SIAM J. MATRIX ANAL. APPL.
Vol. 31, No. 5, pp. 2899–2920

ROBUST APPROXIMATE CHOLESKY FACTORIZATION OF RANK-STRUCTURED SYMMETRIC POSITIVE DEFINITE MATRICES*

JIANLIN XIA† AND MING GU‡

Abstract. Given a symmetric positive definite matrix A, we compute a structured approximate Cholesky factorization $A \approx \mathbf{R}^T\mathbf{R}$ up to any desired accuracy, where $\mathbf{R}$ is an upper triangular hierarchically semiseparable (HSS) matrix. The factorization is stable, robust, and efficient. The method compresses off-diagonal blocks with rank-revealing orthogonal decompositions. In the meantime, positive semidefinite terms are automatically and implicitly added to Schur complements in the factorization so that the approximation $\mathbf{R}^T\mathbf{R}$ is guaranteed to exist and be positive definite. The approximate factorization can be used as a structured preconditioner which does not break down. No extra stabilization step is needed. When A has an off-diagonal low-rank property, or when the off-diagonal blocks of A have small numerical ranks, the preconditioner is data sparse and is especially efficient. Furthermore, the method has a good potential to give satisfactory preconditioning bounds even if this low-rank property is not obvious. Numerical experiments are used to demonstrate the performance of the method. The method can be used to provide effective structured preconditioners for large sparse problems when combined with some sparse matrix techniques. The hierarchical compression scheme in this work is also useful in the development of more HSS algorithms.

Key words. robust preconditioner, Schur compensation, hierarchically semiseparable structure, off-diagonal compression

AMS subject classifications. 15A12, 65F05, 65F30

DOI. 10.1137/090750500

1. Introduction.

This paper is concerned with robust and efficient structured factorization and preconditioning for symmetric positive definite (SPD) matrices. In iterative solutions of linear systems, it is often critical to find effective preconditioners. Approximate factorizations such as incomplete LU and incomplete orthogonalization are often used to provide preconditioners. For SPD matrices, incomplete Cholesky factorizations have been shown to exist for some particular matrices such as M-matrices and H-matrices [41, 42]. Some robust or stabilized methods have been proposed for general SPD matrices (see, e.g., [1, 7, 8, 37]).

Given a dense SPD matrix A, we propose a method to compute a structured approximate Cholesky factorization up to a user-specified accuracy. The factorization can be used as an efficient approximate direct solver with reasonable accuracy or as a robust and effective preconditioner. The factorization uses off-diagonal block compression or rank-revealing orthogonal decompositions. In the meantime, the off-diagonal compression causes certain positive semidefinite matrices to be automatically and implicitly added to Schur complements. That is, the approximate Schur complements differ from the exact ones by positive semidefinite terms which are related to the information dropped in the compression (see, e.g., (2.7)). We call such a robust-

*Received by the editors February 23, 2009; accepted for publication (in revised form) by M. H. Gutknecht August 31, 2010; published electronically November 30, 2010.
http://www.siam.org/journals/simax/31-5/75050.html

†Department of Mathematics, Purdue University, West Lafayette, IN 47907 (xiaj@math.purdue.edu). The research of this author was supported in part by NSF grant CHE-0957024.

‡Department of Mathematics, University of California, Berkeley, CA 94720 (mgu@math.berkeley.edu).

ness technique *Schur compensation*, and the factorization method *Schur-compensated factorization*, named in a similar way to the techniques of diagonal compensation and diagonally compensated factorization [2, 8], where some positive off-diagonal entries are replaced by zeros and their original values are added to the diagonal.

Our preconditioner uses semiseparable structured matrices. Semiseparable matrices have attracted a lot of interests in the recent years, and are very useful in solving some linear systems, eigenvalue problems, PDEs, integral equations, and many more [4, 9, 14, 21, 25, 26, 30, 34, 40, 45]. See [46] for a bibliography. Methods based on semiseparable matrices exploit a low-rank property; that is, appropriate off-diagonal blocks have small (numerical) ranks (based on a reasonable tolerance) and can be compressed. Here, we introduce robustness techniques into semiseparable matrix factorization or preconditioning. That is, we compute a structured approximate factorization $A \approx \mathbf{R}^T\mathbf{R}$ with Schur compensation, where $\mathbf{R}$ is an upper triangular matrix in *hierarchically semiseparable* (HSS) form [20, 48]. The HSS representation has a nice hierarchical tree structure called an HSS tree. Our factorization is conducted along the traversal of a given HSS tree. The Schur compensation guarantees that $\mathbf{R}^T\mathbf{R}$ is positive definite independent of the compression tolerance. No extra stabilization step is needed to avoid breakdown. The structured approximate factorization can then be used as a robust preconditioner. Similar robustness techniques have been proposed in [6] in terms of hierarchical matrices. An algorithm using sequentially semiseparable (SSS) matrices [21] is also developed in [32] without related analysis.

Our HSS preconditioner is very efficient and effective for rank-structured problems where dense off-diagonal blocks have relatively small numerical ranks or have decaying singular values. It only costs $O(n^2)$ flops to compute the approximate HSS factorization and $O(n)$ to apply the preconditioner to a vector, where n is the order of the matrix. The HSS preconditioner needs only $O(n)$ storage or is data sparse even if A is dense. By dropping information in the off-diagonal compression, the method both improves the reliability and reduces the cost of the factorization. The factorization can also be used as an approximate direct solver with a reasonable accuracy.

Furthermore, the preconditioning technique has a good potential to still perform well even if the low-rank property is not obvious. The preconditioned matrix has significantly better condition than the original one, in general, even if we use a relatively large tolerance or manually specify a small numerical rank when building the preconditioner. See, e.g., Figure 2.1.

In addition, with this HSS algorithm, it is possible to develop structured sparse factorization methods which fully take advantage of sparse matrix techniques. For example, we can use the HSS factorization as a kernel routine for the dense fill-in in the multifrontal method [24] together with nested dissection [27]. Structured sparse solvers based on this idea are useful for solving some discretized PDEs [4, 5, 16, 30, 39, 47], where the fill-in in the factorization of the discretized matrix has the low-rank property. (Such HSS-based sparse solvers are generally restricted to some two-dimensional (2D) problems, since the special low-rank block structure in HSS representations limits the effective application of HSS matrices to only one-dimensional (1D) problems in general.)

The remaining sections are organized as follows. In section 2, the ideas of Schur-compensated factorization and preconditioning based on off-diagonal compression are discussed and analyzed. In section 3, we briefly review HSS structures. Section 4 presents the algorithm for general multiple block HSS factorization with Schur compensation. Numerical experiments are provided in section 5. We test dense intermediate matrices in the factorizations of some ill-conditioned discretized PDEs. Section

6 gives some concluding remarks.

2. Schur-compensated factorization and preconditioning based on off-diagonal compression. Compression of off-diagonal blocks is essential for the effectiveness of semiseparable structures and their operations. In this section, we provide the fundamental ideas of our preconditioning which uses structured approximate factorization with Schur compensation. For convenience, our discussions concentrate on real matrices, but they can be easily extended to complex ones.

2.1. Block compression. Here, by the *compression* of an $n_1 \times n_2$ block Ω, we mean an approximate factorization of Ω which reveals its numerical rank r. Such an approximate factorization can be a skeleton approximation [29, 44], a τ-accurate SVD (SVD with an absolute or relative tolerance τ for the singular values), a rank-revealing QR (RRQR) factorization, etc. In this paper, we mainly employ RRQR, although we use τ-accurate SVD in presenting the basic idea. Many RRQR algorithms have been developed [17, 18, 19, 31, 38]. As an example, the QR with column pivoting method [17, 28] computes the following factorization after k steps:

$$Q^{(k)}\Omega\Pi^{(k)} = \begin{pmatrix} R_{11}^{(k)} & R_{12}^{(k)} \\ 0 & R_{22}^{(k)} \end{pmatrix},$$

where $Q^{(k)}$ is a product of k Householder transformations, $\Pi^{(k)}$ consists of k permutations, and $R_{11}^{(k)}$ is nonsingular and upper triangular. In the next step, the column of $R_{22}^{(k)}$ with the largest norm and the smallest index is the pivot column. The factorization stops when the norm of $R_{22}^{(k)}$ or the norm of the pivot column drops below a given tolerance. Then k is the numerical rank r. (Sometimes, we may also set a bound for k instead of using a tolerance.) Ω is now approximated by a compressed form

$$\Omega \approx \underbrace{(Q^{(r)}(1:r,:))^T}_{n_1 \times r} \underbrace{\begin{pmatrix} R_{11}^{(r)} & R_{12}^{(r)} \end{pmatrix}(\Pi^{(r)})^T}_{r \times n_2},$$

where $Q^{(r)}(1:r,:)$ denotes the first r rows of $Q^{(r)}$. This procedure costs $O(rn_1n_2)$ flops. Similarly, a modified Gram–Schmidt process with column pivoting or other more complicated procedures can also be used.

2.2. Schur-compensated factorization and preconditioning. We introduce the preconditioner using block 2×2 cases. Generalizations to more blocks are shown in the remaining sections. Assume an $n \times n$ SPD matrix A has the following partition:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} \begin{matrix} m \\ n-m \end{matrix}. \tag{2.1}$$

Without loss of generality, we assume $m \le n/2$. First, compute a Cholesky factorization $A_{11} = \mathbf{R}_{11}^T\mathbf{R}_{11}$, where $\mathbf{R}_{11}$ is the (upper triangular) Cholesky factor of A_{11}. The traditional block Cholesky factorization of A proceeds as

$$A = \begin{pmatrix} \mathbf{R}_{11}^T & 0 \\ \Omega^T & \mathbf{R}_{22}^T \end{pmatrix} \begin{pmatrix} \mathbf{R}_{11} & \Omega \\ 0 & \mathbf{R}_{22} \end{pmatrix} \equiv \mathbf{R}^T\mathbf{R}, \tag{2.2}$$

where $\Omega = \mathbf{R}_{11}^{-T}A_{12}$, and $\mathbf{R}_{22}$ is the Cholesky factor of the Schur complement $S = A_{22} - \Omega^T\Omega$. In our approximate factorization, before $\mathbf{R}_{22}$ is computed, we

first compute an SVD of the off-diagonal block Ω:

$$\Omega = \begin{pmatrix} U & \hat{U} \end{pmatrix} \begin{pmatrix} \Sigma & \\ & \hat{\Sigma} \end{pmatrix} \begin{pmatrix} V^T \\ \hat{V}^T \end{pmatrix} = U\Sigma V^T + \hat{U}\hat{\Sigma}\hat{V}^T = U\Sigma V^T + O(\tau), \tag{2.3}$$

where $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$, $\hat{\Sigma} = \mathrm{diag}(\sigma_{r+1}, \ldots, \sigma_m)$, and τ is a tolerance so that

$$\sigma_1 \geq \cdots \geq \sigma_r \geq \tau \geq \sigma_{r+1} \geq \cdots \geq \sigma_m. \tag{2.4}$$

(We can also use a relative tolerance τ, or can manually specify the numerical rank r instead of τ.) The decomposition $\tilde{\Omega} = U\Sigma V^T$ is the τ-accurate SVD of Ω, or the compressed form of Ω. The Schur complement is now

$$S = A_{22} - V\Sigma^2 V^T - \hat{V}\hat{\Sigma}^2\hat{V}^T. \tag{2.5}$$

To obtain a preconditioner, we approximate Ω by $\tilde{\Omega}$ and then the Schur complement is approximated by

$$\tilde{S} = A_{22} - V\Sigma^2 V^T. \tag{2.6}$$

Clearly,

$$\tilde{S} = S + \hat{V}\hat{\Sigma}^2\hat{V}^T = S + O(\tau^2) \quad \text{(Schur compensation)}. \tag{2.7}$$

That is, the Schur complement is automatically compensated with a positive semidefinite term $\hat{V}\hat{\Sigma}^2\hat{V}^T$. The matrix U has orthonormal columns and does not participate in the computation of $\tilde{S}$. That is, $\tilde{S}$ is computed by a low-rank update $A_{22} - (\Sigma V^T)^T(\Sigma V^T)$ when r is small.

We then compute the Cholesky factorization $\tilde{S} = \tilde{\mathbf{R}}_{22}^T\tilde{\mathbf{R}}_{22}$ and obtain an approximate factorization of A,

$$A \approx \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}, \quad \tilde{\mathbf{R}} = \begin{pmatrix} \mathbf{R}_{11} & U\Sigma V^T \\ 0 & \tilde{\mathbf{R}}_{22} \end{pmatrix}. \tag{2.8}$$

$\tilde{\mathbf{R}}$ is thus guaranteed to exist so that $\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}$ is positive definite. The matrix $\tilde{\mathbf{R}}$ can be used as a preconditioner. The following proposition provides some results related to the preconditioned matrix $\tilde{A} = \tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1}$.

PROPOSITION 2.1. *If $\tilde{\mathbf{R}}$ in (2.8) is used as a preconditioner, then the preconditioned matrix is given by*

$$\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1} = \begin{pmatrix} I & C \\ C^T & I \end{pmatrix}, \tag{2.9}$$

where $C = (\hat{U}\hat{\Sigma}\hat{V}^T)\tilde{\mathbf{R}}_{22}^{-1}$ satisfies $||C||_2 < 1$. This yields

$$||\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1} - I||_2 = ||C||_2, \tag{2.10}$$

$$\kappa(\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1}) = \frac{1 + ||C||_2}{1 - ||C||_2}, \tag{2.11}$$

where $\kappa(\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1})$ is the 2-norm condition number of $\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1}$.

Proof. Let $\mathbf{R}$ be the exact Cholesky factor of A. The preconditioned matrix is $\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1} = (\mathbf{R}\tilde{\mathbf{R}}^{-1})^T(\mathbf{R}\tilde{\mathbf{R}}^{-1})$ and

$$\begin{aligned}\mathbf{R}\tilde{\mathbf{R}}^{-1} &= \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{11}^{-T}A_{12} \\ 0 & \mathbf{R}_{22} \end{pmatrix}\begin{pmatrix} \mathbf{R}_{11}^{-1} & -\mathbf{R}_{11}^{-1}(U\Sigma V^T)\tilde{\mathbf{R}}_{22}^{-1} \\ 0 & \tilde{\mathbf{R}}_{22}^{-1} \end{pmatrix} \\ &= \begin{pmatrix} I & [-(U\Sigma V^T) + \mathbf{R}_{11}^{-T}A_{12}]\tilde{\mathbf{R}}_{22}^{-1} \\ 0 & \mathbf{R}_{22}\tilde{\mathbf{R}}_{22}^{-1} \end{pmatrix} = \begin{pmatrix} I & (\hat{U}\hat{\Sigma}\hat{V}^T)\tilde{\mathbf{R}}_{22}^{-1} \\ 0 & \mathbf{R}_{22}\tilde{\mathbf{R}}_{22}^{-1} \end{pmatrix}.\end{aligned}$$

Thus,

$$\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1} = \begin{pmatrix} I & C \\ C^T & \tilde{\mathbf{R}}_{22}^{-T}[(\hat{U}\hat{\Sigma}\hat{V}^T)^T(\hat{U}\hat{\Sigma}\hat{V}^T) + \mathbf{R}_{22}^T\mathbf{R}_{22}]\tilde{\mathbf{R}}_{22}^{-1} \end{pmatrix},$$

where $C = (\hat{U}\hat{\Sigma}\hat{V}^T)\tilde{\mathbf{R}}_{22}^{-1}$. According to (2.5) and (2.6)

$$(\hat{U}\hat{\Sigma}\hat{V}^T)^T(\hat{U}\hat{\Sigma}\hat{V}^T) + \mathbf{R}_{22}^T\mathbf{R}_{22} = \hat{V}\hat{\Sigma}^2\hat{V}^T + S = \tilde{S}.$$

Equation (2.9) then follows since $\tilde{S} = \tilde{\mathbf{R}}_{22}^T\tilde{\mathbf{R}}_{22}$.

According to (2.9), $\left(\begin{smallmatrix} I & C \\ C^T & I \end{smallmatrix}\right)$ is equal to $\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1}$, which is SPD. Thus, $I - C^TC$ is SPD, since it is the Schur complement of the $(1,1)$ block in the Cholesky factorization of the SPD matrix $\left(\begin{smallmatrix} I & C \\ C^T & I \end{smallmatrix}\right)$. Therefore, $||C||_2 < 1$. Next, from (2.9), (2.10) is obvious. Equation (2.11) follows from a simple result that the 2-norm condition number of any matrix $\left(\begin{smallmatrix} I & C \\ C^T & I \end{smallmatrix}\right)$, with $||C||_2 < 1$, is $\frac{1+||C||_2}{1-||C||_2}$. This result can be found in, say, [23]. ∎

This proposition relates the error and the condition number in the preconditioning to the off-diagonal singular values. In particular, this is clear in a special case where $A_{22} \equiv I$ in the original matrix A. In such a situation, we can actually show that $||C||_2 = \sigma_{r+1}$ using standard algebra (the details are omitted since this is only a special example). This means that the error $||\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1} - I||_2$ is controlled by the largest dropped singular value σ_{r+1} in the off-diagonal compression, according to (2.10). Furthermore, we then also have $\kappa(\tilde{\mathbf{R}}^{-T}A\tilde{\mathbf{R}}^{-1}) = \frac{1+\sigma_{r+1}}{1-\sigma_{r+1}}$. To illustrate the effectiveness of this preconditioning technique in such a situation, we use an example where the singular values σ_i decay linearly in $(0,1)$. See Figure 2.1. The decay of $\kappa_i = \frac{1+\sigma_i}{1-\sigma_i}$ is also shown. We see that the values of κ_i decay much faster than those of σ_i. For a σ_i value which is not so small, κ_i is already reasonably close to 1. This means, by keeping only a small amount of the largest singular values in the off-diagonal compression, we can still get satisfactory preconditioning results. Thus, this preconditioning technique does not need the off-diagonal blocks to have strong low-rank structure, and we can still manually set a small r. In the meantime, dropping more singular values leads to better efficiency and storage.

In order for the results in this special case to apply to a general SPD matrix A with $A_{22} \neq I$, we can first block precondition A as

$$\begin{pmatrix} I & \\ & \bar{\mathbf{R}}_{22}^{-1} \end{pmatrix}^T A \begin{pmatrix} I & \\ & \bar{\mathbf{R}}_{22}^{-1} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12}\bar{\mathbf{R}}_{22}^{-1} \\ \bar{\mathbf{R}}_{22}^{-T}A_{12}^T & I \end{pmatrix},$$

where $\bar{\mathbf{R}}_{22}$ is the Cholesky factor of A_{22}. The previous compression process then applies to this matrix.

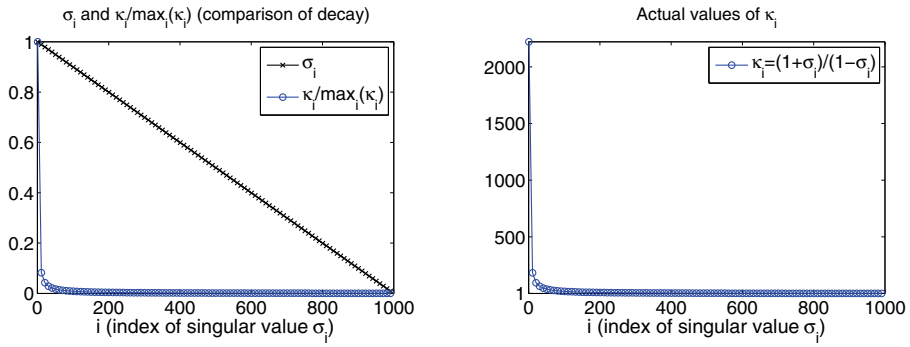


FIG. 2.1. *Example: The preconditoned condition numbers* $\kappa_i = \frac{1+\sigma_i}{1-\sigma_i}$ *decay faster than the off-diagonal singular values* σ_i*, where in the left graph the condition numbers* κ_i *are scaled to be within* $(0, 1]$ *with* $\kappa_i / \max_i(\kappa_i)$*, and the right graph shows the actual* κ_i*.*

Remark 2.2. The discussion here in terms of the block 2×2 situation does not give a final analysis of the error and the condition number in the preconditioning. However, Proposition 2.1 (especially in the special case where $A_{22} \equiv I$) serves as a motivational example showing the potential of structured preconditioning. This block 2×2 procedure uses factorizations of the diagonal blocks. It is possible to recursively apply the procedure to the diagonal blocks so as to reduce the cost. In practice, the procedure without the initial block diagonal preconditioning works very well, and the initial preconditioning may be skipped. This significantly simplifies the generalization of the block 2×2 procedure to multiple blocks. Although a similar analysis of the condition number for the multiblock procedure (section 4) is not yet available, we can demonstrate the effectiveness of preconditioning with numerical examples (section 5).

Remark 2.3. If RRQR (as in section 2.1), say, $\Omega = \begin{pmatrix} U & \hat{U} \end{pmatrix} \begin{pmatrix} R \\ \hat{R} \end{pmatrix} \approx UR$ instead of τ-accurate SVD is used, then the Schur compensation step (2.7) becomes $\tilde{S} = S + \hat{R}^T \hat{R}$. Or we can replace ΣV^T and $\hat{\Sigma}\hat{V}^T$ above by R and $\hat{R}$, respectively. The analysis above can be conducted similarly, with σ_{r+1} being the largest singular value of $\hat{R}$. However, the numerical rank r may be different. This depends on the difference between RRQR and τ-accurate SVD in how they decide the numerical rank. Since generally RRQR is cheaper (in our actual algorithm in section 4), we use RRQR.

To generalize the block 2×2 procedure to multiple blocks, we use HSS matrices whose hierarchical structure is suitable for multilevel compression. First we briefly review HSS structures.

3. Review of HSS structures.

HSS representations [20, 48] are closely related to SSS matrices and $\mathcal{H}$-matrices [13, 33, 34, 35], and can be viewed as special cases of $\mathcal{H}^2$-matrices [14, 36]. Existing HSS algorithms usually satisfy certain properties to enable stability. For example, in the representation, the basis matrices of off-diagonal blocks are generally made to have orthonormal columns.

A formal definition of HSS structures can be found in [48]. Here, we introduce HSS matrices in terms of a special case of cluster trees [12, 14]. Let $\mathcal{I}$ be the index set $\{1, 2, \ldots, n\}$, and let $\mathcal{T}$ be a binary tree. The nodes of $\mathcal{T}$ are denoted by integers. Each node i of $\mathcal{T}$ is associated with a subset of indices $t_i \subset \mathcal{I}$. We say $\mathcal{T}$ is a *postordered (binary) cluster tree* if it satisfies the following:

1. $\mathcal{T}$ is a full binary tree. That is, each node i either has no child (called a leaf)

or has two children (called a nonleaf node, and its children are denoted $c_{i;1}$ and $c_{i;2}$ (or sometimes c_1 and c_2) when no confusion is caused).

2. $\mathcal{T}$ is in its postordering. That is, the nodes are ordered in the way that a nonleaf node i is ordered after its children, and $c_{i;2}$ is ordered after $c_{i;1}$.
3. $t_i \neq \emptyset$ for each node i. The leaves satisfy $\cup_{\text{all leaves } i}\, t_i = \mathcal{I}$, and each nonleaf node i satisfies $t_{c_{i;1}} \cup t_{c_{i;2}} = t_i$ and $t_{c_{i;1}} \cap t_{c_{i;2}} = \emptyset$. If i is the root node, $t_i = \mathcal{I}$.

Then an order n matrix H is said to be in an HSS form if there exist matrices D_i, U_i, V_i, R_i, W_i, and B_i (called HSS generators) associated with each node i of a postordered cluster tree $\mathcal{T}$ satisfying

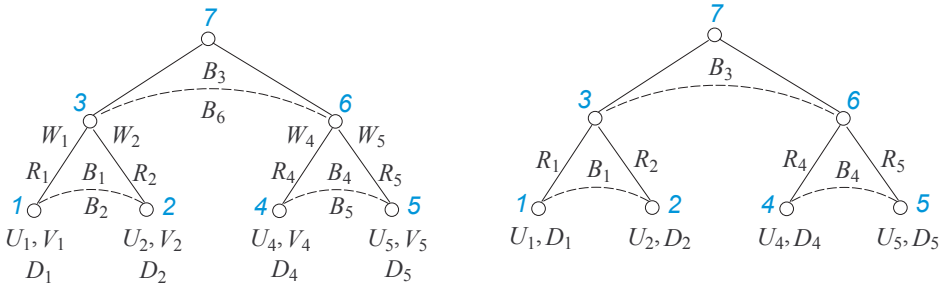
$$D_i = \begin{pmatrix} D_{c_{i;1}} & U_{c_{i;1}} B_{c_{i;1}} V_{c_{i;2}}^T \\ U_{c_{i;2}} B_{c_{i;2}} V_{c_{i;1}}^T & D_{c_{i;2}} \end{pmatrix}, \; U_i = \begin{pmatrix} U_{c_{i;1}} R_{c_{i;1}} \\ U_{c_{i;2}} R_{c_{i;2}} \end{pmatrix}, \; V_i = \begin{pmatrix} V_{c_{i;1}} W_{c_{i;1}} \\ V_{c_{i;2}} W_{c_{i;2}} \end{pmatrix},$$

so that $D_i \equiv A|_{t_i \times t_i}$ (the submatrix of A corresponding to row index set t_i and column index set t_i). Note that if i is the root, then R_i, W_i, and B_i are empty matrices and $D_i \equiv A$. The matrices U_i and V_i are defined recursively similarly to the nested cluster basis in [12, 14]. The R_i, W_i, B_i generators for all nodes i and the D_i, U_i, V_i generators for all leaves are explicitly stored. $\mathcal{T}$ is then also called an *HSS tree*.

The following is a block 4×4 HSS matrix example:

$$\begin{pmatrix} \begin{pmatrix} D_1 & U_1 B_1 V_2^T \\ U_2 B_2 V_1^T & D_2 \end{pmatrix} & \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \end{pmatrix} B_3 \begin{pmatrix} W_4^T V_4^T & W_5^T V_5^T \end{pmatrix} \\ \begin{pmatrix} U_4 R_4 \\ U_5 R_5 \end{pmatrix} B_6 \begin{pmatrix} W_1^T V_1^T & W_2^T V_2^T \end{pmatrix} & \begin{pmatrix} D_4 & U_4 B_4 V_5^T \\ U_5 B_5 V_4^T & D_5 \end{pmatrix} \end{pmatrix}. \tag{3.1}$$

Figure 3.1(i) shows the corresponding HSS tree.



(i) HSS tree for (3.1) (ii) Symmetric case satisfying (3.2)

FIG. 3.1. *HSS tree for* (3.1) *with postordering of the nodes.*

Remark 3.1. It is straightforward to verify that if an HSS matrix satisfies the following conditions, then it is symmetric:

$$D_i = D_i^T, \; U_i = V_i; \quad R_i = W_j, \; B_i = B_j^T \text{ (for sibling nodes } i \text{ and } j). \tag{3.2}$$

On the other hand, for a symmetric matrix A, an HSS form satisfying (3.2) can be constructed [48]. Figure 3.1(ii) shows an HSS tree for a symmetric matrix example.

HSS representations are very useful in reflecting the rank structure of certain off-diagonal blocks of a matrix. These off-diagonal blocks are $A|_{t_i \times (\mathcal{I} \setminus t_i)}$ (and $A|_{(\mathcal{I} \setminus t_i) \times t_i}$), called *HSS blocks*. They are block rows or columns without diagonal blocks and are

defined hierarchically at different levels of splitting of the matrix following $\mathcal{T}$. Figure 3.2 shows two levels of HSS blocks corresponding to the two levels of partitions in (3.1). If all of these HSS blocks have small ranks (or numerical ranks), the matrix is said to have a *low-rank property*. The maximum (numerical) rank of all the HSS blocks is called the *HSS rank* of the matrix. Clearly, U_i and V_i^T matrices are column and row bases, respectively, of appropriate HSS blocks. This can be clarified as follows. Without loss of generality, consider node $i = 2$ whose corresponding HSS block row and column in (3.1) and Figure 3.2 are

$$\begin{pmatrix} U_2B_2V_1^T & U_2R_2B_3\left(W_4^TV_4^T \quad W_5^TV_5^T\right)\end{pmatrix} = U_2\left(B_2V_1^T \quad R_2B_3\left(W_4^TV_4^T \quad W_5^TV_5^T\right)\right),$$

$$\begin{pmatrix} U_1B_1V_2^T \\ \begin{pmatrix} U_4R_4 \\ U_5R_5 \end{pmatrix} B_6W_2^TV_2^T \end{pmatrix} = \begin{pmatrix} U_1B_1 \\ \begin{pmatrix} U_4R_4 \\ U_5R_5 \end{pmatrix} B_6W_2^T \end{pmatrix} V_2^T.$$

We can see that U_2 and V_2^T are the column and row bases of these two blocks, respectively. More general and detailed discussions of similar block structures can be found in [11] in terms of total cluster basis, together with detailed approximation error analysis.

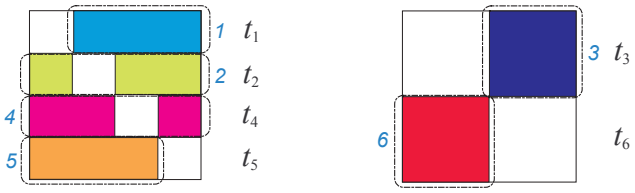


FIG. 3.2. *Two levels of HSS block rows of a matrix, where the blocks are numbered following the postordering of the nodes in Figure* 3.1*, and the row index sets* $t_1, \ldots, t_6$ *satisfy* $t_1 \cup t_2 = t_3$, $t_4 \cup t_5 = t_6$, *and* $t_3 \cup 6_6 = \mathcal{I} \equiv \{1, \ldots, n\}$.

A compact HSS form has R_i, W_i, B_i generators with small sizes which are close to the HSS rank. Generally, only linear storage is needed for a compact HSS form. Thus, even if the original matrix is dense, the compact HSS form is data sparse. Matrix operations with compact HSS forms are generally very efficient. For example, it only needs linear complexity to multiply two compact HSS matrices and to solve compact HSS systems. For a dense matrix with the low-rank property, fast algorithms exist to convert the matrix into an HSS form [20, 48], or more generally, an $\mathcal{H}^2$ form [14].

4. Multiblock HSS factorization with Schur compensation. In this section, we use the idea of Schur compensation in section 2 and extend the procedure to an approximate Cholesky factorization of a general dense SPD matrix A, where the Cholesky factor is in HSS form and can be used as a structured preconditioner.

In addition to all the previous HSS notation, the following is also used:

- First, for simplicity, we use RRQR in the compression. We also simply use $\mathbf{R}$ instead of $\tilde{\mathbf{R}}$ to denote the approximate Cholesky factor of A so that $A \approx \mathbf{R}^T\mathbf{R}$.
- For an index set $t_i \subset \mathcal{I}$, let $t_i^c \cup t_i \cup t_i^r = \mathcal{I}$, where all indices in t_i^c are smaller than those in t_i, and all indices in t_i^r are larger than those in t_i.
- $A|_{t_i \times t_j}$ is the submatrix of A with row index set t_i and column index set t_j. Thus in $\mathbf{R}$, blocks $\mathbf{R}|_{t_i^c \times t_i}$ and $\mathbf{R}|_{t_i \times t_i^r}$ are the (upper) HSS off-diagonal block column and row, respectively, associated with node i (see Figure 4.2).

- The place of $A|_{t_i \times t_i^r}$ is used to store $\mathbf{R}|_{t_i \times t_i^r}$. After the RRQR factorization of $\mathbf{R}|_{t_i \times t_i^r}$, the R-factor is still stored in $A|_{t_i \times t_i^r}$ with row index set $\hat{t}_i$. Also, $A|_{t_i^r \times t_i^r}$ is used to store the (approximate) Schur complement. With these storage arrangements, after each factorization step i (traversal of node i), the matrix A is updated to $A^{(i)}$ from $A^{(i-1)}$, where $A^{(0)} \equiv A$. For example, with $\mathbf{R}|_{t_i \times t_i^r}$ stored in $A^{(i-1)}|_{t_i \times t_i^r}$, its RRQR looks like $\mathbf{R}|_{t_i \times t_i^r} \approx U_i A^{(i)}|_{\hat{t}_i \times t_i^r}$.
- diag(...) represents a block diagonal matrix formed by the blocks in (...).

4.1. Algorithm overview. For a given HSS tree $\mathcal{T}$, the factorization of A is done along the postordering traversal of $\mathcal{T}$. The block rows of $\mathbf{R}$ are computed and the off-diagonal blocks are compressed. The major steps are outlined in Table 4.1, and are briefly explained as follows.

TABLE 4.1

Outline of the factorization, where c_1 and c_2 denote the children of a nonleaf node i. (R_{c_j} denotes an HSS generator associated with c_j and is not directly related to the notation $\mathbf{R}$ for the approximate Cholesky factor of A.)

Stage	Leaf node i	Nonleaf node i
Elimination	compute a block row: $D_i, \mathbf{R}\|_{t_i \times t_i^r}$	$\mathbf{R}\|_{t_{c_1} \times t_{c_1}^r}, \mathbf{R}\|_{t_{c_2} \times t_{c_2}^r} \Longrightarrow \mathbf{R}\|_{t_i \times t_i^r}$
Compression	$\Theta_i \equiv \begin{pmatrix} \text{from previous nodes} \\ (\mathbf{R}\|_{t_i^c \times t_i})^T \end{pmatrix}$ $\begin{pmatrix} \text{from elimination stage} \\ \mathbf{R}\|_{t_i \times t_i^r} \end{pmatrix}$ $\Downarrow$ ignore orthonormal bases $\boxed{\Omega_i}$ $\Downarrow$ QR $\Omega_i \approx U_i \bar{A}_i$	$\Omega_i \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \bar{A}_i$
Schur complement	if i is a left node, then form $A^{(i)}\|_{t_{i+1} \times t_i^r}$ for the next leaf $i+1$	

There is an elimination step associated with each leaf i to compute a block row of $\mathbf{R}$. The block row of $\mathbf{R}$ corresponding to a nonleaf node i is available from the child blocks. In the compression stage, the HSS block column $\mathbf{R}|_{t_i^c \times t_i}$ and HSS block row $\mathbf{R}|_{t_i \times t_i^r}$ are put together and compressed. Since the off-diagonal blocks involve existing compressed blocks, any previously computed orthonormal bases are ignored in further compression. We also maintain a compressed form of the contribution of existing off-diagonal blocks of $\mathbf{R}$ to Schur complements. A Schur complement is only partially formed with the portion corresponding to the next leaf explicitly computed.

4.2. A block 4×4 example. Before presenting the general algorithm, we first show an example. We demonstrate the procedure of factorizing A so that the (approximate) Cholesky factor $\mathbf{R}$ is a block 4×4 HSS matrix. According to Remark 3.1, we can assume the HSS form of $\mathbf{R}$ to be

$$\mathbf{R} = \begin{pmatrix} D_1 & U_1 B_1 U_2^T & U_1 R_1 B_3 R_4^T U_4^T & U_1 R_1 B_3 R_5^T U_5^T \\ & D_2 & U_2 R_2 B_3 R_4^T U_4^T & U_2 R_2 B_3 R_5^T U_5^T \\ & & D_4 & U_4 B_4 U_5^T \\ 0 & & & D_5 \end{pmatrix}, \tag{4.1}$$

whose HSS tree is in Figure 3.1(ii), except that the tree is only used for the block upper triangular part of $\mathbf{R}$. Based on the tree traversal we have the following steps.

(a) *Node* 1. First, we compute a Cholesky factorization $A|_{t_1\times t_1} = D_1^T D_1$, and let $\Omega_1 \equiv \mathbf{R}|_{t_1\times t_1^r} = D_1^{-T} A|_{t_1\times t_1^r}$. Compress it by a RRQR factorization $\Omega_1 \approx U_1 A^{(1)}|_{\hat{t}_1\times t_1^r}$, where U_1 has orthonormal columns. Then A is approximated by

$$A \approx \begin{pmatrix} D_1^T & \\ (A^{(1)}|_{\hat{t}_1\times t_1^r})^T U_1^T & I \end{pmatrix} \begin{pmatrix} D_1 & U_1 A^{(1)}|_{\hat{t}_1\times t_1^r} \\ & A^{(1)}|_{t_1^r\times t_1^r} \end{pmatrix},$$

where $A^{(1)}|_{t_1^r\times t_1^r} = A|_{t_1^r\times t_1^r} - (A^{(1)}|_{\hat{t}_1\times t_1^r})^T A^{(1)}|_{\hat{t}_1\times t_1^r}$ is the (approximate) Schur complement. $A^{(1)}|_{t_1^r\times t_1^r}$ is not fully formed. Instead, since node 1 is immediately followed by a leaf (node 2), only $A^{(1)}|_{t_2\times t_1^r}$, the block corresponding to node 2, is explicitly formed with a low-rank update. That is, let $Y_1 = A^{(1)}|_{\hat{t}_1\times t_1^r}$, and partition it as $\begin{pmatrix} Y_{1,1} & Y_{1,2} \end{pmatrix} \equiv \begin{pmatrix} A^{(1)}|_{\hat{t}_1\times t_2} & A^{(1)}|_{\hat{t}_1\times t_2^r} \end{pmatrix}$, so that

$$A^{(1)}|_{t_2\times t_1^r} = A|_{t_2\times t_1^r} - Y_{1,1}^T Y_1. \tag{4.2}$$

(b) *Node* 2. Note $A^{(1)}|_{t_2\times t_1^r} = \begin{pmatrix} A^{(1)}|_{t_2\times t_2} & A^{(1)}|_{t_2\times t_2^r} \end{pmatrix}$. Compute a Cholesky factorization $A^{(1)}|_{t_2\times t_2} = D_2^T D_2$, and let $\mathbf{R}|_{t_2\times t_r^r} = D_2^{-T} A^{(1)}|_{t_2\times t_2^r}$. Then

$$A \approx \left(\begin{array}{cc|c} D_1^T & & \\ (A^{(1)}|_{\hat{t}_1\times t_2})^T U_1^T & D_2^T & \\ \hline (A^{(1)}|_{\hat{t}_1\times t_2^r})^T U_1^T & (\mathbf{R}|_{t_2\times t_r^r})^T & I \end{array}\right) \left(\begin{array}{cc|c} D_1 & U_1 A^{(1)}|_{\hat{t}_1\times t_2} & U_1 A^{(1)}|_{\hat{t}_1\times t_2^r} \\ & D_2 & \mathbf{R}|_{t_2\times t_r^r} \\ \hline & & A^{(1)}|_{t_2^r\times t_2^r} \end{array}\right),$$

where $A^{(1)}|_{t_2^r\times t_2^r} = A|_{t_2^r\times t_2^r} - (A^{(1)}|_{\hat{t}_1\times t_2^r})^T A^{(1)}|_{\hat{t}_1\times t_2^r} - (\mathbf{R}|_{t_2\times t_r^r})^T \mathbf{R}|_{t_2\times t_r^r}$ is not explicitly formed. At this point, the block to be compressed is

$$\begin{aligned} \Theta_2 &\equiv \begin{pmatrix} (\mathbf{R}|_{t_1\times t_2})^T & \mathbf{R}|_{t_2\times t_r^r} \end{pmatrix} = \begin{pmatrix} (A^{(1)}|_{\hat{t}_1\times t_2})^T U_1^T & \mathbf{R}|_{t_2\times t_r^r} \end{pmatrix} \\ &= \begin{pmatrix} (A^{(1)}|_{\hat{t}_1\times t_2})^T & \mathbf{R}|_{t_2\times t_r^r} \end{pmatrix} \operatorname{diag}(U_1, I). \end{aligned} \tag{4.3}$$

Since $\operatorname{diag}(U_1, I)$ has orthonormal columns, it can be ignored in the compression of Θ_2. That is, we need only compute the following compression:

$$\Omega_2 \equiv \begin{pmatrix} (A^{(1)}|_{\hat{t}_1\times t_2})^T & \mathbf{R}|_{t_2\times t_r^r} \end{pmatrix} \approx U_2 \begin{pmatrix} (A^{(2)}|_{\hat{t}_1\times \hat{t}_2})^T & A^{(2)}|_{\hat{t}_2\times t_2^r} \end{pmatrix}. \tag{4.4}$$

Set $B_1 \equiv A^{(2)}|_{\hat{t}_1\times\hat{t}_2}$. Now, A becomes

$$A \approx \left(\begin{array}{cc|c} D_1^T & & \\ U_2 B_1^T U_1^T & D_2^T & \\ \hline (A^{(2)}|_{\hat{t}_1\times t_2^r})^T U_1^T & (A^{(2)}|_{\hat{t}_2\times t_2^r})^T U_2^T & I \end{array}\right) \left(\begin{array}{cc|c} D_1 & U_1 B_1 U_2^T & U_1 A^{(2)}|_{\hat{t}_1\times t_2^r} \\ & D_2 & U_2 A^{(2)}|_{\hat{t}_2\times t_2^r} \\ \hline & & A^{(2)}|_{t_2^r\times t_2^r} \end{array}\right),$$

where $A^{(2)}|_{\hat{t}_1\times t_2^r} \equiv A^{(1)}|_{\hat{t}_1\times t_2^r}$, and $A^{(2)}|_{t_2^r\times t_2^r}$ is a compact approximation to $A^{(1)}|_{t_2^r\times t_2^r}$:

$$A^{(2)}|_{t_2^r\times t_2^r} = A|_{t_2^r\times t_2^r} - \begin{pmatrix} (A^{(2)}|_{\hat{t}_1\times t_2^r})^T & (A^{(2)}|_{\hat{t}_2\times t_2^r})^T \end{pmatrix} \begin{pmatrix} A^{(1)}|_{\hat{t}_1\times t_2^r} \\ A^{(2)}|_{\hat{t}_2\times t_2^r} \end{pmatrix}.$$

Since the next node (node 3) is not a leaf, $A^{(2)}|_{t_2^r\times t_2^r}$ is not explicitly formed.

(c) *Node* 3. Node 3 has child nodes 1 and 2. We need to compress $\mathbf{R}|_{t_3\times t_3^r}$, which is available from appropriate subblocks of $\mathbf{R}|_{t_1\times t_1^r}$ and $\mathbf{R}|_{t_2\times t_2^r}$ (Figure 3.2) as

$$\Theta_3 \equiv \mathbf{R}|_{t_3\times t_3^r} = \begin{pmatrix} U_1 A^{(2)}|_{\hat{t}_1\times t_2^r} \\ U_2 A^{(2)}|_{\hat{t}_2\times t_2^r} \end{pmatrix} = \operatorname{diag}(U_1, U_2) \begin{pmatrix} A^{(2)}|_{\hat{t}_1\times t_2^r} \\ A^{(2)}|_{\hat{t}_2\times t_2^r} \end{pmatrix}.$$

Further compression is then done with previous U bases ignored by a QR factorization

$$\Omega_3 \equiv \begin{pmatrix} A^{(2)}|_{\hat{t}_1\times t_2^r} \\ A^{(2)}|_{\hat{t}_2\times t_2^r} \end{pmatrix} \approx \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} A^{(3)}|_{\hat{t}_3\times t_3^r}.$$

Let $Y_3 = A^{(3)}|_{\hat{t}_3\times t_3^r}$. Then $A^{(2)}|_{t_2^r\times t_2^r}$ is approximated by a more compact $A^{(3)}|_{t_3^r\times t_3^r} = A|_{t_2^r\times t_2^r} - Y_3^T Y_3$. (Note that $t_2^r = t_3^r$, since they are both equal to $t_6 \equiv t_4 \cup t_5$. More generally, $t_{c_2}^r = t_i^r$ for the right child c_2 of i.) Clearly, $Y_3^T Y_3$ represents the accumulated contribution from previous eliminations to the Schur complement $A^{(3)}|_{t_3^r\times t_3^r}$. Partition Y_3 as $\begin{pmatrix} Y_{3,1} & Y_{3,2} \end{pmatrix} \equiv \begin{pmatrix} A^{(3)}|_{\hat{t}_3\times t_4} & A^{(3)}|_{\hat{t}_3\times t_5} \end{pmatrix}$. Since the next node (node 4) is a leaf and $t_3 = t_4 \cup t_4^r$, we explicitly form

$$A^{(3)}|_{t_4\times t_3^r} = A|_{t_4\times t_3^r} - Y_{3,1}^T Y_3. \tag{4.5}$$

Note that after this step,

$$U_3 = \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \end{pmatrix} = \begin{pmatrix} U_1 & \\ & U_2 \end{pmatrix} \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}$$

is implicitly available and has orthonormal columns, since both factors have orthonormal columns.

(d) *Node* 4. We proceed similarly. One thing we point out is the Schur complement computation. After the elimination of node 4, the approximate Schur complement is $A|_{t_5\times t_5} - Y_{3,2}^T Y_{3,2} - (A^{(4)}|_{\hat{t}_4\times t_5})^T A^{(4)}|_{\hat{t}_4\times t_5}$. We compute a QR factorization $\begin{pmatrix} Y_{3,2} \\ A^{(4)}|_{\hat{t}_4\times t_5} \end{pmatrix} \approx \tilde{Q}_4 Y_4$, so that this Schur complement is approximated by

$$A^{(4)}|_{t_5\times t_4^r} \equiv A^{(4)}|_{t_5\times t_5} = A|_{t_5\times t_5} - Y_4^T Y_4. \tag{4.6}$$

Again, $Y_4^T Y_4$ represents the accumulated update from previous elimination steps.

(e) *Other nodes.* The factorization and compression proceed along the HSS tree. After the compression associated with node 6 is done, we have a factorization $A \approx \mathbf{R}^T\mathbf{R}$, where $\mathbf{R}$ has the form (4.1). The approximation to each Schur complement differs from the exact one by a positive semidefinite term so that $\mathbf{R}$ is guaranteed to exist and $\mathbf{R}^T\mathbf{R}$ is always positive definite.

4.3. General factorization algorithm. An important idea of the algorithm is that *the compressed forms of previously computed off-diagonal blocks of* $\mathbf{R}$ *also participate in later compression steps.* For example, in (4.3), $\mathbf{R}|_{t_1\times t_2}$ is previously compressed as $U_1 A^{(1)}|_{\hat{t}_1\times t_2}$ so that the current compression step (4.4) uses $(A^{(1)}|_{\hat{t}_1\times t_2})^T$ instead of $(\mathbf{R}|_{t_1\times t_2})^T$. To clearly keep track of previously compressed blocks and to help later steps use earlier compression information as much as possible, we introduce a definition, in a way similar to the consideration of the total cluster basis in [11].

DEFINITION 4.1. *For a node* i *of a postordered binary tree* $\mathcal{T}$*, let* $\operatorname{par}(i)$ *be its parent and* $\operatorname{sib}(i)$ *be its sibling. The* set of predecessors *of* i *is defined to be*

$$\operatorname{pred}(i) = \begin{cases} \{i\} & \text{if } i \text{ is the root of } \mathcal{T}, \\ \{i\} \cup \operatorname{pred}(\operatorname{par}(i)) & \text{otherwise.} \end{cases}$$

A visited set $\mathcal{V}_i$ *(of visited nodes whose siblings have not been visited) is defined to be*

$$\mathcal{V}_i = \{j : j \text{ is a left node and } \mathrm{sib}(j) \in \mathrm{pred}(i)\}.$$

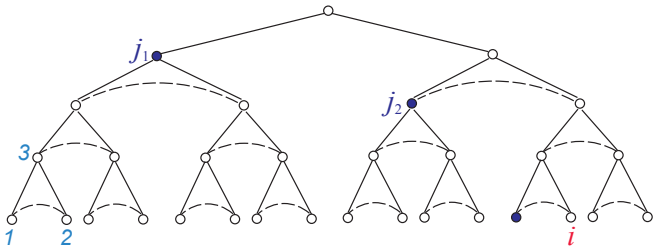


FIG. 4.1. *Visited set $\mathcal{V}_i$ (formed by the solid nodes).*

See Figure 4.1. For a node i of the HSS tree $\mathcal{T}$, the visited set $\mathcal{V}_i$ is used to form t_i^c and then the compressed form of $(\mathbf{R}|_{t_i^c \times t_i})^T$ in Θ_i in Table 4.1. That is, the compressed form of $\mathbf{R}|_{t_j \times t_i}$ for each $j \in \mathcal{V}_i$ contributes to $\mathbf{R}|_{t_i^c \times t_i}$. See, e.g., Figure 4.2.

Remark 4.2. We can conveniently obtain the elements of $\mathcal{V}_i$ with the aid of a stack. For step $j = 1, 2, \ldots$ along the factorization/tree traversal, if j is a left node, push j onto the stack. Otherwise, pop an element from the stack. Then right after step $j = i - 1$, the elements of the stack form $\mathcal{V}_i$. Thus, the actual implementation (see the link in section 5) of Algorithm 1 stores Ω_i (with orthonormal basis ignored in Θ_i) with a stack. However, for the convenience of presenting the algorithm, we assume the actual elements of $\mathcal{V}_i$ to be

$$\mathcal{V}_i = \{j_1, j_2, \ldots, j_s : j_1 < \cdots < j_s\}, \tag{4.7}$$

where s is the cardinality of $\mathcal{V}_i$ and depends on i.

4.3.1. Elimination—computation of a block row of R. First, for a leaf node i, the Schur complement $A^{(i-1)}|_{t_{i-1}^r \times t_{i-1}^r}$ is partially available after the previous $i-1$ elimination steps. That is, $A^{(i-1)}|_{t_i \times t_{i-1}^r}$, the first block row of $A^{(i-1)}|_{t_{i-1}^r \times t_{i-1}^r}$, is explicitly formed. (This is an induction process. See (4.22).) Notice $t_{i-1}^r = t_i \cup t_i^r$. Compute a Cholesky factorization, and get an HSS off-diagonal block row as

$$A^{(i-1)}|_{t_i \times t_i} = D_i^T D_i, \quad \mathbf{R}|_{t_i \times t_i^r} = D_i^{-T} A^{(i-1)}|_{t_i \times t_i^r}. \tag{4.8}$$

If i is a nonleaf node, $\mathbf{R}|_{t_i \times t_i^r}$ is available from the children of i.

4.3.2. Compression. Next, we need to form $\Theta_i \equiv \begin{pmatrix} (\mathbf{R}|_{t_i^c \times t_i})^T & \mathbf{R}|_{t_i \times t_i^r} \end{pmatrix}$ and to compress it. For node i, the HSS block column $\mathbf{R}|_{t_i^c \times t_i}$ can be (implicitly) formed by stacking all the blocks $\mathbf{R}|_{t_j \times t_i}$ for $j \in \mathcal{V}_i$. See Figure 4.2. Since all $j \in \mathcal{V}_i$ have been visited in previous compression steps, $\mathbf{R}|_{t_j \times t_j^r} \approx U_j A^{(j)}|_{\hat{t}_j \times t_j^r}$ due to induction (see (4.14) and (4.18)). Thus, each $\mathbf{R}|_{t_j \times t_i}$ is in a compressed form

$$\mathbf{R}|_{t_j \times t_i} \approx U_j A^{(j)}|_{\hat{t}_j \times t_i}, \tag{4.9}$$

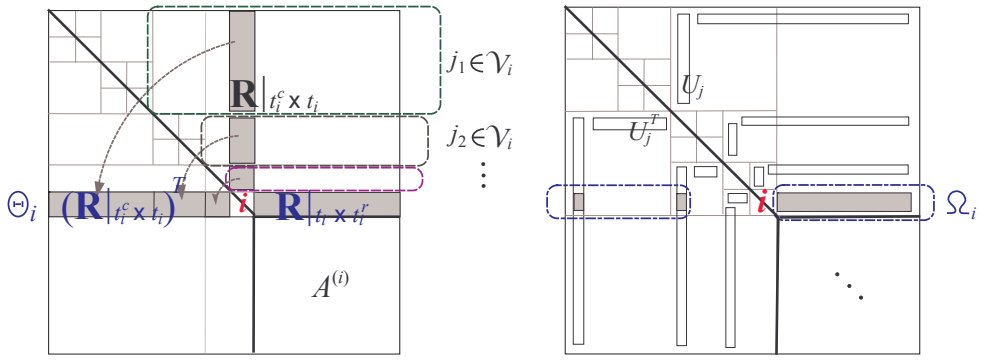


FIG. 4.2. *Forming Θ_i and Ω_i for a leaf i.*

where U_j has orthonormal columns. Thus, after step $i-1$,

$$
\begin{aligned}
(\mathbf{R}|_{t_i^c \times t_i})^T &= \begin{pmatrix} (\mathbf{R}|_{t_{j_1} \times t_i})^T & \cdots & (\mathbf{R}|_{t_{j_s} \times t_i})^T \end{pmatrix} \\
&\approx \begin{pmatrix} (U_{j_1} A^{(i-1)}|_{\hat{t}_{j_1} \times t_i})^T & \cdots & (U_{j_s} A^{(i-1)}|_{\hat{t}_{j_s} \times t_i})^T \end{pmatrix} \\
&= \begin{pmatrix} (A^{(i-1)}|_{\hat{t}_{j_1} \times t_i})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s} \times t_i})^T \end{pmatrix} \tilde{U}_i^T,
\end{aligned} \tag{4.10}
$$

where $\tilde{U}_i = \operatorname{diag}(U_{j_1}, \ldots, U_{j_s})$ has orthonormal columns. We can then write a representation for Θ_i, depending on whether or not i is a leaf.

(a) *U generators.* If i is a leaf, (4.10) yields

$$
\begin{aligned}
\Theta_i &\equiv \begin{pmatrix} (\mathbf{R}|_{t_i^c \times t_i})^T & \mathbf{R}|_{t_i \times t_i^r} \end{pmatrix} \\
&\approx \begin{pmatrix} \begin{pmatrix} (A^{(i-1)}|_{\hat{t}_{j_1} \times t_i})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s} \times t_i})^T \end{pmatrix} & D_i^{-T} A^{(i-1)}|_{t_i \times t_i^r} \end{pmatrix} \operatorname{diag}(\tilde{U}_i^T, I).
\end{aligned} \tag{4.11}
$$

Since $\tilde{U}_i$ has orthonormal columns, the compression can be effectively done on

$$
\Omega_i \equiv \begin{pmatrix} \begin{pmatrix} (A^{(i-1)}|_{\hat{t}_{j_1} \times t_i})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s} \times t_i})^T \end{pmatrix} & D_i^{-T} A^{(i-1)}|_{t_i \times t_i^r} \end{pmatrix}. \tag{4.12}
$$

See Figure 4.2. Compute an RRQR factorization

$$
\Omega_i \approx U_i \bar{A}_i \equiv U_i \begin{pmatrix} \begin{pmatrix} (A^{(i)}|_{\hat{t}_{j_1} \times \hat{t}_i})^T & \cdots & (A^{(i)}|_{\hat{t}_{j_s} \times \hat{t}_i})^T \end{pmatrix} & A^{(i)}|_{\hat{t}_i \times t_i^r} \end{pmatrix},
$$

where $\bar{A}_i$ is partitioned following the column partition in (4.12).

Notice that, at this point,

$$
(\mathbf{R}|_{t_i^c \times t_i})^T \approx U_i \begin{pmatrix} (A^{(i)}|_{\hat{t}_{j_1} \times \hat{t}_i})^T & \cdots & (A^{(i)}|_{\hat{t}_{j_s} \times \hat{t}_i})^T \end{pmatrix} \tilde{U}_i^T, \tag{4.13}
$$

$$
\mathbf{R}|_{t_i \times t_i^r} \approx U_i A^{(i)}|_{\hat{t}_i \times t_i^r}. \tag{4.14}
$$

Equation (4.14) confirms the induction as in (4.9). Equations (4.13)–(4.14) will participate in later compression. (Equations (4.13)–(4.14) also hold for a nonleaf node i, as verified below.)

(b) *R generators.* If i is a nonleaf node, let c_1 and c_2 be the two children of i with $c_1 < c_2$. Clearly, $c_2 = i-1$. To form Θ_i, we merge appropriate subblocks

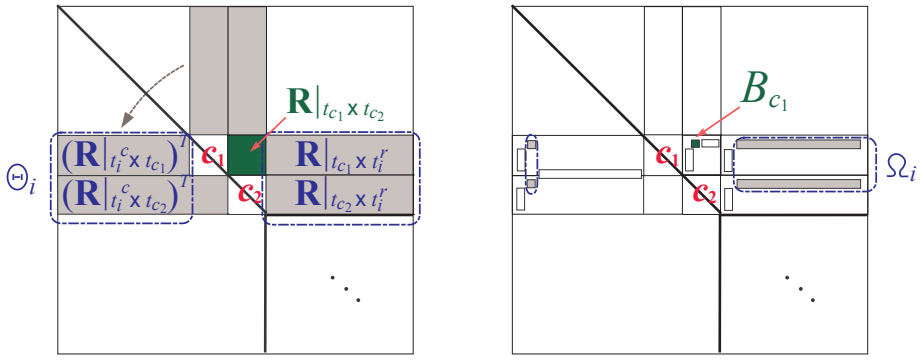


FIG. 4.3. *Forming Θ_i and Ω_i for a nonleaf node i, where c_1 and c_2 are the children of i.*

of $\Theta_{c_1} = (\ (\mathbf{R}|_{t^c_{c_1}\times t_{c_1}})^T \quad \mathbf{R}|_{t_{c_1}\times t^r_{c_1}}\ )$ and $\Theta_{c_2} = (\ (\mathbf{R}|_{t^c_{c_2}\times t_{c_2}})^T \quad \mathbf{R}|_{t_{c_2}\times t^r_{c_2}}\ )$, which, again by induction, are in compressed forms. For example, afterstep $c_2 = i-1$,

$$\mathbf{R}|_{t_{c_1}\times t^r_{c_1}} \approx U_{c_1} A^{(i-1)}|_{\hat{t}_{c_1}\times t^r_{c_1}}, \quad \mathbf{R}|_{t_{c_2}\times t^r_{c_2}} \approx U_{c_2} A^{(i-1)}|_{\hat{t}_{c_2}\times t^r_{c_2}}. \tag{4.15}$$

Including previous compression results (4.13)–(4.14) from induction (with appropriate indices such as in (4.15), and with existing U and $\tilde{U}$ bases ignored), we have

$$\Omega_i = \begin{pmatrix} ((A^{(i-1)}|_{\hat{t}_{j_1}\times\hat{t}_{c_1}})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s}\times\hat{t}_{c_1}})^T\ ) & A^{(i-1)}|_{\hat{t}_{c_1}\times t_i^r} \\ ((A^{(i-1)}|_{\hat{t}_{j_1}\times\hat{t}_{c_2}})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s}\times\hat{t}_{c_2}})^T\ ) & A^{(i-1)}|_{\hat{t}_{c_2}\times t_i^r} \end{pmatrix}, \tag{4.16}$$

where $\mathcal{V}_{c_1} = \mathcal{V}_i$, $\mathcal{V}_{c_2} = \mathcal{V}_i \cup \{c_1\}$ are used. See Figure 4.3. The compression is done with an RRQR factorization

$$\Omega_i \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \bar{A}_i \equiv \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \begin{pmatrix} ((A^{(i)}|_{\hat{t}_{j_1}\times\hat{t}_i})^T & \cdots & (A^{(i)}|_{\hat{t}_{j_s}\times\hat{t}_i})^T) & A^{(i)}|_{\hat{t}_i\times t_i^r} \end{pmatrix}, \tag{4.17}$$

where the row and column partitions of the factors follow those in (4.16). We point out that (4.15)–(4.17) indicate

$$\mathbf{R}|_{t_i\times t_i^r} \approx \operatorname{diag}(U_{c_1}, U_{c_2}) \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} A^{(i)}|_{\hat{t}_i\times t_i^r} = U_i A^{(i)}|_{\hat{t}_i\times t_i^r}, \tag{4.18}$$

where $A^{(i)}|_{\hat{t}_i\times t_i^r}$ will appear in later compression (see (4.9) for the induction). Similarly, we can verify that $(\mathbf{R}|_{t_i^c\times t_i})^T$ also has a form as in (4.13).

(c) *B generators.* In addition, notice that in the compression step for node c_1, the block $\mathbf{R}|_{t_{c_1}\times t_{c_2}} \approx U_{c_1} A^{(c_1)}|_{\hat{t}_{c_1}\times t_{c_2}}$. In the compression step for node c_2, this block is further compressed (with U_{c_1} basis ignored) as $(A^{(c_1)}|_{\hat{t}_{c_1}\times t_{c_2}})^T \approx U_{c_2}(A^{(c_2)}|_{\hat{t}_{c_1}\times\hat{t}_{c_2}})^T$. (This is part of the compression of Ω_{c_2}.) Thus,

$$\mathbf{R}|_{t_{c_1}\times t_{c_2}} \approx U_{c_1} B_{c_1} U_{c_2}^T \text{ with } B_{c_1} = A^{(c_2)}|_{\hat{t}_{c_1}\times\hat{t}_{c_2}}, \tag{4.19}$$

where B_{c_1} is the B generator associated with node c_1 in the HSS form (Figure 4.3). See (4.4) for an example.

4.3.3. Schur complement computation. According to the postordering of the HSS tree, if i is a left node, $i+1$ must be a leaf. Then an elimination step (4.8) is needed for $i+1$. Thus, we *partially* form the new Schur complement $A^{(i)}|_{t_i^r \times t_i^r}$. That is, we compute $A^{(i)}|_{t_{i+1} \times t_i^r}$, the block row of the Schur complement corresponding to node $i+1$. Clearly,

$$A^{(i)}|_{t_{i+1} \times t_i^r} = A|_{t_{i+1} \times t_i^r} - (\mathbf{R}|_{t_{i+1}^c \times t_{i+1}})^T \mathbf{R}|_{t_{i+1}^c \times t_i^r}. \tag{4.20}$$

Such a direct computation is generally costly. Thus, we maintain a compressed form

$$\mathbf{R}|_{t_{i+1}^c \times t_i^r} \approx Q_i Y_i \tag{4.21}$$

for each left node i, where Q_i has orthonormal columns and is not stored, and Y_i is either set to be $Y_i = A^{(i)}|_{\hat{t}_i \times t_i^r}$ when $i \in \mathrm{pred}(1)$ or is obtained approximately with the compression (4.23) below. In this way, (4.20) is replaced by a low-rank update

$$A^{(i)}|_{t_{i+1} \times t_i^r} \approx A|_{t_{i+1} \times t_i^r} - Y_{i,1}^T Y_i, \tag{4.22}$$

where $Y_i = \begin{pmatrix} Y_{i,1} & Y_{i,2} \end{pmatrix}$ so that $Q_i Y_{i,1} \approx \mathbf{R}|_{t_{i+1}^c \times t_{i+1}}$. See (4.2), (4.5), and (4.6) for examples.

Here, Y_i is quickly computed along the traversal of the tree. This is an induction process. When $i = 1$, let $Q_i \equiv U_1$, $Y_1 \equiv A^{(1)}|_{\hat{t}_1 \times t_1^r}$ (see the first step in section 4.2). For a general left node i, assume $j+1$ is the largest leaf before $i+1$ (thus, j is the left node right before $j+1$), and $\mathbf{R}|_{t_{j+1}^c \times t_j^r} = Q_j Y_j$. Then $t_i^r = t_{j+1}^r$ and

$$\mathbf{R}|_{t_{i+1}^c \times t_i^r} = \begin{pmatrix} \mathbf{R}|_{t_{j+1}^c \times t_{j+1}^r} \\ \mathbf{R}|_{t_{j+1} \times t_{j+1}^r} \end{pmatrix} = \begin{pmatrix} Q_j Y_{j,2} \\ U_{j+1} A^{(j+1)}|_{\hat{t}_{j+1} \times t_{j+1}^r} \end{pmatrix} = \begin{pmatrix} Q_j & \\ & U_{j+1} \end{pmatrix} \begin{pmatrix} Y_{j,2} \\ A^{(j+1)}|_{\hat{t}_{j+1} \times t_{j+1}^r} \end{pmatrix}.$$

Since Q_j and U_{j+1} have orthonormal columns, we need only compress the following matrix with a QR factorization:

$$\begin{pmatrix} Y_{j,2} \\ A^{(j+1)}|_{\hat{t}_{j+1} \times t_{j+1}^r} \end{pmatrix} \approx \tilde{Q}_i Y_i, \tag{4.23}$$

so that (4.21) holds with $Q_i = \mathrm{diag}(Q_j, U_{j+1}) \tilde{Q}_i$. The induction then continues. Then the Schur complement is partially formed with the low-rank update (4.22). $Y_{i,1}^T Y_i$ represents the accumulated update from the previous elimination steps to the portion $A^{(i)}|_{t_{i+1} \times t_i^r}$ of the current Schur complement. Y_j is not needed after Y_i is computed. In addition, it is easy to verify that if $i \in \mathrm{pred}(1)$, we can simply set $Y_i = A^{(i)}|_{\hat{t}_i \times t_i^r}$ without using (4.23).

Also, with the compression (4.21), the entire Schur complement $A^{(i)}|_{t_i^r \times t_i^r} = A|_{t_i^r \times t_i^r} - (\mathbf{R}|_{t_{i+1}^c \times t_i^r})^T \mathbf{R}|_{t_{i+1}^c \times t_i^r}$ is approximated by $A|_{t_i^r \times t_i^r} - Y_i^T Y_i$. This means a positive semidefinite term is implicitly added to the Schur complement, just like the Schur compensation in section 2. The submatrix $A|_{t_i^r \times t_i^r}$ is never touched before step $i+1$. Thus, this factorization can be viewed as a structured left-looking method [3].

4.4. Algorithm and complexity.

4.4.1. Algorithm. We summarize the procedure in Algorithm 1. Two points are essential for the efficiency of the algorithm. First, in both the off-diagonal compression and the Schur complement computations, compressed forms are used as much

as possible, with appropriate orthonormal basis matrices ignored. Next, A is accessed locally or block rowwise, and Schur complements are formed partially with the accumulated update from previous elimination steps in a compressed form.

The algorithm returns an approximate Cholesky factor $\mathbf{R}$ in HSS form. When $\mathbf{R}$ is used as a preconditioner, upper and lower triangular HSS systems are solved. The forward (backward) substitution can be done by the forward (backward) traversal of the HSS tree. These solutions cost $O(rn)$ flops when A has HSS rank r. The details are similar to the solvers in [43].

ALGORITHM 1. (HSS Cholesky factorization)

`procedure` hss_chol
`for` $i = 1, 2, \ldots$ `do`

1. `if` i is a leaf `then`
   (a) Compute the Cholesky factorization $A^{(i-1)}|_{t_i \times t_i} = D_i^T D_i$
   (b) Use $\mathcal{V}_i$ in Remark 4.2 and form
   $$\Omega_i = \begin{pmatrix} ((A^{(i-1)}|_{\hat{t}_{j_1} \times t_i})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s} \times t_i})^T \end{pmatrix} \quad D_i^{-T} A^{(i-1)}|_{t_i \times t_i^r} \Big)$$
   (c) QR factorize $\Omega_i \approx U_i \big(((A^{(i)}|_{\hat{t}_{j_1} \times \hat{t}_i})^T \quad \cdots \quad (A^{(i)}|_{\hat{t}_{j_s} \times \hat{t}_i})^T \big) \quad A^{(i)}|_{\hat{t}_i \times t_i^r} \big)$
2. `else`
   `if` $i \neq$ root node `then`
   (a) Let c_1 and c_2 be the children of i. Use $\mathcal{V}_i$ in Remark 4.2 and form
   $$\Omega_i = \begin{pmatrix} ((A^{(i-1)}|_{\hat{t}_{j_1} \times \hat{t}_{c_1}})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s} \times \hat{t}_{c_1}})^T & A^{(i-1)}|_{\hat{t}_{c_1} \times t_i^r} \\ ((A^{(i-1)}|_{\hat{t}_{j_1} \times \hat{t}_{c_2}})^T & \cdots & (A^{(i-1)}|_{\hat{t}_{j_s} \times \hat{t}_{c_2}})^T & A^{(i-1)}|_{\hat{t}_{c_2} \times t_i^r} \end{pmatrix}$$
   (b) QR factorize $\Omega_i \approx \binom{R_{c_1}}{R_{c_2}} \big(((A^{(i)}|_{\hat{t}_{j_1} \times \hat{t}_i})^T \quad \cdots \quad (A^{(i)}|_{\hat{t}_{j_s} \times \hat{t}_i})^T \big) \quad A^{(i)}|_{\hat{t}_i \times t_i^r} \big)$
   `end if`
   $B_{c_1} = A^{(c_2)}|_{\hat{t}_{c_1} \times \hat{t}_{c_2}}$
   `end if`
3. `if` $i + 1$ is a leaf `then`
   (a) `if` i is in the path from node 1 to the root `then`
   Set $Y_i = A^{(i)}|_{\hat{t}_i \times t_i^r}$
   `else`
   Compute Y_i as in (4.23)
   `end if`
   (b) Form a part the Schur complement $A^{(i)}|_{t_{i+1} \times t_i^r}$ as in (4.22)
   `end if`

`end for`
`end procedure`

4.4.2. Complexity. The complexity of the HSS factorization is $O(rn^2)$ flops if the HSS rank of A is r. For simplicity, we assume the HSS tree $\mathcal{T}$ is a perfect binary tree with $2k - 1$ nodes, and any bottom level HSS block row dimension is a constant $m = O(r)$. Since there are k leaves, $km = n$.

Clearly, the operations (compression, possible diagonal factorization, and possible Schur complement calculation) associated with each node cost $O(r^2 n)$ flops. Thus, similarly to [11], the $O(rn^2)$ complexity can be basically understood following the idea that the HSS tree has $O(k) = O(\frac{n}{r})$ nodes, each corresponding to $O(r^2 n)$ flops. This holds even if, for each leaf i, we directly compress the *dense* block $\Theta_i \equiv \begin{pmatrix} (\mathbf{R}|_{t_i^c \times t_i})^T & \mathbf{R}|_{t_i \times t_i^r} \end{pmatrix}$ so as to make the procedure simpler. (We can similarly simplify the compression for nonleaf nodes.) However, in our algorithm, $(\mathbf{R}|_{t_i^c \times t_i})^T$

appears as a compressed form due to early compression steps. With orthonormal bases ignored, only Ω_i needs to be compressed. This enables further savings. See the beginning of section 4.3 and Figures 4.2–4.3. To illustrate the savings, we look at a block 4×4 example as in section 4.2. Assume all HSS blocks have ranks $r = \frac{m}{4}$. Then the compression costs about $43m^3$ flops in total if dense Θ_i is used for all leaves i, while about $27m^3$ if Ω_i is used as in our algorithm. Interested readers can derive how much is saved by using Ω_i instead of dense Θ_i.

Similarly, the storage for the U, R, and B generators associated with each node is $O(r^2)$, so that the total storage for all $O(k) = O(\frac{n}{r})$ nodes is $O(rn)$. The original matrix A is used to store the intermediate matrices.

4.5. Connection to some existing methods and application to sparse problems. The HSS factorization algorithm has some attractive features similar to the hierarchical compression algorithm with $\mathcal{H}^2$-matrices in [12]. For example, the U bases are in nested forms with orthonormal columns, and the matrix A does not have to be kept in storage entirely since it is only accessed block rowwise. The compression here also has a hierarchical nature (bottom-up traversal of the HSS tree), but is designed in a way so that A is only locally accessed. The hierarchical compression scheme in this work is also useful in developing more HSS algorithms.

The algorithm focuses on *dense* SPD matrices instead of sparse ones, and costs $O(n^2)$ for matrices with the low-rank property. This can also be understood by noticing that $O(n^2)$ entries have to be accessed. However, it is possible to reduce the complexity if additional information about A is known, say, when A is from integral equations or Toeplitz problems so that we do not need to access the matrix entries in the fashion of a standard factorization.

In addition, it is possible to improve the constant of the flop count by using the recursive techniques in [10, 12]. We may also design a factorization algorithm with top-down traversal of the HSS tree. For example, for A in (2.1), the top level off-diagonal block A_{12} is compressed, and then A_{11} and A_{22} are recursively processed. The total factorization cost can be reduced from $O(n^2)$ to $O(n \log n)$ if the compression of any order-m off-diagonal block only costs $O(m)$. This is possible when additional information on the matrix is available. However, it is not clear if these recursive procedures can guarantee the existence of the positive definite approximation $\mathbf{R}^T\mathbf{R}$.

In terms of large sparse problems, an important application of our HSS factorization algorithm is to factorize *dense intermediate matrices in a sparse factorization framework*. During the direct factorization of some sparse discretized PDEs, the dense Schur complements or fill-in have the low-rank property [4, 5, 16, 30, 39, 47, etc.]. By taking advantage of this, some $\mathcal{H}$-LU factorization algorithms have been developed, with complexity $O(N \log N)$ [16] or $O(N \log^2 N)$ [15, 30], where N is the order of the sparse matrix. The factors given by these algorithms are $\mathcal{H}$-matrices with zero blocks and Rk-blocks [33]. We can use the HSS algorithm in this paper to handle 2D sparse discretized matrices from a different point of view and with robustness enhancement. That is, the dense HSS factorization is used as an internal *kernel routine* in some sparse matrix factorizations. In this way, we can fully take advantage of sparse matrix techniques such as the multifrontal method [24] and nested dissection [27]. The resulting robust structured sparse factorization method has complexity $O(N \log N)$. (Such structured sparse factorization is generally restricted to 2D problems, since HSS representations are generally only useful for 1D problems due to the special HSS block structure.) The details are expected to appear in future work. Since the focus of this paper is dense matrices, the reader is referred to [47] for an example of using dense

HSS algorithms in a multifrontal framework.

5. Numerical experiments. We use numerical examples to demonstrate the effectiveness of preconditioning with our method. Algorithm 1 is implemented in both MATLAB and FORTRAN 90, and the codes are available from

http://www.math.purdue.edu/∼xiaj/work/hsschol.

We test the HSS preconditioner on dense fill-in arising from the factorizations of some sparse discretized PDE problems. Note that each matrix A we test is a dense intermediate matrix instead of the entire sparse discretized matrix $\mathcal{A}$.

Example 1. Consider the following problem defined on the unit square:

$$a(x,y)\frac{\partial^2 u}{\partial x^2} + 2b(x,y)\frac{\partial^2 u}{\partial x \partial y} + c(x,y)\frac{\partial^2 u}{\partial y^2} = f(x,y)$$

with $\begin{pmatrix} a(x,y) & b(x,y) \\ b(x,y) & c(x,y) \end{pmatrix} = \alpha I + dd^T$, where $\alpha > 0$ and d is a unit vector. We assume a mixture of Dirichlet and Neumann boundary conditions.

The problem is discretized on an $n \times n$ regular mesh with nested dissection ordering of the mesh points. We look at a matrix A which is the last Schur complement corresponding to the top level separator of nested dissection during the factorization of the stiffness matrix $\mathcal{A}$. The matrix A is dense and SPD and has small off-diagonal numerical ranks. It is preconditioned with our HSS algorithm. In all the compression we specify the maximum numerical rank r (see (2.4)) and denote the preconditioned matrix by $\tilde{A}_r$. If $r = 0$, then we have the regular block diagonal preconditioning. Table 5.1 shows that, for r as small as 3, the matrices $\tilde{A}_r$ after the HSS preconditioning are very well conditioned. The cost of applying the HSS preconditioner to a vector is a small multiple of n, and so is the storage of the preconditioner.

TABLE 5.1

Condition numbers before and after preconditioning dense Schur complements in Example 1 *with* $d = (\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2})^T$, *where* $n = 200$ *and the bottom level block row dimensions* $m \approx 5$.

α	1	10^{-2}	10^{-4}	10^{-6}	10^{-8}
$\kappa(A)$	4.3×10^2	5.7×10^3	1.9×10^5	4.6×10^5	4.7×10^5
$\kappa(\tilde{A}_0)$	1.2×10^2	1.1×10^3	3.4×10^3	8.3×10^4	8.5×10^4
$\kappa(\tilde{A}_2)$	12.0	6.9	6.1×10^2	1.9×10^2	2.0×10^2
$\kappa(\tilde{A}_3)$	2.7	2.0	6.7	19.7	20.2
$\kappa(\tilde{A}_4)$	1.6	1.0	2.0	3.4	3.4
$\kappa(\tilde{A}_5)$	1.1	1.0	1.1	1.2	1.2

Example 2. Next, consider a linear elasticity equation

$$-(\mu\Delta\mathbf{u} + (\lambda+\mu)\nabla\nabla\cdot\mathbf{u}) = \mathbf{f} \text{ in } \Omega = (0,1)\times(0,1),$$
$$\mathbf{u} = 0 \text{ on } \partial\Omega,$$

where $\mathbf{u}$ is the displacement vector field, and λ and μ are the Lamé constants.

This PDE is very ill conditioned when λ/μ is large. The limit is known as the incompressible limit, which is a very important situation in practical problems, and for example, is associated with the mechanical behavior of elastomeric materials and

plastic flow in metals. Without an effective preconditioner, iterative methods including multigrid diverge or converge very slowly. Again, we use nested dissection on a regular mesh and consider the last Schur complement A corresponding to the top level separator in nested dissection. The preconditioning results are shown in Table 5.2. With a small $r = 8$, we get favorable conditioning which is much more satisfactory than block diagonal preconditioning. Furthermore, the preconditioning results are relatively insensitive to λ/μ. Or the preconditioner is very suitable for problems where λ/μ is near the incompressible limit.

TABLE 5.2
Preconditioning dense Schur complements in Example 2, where $n = 802$ and the bottom level block row dimensions $m \approx 8$.

λ/μ	1	10^3	10^6	10^9	10^{12}
$\kappa(A)$	6.9×10^2	8.2×10^4	8.1×10^7	8.1×10^{10}	7.3×10^{13}
$\kappa(\tilde{A}_0)$	1.9×10^2	4.4×10^2	1.1×10^5	1.1×10^8	9.0×10^{10}
$\kappa(\tilde{A}_8)$	2.4	15.7	16.2	16.2	14.2

Example 3. We also test the preconditioner on the following time harmonic Maxwell equation similar to a problem in [22]:

$$\nabla \times \nabla \times \mathbf{E} - k^2 \mathbf{E} = \mathbf{J} \text{ in } \Omega,$$
$$\mathbf{E} \times \mathbf{n} = 0 \text{ on } \partial\Omega,$$

where $k \in \mathbb{R}$ ($k \neq 0$), $\mathbf{J} = (1, 1, 1)$, $\Omega = \Omega_0 \setminus \Sigma$, $\Omega_0 = (-1, 1)^3$, $\Sigma = \{y = 0, (x, z) \in (-\frac{1}{2}, \frac{1}{2})^2\}$, and $\mathbf{n}$ is the unit outer normal of $\partial\Omega$.

The problem is discretized on a tetrahedral mesh. The discretized finite element matrix $\mathcal{A}$ is not SPD, so we use $\mathcal{A}^H\mathcal{A}$. Again, we consider a Schur complement A corresponding to the top level separator in nested dissection. Here, k is related to the wave number and is set to be 16. The dense matrix A has order $n = 3947$ and condition number 4.8×10^9. In our HSS preconditioner, we use bottom level HSS block row sizes $m \approx 123$. Numerical results indicate that the maximum off-diagonal numerical rank r is not very small when a small tolerance is used. Thus, instead, we manually specify small r. Figure 5.1 shows the convergence of the conjugate gradient method (CG) and preconditioned CG (PCG) with different r in building the HSS preconditioners. The preconditioners still significantly improve the convergence. The storage of each structured preconditioner is only a small portion of the storage of A. For example, the preconditioners need only 6.9% and 12.3% of the storage of A for $r = \frac{1}{2}m$ and $\frac{3}{4}m$, respectively, for this particular matrix. The cost of generating the preconditioner is insignificant as compared with the solution cost, and so is the cost of preconditioning as compared with other costs in each PCG step.

Similar convergence behaviors are observed for different k, although a theoretical justification is not yet available. This indicates, based on our new preconditioner, that there is a good potential to develop an efficient and robust preconditioner for PDEs with large wave numbers.

Remark 5.1. These examples are used to illustrate the potential of the HSS algorithm in effectively preconditioning dense intermediate matrices during the direct factorization of discretized PDEs. When the multifrontal method [24] is used together with nested dissection, these dense matrices are frontal matrices associated with the separators or interfaces. (Each Schur complement A chosen above is also the top level

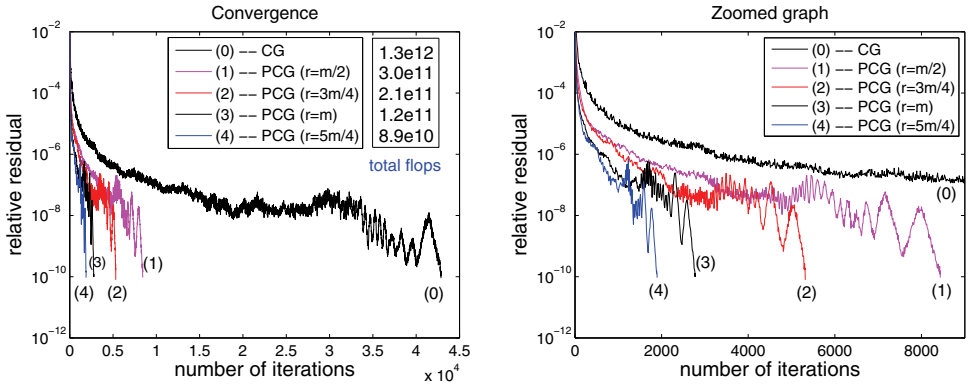


FIG. 5.1. *Convergence of PCG for dense Schur complements arising in the factorization of* $\mathcal{A}^H\mathcal{A}$ *in Example* 3*, where different numerical ranks* r *are used to build the HSS preconditioners. Total solution costs are shown. Part of the left graph is zoomed in and shown in the right graph.*

frontal matrix in the multifrontal factorization.) Thus, for sparse 2D discretized problems, HSS matrices are used to precondition dense matrices roughly corresponding to 1D separators. See also section 4.5. The off-diagonal numerical ranks of these dense matrices may not be small enough, but we may still employ large compression rates by manually setting r to be small. A comprehensive analysis is expected to be done in future work.

Remark 5.2. In the examples, we have not considered issues such as advanced discretization or smoothing techniques. These are expected to be included in our future work when we apply the HSS algorithm to sparse problems as described in section 4.5. In such a situation, we will be able to conduct an effective comparison of the structured sparse factorization and other sparse methods such as multigrid.

6. Conclusions. This paper presents a structured approximate factorization method with up to a user-specified accuracy for dense SPD matrices. After the factorization, an approximate factor in compact HSS form is obtained. Positive semidefinite terms are automatically added to Schur complements in the factorization so that the matrix is approximated by a factorized form which is guaranteed to be positive definite. This leads to enhanced robustness. The factorization can be used as a robust and effective preconditioner. Satisfactory preconditioning results can be achieved even if the low-rank structure is not highly significant.

Acknowledgments. The authors are very grateful to the editor and the anonymous referees for their valuable comments. The authors would also like to thank Xiaoye S. Li for some useful discussions and Zhiqiang Cai, Jie Shen, Panayot Vassilevski, and Linbo Zhang for providing related test problems.

REFERENCES

[1] M. A. AJIZ AND A. JENNINGS, *A robust incomplete Choleski-conjugate gradient algorithm*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 949–966.

[2] O. AXELSSON AND L. KOLOTILINA, *Diagonally compensated reduction and related preconditioning methods*, Numer. Linear Algebra Appl., 1 (1994), pp. 155–177.

[3] Z. BAI, J. DEMMEL, J. DONGARRA, A. RUHE, AND H. VAN DER VORST, EDS., *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.

[4] M. BEBENDORF, *Efficient inversion of Galerkin matrices of general second-order elliptic differential operators with nonsmooth coefficients*, Math. Comp., 74 (2005), pp. 1179–1199.

[5] M. Bebendorf and W. Hackbusch, *Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients*, Numer. Math., 95 (2003), pp. 1–28.
[6] M. Bebendorf and W. Hackbusch, *Stabilized rounded addition of hierarchical matrices*, Numer. Linear Algebra Appl., 14 (2007), pp. 407–423.
[7] M. Benzi, J. K. Cullum, and M. Tuma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., 22 (2000), pp. 1318–1332.
[8] M. Benzi and M. Tuma, *A robust incomplete factorization preconditioner for positive definite matrices*, Numer. Linear Algebra Appl., 10 (2003), pp. 385–400.
[9] D. A. Bini, L. Gemignani, and V. Y. Pan, *Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equations*, Numer. Math., 100 (2005), pp. 373–408.
[10] S. Börm, *$\mathcal{H}^2$-matrix arithmetics in linear complexity*, Computing, 77 (2006), pp. 1–28.
[11] S. Börm, *Data-sparse approximation of non-local operators by $\mathcal{H}^2$-matrices*, Linear Algebra Appl., 422 (2007), pp. 380–403.
[12] S. Börm, *Construction of data-sparse $\mathcal{H}^2$-matrices by hierarchical compression*, SIAM J. Sci. Comput., 31 (2009), pp. 1820–1839.
[13] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem, 27 (2003), pp. 405–422.
[14] S. Börm and W. Hackbusch, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.
[15] S. Le Borne and L. Grasedyck, *$\mathcal{H}$-matrix preconditioners in convection-dominated problems*, SIAM. J. Matrix Anal. Appl., 27 (2006), pp. 1172–1183.
[16] S. Le Borne, L. Grasedyck, and R. Kriemann, *Domain-decomposition based $\mathcal{H}$-LU preconditioners*, in Domain Decomposition Methods in Science and Engineering XVI, O. B. Widlund and D. E. Keyes, eds., Lect. Notes Comput. Sci. Eng. 55, Springer, Berlin, 2006, pp. 667–674.
[17] P. Businger and G. H. Golub, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.
[18] T. F Chan, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.
[19] S. Chandrasekaran and I. C. F. Ipsen, *On rank-revealing factorisations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 592–622.
[20] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.
[21] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.
[22] Z. Chen, L. Wang, and W. Zheng, *An adaptive multilevel method for time-harmonic Maxwell equations with singularities*, SIAM J. Sci. Comput., 29 (2007), pp. 118–138.
[23] J. Demmel, *The condition number of equivalence transformations that block diagonalize matrix pencils*, SIAM J. Numer. Anal., 20 (1983), pp. 599–610.
[24] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.
[25] Y. Eidelman, I. Gohberg, and V. Olshevsky, *The QR iteration method for Hermitian quasiseparable matrices of an arbitrary order*, Linear Algebra Appl., 404 (2005), pp. 305–324.
[26] L. Gemignani, M. Van Barel, and S. Delvaux, *Fast QR factorization of Cauchy-like matrices*, Linear Algebra Appl., 428 (2008), pp. 697–711.
[27] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
[28] G. H. Golub and C. V. Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[29] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *A theory of pseudoskeleton approximations*, Linear Algebra Appl., 261 (1997), pp. 1–21.
[30] L. Grasedyck, R. Kriemann, and S. Le Borne, *Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems*, Comput. Vis. Sci., 11 (2008), pp. 273–291.
[31] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong-rank revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.
[32] M. Gu, X. S. Li, and P. S. Vesselevski, *Direction-preserving and Schur-monotonic semiseparable approximations of symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl, 31 (2010), pp. 2650–2664.
[33] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part* I*: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[34] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part* II*: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[35] W. Hackbusch, B. Khoromskij, and R. Kriemann, *Hierarchical matrices based on a weak admissibility criterion*, Computing, 73 (2004), pp. 207–243.

[36] W. Hackbusch, B. Khoromskij, and S. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, H. Bungartz, R. H. W. Hoppe, C. Zenger, eds., Springer, Berlin, 2000, pp. 9–29.

[37] I. E. Kaporin, *High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$-decomposition*, Numer. Linear Algebra Appl., 5 (1998), pp. 483–509.

[38] T.-L. Lee, T.-Y. Li, and Z. Zeng, *A rank-revealing method with updating, downdating, and applications. Part* II, SIAM. J. Matrix Anal. Appl., 31 (2009), pp. 503–525.

[39] P.-G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 38 (2009), pp. 316–330.

[40] P. G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.

[41] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473–497.

[42] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[43] Z. Sheng, P. Dewilde, and S. Chandrasekaran, *Algorithms to solve hierarchically semi-separable systems*, in System Theory, the Schur Algorithm and Multidimensional Analysis, Oper. Theory Adv. Appl. 176, Birkhäuser, Basel, 2007, pp. 255–294.

[44] E. E. Tyrtyshnikov, *Incomplete cross approximation in the mosaic-skeleton method*, Computing, 64 (2000), pp. 367–380.

[45] M. Van Barel, E. Van Camp, and N. Mastronardi, *Orthogonal similarity transformation into block-semiseparable matrices of semiseparability rank k*, Numer. Linear Algebra Appl., 12 (2005), pp. 981–1000.

[46] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, *A bibliography on semiseparable matrices*, Calcolo, 42 (2005), pp. 249–270.

[47] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.

[48] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., to appear; available online at http://doi.wiley.com/10.1002/nla.691.