A Chordal Preconditioner for Large Scale Optimization*

Thomas F. Coleman

TR 86-762
June 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

* This reserch was supported in part by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under contract DE-AC02-83-ER10369 and grant DE-FG02-86ER25013.

A Chordal Preconditioner

For

Large Scale Optimization

Thomas F. Coleman
Computer Science Department
Cornell University
Ithaca New York 14853

June 22, 1986

Abstract: We propose an automatic preconditioning scheme for large sparse numerical optimization. The strategy is based on an examination of the sparsity pattern of the Hessian matrix: using a graph-theoretic heuristic, a block diagonal approximation to the Hessian matrix is induced. The blocks are submatrices of the Hessian matrix; furthermore, each block is chordal. That is, under a positive definiteness assumption, each block can be Cholesky factored without creating any new nonzeroes (fill). Therefore the preconditioner is space efficient. We conduct a number of numerical experiments to determine the effectiveness of the preconditioner in the context of a linear conjugate gradient algorithm for optimization.

0. Introduction

In this paper we are concerned with the minimization of a smooth function $f: R^n \to R^1$ in the circumstance where n is large and the Hessian matrix $H(x)$ is sparse. An efficient method for such a problem must exploit sparsity at every turn: In particular, sparsity attention must be paid both to the determination of $H(x)$ (or an approximation) as well as to subsequent computations involving $H(x)$ - especially the solution of linear systems. Theory, algorithms, and software for the efficient sparse finite-difference approximation to $H(x)$ have recently been developed (eg. Powell and Toint[1979]; Coleman and More[1983, 1984]; Coleman, Garbow, and More[1984, 1985]; Coleman and Cai[1986]) ; we propose no improvements here. Indeed, the algorithm proposed here can use this previous work in a straightforward way. The focus of this present work is on the exploitation of sparsity after H has been determined (perhaps via a sparse finite-difference procedure). We assume throughout that the sparsity pattern of H is both known and fixed (for all x).

A Newton-like method for minimization involves the repeated solution of linear systems

$$H(x)s = -\nabla f(x) \qquad (0.1)$$

When x is close to a local minimizer x_* of f then we can expect H to be positive definite: In such a case the method of conjugate gradients (CG) is appropriate for solving *(0.1)*. When x is not close to x_* the symmetric matrix H may not be positive definite. Nevertheless, it is possible to 'apply' CG until indefiniteness is discovered - at this point some alternative action must be taken. Steihaug's [1983] algorithm is an example: this method generalizes the notion of Powell's [1970] 'dogleg' step and allows for the use of a conjugate gradient procedure to determine a suitable step even in the presence of an indefinite Hessian matrix.

It is well known that a CG-based method can generally be improved (ie. made more efficient) with the use of a preconditioner (PCCG). The purpose of this paper is to propose and experimentally explore a general algorithm for determining preconditioning matrices for use in an optimization context.

The basic idea behind the construction of a preconditioning matrix is as follows. Given H, we find a permutation matrix P so that a block-diagonal of the matrix PHP^T, $D = diag(D_i)$, is chordal. That is, each diagonal block D_i can be symmetrically permuted so that it incurs no fill under Cholesky factorization (assuming D_i to be positive definite). The proposed algorithm attempts to find chordal blocks D_i which are as 'heavy' as possible. The no-fill property allows for the implementation of the preconditioned conjugate gradient algorithm in space $O(\eta(H))$ where $\eta(\cdot)$ is the number of nonzeroes. Note that since each diagonal block D_i is a submatrix of H, if H is positive definite then so is D. If D_i is indefinite then this fact will be discovered during the Cholesky factorization of D; D_i can then be replaced with a diagonal matrix with positive elements. Alternatively, since indefiniteness of D_i implies indefiniteness of H, it is also possible to construct a direction of negative curvature using the partially formed factor (Section 2).

The use of chordal preconditioning matrices is not new: banded matrices are chordal and they have been a popular choice for such a role. However, sparse optimization problems cannot be expected to possess banded Hessian matrices in general. Many other suggestions have been made regarding (non-chordal) preconditioners (eg. Gustafsson[1980], Manteuffel[1980], Munksgaard[1980]) but little work has been done with respect to general preconditioners for optimization. (Thapa [1984] has conducted a numerical study of some possibilities.)

Dearing, Shier, and Warner[1983] have also suggested a heuristic for determining a chordal matrix C that is 'close' to a given symmetric matrix H. Their heuristic eliminates off-diagonal nonzeroes from H, in a prescibed way, until the resultant matrix is chordal. This mechansim is entirely different from the procedure described here. Furthermore, their resultant matrix C is not comprised of submatrices of H (which is the case in our proposal). Hence one cannot conclude that if H is positive definite then so is C. This complicates the use of C as a preconditioner.

Our paper is organized as follows. Section 1 discusses chordality and introduces our algorithm for finding heavy chordal submatrices. In Section 2 we present the results of experiments on symmetric positive definite linear systems. Section 3 discusses the integration of our chordal preconditioner with Steihaug's trust region algorithm for sparse nonlinear minimization. Computational results on nonlinear minimization problems are presented in Section 4. Section 5 provides concluding remarks.

1. Chordal Submatrices

Chordal graphs have been much studied (eg. Golumbic[1980], Rose, Tarjan and Leuker[1976]) under various names: perfect elimination, triangulated, rigid circuit, or monotone graphs. There are several equivalent characterizations. For example, a graph $G=(V, E)$ is chordal if and only if every cycle of length 4 or more, $v_1, \ldots, v_t, v_1$ has a chord: $(v_i, v_j) \in E$ for some $1 \leq i < j \leq t$. Or, a graph is chordal if and only if every minimal x, y separator is completely connected (An x,y separator is a set of vertices whose removal leaves no path from x to y). However, the most relevant characterization is this: G is chordal if the vertices V can be ordered, $v_1, v_2, \ldots, v_n$ so that if $(v_i, v_j) \in E$, $(v_i, v_k) \in E$, and $i < j \leq k$ then $(v_j, v_k) \in E$. The importance of this characterization is that it is now easy to show (eg. George and Liu [1981]) that if G is the adjacency graph of a sparse symmetric matrix H, then H suffers no fill under Cholesky factorization over all feasible assignments of numerical values to the nonzeroes of H. (Element (i,j) is a fill element if $H_{ij} = 0$ but $L_{ij} \neq 0$ where L is the Cholesky factor of H.)

We define a sparse symmetric matrix to be chordal if its corresponding adjacency graph is chordal.

Chordal matrices can be recognized in time $O(n+\eta(H))$, where $\eta(H)$ is the number of nonzeroes in matrix H. To do this, first order the nodes according to Maximum Cardinality Search (MCS):

> Number the vertices from n to 1. As the next vertex to number, select a vertex adjacent to the most numbered vertices.

Then test for chordality by applying the last chordality characterization mentioned above. For more information see Rose, Tarjan, and Leuker 1976]. Golumbic [1980] also provides an interesting discussion of chordal graphs.

Unfortunately, very few practical problems give rise to sparse chordal matrices. Nevertheless, a purpose of this paper is to discover if chordal submatrices can often be found in practise; the chordal submatrix structure can be used to define a preconditioner which can then be efficiently factored (since no fill is incurred).

1.1 An Algorithm For Locating a Chordal Preconditioner

The purpose of this section is to describe a fast algorithm for identifying chordal submatrices. The algorithm is a greedy one; the driving force behind the algorithm is Theorem 1.1.1, which we state immediately following a few definitions.

A graph is a <u>clique</u> if it is completely connected. Let $G=(V,E)$ be the graph under consideration with $X \subseteq V$; $\langle X \rangle = (X, E(X))$ denotes the <u>subgraph induced</u> by X in G. That is, if $v,w \in X$ then $(v,w) \in E(X) \Leftrightarrow (v,w) \in E$. If $X \subseteq V$, $y \in V$, let $N_X(y) \subseteq V$ be the adjacency set of y in $\langle X \rangle$.

Theorem 1.1.1: Let $V_i \subseteq V$ induce a <u>chordal</u> subgraph $\langle V_i \rangle$, $i=1{:}k$. Assume that V_i and V_j are distinct sets of vertices and there is no edge in E incident to both V_i and V_j for $i,j=1{:}k$. Let $x \in V - \bigcup V_i$; further, assume $N_{V_i}(x) \neq \emptyset$ for $i=1{:}k$.

Then,

$$\langle N_{V_i}(x) \rangle \text{ is a clique for } i=1{:}k$$

$$\Rightarrow \quad \langle x \bigcup_{i=1}^{k} V_i \rangle \text{ is chordal .}$$

Proof: The proof rests on the fact that a chordal graph has no unchorded cycles. Assume that the induced graph $< x \bigcup_{i=1}^{k} V_i >$ is not chordal.

Hence there is an unchorded cycle. Clearly the cycle must contain x and must otherwise be entirely included in some set V_j. It follows that there are 2 vertices v, w in $N_{V_j}(x)$ which are also on this unchorded cycle. But this violates the assumption that $< N_{V_j}(x) >$ is a clique. $\square$

Figure 1.1.1 illustrates the construction used in the proof. Vertex sets V_1, V_2, and V_3 each induce disjoint chordal subgraphs. Vertex x is not in any of the three sets and $N_{V_i}(x)$ induces a clique for $i=1{:}3$.

These 3 neighbour sets are identified by the rectangles in Figure 1.1.1.

Hence the induced graph, $< x \bigcup_{i=1}^{3} V_i >$, is chordal.

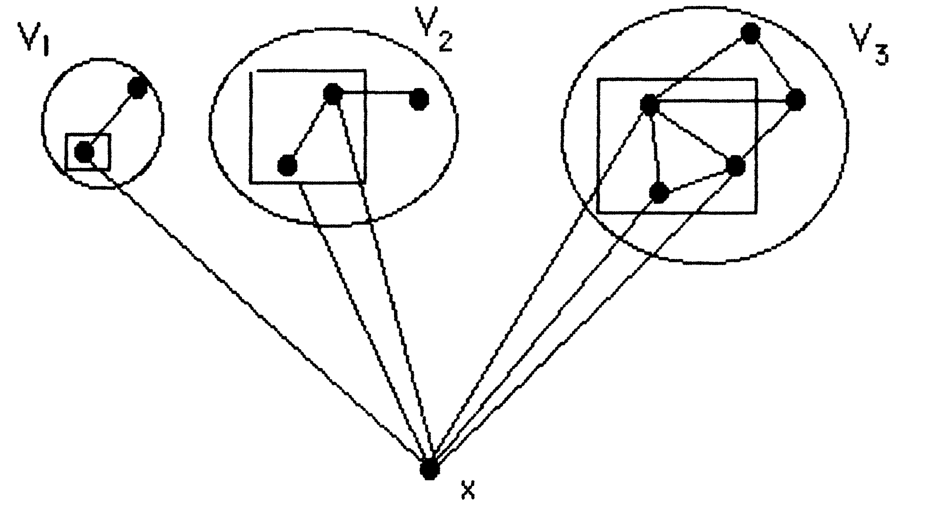

Figure 1.1.1

Theorem 1.1.1 exposes the central idea in our algorithm to find a large chordal submatrix C. Suppose the nodes of the adjacency graph have been ordered $v_1, v_2, \ldots, v_n$. At the beginning of step p, the first $p-1$ nodes have been considered for inclusion in V_C ; V_C can be partitioned into disjoint sets $V_1, \ldots, V_k$ such that $\langle V_i \rangle$ is chordal for $i=1:k$. Furthermore, there is no edge in E from V_i to V_j for $i,j=1:k$. Step p then considers including v_p in V_C: Let $I \subseteq \{1,2,\ldots,k\}$ identify sets V_i such that $N_{V_i}(v_p) \neq \emptyset$.

Then if each such set induces a clique, v_p is added to V_C and the sets identified by I are merged together (with v_p).

Algorithm Find_Chordal ($G=(V,E), V_C$)

- Order nodes $v_1, \ldots, v_n$
- $V_C := \emptyset, \quad k := 0$
- *For* $p=1:n$
  - $I(v_p) := \{i : N_{V_i}(v_p) \neq \emptyset, \ i=1:k\}$
  - If $\{ \langle N_{V_i}(v_p) \rangle = clique \ \ \forall \, i \in I(v_p) \}$ *then*
    - $V_C := V_C \cup \{v_p\}$
    - union V_i *for* $i \in I(v_p)$ and v_p, adjust k

Notice that algorithm Find_Chordal affords an efficient implementation since it is merely a particular expression of the well known union-find problem (eg. Aho, Hopcroft and Ullman [1974]).

In order to find a symmetric permutation that induces a chordal block diagonal, one merely applies algorithm *Find_Chordal* repeatedly:

Algorithm Find_P

Repeat until $V = \emptyset$
$\quad$ *find_chordal*$(G=(V,E), V_C)$
$\quad$ $G := (V - V_C, \ E := E(V - V_C))$

We have not yet specified the vertex ordering scheme. It is reasonable to put as much 'weight' in the chordal diagonal blocks as possible. Hence we have implemented a dynamic ordering scheme in which the next vertex to be considered is that which is most connected to the partially formed chordal block under construction. Specifically, if C is the partially formed chordal block under construction and V_C the vertex set, then for each vertex not yet assigned to a chordal block, define the connectivity weight:

$$cw(v) = \sum_{w \in V_C} |h(w,v)| - \sum_{w \in W_C} |h(w,v)|$$

The set W_C is the set of nodes not yet assigned to a chordal block; $h(w,v)$ is the value of the matrix element. The next vertex chosen is a maximizer of $\{cw(v): v \in W_C\}$.

1.1.1 Some Properties of Algorithm Find_P

Unfortunately, algorithm Find_P does not necessarily choose $C=H$ when H is chordal. To see this suppose $G(H)$ is the adjacency graph depicted in Figure 1.1.2.

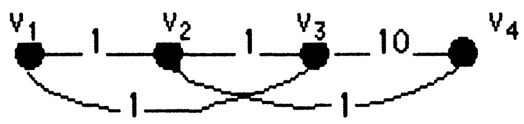

Figure 1.1.2

An ordering by connectivity weight is v_1, v_3, v_4, v_2. Note that G is chordal; however, the preconditioner produced by Find_P consists of 2 blocks - $\{v_1, v_3, v_4\}$ and $\{v_2\}$- because v_2 is adjacent to 3 vertices in the same component but they themselves are not pairwise adjacent. Despite this failure, Find_Chordal will recognize certain classes of chordal graphs. A forest is a graph without cycles; if it is also connected, it is a tree.

Theorem 1.1.2: Let $v_1, v_2, \ldots, v_n$ be an arbitrary ordering of the nodes of graph $G=(V,E)$. If G is a forest then Find_Chordal assigns $V_C = V$.

Proof: Suppose that after $k-1$ steps $V_C = \{v_1, v_2, \ldots v_{k-1}\}$. Divide V_C into a maximal number of disjoint vertex sets $V_1, V_2, \ldots, V_l$ such that there is no edge from V_i to V_j for $i,j = 1{:}l$. But there is at most 1 edge from v_k to every set V_i, $i=1{:}l$; otherwise, by connectivity of each V_i, there is a cycle, which is a contadiction. But clearly then Find_Chordal assigns $V_C = V_C \cup \{v_k\}$: hence, Theorem 1.1.2 is established. □

Notice that Theorem 1.1.2 is applicable under any ordering of the vertices. If the vertices are ordered according to connectivity weight, then we have the following theorem concerning band graphs.

Definition: $G=(V,E)$ is a band graph with bandwidth β if there is an ordering $v_1, v_2, \ldots, v_n$ of the vertices of G such that for $i \neq j$,

$$|i-j| \leq \beta(G) \Leftrightarrow (v_i, v_j) \in E$$

For example, in Figure 1.1.3, G_1 is a band graph with $\beta = 1$ (corresponding to a tri-diagonal matrix); G_2 is a band graph with $\beta = 2$ (corresponding to a penta-diagonal matrix).

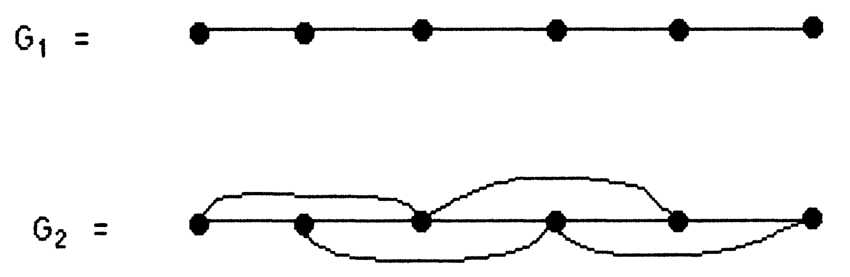

Figure 1.1.3

Theorem 1.1.3: Let $v_1, v_2, \ldots, v_n$ be a connectivity weight ordering of the nodes of G. Let G be a band graph of bandwidth β with the additional property that all edges have equal weight (all off-diagonal elements inside the band are equal-valued). Then, algorithm Find_Chordal yields $C = G$.

Proof: Assume that after k steps ,

$$V_C = \{v_{j+1}, \ldots, v_{j+k}\}.$$

But if $j+1 \neq 1$ then $cw(j) > cw(i)$ for $i=1:j-1$. Similarily, if $j+k \neq n$ then $cw(j+k+1) > cw(j+k+i)$ for $i=2:n-j-k$. Without loss of generality, assume the former. But $\langle N_{V_C}(v_j) \rangle$ is a clique and hence Find_Chordal assigns

$V_C = V_C \cup \{v_j\}$. Thus Theorem 1.1.3 is established by induction.□

1.2 Algorithms for Finding t-Chordal Submatrices

The chordal preconditioner defined above has the useful feature that it can be factored without using extra space. Hence, an upper bound on the required space to factor the preconditioner is just the number of nonzeroes in the original matrix, $\eta(H)$. Moreover, this number is known before computations begin, provided the structure of H is known. This allows for a static data structure. Nevertheless, it may be useful to exercise greater control over space. For example, the user may feel that $3n$ is the most space available for a preconditioner; this is the space required by a tri-diagonal preconditioner. Fortunately, the algorithms proposed above can be modified to allow for this flexibility.

Algorithm *find_chordal* will find a forest (instead of a general chordal graph) if the 'clique test' is replaced with a 'clique of size 1' test. More generally, a chordal submatrix of order r, with at most $2tr + 1$ nonzeroes, will be found if the 'clique test' is replaced with a 'clique of size t' test.

1.3 A Bordering Scheme

If extra space is available, it is possible to extend a chordal submatrix to a larger submatrix. This extra 'weight' in the preconditioner should decrease the number of required CG iterations. Of course this gain is partially mitigated by increased storage requirements and factorization expense. Nevertheless, if some limited extra space is available, it may well be worthwhile to use it.

To illustrate the idea, consider the following blocked matrix:

$$D = \left[\begin{array}{c|c} C & V \\ \hline V^T & B \end{array}\right]$$

where C is a chordal matrix, V is arbitrary (but sparse) and B is a small dense matrix (at least it will be treated as a dense matrix). Since D is a submatrix of H, D is positive definite. As George and Liu[1981] point out, it is possible to solve systems with a positive definite blocked matrix without explicitly forming the factor of the entire matrix. In particular, it is only necessary to factor C and the Schur-complement $B - V^T C^{-1} V$. For example, an algorithm to solve $Dx = b$ is as follows:

FACTOR:

$$C = L_1 L_1^T$$

Form $\bar{B} = B - V^T C^{-1} V$

$$\bar{B} = L_2 L_2^T$$

SOLVE:

$$Cy = b^1$$

$$\bar{B} x^2 = b^2 - V^T y$$

$$C x^1 = b^1 - V x^2$$

where $b^T = (b^1, b^2)^T$, $x = (x^1, x^2)^T$. If the block diagonal preconditioner consists of m blocks D_i of this form -each with a dense m_i by m_i corner submatrix - then the total space requirements are bounded by the number of nonzeroes in H plus

$$\sum_{i=1}^{m} m_i^2.$$

In our implementation, the user can specify in advance the space available for a preconditioner. The chordal blocks are then bordered (as above) as they are formed until a space bound is met. The vertices are added to submatrix B_i according to connectivity weight.

2. Experiments on Positive Definite Linear Systems

In this section we numerically compare 2 chordal preconditioners (tree, general) to a diagonal preconditioner (as well as to CG without preconditioning) on several sparse symmetric positive definite linear systems. We assume the reader is familiar with the conjugate gradient method for linear systems (eg. Golub and Van Loan [1983]). All computations were performed on a VAX 780, in Fortran 77, in double precision.

The sparse matrix structures we have chosen come from the Everstine [1979] collection of symmetric sparse matrix structures (Naval Problems) and a well-known set of sparse linear programming problems (obtained from Dr. Michael Saunders, Stanford University). The LP collection yields symmetric matrix structures through the product A^TA.

Our first set of experiments are concerned only with relatively small dimensional matrices from these collections: the extensive number of runs and the monitoring of condition number demanded a modest value of n. (We conclude this section with several larger experiments.) The densities of the symmetric test matrices are listed in the Table 1. Density is $\eta(H)/n{*}{*}2$.

Table 1: Matrix Density

Class	Dimension	Density
Naval	59	7.67
	66	7.35
	72	4.28
	87	7.15
	162	4.50
LP	27	21.
	56	23.
	96	18.
	173	18.

Values were assigned to the nonzeroes in a manner which allowed some (limited) control over conditioning while maintaining symmetry and positive definiteness. For the Naval Problems our technique was the following: First we generated a sparse upper triangular matrix R such that the structure of $R^T R$ was contained in the structure of the given symmetric matrix H. Random values were then assigned to the off-diagonal nonzeroes so that each entry was in the range [-1,1]. Finally, the diagonal elements were assigned values uniformly distributed over different ranges: this range was varied until matrices of different but 'reasonable' condition were generated. (An estimate of the condition number of each matrix is listed in our reported results.)

The LP data was used to generate symmetric positive definite matrices by forming

$$H = ADA^T$$

where the structure and value of A was given; the diagonal elements of D were assigned to be positive random numbers uniformly distributed over different ranges (varied to yield matrices of different condition).

Right hand sides were generated randomly in the range [-1,1].

The first experimental question we address is this: How well does C (produced by algorithm *Find_P*) approximate H? Since C is comprised of submatrices of H, a reasonable and convenient measure is

$$\|C\|_F / \|H\|_F$$

These ratios are recorded in the Chordal Weight Table. Note that t=1,t=2, and t=∞ correspond to diagonal, tree, and general chordal preconditioners respectively.

Table 2: Chordal Weights

Class	n	t=1	t=2	t=∞
naval	59	48	70	82
	66	45	70	94
	72	53	97	97
	87	49	71	83
	162	40	59	77
LP	27	71	83	87
	56	81	88	88
	96	23	44	99
	173	39	49	70

Roughly speaking, the diagonal preconditioner recovers about 45-55% of H; the tree preconditioner recovers about 65-75% of H; the general chordal preconditioner yields 80-95% . Hence in general we expect to observe a significant variation in the number of required CG iterations as t varies.

We present results on 2 sets of experiments on symmetric positive definite linear systems. First, we consider a 'high accuracy' solution to $Hs = -g$. In particular, PCCG is terminated when

$$\|Hs+g\|_2 \le \epsilon \|g\|_2$$

where $\epsilon = 10^{-5}$. The second table corresponds to low accuracy: ϵ is set to 10^{-1} in this case. The entries in the tables correspond to the number of PCCG iterations. The * column records the exponent of the estimated condition number (ie. cond(H) $\approx 10^{*}$) obtained by Linpack's DPOCO (Dongarra et al., [1978]).

Table 3: High Accuracy

Class	n	#	t=0	t=1	t=2	t=∞
Naval	59	2	33	31	23	19
		3	52	45	31	27
		6	95	63	40	30
	66	2	26	23	17	3
		3	55	38	28	3
		6	84	72	52	11
	72	2	41	41	8	8
		3	64	54	6	6
		6	93	64	6	6
	87	2	51	47	33	27
		3	63	58	42	34
		4	87	74	55	44
		6	124	84	63	49
	162	3	113	99	80	59
		4	153	141	108	81
		5	241	193	145	108
LP	27	3	27	14	11	8
		5	32	14	12	7
		7	>400	14	12	7
	56	6	78	26	17	14
		8	>400	29	27	17
	96	7	>400	>400	85	26
		8	>400	185	122	54
LP	173	6	>400	28	27	27
		7	259	41	40	37

Table 4: Low Accuracy

Class	n	#	t=0	t=1	t=2	t=∞
naval	59	2	8	7	6	4
		3	22	21	17	9
		6	77	56	40	30
	66	2	2	2	1	2
		3	3	2	2	1
		6	4	2	2	2
	72	2	7	5	2	2
		3	23	11	2	2
		6	84	59	6	6
	87	2	9	7	6	5
		3	24	23	17	14
		4	62	45	39	27
		6	95	80	57	44
	162	3	23	24	19	14
		4	65	59	44	31
		5	170	143	82	61
LP	27	3	18	3	3	2
		5	25	4	4	2
		7	>400	4	4	2
	56	6	64	20	11	9
		8	97	22	22	11
	96	7	>400	164	111	12
		8	>400	117	75	28
	173	6	163	25	26	23
		7	227	16	16	16

Comments:

1. In general the number of required iterations decreases as t increases. The difference between $t=0$ and $t=\infty$ becomes more marked as the condition number increases, in general. ($t=0$ corresponds to CG without preconditioning (or PCCG with $C = I$).)

2. For $t=0$ the number of required iterations often increases dramatically with condition number; for $t > 0$ the number of required iterations is considerably more insensitive to condition number.

3. The separation amongst the cases $t=1$, $t=2$, and $t=\infty$ is distinct in some cases and insignificant in others. This variation is usually related to the relative weights of the different preconditioners: see Table 1.

4. Except for 3 cases ($n=66$, $n=72$, $n=96$), the number of iterations for $t=\infty$ is never significantly less then half that required by the diagonal preconditioner ($t=1$). Notice that all three are in the 90% 'weight' range - the corresponding diagonal preconditioners 'weigh' about 50%.

As mentioned in Section 1.3, it is possible to increase the weight of the preconditioner by relaxing the chordality restriction somewhat. We have experimented with this idea, using the bordering scheme: the Extra Space Table records some of the results. The preconditioner then is still block diagonal; however, each chordal block is augmented with a dense block (Section 1.3). We have 2 sets of experiments - one with a dense block space of size $x=n/4$ and one dense block space of size $x=n/16$. The Extra Space Table records the results for the high accuracy runs on the most ill-conditioned Naval Problems.

Table 5: Extra Space

n	*	weight x=0	weight x=n/16	weight x=n/4	its x=0	its x=n/16	its x=n/4
59	6	82	85	88	30	24	18
66	6	94	97	99	11	8	5
72	6	97	98	98	6	4	2
87	6	83	84	86	49	42	26
162	5	77	76	81	108	104	76

As predicted, the number of requires iterations decreases as x increases.

We conclude this section with results on 3 larger problems from the Everstine collection: n = 758, 918, 1007 with densities 1.04, .88, and .85 respectively. We did not estimate the condition number of these matrices; however, the numerical values were generated in the fashion used on the previous (smaller) matrix structures. Hence we can expect the conditioning of the matrices to vary in a similar way. The asterisks indicate the relative ordering of the 'estimated' condition : eg. ** is more poorly conditioned than *, etc.

Table 6: Chordal Weights

n	t=1	t=2	$t=\infty$
758	39	57	71
918	45	64	78
1005	38	58	69

Table 7: High Accuracy

n	#	t=0	t=1	t=2	t=∞
758	*	18	18	17	15
	**	26	23	21	16
	***	52	40	37	23
	****	286	179	90	75
918	*	19	18	17	15
	**	24	22	19	16
	***	41	30	25	20
	****	307	146	46	38
1005	*	17	17	17	15
	**	25	22	17	15
	***	49	38	33	23
	****	381	252	116	166

Table 8: Low Accuracy

n	#	t=0	t=1	t=2	t=∞
758	*	2	2	2	2
	**	2	2	2	2
	***	4	3	2	2
	****	14	8	8	7
	*****	1061	586	445	334

n	#	t=0	t=1	t=2	t=∞
918	*	2	2	2	2
	**	3	2	2	2
	***	3	2	3	2
	****	6	4	3	2
1007	*	2	2	1	1
	**	2	2	2	1
	***	3	2	2	1
	****	7	3	2	2

Once again the variation is predictable: the number of CG iterations decreases as t increases. In this case differences are quite manner on the low accuracy examples.

In summary, the experiments in this section certainly support the hypothesis that the number of CG iterations will decrease as t increases. In some cases the move from t=1 to t= ∞ results in a dramatic decrease; in others, the decrease is modest. The difference is related to weight difference, condition number, and accuracy demanded.

Of course the most important computational question is this: Does the chordal preconditioner provide a <u>net</u> computational gain over a diagonal preconditioning scheme. We defer a discussion of this question to Section 5 so that the nonlinear experiments is Section 4 can also bear on the discussion.

3. Preconditioned Conjugate Gradient Trust Region Algorithm

In this section we describe the integration of our chordal preconditioner with Steihaug's[1983] conjugate gradient trust region algorithm.

Suppose we are interested in finding a local minimizer of the function $f: R^n \to R^1$ where f is twice continuously differentiable. Let x be the current point. In general, a trust region method computes a trial correction, s, to the current point by modelling f with a quadratic function defined at x, $q(s) \equiv g^T s + 1/2\, s^T Hs$, where g is the gradient vector evaluated at x and H is the Hessian matrix (at x). The following problem is then solved (perhaps approximately):

$$\min \{ q(s): \|s\| \leq \Delta \} \qquad (3.1)$$

Steihaug suggests using a weighted norm: $\|s\|_C \equiv (s^T Cs)^{1/2}$, where C is symmetric positive definite; problem (3.1) is solved approximately using conjugate gradients. The method of conjugate gradients is an iterative method in itself: a sequence of approximate solutions to (3.1) is generated, $s_1, s_2, \ldots, s_m$, with the properties

$$\|s_i\|_C > \|s_{i-1}\|_C, \quad q(s_i) \leq q(s_{i-1}) \quad i=1:m$$

The index m is determined when either the boundary Δ is crossed or the residual, $Hs_i + g$ is acceptably small.

The method of conjugate gradients is generally reserved only for positive definite systems; however, the Hessian matrix $H(x)$ need not be positive definite. Nevertheless, Steihaug points out that CG can be applied to (3.1) in the following way. CG is well-defined provided only positive curvature is encountered: ie. $d_i^T Hd_i > 0$, where d_i is the direction along which s_i is to be corrected. If $d_i^T Hd_i \leq 0$ then d_i is a direction of non-positive curvature; an approximate solution to (3.1), $s_{i+1} \equiv s_m$ with $\|s_m\| = \Delta$, is immediately available. For completeness, we reproduce Steihaug's algorithm to approximately solve *(3.1)*:

Step 1:

- $s_0 := 0,\ r_0 := -g$
- solve $C\tilde{r}_0 = r_0$
- $d_0 := \tilde{r}_0,\ \ i := 0.$

Step 2:

- $\gamma_i := d_i^T H d_i$
- If $\gamma_i \leq 0$ then

  compute $\tau \ni \|s_i + \tau d_i\|_C = \Delta$

  $s := s_i + \tau d_i$ return
- else

Step 3:

- $\alpha_i := r_i^T \tilde{r}_i / \gamma_i\ ,\quad s_{i+1} := s_i + \alpha_i d_i$
  - If $\| s_{i+1} \| \geq \Delta$ then

    compute $\tau \ni \|s_i + \tau d_i\|_C = \Delta$

    $s := s_i + \tau d_i$ return
- else

Step 4:

- $r_{i+1} := r_i - \alpha_i H d_i$
- If $\| r_{i+1} \|_C \leq \epsilon \|g\|_C$ then

  $s := s_i,$ return
- else

Step 5:

- solve $C\tilde{r}_{i+1} = r_i$
- $\beta_i := r_{i+1}{}^T \tilde{r}_{i+1} / r_i^T \tilde{r}_i\ ,\quad d_{i+1} := \tilde{r}_{i+1} + \beta_i d_i$
- $i := i+1$, go to **Step 2**

Of course the step s produced above is only a trial step: it may or may not be accepted depending on the change in the objective function f. The reader is referred to Powell[1970] (also Steihaug [1983]) for a full description of this acceptance test and the overall trust region strategy (eg. Δ will need to be adjusted occasionally).

Step 1 needs to be expanded to include the computation of the preconditioner. Specifically, the Cholesky factorization of the block diagonal preconditioner will be attempted in this step: If it is unsucessful then the offending blocks are replaced by diagonal matrices.

Step 1.

- $r_0 := -g$
- find chordal blocks $D = (D_1, D_2, \ldots, D_k)$ (Find_P)
- for $i=1:k$
  - if $D_i = L_i L_i^T$ then
    $C_i := D_i$
  - else
    $C_i := |\text{diagonal of } D_i|$
  - endif
  - $\tilde{r}_0^i := C_i^{-1} r_0^i, \quad d_0^i := \tilde{r}_0^i$
- $s_0 := 0, \ i := 0$

End of Step 1

The failure of the Cholesky factorization of a block can also allow for the determination of a direction of negative curvature: we have not employed the use of such a direction in our implementation but we sketch the computation below - this approach is based on the work of Gill, Murray, and Wright [1981].

Assume for the moment that H is symmetric positive definite. We can write

$$H = \left[\begin{array}{c|c} \bar{H} & v \\ \hline v^T & a \end{array}\right]$$

where $\bar{H}$ is symmetric positive definite. Cholesky factorization can be implemented recursively:

- factor $\bar{H} = \bar{L}\bar{L}^T$
- solve $\bar{L}w = v$
- $u := (a - w^T w)^{1/2}$

yielding the Cholesky factor of H: $L = \left[\begin{array}{c|c} \bar{L} & \\ \hline w^T & u \end{array}\right]$

If $\bar{H}$ is positive definite but H is indefinite, then the above procedure fails because $a - w^T w < 0$. Nevertheless, in this case we can write

$$H = \tilde{L} D \tilde{L}^T \text{ where } \tilde{L} = \left[\begin{array}{c|c} \bar{L} & \\ \hline w^T & u \end{array}\right]$$

and $D = diag(1, 1, \ldots, 1, -1)$. The solution to the system

$$\tilde{L}^T d = e_n$$

yields a direction of negative curvature since $d^T H d = -1$.

To apply this reult in our context, notice that if $\bar{d}_i$ is a direction of negative curvature for block D_i then $d_i = (0, \bar{d}_i, 0)$ satisfies the negative curvature condition $d_i^T H d_i < 0$. In a similar way define d_i for

$i \in J$ (an index set).

Then $d = \sum_{i \in J} d_i$ satisfies $d^T H d < 0$. Hence every indefinite block can contribute to the negative curvature direction.

4. Experiments on Sparse Nonlinear Minimization Problems

We have implemented the preconditioned conjugate gradient/trust region algorithm of Section 3 and conducted experiments on a number of sparse nonlinear optimization problems. The purpose of this section is to present and discuss our results.

The first problem class of test problems is due to Toint [1978]. Let $S = \{(i,j)\}$ be the index set of nonzeroes of H. We assume that S is symmetric ($(i,j) \in S \Rightarrow (j,i) \in S$) and all diagonal elements belong to S $((i,i) \in S,\ i=1{:}n)$ Define

$$f(x) = \sum_{(i,j) \in S} \sin(\beta_i x_i + \beta_j x_j + \gamma_{ij})$$

where $\beta_i = i/n, \quad \gamma_{ij} = (i+j)/n$.

Numerical results for sparsity patterns defined by 2 Naval Problems are presented (n=59, n=162). We have experimented with several other sparsity patterns from this collection; however, the presented results are representative of the rest.

In the test problem tables, **t** indicates the type of preconditioner (as before), **weight** reflects the Frobenius norm of the preconditioner divided by the Frobenius norm of the initial Hessian matrix, **high-CG** is the total number of CG iterations required when high accuracy is demanded from each linear system: ie. $\|Hs+g\|_2 \leq 10^{-5} \| g\|_2$. **Low-CG** gives the total number of CG iterations required when each linear system was solved to low accuracy: $\|Hs+g\|_2 \leq 10^{-1} \| g\|_2$. The **high-M** and **low-M** numbers represent the total number of **M**ajor iterations that were performed in the high and low accuracy situations respectively. Finally, the **neg. curve** entries reflect the number of major iterations in which negative curvature was discovered. The stopping criterion used was $\|g(x)\|_2 \leq 10^{-5}$.

Table 9 : Trigonometric Test Problem

$n = 59$ cond # $= 10^4$

t	weight	high-CG	low-CG	high-M	low-M	neg. curve
0		344	349	35	37	31
1	60	89	57	36	38	32
2	76	77	51	36	37	32
∞	82	78	53	36	38	32

$n = 162$ cond # $= 10^5$

t	weight	high-CG	low-CG	high-M	low-M	neg. curve
0		873	920	50	52	46
1	57	81	66	52	55	48
2	66	103	67	44	47	37
∞	88	79	57	43	45	38

Obviously there is a big jump in total CG iterations from t=0 to t=1 but little to distinguish t=1 from t=2 from t= ∞ . This similarity in performance for positive values of t is directly related to the presence of negative curvature. Indeed, in each problem, negative curvature was found in all but the last few iterations; negative curvature was always discovered after just a few CG iterations on each linear system - an approximate solution on the boundary of the trust region was then determined. The different preconditioners had little effect until the last few major iterations.

The t=0 runs were considerably more expensive than the rest. Once again negative curvature was frequently discovered but in this case the discovery was made only after many CG iterations.

To contrast with this class of problems, the next problem contained only positive curvature. A nonlinear minimization problem can be obtained by transforming a sparse LP to a nonlinear form using a penalty/barrier function. Specifically, a problem of the form

$$\min \{ c^T x : Ax=b, \ x_i \geq 0, \ i=1:n \} \qquad (LP)$$

yields the following nonlinear penalty/barrier function:

$$f(x) = \mu c^T x - \mu^2 \sum \ln x_i \ + 1/2 \, \|Ax-b\|_2^{\,2}.$$

Hence the gradient $g(x)$ equals $\mu c - \mu^2 D^{-1} e + A^T(Ax-b)$ and the Hessian $H(x)$ equals $\mu^2 D^{-2} + A^T A$, where $D = diag(x_1, x_2, \ldots, x_n)$.
Clearly the Hessian matrix is positive definite for all strictly positive x and $\mu > 0$. We define $f(x) = \infty$ if $x_i < 0$ for any i.

Our purpose here is not to advocate a particular method for solving (LP): rather, we are interested in solving an interesting sparse optimization problem with positive curvature. Our experiments were based on a fixed value of μ: $\mu = 1$ (hence only a <u>very</u> inaccurate solution to (LP) was determined).

Table 10: LP Test Problem

n = 51 cond * = 10^6

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	30	1516	1640	136	175
2	39	1203	1130	140	140
∞	71	517	475	72	112

In this case we terminated the t=0 runs because of the excessive number of CG-iterations (>2000). The number of required CG iterations for the diagonal preconditioning case (t=1) is still quite high and the tree pre-conditioner (t=2) does not help much. (Notice the the weights differ only slightly between t=1 and t=2). The full chordal preconditioner is able to substantially reduce the number of required CG-iterations for both the low and high accuracy approaches.

We derived the final class of nonlinear problems using the Naval structures. In particular, define

$$f(x) = 1/2\,(c - b^T x + 1/2\, x^T H x)^2$$

where H is a matrix used in the Naval problems of Section 2, b is the right-hand-side used in Section 2, and the constant c is chosen so that the function $f(x)$ is strictly bounded from zero. If we define

$$q(x) = c - b^T x + 1/2\, x^T H x$$

then the gradient of f is $g(x) = q(x) * (Hx-b)$ and the Hessian of f is

$q(x) * H + g(x)g(x)^T$. Since the Hessian of f at the solution x_* is $q(x_*) * H$, we have approximated the Hessian throughout the computations with $q(x) * H$. The starting point is the zero vector in all cases.

Table 11: Nonlinear Naval Problems

n = 59 cond # = 10^2

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	52	65	54	5	10
2	72	45	41	5	9
∞	85	33	29	5	9

n = 59 cond # = 10^4

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	50	109	167	11	15
2	72	75	114	10	14
∞	85	51	221	10	48

n = 66 cond # = 10^2

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	47	88	91	6	10
2	65	45	65	5	10
∞	99	5	7	4	5

n = 66 cond # = 10^8

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	46	593	435	68	64
2	70	505	318	72	70
∞	94	251	136	61	64

n = 72 cond # = 10^2

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	62	25	30	4	9
2	97	7	14	4	8
∞	97	7	14	4	8

n = 72 cond # = 10^3

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	60	69	75	7	11
2	97	7	21	4	9
∞	97	7	21	4	9

n = 87 cond # = 10^3

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	45	93	107	8	13
2	60	80	91	8	13
∞	78	58	59	7	11

n =162 cond # = 10^3

t	weight	high-CG	low-CG	high-M	low-M
0		*	*	*	*
1	34	83	123	6	11
2	52	89	74	6	11
∞	75	57	44	6	10

Clearly the number of required CG iterations decreases as t increases. Dramatic decreases occur in cases 3,5, and 6 corresponding to weights in the mid to upper 90's. (In case 4, where weight = 94 for t = ∞, there is no corresponding jump: this is due to the very high condition number.) Surprisingly, in the majority of the cases it is cheaper to solve the subproblems to high accuracy: the total number of major iterations is sufficiently lower to cause a net win for the high accuracy strategy.

5. Concluding Remarks

We have proposed an automatic preconditioning mechanism for use within a general large sparse optimization context. The preconditioning matrices are chordal: ie. the Cholesky factorization incurs no fill. Hence the Cholesky factor of the preconditioner can fit within the data structure needed for the sparse Hessian matrix. Furthermore, the heuristic procedure used to find chordal submatrices can be further restricted to allow for t-chordality. So, for example, if t=1 then the structure of the preconditioner is a <u>forest</u> with at most $3n$ nonzeroes (a tridiagonal matrix is a particular example of a forest).

It is clear from the numerical results presented in Sections 2 & 4 that chordal preconditioning, as defined by algorithm Find_P, can greatly reduce the number of CG iterations required. In general as t increases the weight of the preconditioner increases; in general as the weight of the preconditioner increases the number of CG iterations decreases. Moreover, our experiments provide several dramatic reductions. Specifically, if we compare diagonal preconditioning with full chordal preconditioning, we observe an order of magnitude reduction in CG iterations in the following cases:

- Table 1, n = 66, * = 2, 3, 6
- n=72, * 3, 6
- n=96, * 7,8
- Table 8, n = 918, * = ****
- Table 11, n = 66, * = 7, 8
- n = 72 , * = 2, 3

There are also examples of dramatic reductions as t changes from 2 (forest) to ∞ . For example, consider n=66, * = 2, in Table 11.

Unfortunately, we believe that the dramatic examples mentioned above represent the only cases where the chordal preconditioner pays for itself relative to a diagonal preconditioner. For example, in many cases the ratio between t=1 and t=∞ , is approximately 3:1 (wrt * CG iterations). It is doubtful that this decrease will compensate for the extra expense in the determination of C, the factorization of C, and the solve with C . We have not done a detailed timing analysis (our experimental code would require extensive fine-tuning) but rough estimates indicate a cost ratio (between diagonal and general chordal preconditioners) of at least 1:5. Hence we can hope for a net decrease in computation time only on the dramatic examples mentioned above. Notice that in each of the dramatic examples, the weight of the chordal preconditioner is very high ($\geq$ 94) . This fact, along with the observation that weights in the 80's produce just modest reductions, suggest that a block preconditioning scheme (with blocks induced by submatrices of H) will be effective only when either a very high percentage of the matrix is 'covered', or there is a very efficient method for solving the block diagonal system (eg. fast parallel computation).

We conclude with a comment on another potential use of the chordal partition produced by algorithm Find_P. Briefly, the chordal matrix C can be used to define a matrix splitting for use within a block Gauss-Seidel setting: it is a very convenient spliiting because of the no-fill property, and because C inherits positive definiteness of H. Indeed, such ideas can also be used to extend the successive projections approach to least squares problems suggested by Dennis and Steihaug[1983] (see also Coleman[1984], pp. 24, # 4). In this approach to large sparse overdetermined least squares problems $Ax \approx b$, let A be partitioned $A = (A_1, A_2, \ldots, A_t)$ with $x = (x^1, x^2, \ldots, x^t)$ partitioned accordingly. Assume that A is m by n. Then, the iteration

$x_0 := 0$
For $k=1:\infty$
  For $i=1:t$

Solve, in the least squares sense

$$A_i x_k^i = b - \sum_{j=1}^{i-1} A_j x_k^j - \sum_{j=i+1}^{n} A_j x_{k-1}^j \qquad (5.0)$$

This iteration is mathematically equivalent to a block Gauss-Seidel process on the normal equations. Dennis and Steihaug[1983] and

Coleman[1984] suggest permuting the columns of A and choosing the partition so that each group A_i consists of structurally independent columns ($a_k, a_j \epsilon A_i \Rightarrow a_{rk} * a_{rj} = 0$, $r=1:m$). Hence $A_i^T A_i$ is diagonal and (5.0) is easily solved using a 2-norm calculation.

The chordal subgraph algorithm (Find_P) allows for a generalization: A can be partitioned, $A = (A_1, A_2, ..., A_t)$, so that each adjacency graph of $A_i^T A_i$ is chordal. Hence the QR factorization of A_i will yield a matrix R_i that fits into the space required by $A_i^T A_i$. This generalization will usually reduce the number of groups in the partition (s< t) and should lead to better convergence behaviour of the successive projection iteration.

6. Acknowledgements

We thank Jorge More' for several useful discussions in the early stages of this research. Moreover, the mechanism for generating positive definite matrices, with a given nonzero structure, (described in Section 2) is his. We are also grateful to colleague John Gilbert for many illuminating discussions on chordality.

REFERENCES

Aho, Hopcroft, and Ullman [1974], *The design and analysis of computer algorithms,* Addison-Wesley.

T.F. Coleman [1984], *Large Sparse Numerical Optimization,* Springer-Verlag.

T.F.Coleman and J.J. More [1983], *Estimation of sparse Jacobian matrices and graph coloring problems,* SIAM Journal on Numerical Analysis, 20, 187-209.

________ [1984], *Estimation of sparse Hessian matrices and graph coloring problems,* Mathematical Programming, 28, 243-270.

T.F.Coleman, B. Garbow and J.J. More[1984], *Software for estimating sparse Jacobian matrices,* ACM Transactions on Mathematical Software, 10, 329-347.

________ [1985], *Software for estimating sparse Hessian matrices,* ACM Transactions on Mathematical Software, 11, 363-378.

T.F. Coleman and Jin-yi Cai[1986], *The cyclic coloring problem and estimation of sparse Hessian matrices,* SIAM Journal on Algebraic and Discrete Methods, 7, 221-235.

P.M. Dearing, D.R. Shier, and D.D. Warner[1983], *Maximal chordal cubgraphs,* Department of Mathematical Sciences Technical Report 406, Clemson University.

J.E. Dennis, JR. and T. Steihaug[1983], *On the successive projections approach to least-squares problems,* Department of Mathematical Sciences Technical Report 83-18, Rice University.

J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart [1978], *LINPACK Users Guide,* SIAM Publications, Philadelphia.

G.C. Everstine[1979], *A comparison of three resequencing algorithms for the reduction of matrix profile and wave front,* International Journal on Numerical Methods in Engineering, 14, 837-853.

A.J. George and J.W. Liu[1981], *Computer solution of large sparse positive definite systems,* Prentice-Hall, Englewood Cliffs.

P.E. Gill, W. Murray, and M.H. Wright[1981], *Practical Optimization,* Academic Press, New York.

G.H. Golub and C.F. Van Loan[1983], *Matrix Computations,* The Johns Hopkins University Press.

M.C. Golumbic [1980], *Algorithmic Graph Theory and Perfect Graphs,* Academic Press.

I. Gustafsson [1980], *A class of first-order factorization methods,* BIT, 18, 142-156.

T.A. Manteuffel[1980], *An incomplete factorization technique for positive definite linear systems,* Mathematics of Computation , 34, 473-497.

N. Munksgaard[1980], *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients,* ACM Transactions on Mathematical Software, 6, 206-219.

M.J.D. Powell[1970], *A new algorithm for unconstrained optimization,* in Nonlinear Programming, J.B. Rosen, O.L. Mangasarian and K. Ritter , eds., Academic Press, New York, 31-65.

M.J.D. Powell and Ph.L. Toint[1979], *On the estimation of sparse Hessian matrices,* SIAM Journal on Numerical Analysis, 16, 1060-1074.

D.J. Rose, R.E. Tarjan and G.S. Leuker[1976], *Algorithmic aspects of vertex elimination on graphs,* SIAM Journal on Computing, 5, 266-283.

T. Steihaug[1983], *The conjugate gradient method and trust regions in l large scale optimization,* SIAM Journal on Numerical Analysis, 20, 626-637.

M.N. Thapa[1984], *Optimization of unconstrained functions with sparse Hessian matrices - Newton-type methods,* Mathematical Programming 29, 156-186.

Ph.L. Toint[1978], *Some numerical results using a sparse matrix updating formula in unconstrained optimization,* Mathematics of Computation, 32, 839-851