence of the support graph preconditioners is independent of the size of the entire domain, but depends on the size of the subdomains. This is observed for different boundary conditions.
- The change in the rate of convergence of SGP for different mesh sizes is unaffected by anisotropy. The preconditioner can be improved by adding heavy weight edges to the support graph.
- Support graph preconditioners outperform incomplete Cholesky factorization preconditioner with zero fill in terms of execution time.

Prior to this work, support graph preconditioners had been developed for diagonally dominant M-matrices only. The extension proposed in the thesis allows support graph preconditioning to be used for a larger class of matrices such as SPD matrices arising from finite element discretizations.

REFERENCES

[1] O. AXELSSON, *Iterative Solution Methods*, Cambridge University Press, Cambridge, UK, 1996.

[2] M. BERN, J. R. GILBERT, B. HENDRICKSON, N. NGUYEN and S. TOLEDO, *Support-Graph Preconditioners*, Technical Report SAND 2001-0456J, Sandia National Labs, Feb 2001

[3] E. G. BOMAN and B. HENDRICKSON, *Support Theory for Preconditioning*, SIAM Journal on Matrix Analysis & Applications, 25/3 (2004), pp. 694-717.

[4] E. G. BOMAN, D. CHEN, B. HENDRICKSON and S. TOLEDO, *Maximum-Weight-Basis Preconditioners*, Technical Report SAND2001-1787J, Sandia National Labs, Livermore, CA, June 2001

[5] D. CHEN, *Analysis, Implementation, and Evaluation of Vaidya's Preconditioners*, M.Sc. Thesis, School of Computer Science, Tel-Aviv University, Israel, Feb. 2001.

[6] D. CHEN and S. TOLEDO, *Vaidya's Preconditioners: Implementation and Experimental Study*, Electronic Transactions on Numerical Analysis, 16 (2003), pp. 30-49.

[7] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST and C. STEIN, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[8] I. DUFF, R. GRIMES, AND J. LEWIS, *Sparse Matrix Test Problems*, ACM Transactions on Mathematical Software, 15 (1989), pp. 1-14.

[9] G. H. GOLUB and C. F. VAN LOAN, *Matrix Computations*, 3rd edition, Johns Hopkins University Press, Baltimore, MD, 1996.

[10] K. D. GREMBAN, *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*, PhD Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Oct. 1996; available as Technical Report CMU-CS-96-123.

[11] K. D. GREMBAN, G. L. MILLER and M. ZAGHA, *Performance Evaluation of a Parallel Preconditioner*, 9th International Parallel Processing Symposium, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 65-69.

[12] S. GUATTERY, *Graph Embedding Techniques for Bounding Condition Numbers of Incomplete Factor Preconditioners*, Technical Report 97-47, ICASE, NASA Langley Research Center, Hampton, VA, 1997.

[13] J. N. REDDY, *An Introduction to the Finite Element Method*, 2nd edition, McGraw-Hill Book Company, New York, 1993, pp. 404-455.

[14] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA, 1996.

[15] J. R. SHEWCHUK, *Triangle: Enginnering a 2D Quality Mesh Generator and Delaunay Triangulator*, Lecture Notes in Computer Science, Springer-Verlag, New York, 1148 (1996), pp. 203-222.

[16] J. R. SHEWCHUK, *Delaunay Refinement Algorithms for Triangular Mesh Generation*, Computational Geometry: Theory and Applications, 22(1-3), May 2002, pp. 21-74.

[17] P. M. VAIDYA, *Solving Linear Equations with Symmetric Diagonally Dominant Matrices by Constructing Good Preconditioners*, unpublished manuscript. Presented at the IMA workshop on Graph Theory and Sparse Matrix Computation, Minneapolis, Oct. 1991.

APPENDIX A

MATRIX DEFINITIONS

- M-matrix: Consider an $n \times n$ real matrix A.
  - A is an M-matrix if $A^{-1} \geq 0$ and $a_{i,j} \leq 0,\ i \neq j$.
  - A is an M-mtrix if A is strictly or irreducibly diagonally dominant. Here, assume that $a_{ij} \leq 0$, $i \neq j$, and $a_{ii} > 0, i = 1, \ldots, n$.
- Irreducible matrix: A matrix is irreducible if and only if it's associated directed graph is strongly connected, i.e, there is a path from node i to node j, for all pairs (i, j).
- Diagonally dominant matrix: A real $n \times n$ matrix A is diagonally dominant if

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|, \qquad i = 1, \ldots, n$$

  - The matrix is Strictly diagonally dominant if strict inequality holds for all i.
  - The matrix is Irreducibly diagonally dominant if it is irreducible, diagonally dominant, and strict inequality hold for atleast one i.

VITA

Name	:	Radhika Gupta
Permanent Address	:	c/o Dr. Vivek Sarin Department of Computer Science Texas A&M University College Station, Texas 77843-3112
Education	:	M.S., Computer Science, Texas A&M University, College Station, TX, 2004; M.S., Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, 2001; B.E., Aerospace Engineering, Indian Institute of Technology, Bombay, India, 2000.

The typist for this thesis was Radhika Gupta.