RANDOMIZED SPARSE DIRECT SOLVERS

JIANLIN XIA*

Abstract. We propose randomized direct solvers for large sparse linear systems, which integrate randomization into rank structured multifrontal methods. The use of randomization highly simplifies various essential steps in structured solutions, where fast operations on skinny matrix-vector products replace traditional complex ones on dense or structured matrices. The new methods thus significantly enhance the flexibility and efficiency of using structured methods in sparse solutions. We also consider a variety of techniques, such as some graph methods, the inclusion of additional structures, the concept of reduced matrices, information reuse, and adaptive schemes. The methods are applicable to various sparse matrices with certain rank structures. Particularly, for discretized matrices whose factorizations yield dense fill-in with some off-diagonal rank patterns, the factorizations cost about $O(n)$ flops in 2D, and about $O(n)$ to $O(n^{4/3})$ flops in 3D. The solution costs and memory sizes are nearly $O(n)$ in both 2D and 3D. These counts are obtained based on two optimization strategies and a sparse rank relaxation idea. The methods are especially useful for approximate solutions and preconditioning. Numerical tests on both discretized PDEs and more general problems are used to demonstrate the efficiency and accuracy. The ideas here also have the potential to be generalized to matrix-free sparse direct solvers based on matrix-vector multiplications in future developments.

Key words. randomized sparse solver, structured multifrontal method, skinny extend-add operation, HSS matrix, adaptive scheme, rank relaxation

AMS subject classifications. 15A23, 65F05, 65F30, 65F50

1. Introduction.

Large sparse linear systems arise frequently in numerical solutions of mathematical and engineering problems. It is often critical to quickly solve these systems. Classical direct solvers have various advantages, such as the robustness and the suitability for multiple right-hand sides. However, the issue of fill-in or loss of sparsity often makes direct factorizations unaffordable. On the other hand, iterative solvers have cheap memory requirements. But they generally need effective preconditioners to converge quickly, and can be inefficient for multiple right-hand sides.

In recent years, a lot of efforts have been made in the development of fast approximate direct solvers, especially those using the idea of the fast multipole method (FMM) [18] or based on rank structures. Such rank structures mean that certain intermediate dense matrices have low-rank (or low-numerical-rank) off-diagonal blocks. Some useful structured or data-sparse matrix representations are developed to characterize low-rank structures, such as hierarchical ($\mathcal{H}$-, $\mathcal{H}^2$-) [3, 4, 20], quasiseparable [2, 13], and semiseparable matrices [7, 34]. These representations are used to develop fast structured direct solvers for sparse linear systems, such as $\mathcal{H}$-LU or inversion methods [1, 16, 17, 27] and structured multifrontal methods [40]. Such solvers compress dense intermediate fill-in, which can often help significantly improve the factorization efficiency. For example, for discretized elliptic equations in 2D, these methods can achieve nearly linear complexity, while classical exact factorizations need at least $O\left(n^{1.5}\right)$ flops [22], where n is the matrix size.

Here, we concentrate on structured multifrontal methods, which are first proposed in [40] and then generalized or simplified in [32, 35, 37, 39]. These methods use a factorization framework called the multifrontal method [12, 25], which is one of the most important direct solvers and has good scalability. The method conducts the

*Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA (xiaj@math.purdue.edu). The research of Jianlin Xia was supported in part by NSF grants DMS-1115572 and CHE-0957024.

factorization in terms of a series of local factorizations of smaller dense matrices (called *frontal matrices*), and these local factorizations are organized following a tree called elimination or *assembly tree* [12, 25]. The elimination performs a partial factorization of a frontal matrix and yields a Schur complement called an *update matrix.* Following the assembly tree, update matrices associated with sibling nodes are assembled to form the parent frontal matrix. This operation is called an *extend-add* operation [12, 25]. In [40], the frontal matrices are further approximated by low-rank structures called hierarchically semiseparable (HSS) forms [5, 6, 26, 41], and the computation of the update matrices and the extend-add operation are also performed in HSS forms.

The structured multifrontal method in [40] mainly works for certain 2D mesh structures, and even so, the HSS version extend-add operation is still very complicated. In fact, it is not clear how to design a general HSS extend-add operation or how efficient it can be. More general meshes are considered in [32, 37, 39]. But the method in [32] still requires the 2D mesh to roughly follow the pattern of a regular mesh, since it uses horizontal and vertical lines to divide the mesh into a uniform partition. The methods in [37, 39] use simplifications by keeping the update matrices dense, so as to avoid complex extend-add operations. This then requires to store potentially large dense matrices, and the overall complexity and memory are higher than those in [40].

1.1. Main results. In this work, we seek to avoid all such difficulties with randomization. 2D, 3D, and more general problems are considered. In recent researches, the idea of randomization has been used to assist efficient low-rank computations [23, 24, 28, 29, 30]. See [21] for a comprehensive review. In particular, if a large matrix Φ is rank deficient, its compression can be conducted in terms of certain matrix-vector products in randomized sampling, instead of handling the original Φ as in classical rank-revealing factorizations. Based on this, fast HSS construction algorithms are proposed in [24, 28], which apply randomize sampling hierarchically to the low-rank off-diagonal blocks of a matrix F. In particular, the method in [28] only needs $O(1)$ matrix-vector multiplications, provided that selected entries of F are available. This scheme is then further improved in [42] with additional structures.

We bring such randomization techniques into sparse solutions by using matrix-vector products as the information passed along the assembly tree instead of dense or HSS update matrices. That is, at each step $\mathbf{i}$ of the multifrontal factorization, certain matrix-vector products $Y_{\mathbf{i}} = \mathcal{F}_{\mathbf{i}} X_{\mathbf{i}}$ are used to compute an HSS approximation to a frontal matrix $\mathcal{F}_{\mathbf{i}}$, where $X_{\mathbf{i}}$ is a tall and skinny random matrix. Then instead of passing the Schur complement $\mathcal{U}_{\mathbf{i}}$ to the parent step $\mathbf{p}$, we pass its product with a submatrix of $X_{\mathbf{i}}$: $Z_{\mathbf{i}} = \mathcal{U}_{\mathbf{i}} \tilde{X}_{\mathbf{i}}$. The product $Z_{\mathbf{i}}$ is a skinny matrix and can be easily used to assemble the parent step $Y_{\mathbf{p}}$ in a highly simplified extend-add operation, where we then repeat these steps.

The overall method includes these main steps, where the last step is used to compute certain matrix entries required by the methods in [28, 42] for HSS constructions:

- Fast randomized HSS construction.
- Fast partial ULV-type HSS factorization [6] and Schur complement computation.
- Fast multiplication of HSS Schur complements with random vectors.
- Simple skinny extend-add operation for the rows of the matrix-vector products, instead of both the rows and the columns of dense or HSS update matrices.
- Fast formation of selected entries of HSS frontal matrices.

The following useful strategies are developed or used to achieve high efficiency and flexibility:

- The inclusion and preservation of additional structures in the randomized HSS construction and factorization, and then in the structured multifrontal method. Some existing HSS algorithms are modified to improve efficiency.
- The idea of reduced matrices in the Schur complement computation after a ULV factorization. This idea replaces the inversion or solution of an HSS matrix by the corresponding operation on a small reduced matrix [39].
- Flexible choices of methods for computing HSS matrix-vector products.
- Three levels of adaptivity in the factorization.
- Strategies for reusing information as much as possible when forming the entries of an HSS matrix.
- Various graph methods that facilitate the computations.

Analysis is provided for the different stages. In particular, the complexity is studied in terms of an idea of rank relaxation in [38, 39]. That is, we do not require the off-diagonal ranks of the frontal matrices to be very low. In fact, some rank patterns are allowed for the off-diagonal blocks, so that even if the ranks depend on the frontal matrix sizes, we can still achieve satisfactory performance. This is especially useful for 3D PDEs and more general problems.

Specifically, assume a frontal matrix $\mathcal{F}$ in the multifrontal factorization is hierarchically partitioned into $l_{\max}$ levels and follows a off-diagonal rank pattern r_l, which is the maximum (numerical) rank of the off-diagonal blocks at level l. Also assume the level l off-diagonal blocks have maximum size N_l. Then, say, if $r_l = O(\log^p N_l)$ $(p \in \mathbb{N})$, $O(N_l^{1/p})$ $(p \in \mathbb{N},\ p \geq 2)$, or $O(\alpha^{l_{\max}-l} r_0)$ $(\alpha, r_0 > 0)$, then the factorization complexity is about $O(n)$ for 2D discretized matrices, and about $O(n)$, $O\left(n^{10/9}\right)$, or $O\left(n^{4/3}\right)$ for 3D (see Tables 4.2 and 4.3 for the details). The solution costs and memory are nearly $O(n)$ for both 2D and 3D, which makes the methods very attractive for preconditioning and for problems with many right-hand sides (such as in seismic imaging). Unlike [40], two different optimization strategies in [39] are used here. We minimize the factorization cost in 2D, while the solution cost in 3D. Note that the method in [39] needs at least $O\left(n^{4/3}\right)$ flops for 3D problems, and is generally slower than the new method. Moreover, we avoid storing large dense update matrices. The new methods also use multiple tree structures and various graph techniques. They thus have nice data locality and are suitable for parallel computations.

1.2. Outline. The remaining part of the paper is organized as follows. Section 2 provides some preliminaries about HSS structures and a randomized HSS construction. The randomized multifrontal methods are discussed step by step in Section 3, and are then summarized and analyzed in Section 4. Section 5 presents the numerical tests. We draw some conclusions in Section 6.

The presentation uses the following notation:

– $1 : N$ or $\{1 : N\}$ means the index set $\{1, 2, \ldots, N\}$.
– $F_{i,j}$ or $F|_{i,j}$ denotes the (i, j) entry of a matrix F.
– $F|_{\mathbf{I}_i \times \mathbf{I}_j}$ is the submatrix of F given by the row and column index sets $\mathbf{I}_i$ and $\mathbf{I}_j$, respectively.
– $F|_{\mathbf{I}_i}$ or $F|_{\mathbf{I}_i \times :}$ is the submatrix of F given by the rows specified by the index set $\mathbf{I}_i$. Similarly, define $F|_{: \times \mathbf{I}_j}$.

2. HSS representations and randomized HSS construction. We first give some preliminaries about HSS representations and their construction via randomized sampling.

2.1. Review of HSS representations. In the HSS definition, an $N \times N$ dense matrix F is hierarchically partitioned, and the off-diagonal blocks are represented (or approximated) by low-rank forms [5, 6, 41]. This follows the structure of a binary tree T. Suppose T has k nodes which are labeled as $1, 2, \ldots, k$ in its postordering. That is, k is the root of T, and the children c_1 and c_2 of any non-leaf node i are ordered as $c_1 < c_2 < i$, where c_1 is the left child of i. Assume i is associated with an index set $\mathbf{I}_i$ that is defined recursively as

$$\mathbf{I}_i = \mathbf{I}_{c_1} \cup \mathbf{I}_{c_2}, \quad \mathbf{I}_{c_1} \cap \mathbf{I}_{c_2} = \emptyset,$$

so that $\mathbf{I}_k = \mathcal{I} \equiv \{1 : N\}$.

An HSS form of F is then defined recursively to be [5, 6, 41]

$$F = D_k,$$

where for each non-leaf node i of T with left child c_1 and right child c_2,

$$D_i \equiv F|_{\mathbf{I}_i \times \mathbf{I}_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_2}^T & D_{c_2} \end{pmatrix}, \; U_i = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}, \; V_i = \begin{pmatrix} V_{c_1} W_{c_1} \\ V_{c_2} W_{c_2} \end{pmatrix}. \tag{2.1}$$

We say D_i, U_i, V_i, R_i, W_i, and B_i are (HSS) *generators* associated with node i. Due to the recursion, we only store the D, U, V generators corresponding to the leaves of T as well as all the R, W, B generators. If F is symmetric, only the D, U, R, B generators are involved [41]. Also, T is called the corresponding *HSS tree*. See Figure 2.1.

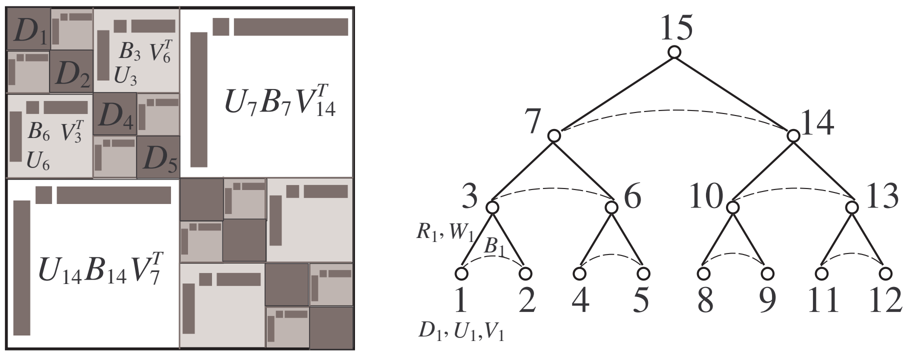


FIG. 2.1. *Pictorial illustrations of an HSS matrix example and the corresponding HSS tree.*

Later for convenience, we use the following names and notation for HSS matrices:
- $F_i^- \equiv F|_{\mathbf{I}_i \times (\mathcal{I} \backslash \mathbf{I}_i)}$ and $F_i^| \equiv F|_{(\mathcal{I} \backslash \mathbf{I}_i) \times \mathbf{I}_i}$ are called the i-th *HSS block row and column* of F, respectively. The maximum (numerical) rank of all HSS blocks of F is called the *HSS rank* of F.
- The block row (or column) of F corresponding to node i is said to be the i-th block row (or column) of F.
- $k = \text{root}\,(T)$ denotes the root of T, and $j = \text{sib}\,(i)$ denotes the sibling node of node i of T.
- The generators associated the nodes at level l of T are also said to be the level-l generators, where root (T) is at level 0.

2.2. Randomized HSS construction. We then review an HSS construction method in [28, 42] that uses randomization, which is what we need in our sparse factorization. The construction applies hierarchically a randomized compression scheme to the off-diagonal blocks of F.

First, given an $M_1 \times M_2$ matrix Φ with numerical rank r and $M_1 \geq M_2$, the randomized method in [21, 23] compresses Φ as follows. (The case $M_1 < M_2$ can be similarly considered.) Choose an $M_2 \times (r + \mu)$ Gaussian random matrix X, where μ is a small constant, and compute

$$Y = \Phi X.$$

(We call $\mathbf{r} \equiv r + \mu$ the *sampling size*.) Then compute a strong rank-revealing factorization [19] of Y:

$$Y \approx \Pi \begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix} \Omega = \Pi \begin{pmatrix} I \\ Z_2 Z_1^{-1} \end{pmatrix} (Z_1 \Omega) = \Pi \begin{pmatrix} I \\ E \end{pmatrix} Y|_{\hat{\mathbf{I}}},$$

where Π is a permutation matrix, Z_1 is $r \times r$, $E = Z_2 Z_1^{-1}$, and $Y|_{\hat{\mathbf{I}}}$ is a submatrix of Y with row index set $\hat{\mathbf{I}}$. Then it is shown in [21, 23] that Φ can be approximated by:

$$\Phi \approx U \Phi|_{\hat{\mathbf{I}}}, \text{ with } U = \Pi \begin{pmatrix} I \\ E \end{pmatrix}. \tag{2.2}$$

Moreover, under certain mild assumptions for μ, the probability for the approximation error $\left\| \Phi - U_1 \Phi|_{\hat{\mathbf{I}}} \right\|_2$ to satisfy the following bound is $1 - 6\mu^{-\mu}$ [21, 23, 29]:

$$\left\| \Phi - U_1 \Phi|_{\hat{\mathbf{I}}} \right\|_2 \leq (1 + 11\sqrt{\mathbf{r}}\sqrt{\min(M_1, M_2)})\sigma_{r+1}, \tag{2.3}$$

where σ_{r+1} is the $(r+1)$-st singular value of Φ. In [28], $\mu = 10$ is used. The above compression scheme is called a *structure-preserving rank-revealing* (*SPRR*) factorization in [42].

Then to construct an HSS form with randomized sampling for an order N dense matrix F with HSS rank r, SPRR factorizations are applied to the HSS blocks of F. We follow the basic method in [28], with additional structures considered as in [42]. Here, we focus on a symmetric F, and the scheme can be further simplified.

Choose an $N \times (r + \mu)$ Gaussian random matrix X and compute

$$Y = FX. \tag{2.4}$$

We traverse a given HSS tree T in a bottom-up order for its nodes $i = 1, 2, \ldots$

If i is a leaf, set the generator $D_i \equiv F|_{\mathbf{I}_i \times \mathbf{I}_i}$, where $\mathbf{I}_i$ is given as in Section 2.1. Let $X_i \equiv X|_{\mathbf{I}_i}$ and $Y_i \equiv Y|_{\mathbf{I}_i}$. Since $F_i^- X|_{(\mathcal{I} \backslash \mathbf{I}_i)} + D_i X_i = Y_i$, let

$$\Phi_i \equiv F_i^- X|_{(\mathcal{I} \backslash \mathbf{I}_i)} = Y_i - D_i X_i. \tag{2.5}$$

Use the SPRR factorization to compute an approximation (like in (2.2))

$$\Phi_i \approx U_i \Phi_i|_{\hat{\mathbf{I}}_i}, \text{ with } U_i = \Pi_i \begin{pmatrix} I \\ E_i \end{pmatrix}. \tag{2.6}$$

Then

$$F_i^- \approx U_i F_i^-|_{\hat{\mathbf{I}}_i}. \tag{2.7}$$

Since $F_i^-|_{\hat{\mathbf{I}}_i}$ is a submatrix of F, we write its row index set in F as $\tilde{\mathbf{I}}_i$. Also, let

$$\hat{Y}_i = U_i^T X_i = \begin{pmatrix} I & E_i^T \end{pmatrix} \Pi_i^T X_i. \tag{2.8}$$

$\hat{Y}_i$ is used in (2.9) later and its meaning is explained after (2.10).

If i is a non-leaf node with the left child c_1 and right child c_2, assume we already obtain $\tilde{\mathbf{I}}_{c_1}$, $\hat{Y}_{c_1}$, $\tilde{\mathbf{I}}_{c_2}$, and $\hat{Y}_{c_2}$ as before. Set

$$B_{c_1} = F|_{\tilde{\mathbf{I}}_{c_1} \times \tilde{\mathbf{I}}_{c_2}}.$$

According to [28, 42], we compute the following SPRR compression:

$$\Phi_i \equiv \begin{pmatrix} \Phi_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_1}\hat{Y}_{c_2} \\ \Phi_{c_2}|_{\mathbf{I}_{c_2}} - B_{c_2}\hat{Y}_{c_1} \end{pmatrix} \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \Phi|_{\hat{\mathbf{I}}_i}, \text{ with } \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} = \Pi_i \begin{pmatrix} I \\ E_i \end{pmatrix}. \tag{2.9}$$

Similarly, it can be shown that (2.7) holds, with U_i recursively given as in (2.1). We also write the row index set of $\Phi|_{\hat{\mathbf{I}}_i}$ in F as $\tilde{\mathbf{I}}_i$, and let

$$\hat{Y}_i = \begin{pmatrix} R_{c_1}^T & R_{c_2}^T \end{pmatrix} \begin{pmatrix} \hat{Y}_{c_1} \\ \hat{Y}_{c_2} \end{pmatrix} = \begin{pmatrix} I & E_i^T \end{pmatrix} \Pi_i^T \begin{pmatrix} \hat{Y}_{c_1} \\ \hat{Y}_{c_2} \end{pmatrix}. \tag{2.10}$$

Due to the recursion as in (2.8) and (2.10), we still have $\hat{Y}_i = U_i^T X_i$. The storage of $\hat{Y}_{c_1}$ and $\hat{Y}_{c_2}$ for the computation of Φ_i in (2.9) can be justified as follows. Consider, say, $\Phi_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_1}\hat{Y}_{c_2}$ in (2.9). Since

$$\begin{aligned} U_{c_1}(\Phi_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_1}\hat{Y}_{c_2}) &= U_{c_1}\Phi_{c_1}|_{\mathbf{I}_{c_1}} - U_{c_1}B_{c_1}\hat{Y}_{c_2} = U_{c_1}\Phi_{c_1}|_{\mathbf{I}_{c_1}} - U_{c_1}B_{c_1}U_{c_2}^T X_{c_2} \\ &= F|_{\mathbf{I}_{c_1}\times(\mathcal{I}\backslash\mathbf{I}_{c_1})} X|_{\mathcal{I}\backslash\mathbf{I}_{c_1}} - F|_{\mathbf{I}_{c_1}\times\mathbf{I}_{c_2}} X_{c_2}, \end{aligned}$$

$\Phi_{c_1}|_{\mathbf{I}_{c_1}} - B_{c_1}\hat{Y}_{c_2}$ can be understood as the (recursive) subtraction of the product $F|_{\mathbf{I}_{c_1}\times\mathbf{I}_{c_2}} X_{c_2}$ from the product $F|_{\mathbf{I}_{c_1}\times(\mathcal{I}\backslash\mathbf{I}_{c_1})} X|_{\mathcal{I}\backslash\mathbf{I}_{c_1}}$ (with the basis matrix U_{c_1} excluded), which is the top portion of $F_i^- X_i$. That is, (2.9) provides a quick way to compute $F_i^- X_i$ (with the basis matrices U_{c_1} and U_{c_2} excluded).

Then the process repeats.

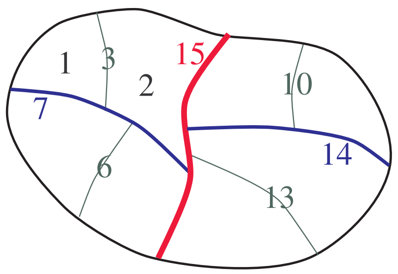
3. Randomized structured multifrontal methods.

For convenience, assume A is a symmetric positive definite (SPD) sparse matrix. (Generalizations of our discussions can be made for nonsymmetric or indefinite matrices and are not shown here.)

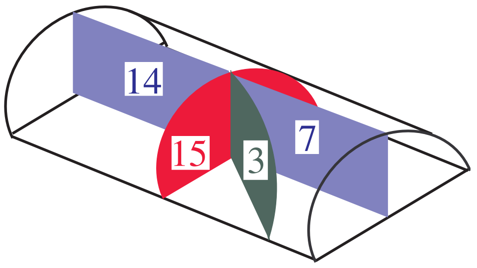
3.1. Review of multifrontal and structured multifrontal methods.

The multifrontal method transforms the overall factorization of the sparse matrix A into a series of smaller local factorizations. Here, we focus on a supernodal version that is combined with the nested dissection ordering of A [15] to reduce fill-in. The reordering of A follows that of its adjacency graph or mesh as follows. Choose some small sets of variables (called *separators*) to recursively divide the graph into smaller pieces at some hierarchical levels. See Figure 3.1. The separators are eliminated following a tree structure, called *assembly tree* [12, 25]. See Figure 3.2(i). Separators corresponding to lower level nodes of the assembly tree $\mathcal{T}$ are eliminated first.

In the multifrontal method, a node (or separator) $\mathbf{i}$ of $\mathcal{T}$ corresponds to a dense matrix $\mathcal{F}_\mathbf{i}$ called a *frontal matrix*, which is obtained as follows. Let $\mathcal{N}_\mathbf{i}$ be the set of

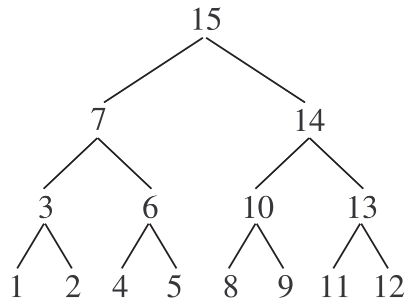


(i) 3 levels of partition in 2D

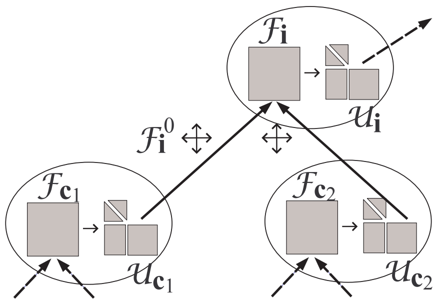


(ii) 3 levels of partition in 3D

FIG. 3.1. *Nested dissection for a 2D and a 3D domain.*



(i) An assembly tree $\mathcal{T}$ for Figure 3.1(i) or (ii)


$\mathcal{F}_{\mathbf{i}}$
$\mathcal{U}_{\mathbf{i}}$
$\mathcal{F}_{\mathbf{i}}^0$
$\mathcal{F}_{\mathbf{c}_1}$
$\mathcal{U}_{\mathbf{c}_1}$
$\mathcal{F}_{\mathbf{c}_2}$
$\mathcal{U}_{\mathbf{c}_2}$

(ii) An intermediate factorization step

FIG. 3.2. *Example of an assembly tree and an intermediate step of the multifrontal method.*

neighbors of $\mathbf{i}$, where a separator $\mathbf{j}$ is a neighbor of $\mathbf{i}$ if $\mathbf{j}$ is at an upper level (closer to root $(\mathcal{T})$) and is connected to $\mathbf{i}$ after the elimination of lower level separators [31, 33].

If $\mathbf{i}$ is a leaf of $\mathcal{T}$, set

$$\mathcal{F}_{\mathbf{i}} \equiv F_{\mathbf{i}}^0, \text{ with } \mathcal{F}_{\mathbf{i}}^0 = \begin{pmatrix} A_{\mathbf{ii}} & (A_{\mathcal{N}_{\mathbf{i}},\mathbf{i}})^T \\ A_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} & 0 \end{pmatrix}, \tag{3.1}$$

where $A_{\mathbf{ii}}$ denotes the submatrix of A whose row and column indices correspond to separator $\mathbf{i}$. $A_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}$ can be similarly understood. If $\mathbf{i}$ is a non-leaf node, let

$$\mathcal{F}_{\mathbf{i}} = \mathcal{F}_{\mathbf{i}}^0 \oplus \mathcal{U}_{\mathbf{c}_1} \oplus \mathcal{U}_{\mathbf{c}_2}, \tag{3.2}$$

where $\mathcal{U}_{\mathbf{c}_1}$ and $\mathcal{U}_{\mathbf{c}_2}$ are the Schur complements obtained from the elimination steps associated with $\mathbf{c}_1$ and $\mathbf{c}_2$, respectively (see the induction step (3.3)–(3.4)), and the symbol $\oplus$ represents an operation to assemble the data, called *extend-add* operation. This operation adds matrix entries according to the indices in A.

The elimination performed on $\mathcal{F}_{\mathbf{i}}$ is a partial factorization. Partition $\mathcal{F}_{\mathbf{i}}$ conformably following $\mathcal{F}_{\mathbf{i}}^0$ in (3.1), and compute the Cholesky factorization of the leading block:

$$\mathcal{F}_{\mathbf{i}} \equiv \begin{pmatrix} \mathcal{F}_{\mathbf{ii}} & \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}^T \\ \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} & \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} \end{pmatrix} = \begin{pmatrix} \mathcal{L}_{\mathbf{ii}} & \\ \mathcal{L}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} & I \end{pmatrix} \begin{pmatrix} I & \\ & \mathcal{U}_{\mathbf{i}} \end{pmatrix} \begin{pmatrix} \mathcal{L}_{\mathbf{ii}}^T & \mathcal{L}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}^T \\ & I \end{pmatrix}, \tag{3.3}$$

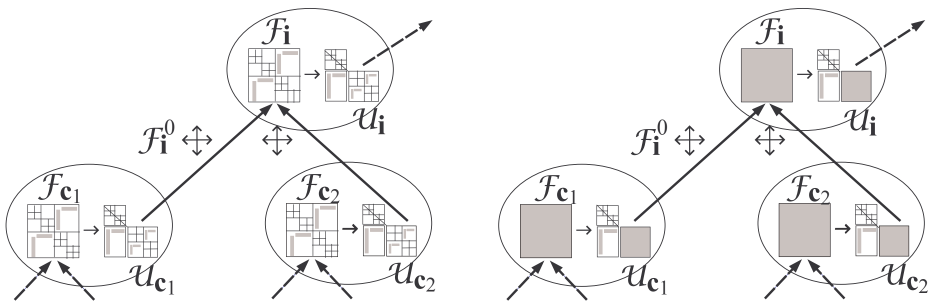
where $\mathcal{U}_{\mathbf{i}}$ is the Schur complement

$$\mathcal{U}_{\mathbf{i}} = \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} (\mathcal{F}_{\mathbf{ii}})^{-1} \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}^T = \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - \mathcal{L}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} \mathcal{L}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}^T. \tag{3.4}$$

$\mathcal{U}_{\mathbf{i}}$ is called the *update matrix* at step $\mathbf{i}$. See Figure 3.2(ii). This then repeats for all $\mathbf{i}$, until $\mathbf{i} = \mathrm{root}\,(\mathcal{T})$ is reached, where $\mathcal{F}_{\mathbf{i}} \equiv \mathcal{F}_{\mathbf{ii}}$ is full factorized. Note that the local eliminations associated with the nodes at a same level of $\mathcal{T}$ can be performed in parallel.

In the structured multifrontal method in [40], $\mathcal{F}_{\mathbf{i}}$ and $\mathcal{U}_{\mathbf{i}}$ are approximated by HSS matrices. See Figure 3.3(i). There are two major stages involved recursively:

1. **Passing information:** an HSS form extend-add operation (3.2) is performed. The HSS forms of $\mathcal{U}_{c_1}$ and $\mathcal{U}_{c_2}$ are permuted and extended to match the HSS partition of $\mathcal{F}_{\mathbf{i}}^0$. This also needs to split and merge existing HSS blocks.
2. **Factorization:** a ULV-type factorization [6] is applied to $\mathcal{F}_{\mathbf{ii}}$, and the HSS form of (3.4) is computed.


$\mathcal{F}_{\mathbf{i}}$
$\mathcal{U}_{\mathbf{i}}$
$\mathcal{F}_{\mathbf{i}}^0$
$\mathcal{F}_{\mathbf{c}_1}$
$\mathcal{U}_{\mathbf{c}_1}$
$\mathcal{F}_{\mathbf{c}_2}$
$\mathcal{U}_{\mathbf{c}_2}$
$\mathcal{F}_{\mathbf{i}}$
$\mathcal{U}_{\mathbf{i}}$
$\mathcal{F}_{\mathbf{i}}^0$
$\mathcal{F}_{\mathbf{c}_1}$
$\mathcal{U}_{\mathbf{c}_1}$
$\mathcal{F}_{\mathbf{c}_2}$
$\mathcal{U}_{\mathbf{c}_2}$

(i) The structured factorization in [40] (ii) The structured factorization in [39]

FIG. 3.3. *Illustration of two existing structured multifrontal methods.*

The HSS form extend-add operation is generally complicated. One reason is that it involves the splitting, permutation, and merging of HSS blocks. Another reason is that the HSS partition of $\mathcal{F}_{\mathbf{i}}$ is decided based on the accumulation of the partitions of $\mathcal{U}_{\mathbf{c}_1}$ and $\mathcal{U}_{\mathbf{c}_2}$. (Such a strategy helps preserve the connectivity among separators and is useful for reducing related HSS ranks.) Thus, the method in [40] focuses on 2D regular meshes, where the pattern of the extend-add operation can be decided in advance.

Simplified versions are then proposed in [37, 39]. A dense $\mathcal{F}_{\mathbf{i}}$ is formed first and then approximated by an HSS form with $\mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}}$ kept as a dense block. $\mathcal{F}_{\mathbf{i}}$ is partially factorized to yield a dense $\mathcal{U}_{\mathbf{i}}$. See Figure 3.3(ii). Then a regular extend-add operation is used. This method thus needs to store dense $\mathcal{F}_{\mathbf{i}}$ and $\mathcal{U}_{\mathbf{i}}$, and is slower than the method in [40] by a factor of about $O(\log n)$ for 2D problems.

3.2. Randomized structured multifrontal factorizations. Here, we present our structured multifrontal methods using randomized sampling techniques to avoid the complicated HSS operations. For convenience, we do not distinguish between a frontal/update matrix and its HSS approximation, or between ranks and numerical ranks. The methods still involve two major stages:

1. **Passing information:** a simple extend-add operation is performed on some vectors, and the result is used to compute an HSS approximation to $\mathcal{F}_{\mathbf{ii}}$.
2. **Factorization:** a ULV factorization is applied to $\mathcal{F}_{\mathbf{ii}}$, and the product of $\mathcal{U}_{\mathbf{i}}$ with certain random vectors is computed.

That is, the main difference between the new methods and the previous structured multifrontal methods is the information passed along the assembly tree and used for HSS constructions. Matrix-vector products are used to replace dense or structured matrices, and complex structured extend-add operations are replaced by a simple skinny one. The main framework of the methods is shown in Table 3.1, and is illustrated by Figure 3.4. Specifically, the five steps in Table 3.1 are marked in Figure 3.5. The details are elaborated in the following subsections.

Table 3.1
Overview of our randomized structured multifrontal methods.

Factorization	1. Construct an HSS approximation to $\mathcal{F}_{\mathbf{i}}$ from selected entries of $\mathcal{F}_{\mathbf{i}}$ and $Y_{\mathbf{i}} = \mathcal{F}_{\mathbf{i}} X_{\mathbf{i}}$, the product of $\mathcal{F}_{\mathbf{i}}$ with a random skinny matrix $X_{\mathbf{i}}$.
	2. Partially factorize the HSS form $\mathcal{F}_{\mathbf{i}}$ with a ULV factorization scheme and compute an HSS form $\mathcal{U}_{\mathbf{i}}$.
Passing information	3. Compute $Z_{\mathbf{i}} = \mathcal{U}_{\mathbf{i}} \tilde{X}_{\mathbf{i}}$, where $\tilde{X}_{\mathbf{i}}$ is a submatrix of $X_{\mathbf{i}}$ that corresponds to $\mathcal{U}_{\mathbf{i}}$.
	4. If $\mathbf{i}$ is a right child of node $\mathbf{p}$ and $\mathbf{j} = \mathrm{sib}(\mathbf{i})$, compute a skinny extend-add operation on $(\mathcal{F}_{\mathbf{p}}^0 X_{\mathbf{p}})$, $Z_{\mathbf{j}}$, and $Z_{\mathbf{i}}$ to form $Y_{\mathbf{p}}$.
	5. Compute selected entries of $\mathcal{F}_{\mathbf{p}}$ from $\mathcal{F}_{\mathbf{p}}^0$, $\mathcal{U}_{\mathbf{i}}$, and $\mathcal{U}_{\mathbf{j}}$.

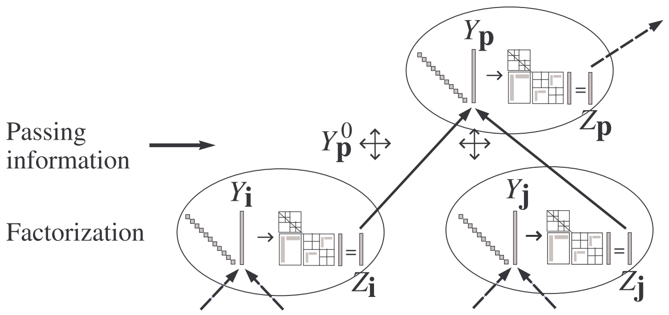


Fig. 3.4. *Illustration of our new randomized structured multifrontal methods, as compared with Figure 3.3.*

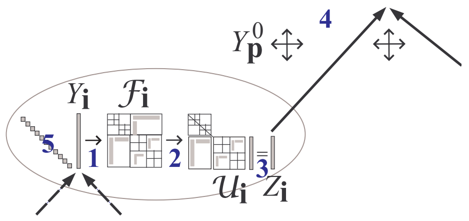


Fig. 3.5. *The operations corresponding to node* $\mathbf{i}$ *in Figure 3.4 with the five steps in Table 3.1 marked.*

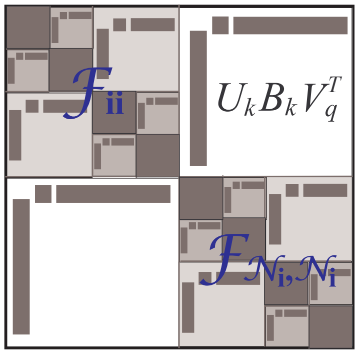
3.2.1. Frontal matrix ($\mathcal{F}_{\mathbf{i}}$) HSS construction. Step 1 in Table 3.1 uses the method in Section 2.2. Assume $\mathcal{F}_{\mathbf{i}}$ has order N and HSS rank r. By induction, Step 4 provides the product of $\mathcal{F}_{\mathbf{i}}$ with an $N \times (r + \mu)$ Gaussian random matrix $X_{\mathbf{i}}$:

$$Y_{\mathbf{i}} = \mathcal{F}_{\mathbf{i}} X_{\mathbf{i}}, \tag{3.5}$$

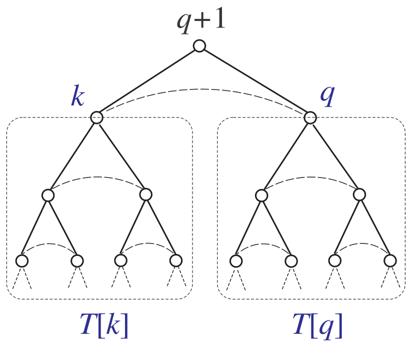
where $r+\mu$ is the sampling size. Then we compute an HSS representation (or approximation) for $\mathcal{F}_{\mathbf{i}}$ with the scheme in Section 2.2, which is assumed to be

$$\mathcal{F}_{\mathbf{i}} = \begin{pmatrix} \mathcal{F}_{\mathbf{ii}} & U_k B_k U_q^T \\ U_q B_k^T U_k^T & \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} \end{pmatrix}. \tag{3.6}$$

See Figure 3.6 with the HSS tree T for $\mathcal{F}_{\mathbf{i}}$. That is, root (T) has two children k and q, which are the roots of two subtrees $T[k]$ and $T[q]$, respectively. $T[k]$ is the HSS trees for $\mathcal{F}_{\mathbf{ii}}$ and $T[q]$ is the HSS tree for $\mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}}$. Unlike in the method in [40], $T[k]$ and $T[q]$ here can be general binary trees.


$\mathcal{F}_{\mathbf{ii}}$
$U_k B_k V_q^T$
$\mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}}$

(i) HSS form for $\mathcal{F}_{\mathbf{i}}$



(ii) HSS tree T for $\mathcal{F}_{\mathbf{i}}$ and subtrees $T[k]$ and $T[q]$

FIG. 3.6. *Illustration of an HSS form and an HSS tree for $\mathcal{F}_{\mathbf{i}}$.*

Notice that the scheme in Section 2.2 needs the D and B generators (submatrices of $\mathcal{F}_{\mathbf{i}}$) to be explicitly available. Here, $\mathcal{F}_{\mathbf{i}}$ is not fully formed in general (except for a leaf $\mathbf{i}$), but we can conveniently get those entries required by the D and B generators. See Section 3.2.5 below.

3.2.2. Partial ULV factorization of $\mathcal{F}_{\mathbf{i}}$ and computation of $\mathcal{U}_{\mathbf{i}}$. Step 2 in Table 3.1 modifies the basic ULV factorization scheme in [42] to take advantage of the symmetry, and it stops when node k is eliminated from T. This is briefly described as follows, and is not justified since similar details can be found in [6, 42].

For node $i=1,2,\ldots,k$ of T, if i is a leaf, the special structure of U_i in (2.6) means [42]

$$\left[\begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} \Pi_i^T\right] U_i = \begin{array}{cc} \begin{pmatrix} 0 \\ I \end{pmatrix} & \begin{array}{l} m-r \\ r \end{array} \end{array}, \tag{3.7}$$

where we assume U_i is $m\times r$ and all the HSS blocks of $\mathcal{F}_{\mathbf{i}}$ have (numerical) ranks r. Note that (3.7) does not cost anything. The multiplication of $\begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix}\Pi_i^T$ to the i-th block row of $\mathcal{F}_{\mathbf{i}}$ thus introduces zeros into the i-th HSS block row. Update D_i as

$$\bar{D}_i = \begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} \Pi_i^T D_i \Pi_i \begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix}^T \equiv \begin{array}{cc} \begin{array}{cc} m-r & r \end{array} & \\ \begin{pmatrix} \bar{D}_{i;1,1} & \bar{D}_{i;1,2} \\ \bar{D}_{i;2,1} & \bar{D}_{i;2,2} \end{pmatrix} & \begin{array}{l} m-r \\ r \end{array} \end{array}. \tag{3.8}$$

The special structures (zeros and identity blocks) help save the computational costs. Then compute a partial Cholesky factorization

$$\begin{pmatrix} \bar{D}_{i;1,1} & \bar{D}_{i;1,2} \\ \bar{D}_{i;2,1} & \bar{D}_{i;2,2} \end{pmatrix} = \begin{pmatrix} L_i & \\ \bar{D}_{i;2,1} L_i^{-T} & I \end{pmatrix} \begin{pmatrix} I & \\ & \hat{D}_i \end{pmatrix} \begin{pmatrix} L_i^T & L_i^{-1}\bar{D}_{i;1,2} \\ & I \end{pmatrix}. \tag{3.9}$$

Now, we can eliminate L_i.

If i is a non-leaf node with left child c_1 and right child c_2, we merge appropriate blocks obtained from the steps associated with c_1 and c_2:

$$\tilde{D}_i = \begin{pmatrix} \hat{D}_{c_1} & B_{c_1} \\ B_{c_1}^T & \hat{D}_{c_2} \end{pmatrix}, \quad \tilde{U}_i = \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}. \tag{3.10}$$

This does not cost anything either, and is significantly simpler than some similar steps in [6, 41, 42].

Then we remove c_1 and c_2 from T. This makes i a leaf, with the corresponding generators $\tilde{D}_i$, $\tilde{U}_i$, R_i, and B_i. The previous process is then repeated on i, until $k = \text{root}\,(T)$ is reached, where we compute a Cholesky factorization

$$\tilde{D}_k = L_k L_k^T. \tag{3.11}$$

Next, we compute $\mathcal{U}_{\mathbf{i}}$. The following definition is useful for replacing some HSS operations by simple operations on smaller matrices.

Definition 3.1. *[39] In the ULV factorization of an HSS matrix with an HSS tree T, assume the children of a non-leaf node i are eliminated, so that the U and V generators corresponding to i are updated to $\tilde{D}_i$ and $\tilde{U}_i$ as in (3.10), respectively. Then the resulting new HSS matrix is called a* reduced (HSS) matrix.

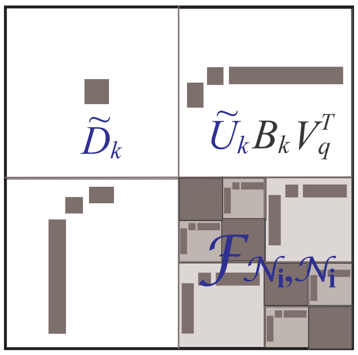
Figure 3.7 shows the reduced matrix right before the removal of node k from T, which corresponds to generators $\tilde{D}_k$, $\tilde{U}_k$, R_k, and B_k. We now have:

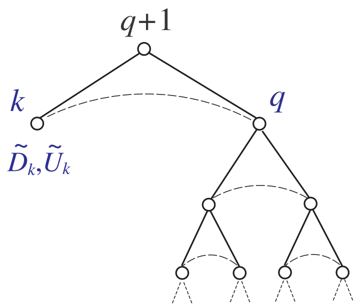
$$\mathcal{F}_{\mathbf{i}} = \begin{pmatrix} \mathcal{L}_{\mathbf{ii}} & \\ U_q B_k^T U_k^T \mathcal{L}_{\mathbf{ii}}^{-T} & I \end{pmatrix} \begin{pmatrix} I & \\ & \mathcal{U}_{\mathbf{i}} \end{pmatrix} \begin{pmatrix} \mathcal{L}_{\mathbf{ii}}^T & \mathcal{L}_{\mathbf{ii}}^{-1} U_k B_k U_q^T \\ & I \end{pmatrix}, \tag{3.12}$$

where $\mathcal{F}_{\mathbf{ii}} = \mathcal{L}_{\mathbf{ii}} \mathcal{L}_{\mathbf{ii}}^T$ represents the ULV factorization of $\mathcal{F}_{\mathbf{ii}}$, and

$$\mathcal{U}_{\mathbf{i}} = \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - U_q B_k^T \left(U_k^T \mathcal{F}_{\mathbf{ii}}^{-1} U_k\right) B_k U_q^T.$$

A direct computation with this can be expensive. In fact, it can be replaced by a fast


$\tilde{D}_k$
$\tilde{U}_k B_k V_q^T$
$\mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}}$
(i) After the partial ULV factorization of $\mathcal{F}_{\mathbf{ii}}$



(ii) The resulting tree

Fig. 3.7. *Illustration of the result after the partial ULV factorization of $\mathcal{F}_{\mathbf{i}}$ in Figure 3.6, where $\tilde{D}_k$ is the reduced matrix from $\mathcal{F}_{\mathbf{ii}}$ before the removal of node k.*

low-rank update. The following result is a direct extension of a theorem in [39].

THEOREM 3.2. (Fast Schur complement computation) *After the above partial ULV factorization of $\mathcal{F}_{\mathbf{i}}$, the Schur complement of $\mathcal{F}_{\mathbf{ii}}$ or the update matrix is*

$$\begin{aligned}\mathcal{U}_{\mathbf{i}} &= \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - U_q B_k^T \left(\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k\right) B_k U_q^T \\ &= \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - \Theta_{\mathbf{i}}^T \Theta_{\mathbf{i}}, \quad \text{with } \Theta_{\mathbf{i}} = (L_k^{-1}\tilde{U}_k) B_k U_q^T,\end{aligned} \tag{3.13}$$

where L_k is given in (3.11).

Note that $\tilde{D}_k$ is the last step reduced matrix in the ULV factorization of $\mathcal{F}_{\mathbf{ii}}$, and its size is only $O(r)$. The matrices $\tilde{D}_k$ and $\tilde{U}_k$ replace the roles of $\mathcal{F}_{\mathbf{ii}}$ and $U_{\mathbf{i}}$, respectively. If $\mathcal{F}_{\mathbf{ii}}$ has size $O(N)$, then the computation of $U_k^T \mathcal{F}_{\mathbf{ii}}^{-1} U_k$ costs $O(r^2 N)$ flops, which that of $\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k$ costs only $O(r^3)$ [39].

We can then conveniently derive the HSS form for $\mathcal{U}_{\mathbf{i}}$ using a graph method in [41]. However, unlike the Cholesky factorization in [41] which performs the following operations for each node $i = 1, 2, \ldots, k$ of $T[k]$, here we only need to preform for node k at the end of the ULV factorization of $\mathcal{F}_{\mathbf{ii}}$.

PROPOSITION 3.3. (Schur complement HSS form) *Assume the HSS generators of $\mathcal{F}$ are D_i, U_i, R_i, and B_i for $i = 1, 2, \ldots, k, k+1, \ldots, q$, and the path connecting node i to k is*

$$\mathcal{P}(i \to k): \; i \to p \to \cdots \to c \to q \to k,$$

where $p = \operatorname{par}(i)$, and c is the child of q that is an ancestor of i. Let

$$S_t = R_p \cdots R_c B_k^T \left(\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k\right) B_k R_c^T \cdots R_p^T. \tag{3.14}$$

Then $\mathcal{U}_{\mathbf{i}}$ is an HSS matrix with generators D_i, $\hat{U}_i$, R_i, and $\hat{B}_i$ for $i = k+1, k+2, \ldots, q$, where

1. *$\hat{D}_i$ for each leaf i is given by*

$$\hat{D}_i = D_i - U_i S_i U_i^T.$$

2. *$\hat{B}_i$ is given by (only needed to compute for left nodes i)*

$$\hat{B}_i = B_i - R_i S_p R_j^T,$$

where $j = \operatorname{sib}(i)$.

S_p in (3.14) can be computed in a top-down order to save the costs. That is, let

$$S_q = B_k^T \left(\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k\right) B_k.$$

Then for each node $p = q-1, q-2, \ldots, k+1$, compute

$$S_p = R_p S_t R_p^T,$$

where $t = \operatorname{par}(p)$,

The total cost for updating the generators of $\mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}}$ to get those of $\mathcal{U}_{\mathbf{i}}$ is $O\left(r^2 N\right)$, if $\mathcal{F}_{\mathbf{i}}$ has order N and HSS rank r. The storage for D_i and B_i can be used for $\hat{D}_i$ and $\hat{B}_i$, respectively, for $i = k+1, k+2, \ldots, q$.

3.2.3. From $\mathcal{U}_{\mathbf{i}}$ to the products of $\mathcal{U}_{\mathbf{i}}$ and random vectors. Next, we want to compute the product

$$Z_{\mathbf{i}} = \mathcal{U}_{\mathbf{i}} \tilde{X}_{\mathbf{i}},$$

where we partition $X_{\mathbf{i}}$ (and also $Y_{\mathbf{i}}$) according to (3.6):

$$X_{\mathbf{i}} = \begin{pmatrix} \hat{X}_{\mathbf{i}} \\ \tilde{X}_{\mathbf{i}} \end{pmatrix}, \; Y_{\mathbf{i}} = \begin{pmatrix} \hat{Y}_{\mathbf{i}} \\ \tilde{Y}_{\mathbf{i}} \end{pmatrix}. \tag{3.15}$$

The purpose of this step is to pass the skinny matrix $Z_{\mathbf{i}}$ instead of the HSS matrix $\mathcal{U}_{\mathbf{i}}$ to the node $\mathbf{p} = \mathrm{par}\,(\mathbf{i})$ (see Figure 3.4).

One way to do this is to compute directly HSS matrix-vector multiplications with the HSS form of $\mathcal{U}_{\mathbf{i}}$. This scheme can be found in [6]. A basic operation it uses is to compute $y = U_q^T x$ for a vector x as follows. Partition x into x_i pieces following the leaf level D_i sizes for $k+1 \le i \le q$. Following the bottom-up traversal of $T\,[q]$ for $i = k+1, k+2, \ldots, q$, compute

$$g_i = \begin{cases} U_i^T x_i, & \text{if } i \text{ is a leaf,} \\ R_{c_1}^T g_{c_1} + R_{c_2}^T g_{c_2} & \text{otherwise.} \end{cases} \tag{3.16}$$

Then $y = g_q$. Using this operation, Algorithm 1 shows how to compute the product of $\mathcal{U}_{\mathbf{i}}$ with a column x of $\tilde{X}_{\mathbf{i}}$. Notice that $\mathcal{U}_{\mathbf{i}}$ has generators $\hat{D}_i$, U_i, R_i, and $\hat{B}_i$.

Algorithm 1 Computing $z = \mathcal{U}_{\mathbf{i}} x$ — Direct method [6]

```
1:  procedure $\mathcal{U}_{\mathbf{i}} x$
2:      Split $x$ into pieces $x_i$ following the leaf level $\hat{D}_i$ sizes for $k+1 \le i \le q$
3:      $g_q = U_q^T x$ as in (3.16)                          ▷ Bottom-up traversal of $T[q]$
4:      $f_q = 0$
5:      for $i = q, q-1, \ldots, k+1$ do                      ▷ Top-down traversal of $T[q]$
6:          if $i$ is a non-leaf node then   ▷ Propagation of the products to the children
7:              $f_{c_1} = \hat{B}_{c_1} g_{c_2} + R_{c_1} f_i$
8:              $f_{c_2} = \hat{B}_{c_1}^T g_{c_1} + R_{c_2} f_i$
9:          else
10:             $z_i \leftarrow \hat{D}_i x_i + U_i f_i$                        ▷ Piece of $z = \mathcal{U}_{\mathbf{i}} x$
11:         end if
12:     end for
13: end procedure
```

Algorithm 1 costs about $9\,(q-k)\,r^2$ flops, if $\mathcal{F}_{\mathbf{i}}$ has HSS rank r. The special structures of the U generators in (2.6) and the R generators in (2.9) also help save the multiplication costs.

The efficiency of Algorithm 1 can be improved by an indirect computation, especially when the size of $\mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}}$ is much larger than that of $\mathcal{F}_{\mathbf{ii}}$. According to (3.6), (3.13), and (3.15), we have

$$\begin{aligned} Z_{\mathbf{i}} = \mathcal{U}_{\mathbf{i}} \tilde{X}_{\mathbf{i}} &= \mathcal{F}_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} \tilde{X}_{\mathbf{i}} - U_q B_k^T \left(\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k \right) B_k U_q^T \tilde{X}_{\mathbf{i}} \\ &= \tilde{Y}_{\mathbf{i}} - U_q B_q^T U_k^T \hat{X}_{\mathbf{i}} - U_q B_k^T \left(\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k \right) B_k U_q^T \tilde{X}_{\mathbf{i}} \\ &= \tilde{Y}_{\mathbf{i}} - U_q B_k^T \left[U_k^T \hat{X}_{\mathbf{i}} + \left(\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k \right) B_k U_q^T \tilde{X}_{\mathbf{i}} \right]. \end{aligned} \tag{3.17}$$

The computation is summarized in Algorithm 2, where we assume $\hat{x}$ is a column of $\hat{X}_{\mathbf{i}}$ and $\tilde{y}$ is a column of $\tilde{Y}_{\mathbf{i}}$. Algorithm 2 costs about $6qr^2 - 3kr^2$. The efficiency improvement over Algorithm 1 is about 1/3. Both algorithms are used for certain circumstances (see Section 3.2.5). The total cost for computing $Z_{\mathbf{i}}$ is $O\left(r^2N\right)$.

Algorithm 2 Computing $z = \mathcal{U}_{\mathbf{i}}x$ — Indirect method

```
1:  procedure $\mathcal{U}_{\mathbf{i}}x$ ($x$, $\hat{x}$, and $\tilde{y}$ have the same column index in $\tilde{X}_{\mathbf{i}}$, $\hat{X}_{\mathbf{i}}$, and $\tilde{Y}_{\mathbf{i}}$,
    respectively)
2:      Split $\tilde{y}$ into pieces $\tilde{y}_i$ following the leaf level $\hat{D}_i$ sizes for $k+1 \le i \le q$
3:      $g_q = U_q^T x$ as in (3.16)                          ▷ Bottom-up traversal of $T[q]$
4:      $g_k = U_k^T \hat{x}$ similarly to (3.16)              ▷ Bottom-up traversal of $T[k]$
5:      $f_q = B_k^T[g_k + (\tilde{U}_k^T \tilde{D}_k^{-1} \tilde{U}_k) B_k g_q]$      ▷ Part of (3.17)
6:      for $j = q, q-1, \ldots, k+1$ do                      ▷ Top-down traversal of $T[q]$
7:          if $j$ is a non-leaf node then    ▷ Propagation of the product to the children
8:              $f_{c_1} = R_{c_1} f_i$
9:              $f_{c_2} = R_{c_2} f_i$
10:         else
11:             $z_i \leftarrow \tilde{y}_i - U_i f_i$                      ▷ Piece of $z = \mathcal{U}_{\mathbf{i}}x$
12:         end if
13:     end for
14: end procedure
```

3.2.4. Skinny extend-add operation. If $\mathbf{i}$ is a right child of $\mathbf{p} = \mathrm{par}(\mathbf{i})$, we perform an extend-add operation to form $\mathcal{F}_{\mathbf{p}}$. We propose a skinny version which is significantly simpler than the regular one and the HSS one in [40]. The regular extend-add operation can be expressed by

$$\mathcal{F}_{\mathbf{p}} = \mathcal{F}_{\mathbf{p}}^0 \mathbin{\updownarrow\!\!\!\leftrightarrow} \mathcal{U}_{\mathbf{j}} \mathbin{\updownarrow\!\!\!\leftrightarrow} \mathcal{U}_{\mathbf{i}} = \mathcal{F}_{\mathbf{p}}^0 + \bar{\mathcal{U}}_{\mathbf{j}} + \bar{\mathcal{U}}_{\mathbf{i}},$$

where $\bar{\mathcal{U}}_{\mathbf{j}}$ and $\bar{\mathcal{U}}_{\mathbf{i}}$ are extended from $\mathcal{U}_{\mathbf{j}}$ and $\mathcal{U}_{\mathbf{i}}$, respectively, and are called subtree update matrices [25]. Such an extension can be represented by

$$\bar{\mathcal{U}}_{\mathbf{j}}|_{\sigma_{\mathbf{j}} \times \sigma_{\mathbf{j}}} = \mathcal{U}_{\mathbf{j}}, \quad \bar{\mathcal{U}}_{\mathbf{i}}|_{\sigma_{\mathbf{i}} \times \sigma_{\mathbf{i}}} = \mathcal{U}_{\mathbf{i}}, \tag{3.18}$$

where $\sigma_{\mathbf{j}}$ and $\sigma_{\mathbf{i}}$ are certain index sets.

Here, the extension and addition are simply performed on the matrix-vector products $Z_{\mathbf{i}}$ and $Z_{\mathbf{j}}$ for $\mathbf{j} = \mathrm{sib}(\mathbf{i})$, so as to compute $Y_{\mathbf{p}}$. Merge and extend $X_{\mathbf{j}}$ and $X_{\mathbf{i}}$ to a parent random matrix $X_{\mathbf{p}}$. That is, we first extend $\tilde{X}_{\mathbf{j}}$, $\tilde{X}_{\mathbf{i}}$, $Z_{\mathbf{j}}$, and $Z_{\mathbf{i}}$ into $\bar{X}_{\mathbf{j}}$, $\bar{X}_{\mathbf{i}}$, $\bar{Z}_{\mathbf{j}}$, and $\bar{Z}_{\mathbf{i}}$, respectively:

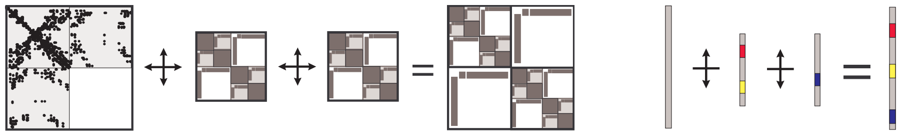
$$\bar{X}_{\mathbf{j}}|_{\sigma_{\mathbf{j}}} = \tilde{X}_{\mathbf{j}}, \quad \bar{X}_{\mathbf{i}}|_{\sigma_{\mathbf{i}}} = \tilde{X}_{\mathbf{i}}, \quad \bar{Z}_{\mathbf{j}}|_{\sigma_{\mathbf{j}}} = Z_{\mathbf{j}}, \quad \bar{Z}_{\mathbf{i}}|_{\sigma_{\mathbf{i}}} = Z_{\mathbf{i}}, \tag{3.19}$$

where the extension of $\tilde{X}_{\mathbf{j}}$ and $\tilde{X}_{\mathbf{i}}$ may introduce additional random entries, and the extension of $Z_{\mathbf{j}}$ and $Z_{\mathbf{i}}$ may introduce zero entries. Then merge $\bar{X}_{\mathbf{j}}$ and $\bar{X}_{\mathbf{i}}$ into $X_{\mathbf{p}}$ by ignoring any repeated entries. Thus, (3.18) is replaced by a simple skinny version (3.19).

Clearly,

$$\begin{aligned} Y_{\mathbf{p}} &= \left(\mathcal{F}_{\mathbf{p}}^0 \mathbin{\updownarrow\!\!\!\leftrightarrow} \mathcal{U}_{\mathbf{j}} \mathbin{\updownarrow\!\!\!\leftrightarrow} \mathcal{U}_{\mathbf{i}}\right) X_{\mathbf{p}} = \mathcal{F}_{\mathbf{p}}^0 X_{\mathbf{p}} + \bar{\mathcal{U}}_{\mathbf{j}} X_{\mathbf{p}} + \bar{\mathcal{U}}_{\mathbf{i}} X_{\mathbf{p}} \\ &= \mathcal{F}_{\mathbf{p}}^0 X_{\mathbf{p}} + \bar{\mathcal{U}}_{\mathbf{j}} \bar{X}_{\mathbf{j}} + \bar{\mathcal{U}}_{\mathbf{i}} \bar{X}_{\mathbf{i}} = \mathcal{F}_{\mathbf{p}}^0 X_{\mathbf{p}} + \bar{Z}_{\mathbf{j}} + \bar{Z}_{\mathbf{i}} \\ &\equiv Y_{\mathbf{p}}^0 \updownarrow Z_{\mathbf{j}} \updownarrow Z_{\mathbf{i}}, \end{aligned} \tag{3.20}$$

where $Y_{\mathbf{p}}^0 \equiv \mathcal{F}_{\mathbf{p}}^0 X_{\mathbf{p}}$, and the symbol $\updownarrow$ is introduced to denote a simplified version of $\oplus$ on the skinny matrices without any extension of the columns. That is, we just need to perform the *skinny extend-add operation* (3.20) (on three tall and skinny matrices) to get $Y_{\mathbf{p}}$, which is then used in Step 1 to compute an HSS approximation to $\mathcal{F}_{\mathbf{p}}$. This is illustrated in Figure 3.8.



(i) Regular HSS extend-add operation
$\mathcal{F}_{\mathbf{p}}^0 \oplus \mathcal{U}_{\mathbf{j}} \oplus \mathcal{U}_{\mathbf{i}} = \mathcal{F}_{\mathbf{p}}$

(ii) Skinny extend-add operation
$Y_{\mathbf{p}}^0 \updownarrow Z_{\mathbf{j}} \updownarrow Z_{\mathbf{i}} = Y_{\mathbf{p}}$

FIG. 3.8. *Skinny extend-add operation in our randomized multifrontal methods as compared with the regular HSS one.*

Thus, it is clear that, a substantial improvement of our randomized multifrontal methods over the classical one and the previous structured ones is, our extend-add operation is performed only on the rows of some vectors, instead of both rows and columns of dense or structured matrices. This skinny extend-add operation is thus significantly simpler and more efficient. In comparison, the structured extend-add operation in Figure 3.8(i) is much more complicated, even for problems discretized on regular meshes. It is not easy to be extended to general sparse matrices either.

3.2.5. Formation of selected entries of a frontal matrix. As stated in Section 3.2.1, computing an HSS approximation to $\mathcal{F}_{\mathbf{p}}$ requires the explicit formation of its D, B generators, which are given by selected entries of $\mathcal{F}_{\mathbf{p}}$, and thus selected entries of $\mathcal{F}_{\mathbf{p}}^0$, $\mathcal{U}_{\mathbf{j}}$, and $\mathcal{U}_{\mathbf{i}}$. Therefore, we need to form some entries of the HSS matrices $\mathcal{U}_{\mathbf{j}}$ and $\mathcal{U}_{\mathbf{i}}$. For convenience, consider $\mathcal{U}_{\mathbf{i}}$ with generators D_i, U_i, R_i, B_i, where we write $\hat{D}_i$ and $\hat{B}_i$ in Proposition 3.3 as D_i and B_i, respectively. There are different versions.

A straightforward version is to traverse the HSS tree $T[q]$ of $\mathcal{U}_{\mathbf{i}}$ (Figure 3.6(ii)) for each specified location (j_1, j_2) given by a row index j_1 and column index j_2. Assume j_1 (j_2) corresponds to a leaf i_1 (i_2) of $T[q]$, and a row index k_1 (k_2) in U_{i_1} (U_{i_2}). Then the desired entry is

$$\mathcal{U}_{\mathbf{i}}|_{j_1,j_2} = U_{i_1}|_{k_1} R_{i_1} R_{t_1} \cdots B_{c_1} \cdots R_{t_2}^T R_{i_2}^T \left(U_{i_2}|_{k_2}\right)^T, \tag{3.21}$$

where the path in $T[q]$ that connects i_1 and i_2 is assumed to be

$$\mathcal{P}\left(i_1 \to i_2\right): \ i_1 \to t_1 \to \cdots \to c_1 \to c_2 \to \cdots \to t_2 \to i_2. \tag{3.22}$$

If $\mathcal{U}_{\mathbf{i}}$ has order $O(N)$ and HSS rank r, and $T[q]$ has $O(\log N)$ levels, the computation of (3.21) costs up to $O\left(r^2 \log N\right)$. Here, we may need to compute up to $O(rN)$ entries of $\mathcal{U}_{\mathbf{i}}$, which makes the total formation cost $O\left(r^3 N \log N\right)$.

An improved version is to take advantage of the fact that, many desired entries of $\mathcal{U}_{\mathbf{i}}$ that contribute to the D, B generators of $\mathcal{F}_{\mathbf{p}}$ are from the same rows or columns. In fact, it is clear that the entries in the same row (column) of $\mathcal{U}_{\mathbf{i}}$ are also in the same row (column) of $\bar{\mathcal{U}}_{\mathbf{i}}$. The randomized HSS construction also means that each row (column) of the D, B generators consists of some entries in a same row (column) of $\mathcal{F}_{\mathbf{p}}$. Thus, we need to traverse at most N paths (instead of $O(rN)$) like (3.22) in $T[q]$ to form the $O(rN)$ entries. We then have the following lemma, which helps us reuse the computations.

LEMMA 3.4. (Computation reuse) *The entries of $\mathcal{U}_{\mathbf{i}}$ that contribute to those in the same row (column) of $\mathcal{F}_{\mathbf{p}}$ are also in the same row (column) of $\mathcal{U}_{\mathbf{i}}$. Thus, it needs $O\left(r^2 N \log N\right)$ flops to compute all those entries in $\mathcal{U}_{\mathbf{i}}$, by traversing at most N paths like (3.22) in $T[q]$.*

The final version we employ is to further reuse the information. That is, different paths $\mathcal{P}\left(i_1 \rightarrow i_2\right)$ may share common pieces also. For this purpose, we define the distance between nodes i_1 and i_2.

DEFINITION 3.5. *The* distance $d\left(i_1, i_2\right)$ *between nodes i_1 and i_2 of T is defined to be the number of nodes in the path (3.22) that are smaller than c_1 (or larger than c_2).*

With these results, we have the formation procedure in Algorithm 3, where to save space, some loops are combined. Another feature to keep in mind is that, the D generators of $\mathcal{F}_{\mathbf{p}}$ can often be quickly obtained, since they generally correspond to the D generators of $\mathcal{U}_{\mathbf{i}}$ and $\mathcal{U}_{\mathbf{j}}$. The reason is that we preserve the partition information of $\mathcal{N}_{\mathbf{i}}$ and $\mathcal{N}_{\mathbf{j}}$ when we partition separator $\mathbf{p}$ and construct an HSS form for $\mathcal{F}_{\mathbf{p}}$.

Algorithm 3 Find the entries $\mathcal{U}_{j_1,j_2}$ of an HSS matrix $\mathcal{U}$ with generators D_i, U_i, R_i, B_i for given sequences of row indices j_1 and column indices j_2

```
 1: procedure hssij
 2:     for each index $j_1$ ($j_2$) do
 3:         Find the corresponding leaf $i_1$ ($i_2$) and row index $k_1$ (column index $k_2$)
 4:     end for
 5:     for each leaf $i_1$ and $i_2$ do
 6:         Find the distance $d(i_1, i_2)$   ▷ For information reuse following the distance
 7:     end for
 8:     for each index $j_1$ ($j_2$) do                                        ▷ Tree traversal
 9:         $(x_{i_1}^1)^T = U_{i_1}|_{k_1}$     $((x_{i_2}^1)^T = U_{i_2}|_{k_2})$
                                                  ▷ Starting to access with row/column index
10:         $t = i_1$     $(t = i_2)$
11:         for $l = 2, 3, \ldots, \max_{i_2} d(i_1, i_2)$     $(l = 2, 3, \ldots, \max_{i_1} d(i_1, i_2))$ do
                          ▷ Bottom up traversal of the HSS tree for at most $\max d(i_1, i_2)$ steps
12:             $(x_{i_1}^l)^T = (x_{i_1}^{l-1})^T R_t$     $((x_{i_2}^l)^T = (x_{i_2}^{l-1})^T R_t)$
                          ▷ Performing the multiplication on the left/right of $B_{c_1}$ in (3.21)
13:             $t \leftarrow \mathrm{par}(t)$
14:         end for
15:     end for
16:     for each index $j_1$ and $j_2$ do
17:         if $i_1 = i_2$ then                                     ▷ Entry of a diagonal block
18:             $\mathcal{U}_{j_1,j_2} \leftarrow D_{i_1}|_{k_1,k_2}$
19:         else                                                 ▷ Entry of an off-diagonal block
20:             $\mathcal{U}_{j_1,j_2} \leftarrow (x_{i_1}^l)^T B_{c_1} x_{i_2}^l$                        ▷ $c_1$ as in (3.22)
21:         end if
22:     end for
23: end procedure
```

3.3. Structured multifrontal solution after randomized factorizations. The above elimination process repeats for nodes $\mathbf{i} = 1, 2, \ldots, \mathrm{root}\,(\mathcal{T})$, and yields a factorization

$$A = \mathcal{L}\mathcal{L}^T.$$

(This is generally approximate, since compression is used. We write equalities for notational convenience.)

$\mathcal{L}$ is a structured Cholesky factor and is generally not triangular. But we can still use forward and backward substitutions to solve

$$\mathcal{L}\mathbf{y} = \mathbf{b}, \text{ and} \tag{3.23}$$

$$\mathcal{L}^T\mathbf{x} = \mathbf{y}, \tag{3.24}$$

respectively. The main framework is similar to the one in [39], except that additional structures are involved. Assume the vectors $\mathbf{b}$, $\mathbf{y}$, and $\mathbf{x}$ are partitioned into pieces $\mathbf{b_i}$, $\mathbf{y_i}$, and $\mathbf{x_i}$, respectively, following the separator ordering in nest dissection.

3.3.1. Forward substitution. In a forward substitution stage, we solve (3.23) with a postordering traversal of $\mathcal{T}$. According to (3.12), in each step, we need to solve a system of the form

$$\begin{pmatrix} \mathcal{L}_{\mathbf{ii}} & \\ U_q B_k^T U_k^T \mathcal{L}_{\mathbf{ii}}^{-T} & I \end{pmatrix} \begin{pmatrix} \mathbf{y_i} \\ \tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} \end{pmatrix} = \begin{pmatrix} \mathbf{b_i} \\ \mathbf{b}_{\mathcal{N}_\mathbf{i}} \end{pmatrix}. \tag{3.25}$$

A ULV forward substitution scheme improved from those in [6, 41] is used to solve $\mathcal{L}_{\mathbf{ii}}\mathbf{y_i} = \mathbf{b_i}$. For convenience, we rewrite this system as

$$\mathcal{L}_{\mathbf{ii}} y = b.$$

Partition b and y into b_i and y_i pieces, respectively, following the leaf level D_i generator sizes of $\mathcal{F}_{\mathbf{ii}}$.

If i is a leaf of the HSS tree $T[k]$, following (3.7), let

$$\tilde{b}_i = \begin{pmatrix} -E_i & I \\ I & 0 \end{pmatrix} \Pi_i^T b_i \equiv \begin{array}{c} \begin{pmatrix} \tilde{b}_{i,1} \\ \tilde{b}_{i,2} \end{pmatrix} \end{array} \begin{array}{l} m-r \\ r \end{array}. \tag{3.26}$$

Then compute

$$y_i \equiv L_i^{-1}\tilde{b}_{i,1}, \tag{3.27}$$

where L_i is given in (3.9). If i is a non-leaf node with children c_1 and c_2, set

$$b_i = \begin{pmatrix} \tilde{b}_{c_1,2} \\ \tilde{b}_{c_2,2} \end{pmatrix} - \begin{pmatrix} \bar{D}_{c_1;2,1} y_{c_1} \\ \bar{D}_{c_2;2,1} y_{c_2} \end{pmatrix}.$$

Then we similarly compute y_i as in (3.26)–(3.27).

The traversal of $T[k]$ proceeds bottom-up. When it reaches k, we compute

$$y_k = L_k^{-1} b_k. \tag{3.28}$$

Merging all the y_i pieces together yields the solution $\mathbf{y_i} \equiv y$.

Then we need to update $\mathbf{b}_{\mathcal{N}_\mathbf{i}}$ in (3.25) as:

$$\mathbf{b}_{\mathcal{N}_\mathbf{i}} \leftarrow \tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - U_q B_k^T U_k^T \mathbf{y_i}. \tag{3.29}$$

A fast computation method can be derived based on the idea of reduced HSS matrices in a similar way as in [39].

THEOREM 3.6. (Fast right-hand side update) *For a separator* $\mathbf{i}$, *write* y_k *in (3.28) as* $\mathbf{y}_{\mathbf{i},k}$. *Assume the conditions in Theorem 3.2 hold. Then*

$$\tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - \Theta_k^T b_k = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - U_q B_k^T \tilde{U}_k^T \mathbf{y}_{\mathbf{i},k}, \tag{3.30}$$

where $\Theta_\mathbf{i}$ *is given in (3.13).*

Note that computing $\tilde{U}_k^T y_k$ in (3.30) costs $O\left(r^2\right)$ flops, while computing $U_k^T \mathbf{y}_\mathbf{i}$ in (3.29) costs $O\left(rN\right)$ flops.

3.3.2. Backward substitution. In a backward substitution stage, we solve (3.24) with a reverse-postordering traversal of $\mathcal{T}$. According to (3.12), we need to solve intermediate systems of the form

$$\begin{pmatrix} \mathcal{L}_{\mathbf{ii}}^T & \mathcal{L}_{\mathbf{ii}}^{-1} U_k B_k U_q^T \\ & I \end{pmatrix} \begin{pmatrix} \mathbf{x}_\mathbf{i} \\ \mathbf{x}_{\mathcal{N}_\mathbf{i}} \end{pmatrix} = \begin{pmatrix} \mathbf{y}_\mathbf{i} \\ \mathbf{x}_{\mathcal{N}_\mathbf{i}} \end{pmatrix},$$

where $\mathbf{x}_{\mathcal{N}_i}$ is already computed in the steps associated with the separators in $\mathcal{N}_\mathbf{i}$. We first compute

$$\tilde{\mathbf{y}}_\mathbf{i} = \mathbf{y}_\mathbf{i} - \mathcal{L}_{\mathbf{ii}}^{-1} U_k B_k U_q^T \mathbf{x}_{\mathcal{N}_\mathbf{i}}. \tag{3.31}$$

Similarly, this can be quickly computed as follows based on Theorem 3.2 and [39].

THEOREM 3.7. (Fast right-hand side update) *Assume the* $y_i \equiv \mathbf{y}_{\mathbf{i},i}$ *pieces in the forward substitution in Section 3.3.1 form* $\mathbf{y}_\mathbf{i}$. *Write* y_k *in (3.28) as* $\mathbf{y}_{\mathbf{i},k}$. *Then the computation of (3.31) is equivalent to updating only the piece* $\mathbf{y}_{\mathbf{i},k}$ *by*

$$\mathbf{y}_{\mathbf{i},k} \leftarrow \tilde{\mathbf{y}}_{\mathbf{i},k} = \mathbf{y}_{\mathbf{i},k} - \Theta_\mathbf{i} \mathbf{x}_{\mathcal{N}_\mathbf{i}}, \tag{3.32}$$

where $\Theta_\mathbf{i}$ *is given in (3.13).*

Then we solve the system $\mathcal{L}_{\mathbf{ii}}^T \mathbf{x}_\mathbf{i} = \tilde{\mathbf{y}}_\mathbf{i}$ with a ULV-type backward substitution. For convenience and by abuse of notation, we rewrite it as

$$\mathcal{L}_{\mathbf{ii}}^T x = y.$$

Also, partition y and x into y_i and x_i pieces, respectively, following the leaf level D_i generator sizes of $\mathcal{F}_{\mathbf{ii}}$.

For node k of $T[k]$, compute

$$x_k = L_k^{-T} y_k.$$

For a non-leaf node i, partition x_i as $x_i = \begin{array}{c} \begin{pmatrix} x_{i,1} \\ x_{i,2} \end{pmatrix} \end{array} \begin{array}{c} r \\ r \end{array}$ and compute

$$x_{c_1} = \begin{pmatrix} -E_{c_1} & I \\ I & 0 \end{pmatrix} \begin{pmatrix} L_{c_1}^{-T} y_{c_1} \\ x_{i,1} \end{pmatrix}, \quad x_{c_2} = \begin{pmatrix} -E_{c_2} & I \\ I & 0 \end{pmatrix} \begin{pmatrix} L_{c_2}^{-T} y_{c_2} \\ x_{i,2} \end{pmatrix},$$

where c_1 and c_2 are the children of i, and the partitions in the matrix-vector multiplications may not be consistent.

When all the leaves are visited, merge all the pieces x_i corresponding to the leaves i to form x.

3.4. Adaptive schemes. There levels of adaptivity can be built into the algorithms to enhance the efficiency. The first level already exists in the SPRR factorization. That is, although the sampling size $r+\mu$ is larger than the actual off-diagonal rank r, SPRR can approximately detect the right r.

The second level is to use variable sampling sizes at different hierarchical levels of the randomized HSS construction algorithm in Section 2.2. That is, the initial X in (2.4) has a column size equal to the maximum rank of the leaf level HSS blocks. Later to compute Φ_i in (2.9), we revisit lower level submatrices (already in HSS forms), so as to use different random vectors. Here, we omit the details (which are similar to the next level adaptivity below). Another strategy is to use a top-down HSS construction method in [24], although $\mathcal{F}_{\mathbf{i}}$ may need to be multiplied by as many as $O(\log N)$ random matrices, instead of one single X in Section 2.2.

Finally, the top level adaptivity is to use flexible sampling sizes at different levels of the assembly tree $\mathcal{T}$. That is, a sampling size $\mathbf{r}_{\mathbf{l}}$ is used at level $\mathbf{l}$ of $\mathcal{T}$, which may be decided based on certain rank patterns (see Remark 4.2 below). We generally use a smaller $\mathbf{r}_{\mathbf{l}}$ for a larger $\mathbf{l}$ (closer to the leaves). This strategy helps save the factorization costs at those levels.

In addition, we can also flexibly choose the algorithm for computing $Z_{\mathbf{i}} = \mathcal{U}_{\mathbf{i}}\tilde{X}_{\mathbf{i}}$ in Section 3.2.3, since Algorithm 2 is generally faster than Algorithm 1. That is, if $\mathbf{r}_{\mathbf{l}} \geq \mathbf{r}_{\mathbf{l}-1}$, only selected columns of $\tilde{X}_{\mathbf{i}}$ (say, $\tilde{X}_{\mathbf{i}}|_{:\times(1:r_{\mathbf{l}-1})}$) contribute to $X_{\mathbf{p}}$. Then we can compute $Z_{\mathbf{i}} = \mathcal{U}_{\mathbf{i}}\tilde{X}_{\mathbf{i}}|_{:\times(1:r_{\mathbf{l}-1})}$ with Algorithm 2. Otherwise, we expand $\tilde{X}_{\mathbf{i}}$ by adding additional columns. Then $Z_{\mathbf{i}}$ is formed from two pieces: $Z_{\mathbf{i}}|_{:\times(1:r_{\mathbf{l}-1})} = \mathcal{U}_{\mathbf{i}}\tilde{X}_{\mathbf{i}}$ computed with Algorithm 2, and $Z_{\mathbf{i}}|_{:\times(r_{\mathbf{l}-1}+1:r_{\mathbf{l}})} = \mathcal{U}_{\mathbf{i}}\hat{X}_{\mathbf{i}}$ computed with Algorithm 1.

See Algorithm 5 in the next section for a summary. More details and additional improvements will be given in future work.

4. Algorithms and performance analysis. In this section, we summarize the algorithms and then show the complexity based a rank relaxation idea. Just like existing structured multifrontal methods in [39, 40], below certain level $\mathbf{l}_s$ of the assembly tree $\mathcal{T}$, exact factorizations are computed. Structured factorizations are computed above (or after) $\mathbf{l}_s$. This level is then called a *switching level*, which helps the algorithms achieve nearly optimal complexity. See Algorithms 4, 5, and 6. These algorithms are highly parallelizable and can be generalized to nonsymmetric and indefinite sparse matrices (ignoring pivoting issues).

For the complexity analysis, we focus on 2D and 3D discretized PDEs. Similar to the complexity results in [38, 39], our analysis relaxes the rank requirements. That is, our methods can be applied to more general problems where the HSS rank of $\mathcal{F}_{\mathbf{i}}$ is not small or depends on N. Such situations arise often, especially in 3D problems [8].

The following lemma summarizes some dense rank relaxation results from [38] with modifications for the randomized case.

Lemma 4.1. (Dense rank relaxation) *Assume an $N \times N$ dense matrix F is partitioned into $l_{\max} = O(\log N)$ levels of HSS blocks following a perfect binary tree T with $l_{\max}$ levels. Then the costs and memory of some HSS algorithms are as shown in Table 4.1, where*

- *$N_l \equiv O(N/2^l)$ is the maximum row dimension of the HSS blocks corresponding to level l of T,*
- *r_l is the maximum (numerical) rank of these HSS blocks, with $r = \max_l r_l$,*
- *$\tilde{\xi}_{\text{constr}}$ is the cost to construct an HSS form for F with Y in (2.4),*

Algorithm 4 Randomized structured multifrontal factorization

```
1: procedure RSMF
2:     for node/separator $\mathbf{i}$ from 1 to root$(\mathcal{T})$ do
3:         if $\mathbf{i}$ is a leaf of $\mathcal{T}$ then
4:             $\mathcal{F}_{\mathbf{i}} = \mathcal{F}_{\mathbf{i}}^0$                          ▷ Initial frontal matrix
5:         end if
6:         if $\mathbf{i}$ is at level $\mathbf{l} > \mathbf{l}_s$ of $\mathcal{T}$ then          ▷ Exact factorization below $\mathbf{l}_s$
7:             $\mathcal{F}_{\mathbf{i},\mathbf{i}} = \mathcal{L}_{\mathbf{ii}}\mathcal{L}_{\mathbf{ii}}^T$                  ▷ Exact Cholesky factorization
8:             $\mathcal{L}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} = F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}\mathcal{L}_{\mathbf{ii}}^{-T}$
9:             $\mathcal{U}_{\mathbf{i}} = F_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - \mathcal{L}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}\mathcal{L}_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}^T$             ▷ Exact update matrix
10:            if $\mathbf{i}$ is a left node then
11:                Push $\mathcal{U}_{\mathbf{i}}$ onto a stack            ▷ For later extend-add operation
12:            else                                          ▷ Assembly of $\mathcal{F}_{\mathbf{p}}$
13:                Pop $\mathcal{U}_{\mathbf{j}}$ from the stack for $\mathbf{j} = \mathrm{sib}(\mathbf{i})$
14:                $\mathcal{F}_{\mathbf{p}} = \mathcal{F}_{\mathbf{p}}^0 \updownarrow\!\!\!\!\leftrightarrow \mathcal{U}_{\mathbf{j}} \updownarrow\!\!\!\!\leftrightarrow \mathcal{U}_{\mathbf{i}}$              ▷ Exact extend-add operation
15:            end if
16:        else                                          ▷ Structured factorization above $\mathbf{l}_s$
17:            if $\mathbf{i}$ is at level $\mathbf{l} = \mathbf{l}_s$ of $\mathcal{T}$ then     ▷ Direct computation of $Y_{\mathbf{i}} = \mathcal{F}_{\mathbf{i}}X_{\mathbf{i}}$
18:                Generate random $X_{\mathbf{i}}$ in (3.5) and compute $Y_{\mathbf{i}} = \mathcal{F}_{\mathbf{i}}X_{\mathbf{i}}$
19:            end if
20:            Compute a partial ULV factorization of $\mathcal{F}_{\mathbf{i}}$ as in (3.12)
21:            $Z_{\mathbf{i}} = \mathcal{U}_{\mathbf{i}}\tilde{X}_{\mathbf{i}}$                                   ▷ Algorithm 2
22:            if $\mathbf{i}$ is a left node then
23:                Push $\tilde{X}_{\mathbf{i}}$ and $Z_{\mathbf{i}}$ onto a stack        ▷ For later skinny extend-add
24:            else                                     ▷ Assembly of $Y_{\mathbf{p}} = \mathcal{F}_{\mathbf{p}}X_{\mathbf{p}}$
25:                Pop $\tilde{X}_{\mathbf{j}}$ and $Z_{\mathbf{j}}$ from the stack for $\mathbf{j} = \mathrm{sib}(\mathbf{i})$
26:                Merge $\tilde{X}_{\mathbf{i}}$ and $\tilde{X}_{\mathbf{j}}$ and introduce new random entries to form $X_{\mathbf{p}}$
27:                $Y_{\mathbf{p}} = \left(\mathcal{F}_{\mathbf{p}}^0 X_{\mathbf{p}}\right) \updownarrow\!\!\!\!\leftrightarrow Z_{\mathbf{j}} \updownarrow\!\!\!\!\leftrightarrow Z_{\mathbf{i}}$           ▷ Skinny extend-add operation
28:            end if
29:        end if
30:    end for
31: end procedure
```

– $\tilde{\xi}_{\text{fact}}$ *is the cost of the ULV factorization for the HSS form,*
– $\tilde{\xi}_{\text{sol}}$ *is the ULV solution cost, and*
– $\tilde{\sigma}_{\text{mem}}$ *is the memory size.*

Here, the formula for r_l is called a *rank pattern* [39]. Clearly, HSS algorithms and related randomized versions perform well for F with various rank patterns.

Note that in our randomized multifrontal methods, Y is accumulated from lower levels of $\mathcal{T}$ via the skinny extend-add operation. According to Lemma 3.4, $\tilde{\xi}_{\text{constr}} = O\left(r^2 N \log N\right)$, which is used in the derivation of the following results based on those in [39].

THEOREM 4.2. (Sparse rank relaxation) *Suppose the randomized algorithms are applied to an order n discretized matrix A, so that the total factorization cost is ξ_{fact}, the solution cost is ξ_{sol}, and the memory size is σ_{mem}. Assume each frontal matrix satisfies the rank patterns in Lemma 4.1. Then,*

- *If A results from the discretization on a 2D $N \times N$ mesh and $n = N^2$, then choose the switching level $\mathbf{l}_s$ so that the* factorization costs *before and after $\mathbf{l}_s$*

Algorithm 5 Adaptive randomized structured multifrontal factorization

```
1: procedure ARSMF (Replace Lines 17–28 of Algorithm 4 by the following)
2:     ...
3:     Decide $\mathbf{r_l}$ if $\mathbf{i}$ is at level $\mathbf{l}$        ▷ Sampling size used at level $\mathbf{l}$ of $\mathcal{T}$
4:     if $\mathbf{i}$ is at level $\mathbf{l} = \mathbf{l}_s$ of $\mathcal{T}$ then        ▷ Direct computation of $Y_\mathbf{i} = \mathcal{F}_\mathbf{i} X_\mathbf{i}$
5:         Generate random $X_\mathbf{i}$ in (3.5) and compute $Y_\mathbf{i} = \mathcal{F}_\mathbf{i} X_\mathbf{i}$
6:     end if
7:     Compute a partial ULV factorization of $\mathcal{F}_\mathbf{i}$ as in (3.12)
8:     if $\mathbf{i}$ is a left node then
9:         Push $\tilde{X}_\mathbf{i}$ onto a stack
10:    else        ▷ Assembly of $Y_\mathbf{p} = \mathcal{F}_\mathbf{p} X_\mathbf{p}$
11:        Pop $\tilde{X}_\mathbf{j}$ from the stack for $\mathbf{j} = \mathrm{sib}(\mathbf{i})$
12:        if $r_\mathbf{l} \geq r_{\mathbf{l}-1}$ then
13:            Merge $\tilde{X}_\mathbf{i}$ and $\tilde{X}_\mathbf{j}$ and introduce new random rows to form $X_\mathbf{p}$
14:            $Z_\mathbf{i} = \mathcal{U}_\mathbf{i} \tilde{X}_\mathbf{i}|_{:\times(1:\mathbf{r}_{\mathbf{l}-1})}$, $Z_\mathbf{j} = \mathcal{U}_\mathbf{j} \tilde{X}_\mathbf{j}|_{:\times(1:\mathbf{r}_{\mathbf{l}-1})}$        ▷ Algorithm 2
15:        else
16:            Merge $\tilde{X}_\mathbf{i}$ and $\tilde{X}_\mathbf{j}$ and introduce new random rows/columns to form $X_\mathbf{p}$
17:            Extend $\tilde{X}_\mathbf{i}$ to $\begin{pmatrix} \tilde{X}_\mathbf{i} & \tilde{X}'_\mathbf{i} \end{pmatrix}$ and $\tilde{X}_\mathbf{j}$ to $\begin{pmatrix} \tilde{X}_\mathbf{j} & \tilde{X}'_\mathbf{j} \end{pmatrix}$ using entries from $X_\mathbf{p}$
18:            $Z_\mathbf{i}|_{:\times(1:\mathbf{r}_{\mathbf{l}-1})} = \mathcal{U}_\mathbf{i} \tilde{X}_\mathbf{i}$,    $Z_\mathbf{j}|_{:\times(1:\mathbf{r}_{\mathbf{l}-1})} = \mathcal{U}_\mathbf{j} \tilde{X}_\mathbf{j}$        ▷ Algorithm 2
19:            $Z_\mathbf{i}|_{:\times(\mathbf{r}_{\mathbf{l}-1}+1:\mathbf{r}_\mathbf{l})} = \mathcal{U}_\mathbf{i} \tilde{X}'_\mathbf{i}$,  $Z_\mathbf{j}|_{:\times(\mathbf{r}_{\mathbf{l}-1}+1:\mathbf{r}_\mathbf{l})} = \mathcal{U}_\mathbf{j} \tilde{X}'_\mathbf{j}$        ▷ Algorithm 1
20:        end if
21:        $Y_\mathbf{p} = \left(\mathcal{F}^0_\mathbf{p} X_\mathbf{p}\right) \updownarrow\!\!\!\!\leftrightarrow Z_\mathbf{j} \updownarrow\!\!\!\!\leftrightarrow Z_\mathbf{i}$        ▷ Skinny extend-add operation
22:    end if
23:    ...
24: end procedure
```

TABLE 4.1
Costs and memory of randomized HSS construction and other HSS algorithms for an $N \times N$ matrix F, where p is a positive integer, $\alpha > 0$, and the results for $\tilde{\xi}_{\mathrm{fact}}$, $\tilde{\xi}_{\mathrm{sol}}$, and $\tilde{\sigma}_{\mathrm{mem}}$ are from [38].

r_l		$r = \max r_l$	$\tilde{\xi}_{\mathrm{constr}}$	$\tilde{\xi}_{\mathrm{fact}}$	$\tilde{\xi}_{\mathrm{sol}}$	$\tilde{\sigma}_{\mathrm{mem}}$
$O(1)$		$O(1)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$
$O(\log^p N_l)$		$O(\log^p N)$	$O(N \log^{2p} N)$			
$O(N_l^{1/p})$	$p > 3$	$O(N^{1/p})$	$O(N^{(p+2)/p})$			
	$p = 3$	$O(N^{1/3})$	$O(N^{5/3})$	$O(N \log N)$		
	$p = 2$	$O(N^{1/2})$	$O(N^2)$	$O(N^{3/2})$	$O(N \log N)$	$O(N \log N)$
$O(\alpha^{l_{\max}-l} r_0)$	$\alpha < \sqrt[3]{2}$	$< O(N^{1/3})$	$< O(N^{5/3})$	$O(N)$	$O(N)$	$O(N)$
	$\alpha = \sqrt[3]{2}$	$O(N^{1/3})$	$O(N^{5/3})$	$O(N \log N)$		
	$\alpha < \sqrt[2]{2}$	$< O(N^{1/2})$	$< O(N^2)$	$O(N^{\log \alpha^3})$		
	$\alpha = \sqrt[2]{2}$	$O(N^{1/2})$	$O(N^2)$	$O(N^{3/2})$	$O(N \log N)$	$O(N \log N)$

are the same, and the counts are given in Table 4.2.

- *If A results from the discretization on a 3D $N \times N \times N$ mesh and $n = N^3$, then choose the switching level $\mathbf{l}_s$ so that the* solution costs *before and after $\mathbf{l}_s$ are the same, and the counts are given in Table 4.3.*

In both cases, $\mathbf{l}_{\max} - \mathbf{l}_s \leq O(\log N)$, where $\mathbf{l}_{\max}$ is the total number of levels in $\mathcal{T}$ and root $(\mathcal{T})$ *is at level* 0.

Proof. The derivation is similar to that in [41]. We only show two cases in 2D

Algorithm 6 Randomized structured multifrontal solution

```
procedure RSMS
    Partition $\mathbf{b}$ into $\mathbf{b_i}$ pieces according to the sizes of the separators
    for node $\mathbf{i}$ from 1 to root$(\mathcal{T})$ do                ▷ Forward substitution for (3.23)
        if $\mathbf{i}$ is at level $\mathbf{l} > \mathbf{l}_s$ of $\mathcal{T}$ then                ▷ Exact solution below $\mathbf{l}_s$
            Solve $\mathcal{L}_{\mathbf{ii}}\mathbf{y_i} = \mathbf{b_i}$
            $\mathbf{b}_{\mathcal{N}_\mathbf{i}} \leftarrow \tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}} = \mathbf{b}_{\mathcal{N}_\mathbf{i}} - \mathcal{L}_{\mathcal{N}_\mathbf{i},\mathbf{i}}\mathbf{y}_{\mathbf{i},k}$
        else                ▷ Structured solution above $\mathbf{l}_s$
            Solve $\mathcal{L}_{\mathbf{ii}}\mathbf{y_i} = \mathbf{b_i}$ as in Section 3.3.1
            $\mathbf{b}_{\mathcal{N}_\mathbf{i}} \leftarrow \tilde{\mathbf{b}}_{\mathcal{N}_\mathbf{i}}$ as in (3.30)   ▷ Fast update via the idea of reduced matrices
        end if
    end for
    $\mathbf{y} \leftarrow \mathbf{b}$                ▷ $\mathbf{y}$ (and also $\mathbf{x}$) in the memory space of $\mathbf{b}$
    for node $\mathbf{i}$ from root$(\mathcal{T})$ to 1 do                ▷ Backward substitution for (3.24)
        if $\mathbf{i}$ is at level $\mathbf{l} > \mathbf{l}_s$ of $\mathcal{T}$ then                ▷ Exact solution below $\mathbf{l}_s$
            Solve $\mathcal{L}_{\mathbf{ii}}^T\mathbf{x_i} = \mathbf{y_i}$
            for all nodes $\mathbf{j}$ such that $\mathbf{i} \in \mathcal{N}_\mathbf{j}$ do
                $\mathbf{y_j} \leftarrow \tilde{\mathbf{y}}_\mathbf{j} = \mathbf{y_j} - \mathcal{L}_{\mathbf{j},\mathbf{i}}\mathbf{y_i}$
            end for
        else                ▷ Structured solution above $\mathbf{l}_s$
            Solve $\mathcal{L}_{\mathbf{ii}}^T\mathbf{x_i} = \mathbf{y_i}$ as in Section 3.3.2
            for node $\mathbf{j}$ such that $\mathbf{i} \in \mathcal{N}_\mathbf{j}$ do          ▷ Fast update via reduced matrices
                $\mathbf{y}_{\mathbf{i},k} \leftarrow \tilde{\mathbf{y}}_{\mathbf{i},k}$ as in (3.32) with $k$ used in step $\mathbf{j}$
            end for
        end if
    end for
end procedure
```

TABLE 4.2

Factorization cost ξ_{fact}, solution cost ξ_{sol}, and storage σ_{mem} of the randomized structured algorithms applied to A discretized on a 2D $N \times N$ mesh, where $p \in \mathbb{N}$ and $\alpha > 0$.

r_l		$r = \max r_l$	ξ_{fact}	ξ_{sol}	σ_{mem}
$O(1)$		$O(1)$	$O(n)$	$O(n)$	$O(n)$
$O(\log^p N_l)$		$O(\log^p N)$			
$O(N_l^{1/p})$	$p \geq 3$	$O(N^{1/p})$	$O(n\log^{3/4} n)$	$O(n\log\log n)$	$O(n\log\log n)$
	$p = 2$	$O(N^{1/2})$	$O(n\log^2 n)$		
$O(\alpha^{l_{\max}-l}r_0)$	$\alpha \leq \sqrt[3]{2}$	$O(N^{1/3})$	$O(n\log^{3/4} n)$		
	$\alpha \leq \sqrt{2}$	$O(N^{1/2})$	$O(n\log^2 n)$		

and 3D using Lemma 4.1. For the case $r_l = O(N_l^{1/3})$ in 2D, we minimize ξ_{fact}:

$$\xi_{\text{fact}} = \underbrace{\sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 4^{\lfloor \mathbf{l}/2\rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/2\rfloor}}\right)^3\right)}_{\text{before } \mathbf{l}_s} + \underbrace{\sum_{\mathbf{l}=0}^{\mathbf{l}_s} 4^{\lfloor \mathbf{l}/2\rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/2\rfloor}}\right)^{5/3} \log\frac{N}{2^{\lfloor \mathbf{l}/2\rfloor}}\right)}_{\text{after } \mathbf{l}_s}$$

$$= O\left(\frac{N^3}{2^{\lfloor \mathbf{l}_s/2\rfloor}}\right) + O\left(N^{5/3} 2^{\lfloor \mathbf{l}_s/2\rfloor/3}\left(\lfloor \mathbf{l}_{\max}/2\rfloor - \lfloor \mathbf{l}/2\rfloor\right)\right).$$

Table 4.3

Factorization cost ξ_{fact}, solution cost ξ_{sol}, and storage σ_{mem} of the randomized structured algorithms applied to A discretized on a 3D $N \times N \times N$ mesh, where $p \in \mathbb{N}$ and $\alpha > 0$.

r_l		$r = \max r_l$	ξ_{fact}	ξ_{sol}	σ_{mem}
$O(1)$		$O(1)$	$O(n)$	$O(n)$	$O(n)$
$O(\log^p N_l)$		$O(\log^p N)$			
$O(N_l^{1/p})$	$p > 3$	$O(N^{1/p})$	$O(n \log n)$		
	$p = 3$	$O(N^{1/3})$	$O\left(n^{10/9} \log n\right)$		
	$p = 2$	$O(N^{1/2})$	$O(n^{4/3} \log n)$	$O(n \log n)$	$O(n \log n)$
$O(\alpha^{l_{\max}-l} r_0)$	$\alpha \leq \sqrt[3]{2}$	$O(N^{1/3})$	$O\left(n^{10/9} \log n\right)$	$O(n)$	$O(n)$
	$\alpha \leq \sqrt{2}$	$O(N^{1/2})$	$O(n^{4/3} \log n)$	$O(n \log n)$	$O(n \log n)$

This can be roughly optimized if we set the dense factorization cost before the switching level $\mathbf{l}_s$ to be equal to the structured factorization cost after $\mathbf{l}_s$. That yields

$$\lfloor \mathbf{l}_{\max}/2 \rfloor - \lfloor \mathbf{l}_s/2 \rfloor = O\left(\log \log N\right). \tag{4.1}$$

We then have

$$\begin{aligned}
\xi_{\text{fact}} &= O(N^2 \left(\lfloor \mathbf{l}_{\max}/2 \rfloor - \lfloor \mathbf{l}_s/2 \rfloor\right)^{3/4}) = O(n \log^{3/4} n), \\
\xi_{\text{sol}} &= \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 4^{\lfloor \mathbf{l}/2 \rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/2 \rfloor}} \right)^2 \right) + \sum_{\mathbf{l}=0}^{\mathbf{l}_s} 4^{\lfloor \mathbf{l}/2 \rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/2 \rfloor}} \right) \right) \\
&= O\left(N^2 \left(\lfloor \mathbf{l}_{\max}/2 \rfloor - \lfloor \mathbf{l}_s/2 \rfloor\right)\right) + O\left(N 2^{\lfloor \mathbf{l}_s/2 \rfloor}\right) = O\left(n \log \log n\right).
\end{aligned}$$

For the case $r_l = O(N_l^{1/3})$ in 3D, we minimize ξ_{sol} instead. This guarantees that ξ_{sol} is roughly $O\left(n\right)$ while ξ_{fact} is in the same order as otherwise when we minimize ξ_{fact}. That is,

$$\begin{aligned}
\xi_{\text{sol}} &= \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 8^{\lfloor \mathbf{l}/3 \rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/3 \rfloor}} \right)^4 \right) + \sum_{\mathbf{l}=0}^{\mathbf{l}_s} 8^{\lfloor \mathbf{l}/3 \rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/3 \rfloor}} \right)^2 \right) \\
&= N^4 \left(\frac{1}{2}\right)^{\lfloor \mathbf{l}_s/3 \rfloor} + O(N^2 2^{\lfloor \mathbf{l}_s/3 \rfloor}).
\end{aligned}$$

The optimality condition is

$$\lfloor \mathbf{l}_s/3 \rfloor = O\left(\log N\right). \tag{4.2}$$

We then have

$$\begin{aligned}
\xi_{\text{sol}} &= O\left(N^3\right) = O\left(n\right), \\
\xi_{\text{fact}} &= \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} 8^{\lfloor \mathbf{l}/3 \rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/3 \rfloor}} \right)^6 \right) + \sum_{\mathbf{l}=0}^{\mathbf{l}_s} 8^{\lfloor \mathbf{l}/3 \rfloor} O\left(\left(\frac{N}{2^{\lfloor \mathbf{l}/3 \rfloor}} \right)^{10/3} \log \left(\frac{N}{2^{\lfloor \mathbf{l}/2 \rfloor}} \right)^2 \right) \\
&= N^6 \sum_{\mathbf{l}=\mathbf{l}_s+1}^{\mathbf{l}_{\max}} \left(\frac{1}{8}\right)^l + N^{10/3} \sum_{\mathbf{l}=0}^{\mathbf{l}_s} \left(\frac{1}{2^{1/3}}\right)^l \left(\lfloor \mathbf{l}_{\max}/2 \rfloor - \lfloor \mathbf{l}_s/2 \rfloor\right) = O\left(n^{10/9} \log n\right).
\end{aligned}$$

□

These results are generally better than those in [39]. The optimality conditions (4.1) and (4.2) show how to choose the switching level $\mathbf{l}_s$. For simplicity, we generally choose $\mathbf{l}_{\max} - \mathbf{l}_s$ to be roughly constant or slowly increasing.

REMARK 4.1. Notice that the solution costs and memory sizes in all the cases in Theorem 4.2 are roughly linear in n. This makes the algorithms especially attractive for problems with multiple right-hand sides and for preconditioning.

REMARK 4.2. The rank patterns are also useful in deciding the values of the sampling sizes $r_{\mathbf{l}}$ in Section 3.4. That is, when such a pattern is known, then we can roughly decide $r_{\mathbf{l}-1}$ from $r_{\mathbf{l}}$ based on their corresponding frontal matrix sizes. Although not theoretically justified, this works well in our numerical tests below.

5. Numerical experiments.

In this section, we test our randomized algorithms with a Helmholtz equation and some more general problems. For convenience, the following notation is used:

- `NEW`: The new randomized algorithms, as specified below.
- `MF`: The classical multifrontal method, which uses the code for `NEW` by setting $\mathbf{l}_s = 0$. (In [37], a similar version is compared with the serial SuperLU solver [11] and has comparable performance.)
- $e_2 = \frac{\|x-\tilde{x}\|_2}{\|x\|_2}$ and $\gamma_2 = \frac{\|A\tilde{x}-b\|_2}{\|b\|_2}$: Relative error and relative residual, respectively, where $\tilde{x}$ is an approximation to the solution x of $Ax = b$.

EXAMPLE 1. We solve sparse linear systems $Ax = b$ arising from the discretization of a Helmholtz equation:

$$[-\Delta - \omega^2 c(\mathbf{x})^{-2}]\mathbf{u} = \mathbf{f}, \tag{5.1}$$

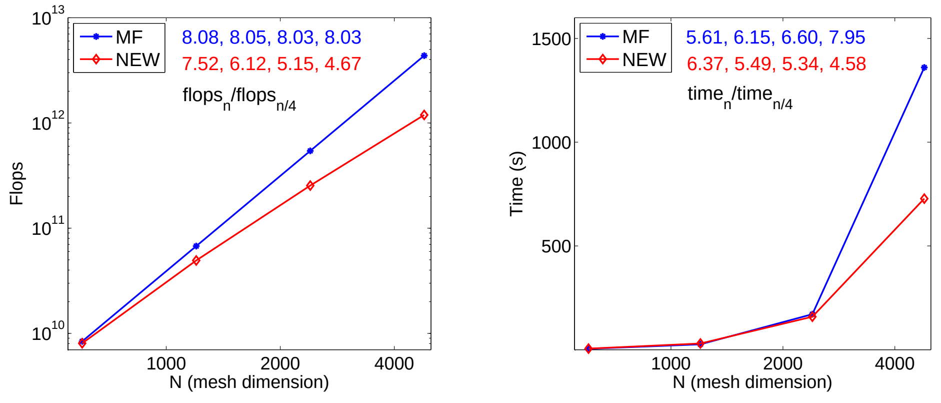
where ω, $c(x)$, and $\mathbf{f}$ are the angular frequency, the velocity field, and the forcing term, respectively. Here, $\omega = 5$Hz. The discretized matrix is indefinite. That is, we are solving an elliptic problem with an indefinite perturbation. By fixing ω and by varying the mesh size, we can conveniently demonstrate the complexity of `NEW` (which is expected to be about $O(N)$ in 2D). Such kind of tests are also used, say, in [35].

The low-rank property of this matrix is studied in [14, 35]. In our factorizations, we choose A with size $n = N^2$ from $N \times N$ meshes, where N varies from 300 to 4800. In `NEW` with Algorithm 4, we set the $r + \mu = 120$ and a relative tolerance $\tau = 10^{-5}$ in the SPRR factorizations used in the intermediate compression steps. See Table 5.1 for the factorization performance. In `NEW`, we choose $\mathbf{l}_{\max} - \mathbf{l}_s = 11$, 11, 11, 10, and 10 for the N values from 300 to 4800. The results are also plotted in Figure 5.1, together with the ratios of the flop counts and timings. We notice that, when n quadruples and gets large, the flop and timing ratios for `MF` and `NEW` approach 8 and 4, respectively, which is consistent with the flop counts $O(n^{1.5})$ for `MF` and $O(n)$ for `NEW`.

Table 5.2 shows the memory of the methods and the ratios when n increases. The solution costs and accuracies (with single precisions) are given in Tables 5.3 and 5.4, respectively. The memory and solution costs of `NEW` scale nearly linearly and better than `MF`. The difference in the solution costs increases when n increases. For example, when $n = 1200^2$, 2400^2, and 4800^2, the ratios of the solution flops of `MF` over `NEW` are 1.21, 1.42, and 1.56, respectively. (We expect this advantage to be even more significant when n is large in our future tests, specially in 3D.) After few steps of iterative refinement, `NEW` reaches similar accuracies as `MF`. The timing for the iterative refinement is roughly the solution time in Table 5.3 times the number of steps in Table

Table 5.1
Example 1: Factorization flops and timing (in seconds) of `NEW` *and* `MF` *for (5.1) with various matrix sizes n.*

n $(=N^2)$		300^2	600^2	1200^2	2400^2	4800^2
$\mathbf{l}_{\max}$		13	15	17	19	21
Flops	`MF`	$1.04E9$	$8.40E9$	$6.76E10$	$5.43E11$	$4.36E12$
	`NEW`	$1.07E9$	$8.05E9$	$4.93E10$	$2.54E11$	$1.19E12$
Time (s)	`MF`	$7.51E-1$	$4.21E0$	$2.59E1$	$1.71E2$	$1.35E3$
	`NEW`	$8.53E-1$	$5.43E0$	$2.98E1$	$1.59E2$	$7.28E2$



(i) Flop counts ξ_{fact} (ii) Timings

Fig. 5.1. *Example 1: Factorization flops and timing (in seconds) of* `NEW` *and* `MF` *for (5.1) with various n, where the scaling factors $\frac{\text{flops}_n}{\text{flops}_{n/4}}$ and $\frac{\text{time}_n}{\text{time}_{n/4}}$ for each method are marked.*

5.4. (This makes the overall `NEW` solution cost larger here, but the total cost is still much lower since the solution is much faster than the factorization.)

Table 5.2
Example 1: Memory (number of nonzeros in the factors) of `NEW` *and* `MF` *for (5.1) with various n.*

n $(=N^2)$		300^2	600^2	1200^2	2400^2	4800^2
$\mathbf{l}_{\max}$		13	15	17	19	21
Memory	`MF`	$7.97E6$	$3.73E7$	$1.71E8$	$7.73E8$	$3.45E9$
	`NEW`	$7.72E6$	$3.35E7$	$1.40E8$	$5.39E8$	$2.20E9$
$\frac{\text{Memory}_n}{\text{Memory}_{n/4}}$	`MF`	/	4.68	4.58	4.52	4.46
	`NEW`	/	4.34	4.18	3.85	4.08

In addition, unlike the method in [39] (denoted `MFHSS`), `NEW` avoids storing large dense frontal matrices. We thus compare their memory requirements for constructing an HSS approximation to the last frontal matrix. See Figure 5.2. Such memory includes the matrix-vector products in `NEW` and the dense frontal matrices in `MFHSS`. It is clear that the memory scales linearly for `NEW`, while quadratically for `MFHSS`.

Remark 5.1. We only compare the storage of the frontal matrices in `NEW` and `MFHSS`. The overall performance is not compared, since the matrices are not large enough for `NEW` to be significantly faster, and `MFHSS` generally uses tolerances to control

Table 5.3
Example 1: Solution flops and timing (in seconds) of `NEW` *and* `MF` *for (5.1) with various n (the timing for* `MF` *with $n = 4800^2$ is left blank since it is too long to be meaningful).*

n $(= N^2)$		300^2	600^2	1200^2	2400^2	4800^2
$\mathbf{l}_{\max}$		13	15	17	19	21
Flops	`MF`	$1.58E7$	$7.39E7$	$3.40E8$	$1.53E9$	$6.85E9$
	`NEW`	$1.53E7$	$6.68E7$	$2.80E8$	$1.08E9$	$4.38E9$
Time (s)	`MF`	$9.80E-2$	$4.06E-1$	$1.70E0$	$6.99E0$	
	`NEW`	$9.80E-2$	$4.08E-1$	$1.70E0$	$6.69E0$	$2.75E1$

Table 5.4
Example 1: Solution accuracies of `NEW` *and* `MF` *for (5.1) with various n, and few steps of iterative refinements are used in* `NEW`*.*

n $(= N^2)$			300^2	600^2	1200^2	2400^2	4800^2
$\mathbf{l}_{\max}$			13	15	17	19	21
`MF`		γ_2	$1.21E-7$	$1.26E-7$	$1.34E-7$	$1.53E-7$	$1.90E-7$
		e_2	$1.79E-5$	$2.62E-5$	$3.92E-5$	$5.98E-5$	$1.14E-4$
`NEW`	Original	γ_2	$1.47E-7$	$2.38E-7$	$2.82E-7$	$2.07E-6$	$6.39E-6$
		e_2	$2.35E-5$	$8.65E-5$	$2.06E-4$	$4.32E-3$	$1.77E-2$
	After iterative refinement	Steps	1	1	1	2	3
		γ_2	$5.65E-8$	$5.60E-8$	$5.57E-8$	$5.82E-8$	$8.62E-8$
		e_2	$1.13E-5$	$9.7E-6$	$1.10E-5$	$2.24E-5$	$1.40E-4$

the accuracy. `NEW` is predicted to be much faster when n is large. A parallel code is expected to be developed, so as to test larger matrices.

Example 2. Next, we show some broader applications by considering some 2D, 3D, and more general matrices, as listed in Table 5.5. They include matrices from the University of Florida Sparse Matrix Collection [10] and from some finite element methods with iterative mesh refinements [9], as well as a random sparse matrix.

In particular, we illustrate the potential of `NEW` for obtaining approximate solutions with modest accuracy, using the adaptive Algorithm 5. From the switching level $\mathbf{l} = \mathbf{l_s}$ to the root level $\mathbf{l} = 0$ of $\mathcal{T}$, we use sampling sizes $r_\mathbf{l}$, which are decided adaptively based on the corresponding frontal matrix sizes. The relative tolerance in the SPRR compression is about $\tau = 2 \times 10^{-3}$. Denote

$$[r_{\min}, r_{\max}] = [\min_{\mathbf{l}=\mathbf{l_s}}^{0} r_\mathbf{l},\ \max_{\mathbf{l}=\mathbf{l_s}}^{0} r_\mathbf{l}].$$

We first show the efficiency improvement with `NEW` over `MF`. See Table 5.6. The pairs $[r_{\min}, r_{\max}]$ are also shown. Although the matrices are from different background and the sizes are relatively small, `NEW` improves the costs by factors of 2.8, 1.9, 1.5, 1.4, 2.1, 1.9, 1.8, 7.4 for the matrices from the top to the bottom in Table 5.6. (The timing is not shown since the matrices are relatively small and `NEW` has no significant advantage in the timing over `MF`. See the previous example for the timing for larger matrices.)

Specifically, for the SPD matrix `ecology2` in Table 5.5, we show the effectiveness of `NEW` as a preconditioner in the preconditioned conjugate gradient method (`PCG`). For $\mathbf{l} = \mathbf{l_s}, \mathbf{l_s}-1, \ldots, 0$, we use the sampling sizes $r_\mathbf{l} = 40, 46, 54, 62, 62, 61, 47$. The largest frontal matrix size is $1,851$. The cost to get the preconditioner is $1.32E10$ flops, as

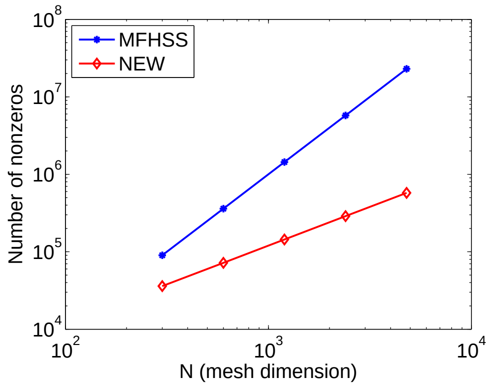


FIG. 5.2. *Example 1: Memory requirements of `NEW` for constructing an HSS approximation to the last frontal matrix, as compared with that in the method in [39].*

TABLE 5.5
Example 2: 2D, 3D, and more general test matrices, where `2cubes_sphere`, `apache2`, `Dubcova3`, `ecology2`, and `parabolic_fem` are from the University of Florida Sparse Matrix Collection [10], `crack` and `jumpMG` are from a toolbox iFEM [9], and nnz denotes the number of nonzeros of the matrix.

Matrix	n	nnz	Description
`2cubes_sphere`	$101,492$	$1,647,264$	3D electromagnetic diffusion equation on two cubes in a sphere
`apache2`	$715,176$	$4,817,870$	3D SPD matrix from "APACHE small"
`crack`	$280,307$	$1,410,269$	2D Poisson equation in a crack domain
`Dubcova3`	$146,689$	$3,636,643$	Matrix collected by Dubcova
`ecology2`	10^6-1	$4,995,991$	Landscape ecology problem using electrical network theory to model animal movement and gene flow
`jumpMG`	$114,211$	$1,303,519$	3D PDE with jump coefficients in an L-shaped partial cubic domain
`parabolic_fem`	$525,825$	$3,674,625$	Parabolic FEM problem for a diffusion-convection reaction in computational fluid dynamics
`random`	$216,000$	$3,154,318$	Ill-conditioned random matrix generated by Matlab `sprandsym` with the nonzero pattern following a 3D tetrahedral grid

compared with the exact factorization cost $2.59E10$ and a randomized factorization cost $1.46E10$ with a uniform sampling size $r_1 \equiv 62$.

`PCG` with `NEW` as the preconditioner (`PCG-NEW`) converges significantly faster and costs much less than the regular conjugate gradient method (`CG`) and `PCG` with a block diagonal preconditioner (`PCG-bdiag`). See Figure 5.3. The total iterative solution costs are shown in Table 5.7. For simplicity, the test is done in Matlab, and `CG`, `PCG-bdiag`, and `PCG-NEW` take $4.29E2$, $6.83E3$, and $1.63E3$ seconds, respectively. `CG` is faster due to its simplicity and fast data access in Matlab and the relatively small matrix size. We expect `PCG-NEW` to be faster in timing for larger tests, say, in Fortran,

TABLE 5.6
Example 2: Costs and accuracies of `NEW` *and* `MF` *for the matrices in Table 5.5.*

Matrix	$l_{\max}$	MF	NEW		
		Flops	Flops	$[r_{\min}, r_{\max}]$	γ_2
`2cubes_sphere`	14	$6.34E10$	$2.24E10$	$[45, 62]$	$2.03E-3$
`apache2`	16	$2.82E11$	$1.50E11$	$[100, 140]$	$7.10E-2$
`crack`	16	$2.84E9$	$1.91E9$	$[20, 30]$	$6.08E-3$
`Dubcova3`	16	$1.89E9$	$1.34E9$	$[25, 32]$	$8.21E-3$
`ecology2`	16	$2.59E10$	$1.24E10$	$[35, 62]$	$7.82E-5$
`jumpMG`	14	$2.07E10$	$1.08E10$	$[30, 63]$	$5.09E-3$
`parabolic_fem`	14	$8.24E9$	$4.57E9$	$[25, 48]$	$9.19E-4$
`random`	15	$3.04E11$	$4.09E10$	$[30, 54]$	$2.61E-3$

and for severely ill-conditioned problems (such as a linear elasticity problem near the incompressible limit in [37]).

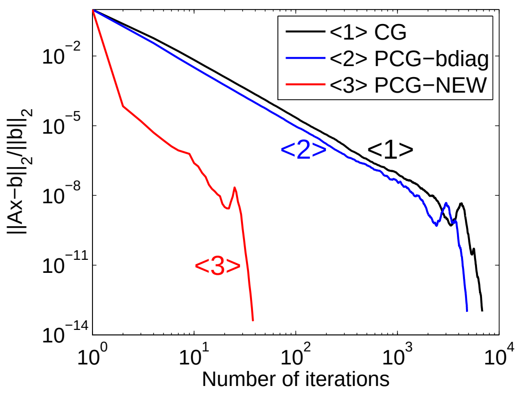


FIG. 5.3. *Example 2: Convergence of* `CG`*,* `PCG` *with a block diagonal preconditioner (*`PCG-bdiag`*), and* `PCG` *with* `NEW` *as the preconditioner (*`PCG-NEW`*) for the matrix* `ecology2` *in Table 5.5, where a diagonal block size* 20 *is used in* `PCG-bdiag`*.*

TABLE 5.7
Example 2: Convergence results for Figure 5.3.

Method	Number of iterations	Total cost (flops)	γ_2
`CG`	$6,821$	$1.43E11$	$9.98E-14$
`CG-bdiag`	$4,835$	$2.90E11$	$9.66E-14$
`CG-NEW`	38	$2.20E10$	$3.86E-14$

6. Conclusions.

This work brings randomized structured techniques into the field of sparse solutions by presenting randomized direct factorizations for sparse matrices. The methods convert complicated HSS operations in some structured factorizations into the operations of skinny matrix-vector products. A series of strategies are used to enhance the efficiency. Additional structures and adaptive schemes are introduced into structured multifrontal methods. The methods have significant advantages over classical factorizations and similar structured ones. They can also work for problems whose intermediate matrices in the factorizations do not have low-rank off-diagonal blocks, especially 3D ones and more general sparse problems. Such approximate factorizations with modest accuracy are especially useful for problems such as seismic imaging [36] and preconditioning.

Efficient implementations, especially parallel ones, are expected to be done for large matrices. Our methods also have the potential to be generalized to matrix-free sparse direct solvers, since various essential operations are already performed on matrix-vector products. We seek to use sparse matrix-vector multiplications (and possibly the stencil information) to obtain data-sparse factors for sparse matrices. A fundamental idea is to construct HSS approximations to submatrices of A based on matrix-vector products, and then perform the factorization as in the method here. The matrix-free randomized HSS construction algorithm in [24] do not need to compute selected entries of the frontal matrices, and can be applied to at least the leaf levels. The work is in progress.

Acknowledgments. The author is very grateful to the two anonymous referees for their useful suggestions, and to Maarten V. de Hoop and Xiaoye S. Li for some discussions. The author would also like to thank Shen Wang for helping with some tests. The research of the author was supported in part by NSF grants DMS-1115572 and CHE-0957024.

REFERENCES

[1] 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.

[2] T. Bella, Y. Eidelman, I. Gohberg, and V. Olshevsky, *Computations with quasiseparable polynomials and matrices*, (409) 2008, Theoret. Comput. Sci., pp. 158–179.

[3] S. Börm, *Efficient Numerical Methods for Non-local Operators*, European Mathematical Society, 2010.

[4] S. Börm and W. Hackbusch, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[5] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.

[6] S. Chandrasekaran, P. Dewilde, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[7] 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.

[8] S. Chandrasekaran, P. Dewilde, M. Gu, and N. Somasunderam, *On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2261–2290.

[9] L. Chen, *iFEM: an integrated finite element methods package in MATLAB*, Technical Report, University of California at Irvine, 2009, http://math.uci.edu/~chenlong/Papers/iFEMpaper.pdf.

[10] T. A. Davis and Y. Hu, *University of Florida Sparse Matrix Collection*, http://www.cise.ufl.edu/research/sparse/matrices/index.html.

[11] J. W. Demmel, J. R. Gilbert, and X. S. Li, *SuperLU Users' Guide*, http://crd.lbl.gov/~xiaoye/SuperLU/, 2003.

[12] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Bans. Math. Software, 9 (1983), pp. 302–325.

[13] Y. Eidelman and I. Gohberg, *On a new class of structured matrices*, Integral Equations Operator Theory, 34 (1999) pp. 293–324.

[14] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: Hierarchical matrix representation*, Pure Appl. Math., LXIV (2011), pp. 0697–0735.

[15] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[16] L. Grasedyck, R. Kriemann, and S. Le Borne, *Parallel black box domain decomposition based $\mathcal{H}$-LU preconditioning*, Technical Report 115, Max Planck Institute for Mathematics in the Sciences, Leipzig, 2005.

[17] L. Grasedyck, R. Kriemann, and S. Le Borne, *Domain-decomposition based $\mathcal{H}$-LU preconditioners*, in Domain Decomposition Methods in Science and Engineering XVI, O.B.Widlund and D.E.Keyes (eds.), Springer LNCSE, 55 (2006), pp. 661–668.

[18] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Phys., 73 (1987), pp. 325–348.

[19] 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.

[20] W. Hackbusch, *A Sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[21] N. Halko, P.G. Martinsson, and J. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), pp. 217–288.

[22] A. J. Hoffman, M. S. Martin, and D. J. Rose, *Complexity bounds for regular finite difference and finite element grids*, SIAM J. Numer. Anal., 10 (1973), pp. 364–369.

[23] E. Liberty, F. Woolfe, P. G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.

[24] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.

[25] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.

[26] W. Lyons, *Fast Algorithms with Applications to PDEs*, PhD thesis, University of California, Santa Barbara, June. 2005.

[27] P. G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 3, 2009, pp. 316–330

[28] P. G. Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM. J. Matrix Anal. Appl., 32 (2011), pp. 1251–1274.

[29] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the decomposition of matrices*, Appl. Comput. Harmon. Anal., 30 (2011), pp. 47–68.

[30] V. Y. Pan and G. Qian, *Randomized preprocessing of homogeneous linear systems of equations*, Linear Algebra Appl., 432 (2010), pp. 3272–3318.

[31] S. V. Parter, *The use of linear graphs in gaussian elimination*, SIAM Rev., 3 (1961), pp. 119–130.

[32] P. G. Schmitz and L. Ying, *A fast direct solver for elliptic problems on general meshes in 2D*, J. Comput. Phys., 231 (2012), pp. 1314–1338.

[33] R. P. Tewarson, *On the product form of inverses of sparse matrices and graph theory*, SIAM Rev., 9 (1967), pp. 91–99.

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

[35] S. Wang, M. V. de Hoop, and J. Xia, *Acoustic inverse scattering via Helmholtz operator factorization and optimization*, J. Comput. Phys., 229 (2010), pp. 8445–8462.

[36] S. Wang, M. V. de Hoop, and J. Xia, *On 3D modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver*, Geophys. Prospect., 59 (2011), pp. 857–873.

[37] J. Xia, *Robust and efficient multifrontal solver for large discretized PDEs*, High-Perform. Sci. Comput., M. W. Berry et al. (eds.), Springer (2012), pp. 199–217.

[38] J. Xia, *On the complexity of some hierarchical structured matrices*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.

[39] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., revised, 2012, http://www.math.purdue.edu/˜xiaj/work/mfhss.pdf.

[40] 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.

[41] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.

[42] J. Xia, Y. Xi, and M. Gu, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 837–858.