SIAM J. SCI. COMPUT.
Vol. 38, No. 4, pp. A1959–A1986

COMPRESSING RANK-STRUCTURED MATRICES VIA RANDOMIZED SAMPLING*

PER-GUNNAR MARTINSSON†

Abstract. Randomized sampling has recently been proven a highly efficient technique for computing approximate factorizations of matrices that have low numerical rank. This paper describes an extension of such techniques to a wider class of matrices that are not themselves rank-deficient but have off-diagonal blocks that are—specifically, the classes of so-called *hierarchically off-diagonal low rank* (HODLR) matrices and *hierarchically block separable* (HBS) matrices (a.k.a. *hierarchically semiseparable* (HSS) matrices). Such matrices arise frequently in numerical analysis and signal processing, in particular in the construction of fast methods for solving differential and integral equations numerically. These structures admit algebraic operations (matrix-vector multiplications, matrix factorizations, matrix inversion, etc.) to be performed very rapidly, but only once a data-sparse representation of the matrix has been constructed. This paper demonstrates that if an $N \times N$ matrix, and its transpose, can be applied to a vector in $O(N)$ time, and if the ranks of the off-diagonal blocks are bounded by an integer k, then the cost for constructing an HODLR representation is $O(k^2\, N\, (\log N)^2)$, and the cost for constructing an HBS representation is $O(k^2\, N\, \log N)$ (assuming that the matrix is compressible in the respective format). The point is that when legacy codes (based on, e.g., the fast multipole method) can be used for the fast matrix-vector multiply, the proposed algorithm can be used to obtain the data-sparse representation of the matrix, and then well-established techniques for HODLR/HBS matrices can be used to invert or factor the matrix. The proposed scheme is also useful in simplifying the implementation of certain operations on rank-structured matrices such as matrix-matrix multiplication, low rank update, and addition.

Key words. randomized approximation of matrices, rank-structured matrices, HODLR matrix, hierarchically block separable matrix, hierarchically semiseparable matrix, fast direct solver

AMS subject classifications. 65N22, 65N38, 15A23, 15A52

DOI. 10.1137/15M1016679

1. Introduction.

A ubiquitous task in computational science is to rapidly perform linear algebraic operations involving very large matrices. Such operations typically exploit special "structure" in the matrix since the costs of standard techniques tend to scale prohibitively quickly with matrix size; for a general $N \times N$ matrix, it costs $O(N^2)$ operations to perform a matrix-vector multiplication, $O(N^3)$ operations to perform Gaussian elimination or to invert the matrix, etc. These estimates can be greatly improved upon, often to $O(N)$, for matrices that are *rank-structured*, which means that their off-diagonal blocks have low rank (to some specified tolerance). Several different formats for rank-structured matrices have been proposed in the literature. In this paper, we rely on the so-called *hierarchically off-diagonal low rank* (HODLR) format. This name was minted in [1], but this class of matrices has a long history. It is a special case of the $\mathcal{H}$-matrix format introduced by Hackbusch and coworkers [18, 3], and it was used explicitly in [26, sec. 4]. The HODLR format is very easy to describe and easy to use, but it can lead to less than optimal performance due to the fact that the basis matrices used to represent large blocks are stored explicitly, leading to an $O(k\, N\, \log N)$ storage requirement for an HODLR matrix whose

*Submitted to the journal's Methods and Algorithms for Scientific Computing section April 13, 2015; accepted for publication (in revised form) April 5, 2016; published electronically July 6, 2016. The research reported was supported by DARPA under contract N66001-13-1-4050 and by the NSF under contract DMS-1407340.

http://www.siam.org/journals/sisc/38-4/M101667.html

†Dept. of Applied Math., Univ. of Colorado Boulder, Boulder, CO 80309 (martinss@colorado.edu).

off-diagonal blocks have rank at most k. To attain *linear* storage requirements and arithmetic operations, one can switch to a format that expresses all basis matrices *hierarchically*; in other words, the basis matrices used on one level are expressed implicitly in terms of the basis matrices on the next finer level. We sometimes say that the basis matrices are *nested*. To be precise, we use the *hierarchically block separable* (HBS) format that was described in [29, 15]. This format is closely related to the *hierarchically semiseparable* (HSS) [7, 34] format and is also related to the $\mathcal{H}^2$-matrix format [20, 4].

The most straightforward technique for computing a data-sparse representation of a rank-structured $N \times N$ matrix $\mathbf{A}$ is to explicitly form all matrix elements and then compress the off-diagonal blocks using, e.g., the SVD. This approach can be executed stably [35, 19], but it is often prohibitively expensive, with an $O(k\,N^2)$ asymptotic cost, where k is the rank of the off-diagonal blocks. Fortunately, there exist for specific applications much faster methods for constructing rank-structured representations. When the matrix $\mathbf{A}$ approximates a boundary integral operator in the plane, the technique of [29] computes a representation in $O(k^2\,N)$ time by exploiting representation results from potential theory. In other environments, it is possible to use known regularity properties of the off-diagonal blocks in conjunction with interpolation techniques to obtain a rough initial factorization and then recompress these to obtain factorizations with close to optimal ranks [4, 30]. A particularly popular version of the "regularity + recompression" method is the so-called *adaptive cross approximation* technique, which was initially proposed for $\mathcal{H}$-matrices [2, 5, 22] but has recently been modified to obtain a representation of a matrix in a format similar to the HSS [12].

This paper describes a fast and simple randomized technique for computing a data-sparse representation of a rank-structured matrix $\mathbf{A}$ which can rapidly be applied to a vector. The existence of such a technique means that the advantages of the HODLR and HBS formats—fast inversion and factorization algorithms, in particular—become available for any matrix that can be applied to a vector using techniques at hand. (For instance, $\mathbf{A}$ could be the product of two data-sparse matrices, it could be an operator that can be applied using the fast multipole method (FMM), etc.) In order to describe the cost of the algorithm precisely, we introduce some notation: We let $\mathbf{A}$ be an $N \times N$ matrix whose off-diagonal blocks have maximal rank k, we let T_{mult} denote the time required to perform a matrix-vector multiplication $\mathbf{x} \mapsto \mathbf{A}\,\mathbf{x}$ or $\mathbf{x} \mapsto \mathbf{A}^*\,\mathbf{x}$, we let T_{rand} denote the cost of constructing a pseudorandom number from a normalized Gaussian distribution, and we let T_{flop} denote the cost of a floating point operation. The computational cost T_{total} of the algorithm for the HBS format then satisfies

$$T_{\text{total}} \sim T_{\text{mult}} \times k\,\log(N) + T_{\text{rand}} \times k\,N\,\log(N) + T_{\text{flop}} \times k^2\,N\,\log(N). \tag{1}$$

In particular, if T_{mult} is $O(N)$, then the method presented here has overall complexity $O(k^2\,N\,\log(N))$. For the HODLR format, an additional factor of $\log N$ arises; cf. (13).

The work presented is directly inspired by [27] (which is based on a 2008 preprint [25]), which described a similar algorithm with $O(k^2\,N)$ complexity for the compression of an HBS matrix. This is better by a factor of $\log(N)$ compared to the present work, but the algorithm of [27] has a limitation in that it requires the ability to evaluate $O(k\,N)$ entries of the matrix to be compressed. This algorithm was refined by Xia [33] and applied to the task of accelerating a "nested dissection" direct solver [13, 11] for elliptic PDEs to $O(N)$ complexity. In 2011, Lin, Lu, and Ying presented an alternative algorithm [24] that interacts with the matrix *only* via matrix-vector

multiplication, which makes the randomized compression idea much more broadly applicable than the algorithm in [27]. However, this came at the cost of requiring $O(k \log(N))$ matrix-vector multiplies (just like the present work). The algorithm proposed here is an evolution of [24] that does away with a step of least-square fitting in the randomized approximation. Moreover, we here present a new strategy for enforcing the "nestedness" of the basis matrices that is required in the HBS format, and unlike [24], we allow the matrix to be nonsymmetric (which means that we must require that the map $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ also be available).

Remark 1. The technique proposed utilizes a method for computing approximate low rank factorizations of matrices that is based on randomized sampling [31, 32, 21]. As a consequence, there is in principle a nonzero risk that any given realization of the algorithm fails to meet the requested tolerance. This risk can be controlled by the user via the choice of a tuning parameter and can at low cost be set to 10^{-10} or less; cf. section 2.5.

2. Preliminaries.

2.1. Notation. Throughout the paper, we measure vectors in $\mathbb{R}^n$ using their Euclidean norm. The default norm for matrices will be the corresponding operator norm $\|\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|$, although we will sometimes also use the Frobenius norm $\|\mathbf{A}\|_{\text{Fro}} = (\sum_{i,j} |\mathbf{A}(i,j)|^2)^{1/2}$. We use the notation of Golub and Van Loan [16] to specify submatrices: If $\mathbf{B}$ is an $m \times n$ matrix, and $I = [i_1, i_2, \ldots, i_k]$ and $J = [j_1, j_2, \ldots, j_\ell]$ are index vectors, then $\mathbf{B}(I, J)$ denotes the $k \times \ell$ matrix

$$\mathbf{B}(I,J) = \begin{bmatrix} \mathbf{B}(i_1,j_1) & \mathbf{B}(i_1,j_2) & \cdots & \mathbf{B}(i_1,j_\ell) \\ \mathbf{B}(i_2,j_1) & \mathbf{B}(i_2,j_2) & \cdots & \mathbf{B}(i_2,j_\ell) \\ \vdots & \vdots & & \vdots \\ \mathbf{B}(i_k,j_1) & \mathbf{B}(i_k,j_2) & \cdots & \mathbf{B}(i_k,j_\ell) \end{bmatrix}.$$

We let $\mathbf{B}(I,:)$ denote the matrix $\mathbf{B}(I,[1,\,2,\ldots,n])$ and define $\mathbf{B}(:,J)$ analogously. The transpose of $\mathbf{B}$ is denoted $\mathbf{B}^*$, and we say that a matrix $\mathbf{U}$ is *orthonormal* if its columns are orthonormal, so that $\mathbf{U}^*\mathbf{U} = \mathbf{I}$.

2.2. The QR factorization. Any $m \times n$ matrix $\mathbf{A}$ admits a *QR factorization* of the form

$$\underset{m\times n}{\mathbf{A}} \quad \underset{n\times n}{\mathbf{P}} \quad = \quad \underset{m\times r}{\mathbf{Q}} \quad \underset{r\times n}{\mathbf{R}}, \tag{2}$$

where $r = \min(m,n)$, $\mathbf{Q}$ is orthonormal, $\mathbf{R}$ is upper-triangular, and $\mathbf{P}$ is a permutation matrix. The permutation matrix $\mathbf{P}$ can more efficiently be represented via a vector $J \in \mathbb{Z}_+^n$ of column indices such that $\mathbf{P} = \mathbf{I}(:,J)$, where $\mathbf{I}$ is the $n \times n$ identity matrix. As a result, the factorization (2) can be written as

$$\underset{m\times n}{\mathbf{A}(:\,,J)} \quad = \quad \underset{m\times r}{\mathbf{Q}} \quad \underset{r\times n}{\mathbf{R}}.$$

The QR factorization is often built incrementally via a greedy algorithm such as column pivoted Gram–Schmidt. This allows one to stop after the first k terms have been computed (and not complete the full factorization process) to obtain a "partial QR factorization of $\mathbf{A}$,"

$$\underset{m\times n}{\mathbf{A}(:\,,J)} \quad \approx \quad \underset{m\times k}{\mathbf{Q}_k} \quad \underset{k\times n}{\mathbf{R}_k}. \tag{3}$$

2.3. The singular value decomposition (SVD). Let $\mathbf{A}$ denote an $m \times n$ matrix, and set $r = \min(m, n)$. Then $\mathbf{A}$ admits a factorization

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times r}{\mathbf{U}} \; \underset{r \times r}{\mathbf{\Sigma}} \; \underset{r \times n}{\mathbf{V}^*}, \tag{4}$$

where the matrices $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{\Sigma}$ is diagonal. We let $\{\mathbf{u}_i\}_{i=1}^r$ and $\{\mathbf{v}_i\}_{i=1}^r$ denote the columns of $\mathbf{U}$ and $\mathbf{V}$, respectively. These vectors are the left and right singular vectors of $\mathbf{A}$. The diagonal elements $\{\sigma_j\}_{j=1}^r$ of $\mathbf{\Sigma}$ are the singular values of $\mathbf{A}$. We order these so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$. We let $\mathbf{A}_k$ denote the truncation of the SVD to its first k terms, so that $\mathbf{A}_k = \sum_{j=1}^k \sigma_j \, \mathbf{u}_j \, \mathbf{v}_j^*$. It is easily verified that

$$\|\mathbf{A} - \mathbf{A}_k\| = \sigma_{k+1} \qquad \text{and that} \qquad \|\mathbf{A} - \mathbf{A}_k\|_{\text{Fro}} = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2}.$$

Moreover, the Eckart–Young theorem states that these errors are the smallest possible errors that can be incurred when approximating $\mathbf{A}$ by a matrix of rank k.

2.4. The interpolative decomposition (ID). Any $m \times n$ matrix $\mathbf{A}$ of rank k admits the factorization

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{A}(:, J)} \; \underset{k \times n}{\mathbf{X}},$$

where J is a vector of indices marking k of the columns of $\mathbf{A}$, and the $k \times n$ matrix $\mathbf{X}$ has the $k \times k$ identity matrix as a submatrix and has the property that all its entries are bounded by 1 in magnitude. In other words, the interpolative decomposition (ID) picks k columns of $\mathbf{A}$ as a basis for the column space of $\mathbf{A}$ and expresses the remaining columns in terms of the chosen ones. The ID can be viewed as a modification to the so-called *rank-revealing QR factorization* [6]. It can be computed in a stable and accurate manner using the techniques of [17], as described in [8]. (Practical algorithms for computing the ID produce a matrix $\mathbf{X}$ whose elements may slightly exceed 1 in magnitude.)

2.5. Randomized compression. Let $\mathbf{A}$ be a given $m \times n$ matrix that can accurately be approximated by a matrix of rank k, and suppose that we seek to determine a matrix $\mathbf{Q}$ with orthonormal columns (as few as possible) such that $\|\mathbf{A} - \mathbf{Q}\,\mathbf{Q}^*\,\mathbf{A}\|$ is small. In other words, we seek a matrix $\mathbf{Q}$ whose columns form an approximate orthornomal basis (ON-basis) for the column space of $\mathbf{A}$. This task can efficiently be solved via the following randomized procedure:

(1) Pick a small integer p representing how much "oversampling" is done. ($p = 10$ is often good.)
(2) Form an $n \times (k + p)$ matrix $\mathbf{\Omega}$ whose entries are independent and identically distributed (i.i.d.) normalized Gaussian random numbers.
(3) Form the "sample matrix" $\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega}$ of size $m \times (k + p)$.
(4) Construct an $m \times (k + p)$ matrix $\mathbf{Q}$ whose columns form an ON-basis for the columns of $\mathbf{Y}$.

Note that each column of the "sample matrix" $\mathbf{Y}$ is a random linear combination of the columns of $\mathbf{A}$. We would therefore expect the algorithm described to have a high probability of producing an accurate result when p is a large number. It is perhaps less obvious that this probability depends only on p (not on m or n, or any other

properties of $\mathbf{A}$) and that it approaches 1 extremely rapidly as p increases. In fact, one can show that the basis $\mathbf{Q}$ determined by the scheme above satisfies

$$\|\mathbf{A} - \mathbf{Q}\,\mathbf{Q}^*\,\mathbf{A}\| \leq \left[1 + 11\sqrt{k+p}\cdot\sqrt{\min\{m,n\}}\right]\sigma_{k+1}, \tag{5}$$

with probability at least $1 - 6\cdot p^{-p}$; see [21, sec. 1.5]. The error bound (5) indicates that the error produced by the randomized sampling procedure can be larger than the theoretically minimal error σ_{k+1} by a factor of $1 + 11\sqrt{k+p}\cdot\sqrt{\min\{m,n\}}$. This crude bound is typically very pessimistic, in particular for matrices whose singular values decay rapidly; cf. [21].

2.6. Functions for low rank factorizations. For future reference, we introduce functions "`qr`," "`svd`," and "`id`" that can operate in three different modes. In the first mode, they produce the full ("economy size") factorizations described in sections 2.2, 2.3, and 2.4, respectively,

$$[\mathbf{Q}, \mathbf{R}, J] = \mathtt{qr}(\mathbf{A}), \qquad [\mathbf{U}, \mathbf{D}, \mathbf{V}] = \mathtt{svd}(\mathbf{A}), \qquad \text{and} \qquad [\mathbf{X}, J] = \mathtt{id}(\mathbf{A}).$$

In practice, we execute these factorizations using standard LAPACK library functions (proceeding as described in [8] for the ID). In the second mode, we provide an integer k and obtain partial factorizations of rank k,

$$[\mathbf{Q}, \mathbf{R}, J] = \mathtt{qr}(\mathbf{A}, k), \qquad [\mathbf{U}, \mathbf{D}, \mathbf{V}] = \mathtt{svd}(\mathbf{A}, k), \qquad \text{and} \qquad [\mathbf{X}, J] = \mathtt{id}(\mathbf{A}, k).$$

Then the matrices $\mathbf{Q}$, $\mathbf{U}$, $\mathbf{D}$, $\mathbf{V}$ have precisely k columns, and $\mathbf{R}$ and $\mathbf{X}$ have precisely k rows. In the third mode, we provide a real positive number ε and obtain partial factorizations

$$[\mathbf{Q}, \mathbf{R}, J] = \mathtt{qr}(\mathbf{A}, \varepsilon), \qquad [\mathbf{U}, \mathbf{D}, \mathbf{V}] = \mathtt{svd}(\mathbf{A}, \varepsilon), \qquad \text{and} \qquad [\mathbf{X}, J] = \mathtt{id}(\mathbf{A}, \varepsilon),$$

such that

$$\|\mathbf{A}(:, J) - \mathbf{QR}\| \leq \varepsilon, \qquad \|\mathbf{A} - \mathbf{UDV}^*\| \leq \varepsilon, \qquad \text{and} \qquad \|\mathbf{A} - \mathbf{A}(:, J)\mathbf{X}\| \leq \varepsilon.$$

In practice, for a *small* input matrix $\mathbf{A}$, we execute mode 2 and mode 3 by calling the LAPACK routine for a *full* factorization (the ID can be obtained from the full QR) and then simply truncating the result. If $\mathbf{A}$ is large, then we use the randomized sampling technique of section 2.5.

Remark 2. The differentiation between modes 2 and 3 for `qr`, `svd`, and `id` is communicated by whether the second argument is an integer (mode 2) or a real number $\varepsilon \in (0, 1)$ (mode 3). This is slightly questionable notation, but it keeps the formulas clean, and we hope it does not cause confusion.

3. Rank-structured matrices. This section provides exact definitions of the HODLR and HBS rank-structured matrix formats.

3.1. A binary tree structure. Both the HODLR and the HBS representations of an $N \times N$ matrix $\mathbf{A}$ are based on a partition of the index vector $I = [1, 2, \ldots, N]$ into a binary tree structure. We let I form the root of the tree and give it the index 1, $I_1 = I$. We next split the root into two roughly equisized vectors I_2 and I_3 so that $I_1 = I_2 \cup I_3$. The full tree is then formed by continuing to subdivide any interval that holds more than some preset fixed number m of indices. We use the integers $\ell = 0, 1, \ldots, L$ to label the different levels, with 0 denoting the coarsest level. A *leaf*

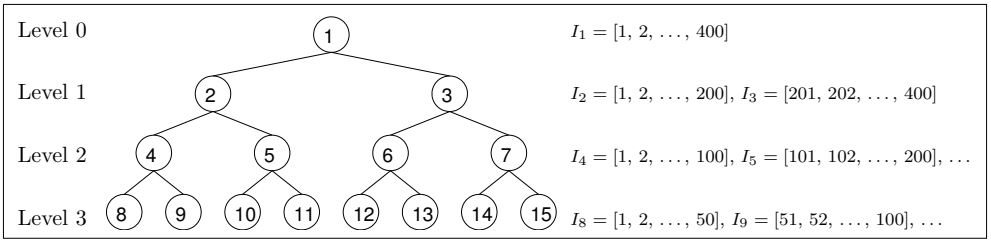

FIG. 1. *Numbering of nodes in a fully populated binary tree with $L = 3$ levels. The root is the original index vector $I = I_1 = [1, 2, \ldots, 400]$.*

is a node corresponding to a vector that never got split. For a nonleaf node τ, its *children* are the two boxes σ_1 and σ_2 such that $I_\tau = I_{\sigma_1} \cup I_{\sigma_2}$, and τ is then the *parent* of σ_1 and σ_2. Two boxes with the same parent are called *siblings*. These definitions are illustrated in Figure 1. For any node τ, let n_τ denote the number of indices in I_τ.

Remark 3. For simplicity, this paper concentrates on the case where the tree is uniform in the sense that all leaves belong to the same level, and all nodes on one level hold roughly equally many points. It is often advantageous to use locally refined trees, and sometimes even nonbinary trees. One possible reason for such tree structures arises when the matrix represents a discretized integral operator, and the tree structure results from a hierarchical partitioning of the points in the discretization based on their *physical* positions in the domain. The techniques described in this paper can be extended to such more general trees. We believe, but have not yet verified, that the impact on computational speed in going to more general trees is very modest.

3.2. The HODLR data-sparse matrix format. The *hierarchically off-diagonal low rank* (HODLR) property is, as the name implies, a condition that the off-diagonal blocks of a matrix $\mathbf{A}$ should have low (numerical) rank. To be precise, given a hierarchical partitioning of the index vector (cf. section 3.1), a computational tolerance ε, and a bound on the rank k, we require that for any sibling pair $\{\alpha, \beta\}$, the corresponding block

$$\mathbf{A}_{\alpha,\beta} = \mathbf{A}(I_\alpha, I_\beta)$$

should have ε-rank at most k. The tessellation resulting from the tree in Figure 1 is shown in Figure 2. We then represent each off-diagonal block via a rank-k factorization

$$\underset{n_\alpha \times n_\beta}{\mathbf{A}_{\alpha,\beta}} \quad = \quad \underset{n_\alpha \times k}{\boldsymbol{\mathcal{U}}_\alpha} \quad \underset{k \times k}{\tilde{\mathbf{A}}_{\alpha,\beta}} \quad \underset{k \times n_\beta}{\boldsymbol{\mathcal{V}}_\beta^*},$$

where $\boldsymbol{\mathcal{U}}_\alpha$ and $\boldsymbol{\mathcal{V}}_\beta$ are orthonormal matrices. It is easily verified that if we required each leaf node to hold at most $O(k)$ points, then it takes $O(k\,N\,\log N)$ storage to store all factors required to represent $\mathbf{A}$, and a matrix-vector multiplication can be executed using $O(k\,N\,\log N)$ flops.

For future reference, we define for a given HODLR matrix $\mathbf{A}$ a "level-truncated" matrix $\mathbf{A}^{(\ell)}$ as the matrix obtained by zeroing out any block associated with levels finer than ℓ. In other words,

$$\mathbf{A}^{(1)} = \begin{array}{|c|c|}\hline \mathbf{0} & \mathbf{A}_{2,3} \\ \hline \mathbf{A}_{3,2} & \mathbf{0} \\ \hline \end{array}, \qquad \mathbf{A}^{(2)} = \begin{array}{|c|c|c|c|}\hline \mathbf{0} & \mathbf{A}_{4,5} & \multicolumn{2}{c|}{\multirow{2}{*}{$\mathbf{A}_{2,3}$}} \\ \cline{1-2} \mathbf{A}_{5,4} & \mathbf{0} & \multicolumn{2}{c|}{} \\ \hline \multicolumn{2}{|c|}{\multirow{2}{*}{$\mathbf{A}_{3,2}$}} & \mathbf{0} & \mathbf{A}_{6,7} \\ \cline{3-4} \multicolumn{2}{|c|}{} & \mathbf{A}_{7,6} & \mathbf{0} \\ \hline \end{array}, \qquad \text{etc.}$$

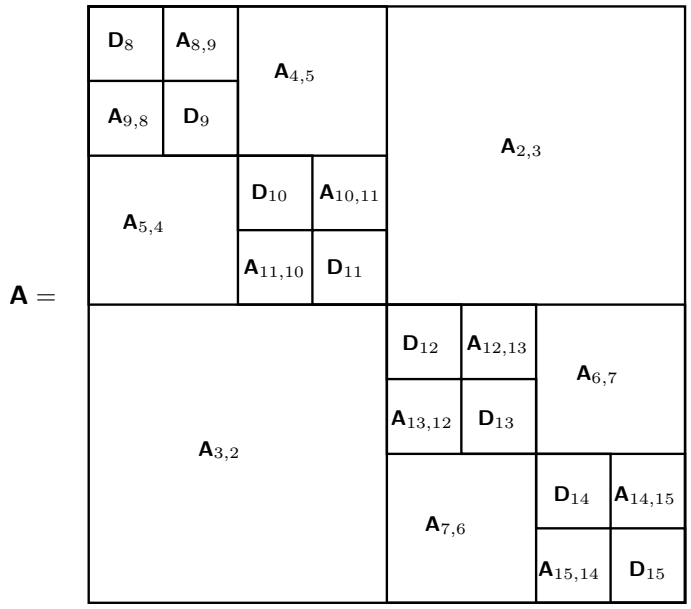

FIG. 2. *An HODLR matrix* $\mathbf{A}$ *tesselated in accordance with the tree in Figure* 1. *Every off-diagonal block* $\mathbf{A}_{\alpha,\beta}$ *that is marked in the figure should have* ε*-rank at most* k.

Remark 4. In the interest of notational simplicity, we generally assume that the numerical rank is the same number k for every off-diagonal block. In practice, we typically estimate the ε-rank for any specific off-diagonal block adaptively to save both storage and flops.

3.3. The HBS data-sparse matrix format. The HODLR format is simple to describe and to use but is slightly inefficient in that it requires the user to store, for every node τ, the basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$, which can be quite long. The *hierarchically block separable* (HBS) format is designed to overcome this problem by expressing these matrices *hierarchically*. (The relationship between HODLR and HBS matrices is entirely analogous to the relationship between $\mathcal{H}$-matrices [18, 3] and $\mathcal{H}^2$-matrices [20, 4].) To be precise, suppose that τ is a node with children $\{\alpha, \beta\}$ and that we can find a *short* matrix $\mathbf{U}_\tau$ such that

$$\underset{n_\tau \times k}{\boldsymbol{\mathcal{U}}_\tau} = \underset{n_\tau \times 2k}{\begin{bmatrix} \boldsymbol{\mathcal{U}}_\alpha & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{U}}_\beta \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_\tau}. \tag{6}$$

The point is that if we have the long basis matrices $\boldsymbol{\mathcal{U}}_\alpha$ and $\boldsymbol{\mathcal{U}}_\beta$ available for the children, then all we need to store in order to be able to apply $\boldsymbol{\mathcal{U}}_\tau$ is the short matrix $\mathbf{U}_\tau$. This process can now be continued recursively. For instance, if $\{\gamma, \delta\}$ are the children of α, and $\{\nu, \mu\}$ are the children of β, we assume there exist matrices $\mathbf{U}_\alpha$ and $\mathbf{U}_\beta$ such that

$$\underset{n_\alpha \times k}{\boldsymbol{\mathcal{U}}_\alpha} = \underset{n_\alpha \times 2k}{\begin{bmatrix} \boldsymbol{\mathcal{U}}_\gamma & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{U}}_\delta \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_\alpha}, \quad \text{and} \quad \underset{n_\beta \times k}{\boldsymbol{\mathcal{U}}_\beta} = \underset{n_\beta \times 2k}{\begin{bmatrix} \boldsymbol{\mathcal{U}}_\mu & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{U}}_\nu \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_\beta}.$$

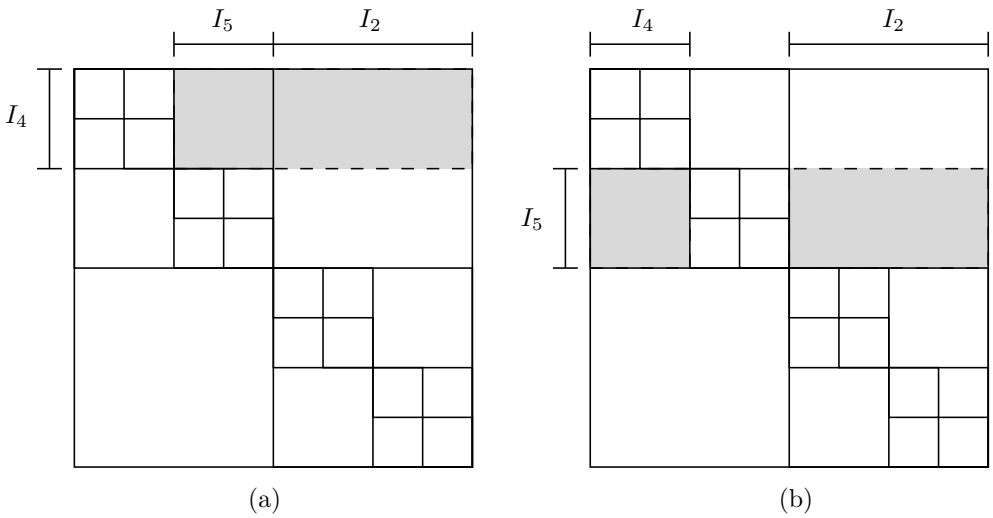

FIG. 3. *Illustration of the* neutered row blocks *for the nodes* 4 *and* 5*, with parent* 2. (a) *The block* $\mathbf{A}(I_4, I_4^{\mathrm{c}})$ *is marked in gray. Observe that* $\mathbf{A}(I_4, I_4^{\mathrm{c}}) = [\mathbf{A}(I_4, I_5), \mathbf{A}(I_4, I_2)]$. (b) *The block* $\mathbf{A}(I_5, I_5^{\mathrm{c}})$ *is marked in gray. Observe that* $\mathbf{A}(I_5, I_5^{\mathrm{c}}) = [\mathbf{A}(I_5, I_4), \mathbf{A}(I_5, I_2)]$.

Then $\boldsymbol{\mathcal{U}}_\tau$ can be expressed as

$$\underset{n_\tau \times k}{\boldsymbol{\mathcal{U}}_\tau} = \underbrace{\begin{bmatrix} \boldsymbol{\mathcal{U}}_\gamma & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{U}}_\delta & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\mathcal{U}}_\mu & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{\mathcal{U}}_\nu \end{bmatrix}}_{n_\tau \times 4k} \underbrace{\begin{bmatrix} \mathbf{U}_\alpha & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_\beta \end{bmatrix}}_{4k \times 2k} \underbrace{\mathbf{U}_\tau}_{2k \times k}.$$

By continuing this process down to the leaves, it becomes clear that we only need to store the "long" basis matrices for a leaf node (and they are not in fact long for a leaf node!); for every other node, it is sufficient to store the small matrix $\mathbf{U}_\tau$. The process for storing the long basis matrices $\boldsymbol{\mathcal{V}}_\tau$ via small matrices $\mathbf{V}_\tau$ of size $2k \times k$ is of course exactly analogous.

In order to guarantee that a relationship such as (6) holds, we need to impose an additional condition (beyond the HODLR conditions) on the long basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$. To this end, given a node τ, let us define a *neutered row block* as the off-diagonal block $\mathbf{A}(I_\tau, I_\tau^{\mathrm{c}})$, where I_τ^{c} is the complement of I_τ within the vector $[1, 2, 3, \ldots, N]$; cf. Figure 3. We then require that the columns of the long basis matrix $\boldsymbol{\mathcal{U}}_\tau$ span the columns of $\mathbf{A}(I_\tau, I_\tau^{\mathrm{c}})$. Observe that for a node τ with sibling σ, the sibling matrix $\mathbf{A}(I_\tau, I_\sigma)$ is a submatrix of the neutered row block $\mathbf{A}(I_\tau, I_\tau^{\mathrm{c}})$ since $I_\sigma \subseteq I_\tau^{\mathrm{c}}$. This means that the new requirement of the basis matrices is more restrictive and that typically the ranks required will be larger for any given precision. However, once the long basis matrices satisfy the more restrictive requirement, it is *necessarily* the case that (6) holds for some small matrix $\mathbf{U}_\tau$. We analogously define the *neutered column block* for τ as the matrix $\mathbf{A}(I_\tau^{\mathrm{c}}, I_\tau)$ and require that the columns of $\boldsymbol{\mathcal{V}}_\tau$ span the rows of $\mathbf{A}(I_\tau^{\mathrm{c}}, I_\tau)$.

DEFINITION 1. *We say that an HODLR matrix* $\mathbf{A}$ *is an HBS matrix if, for every parent node* τ *with children* $\{\alpha, \beta\}$*, there exist "small" basis matrices* $\mathbf{U}_\tau$ *and* $\mathbf{V}_\tau$ *such*

that

$$\underset{n_\tau \times k}{\boldsymbol{\mathcal{U}}_\tau} = \underset{n_\tau \times 2k}{\begin{bmatrix} \boldsymbol{\mathcal{U}}_\alpha & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{U}}_\beta \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_\tau} \quad \textit{and} \quad \underset{n_\tau \times k}{\boldsymbol{\mathcal{V}}_\tau} = \underset{n_\tau \times 2k}{\begin{bmatrix} \boldsymbol{\mathcal{V}}_\alpha & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{V}}_\beta \end{bmatrix}} \underset{2k \times k}{\mathbf{V}_\tau}.$$

While standard practice is to require all basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$ to be *orthonormal*, we have found that it is highly convenient to use the interpolative decomposition (ID) to represent the off-diagonal blocks. The key advantage is that then the sibling interaction matrices will be *submatrices* of the original matrix. This improves interpretability and also slightly reduces storage requirements.

DEFINITION 2. *We say that an HBS matrix* $\mathbf{A}$ *is in HBSID format if every basis matrix* $\mathbf{U}_\tau$ *and* $\mathbf{V}_\tau$ *contains a* $k \times k$ *identity matrix, and every sibling interaction matrix* $\tilde{\mathbf{A}}_{\alpha,\beta}$ *is a submatrix of* $\mathbf{A}$*. In other words, there exist some index sets* $\tilde{I}_\alpha^{\text{in}}$ *and* $\tilde{I}_\beta^{\text{out}}$ *such that*

$$\tilde{\mathbf{A}}_{\alpha,\beta} = \mathbf{A}(\tilde{I}_\alpha^{\text{in}}, \tilde{I}_\beta^{\text{out}}).$$

The index sets $\tilde{I}_\tau^{\text{in}}$ *and* $\hat{I}_\tau^{\text{out}}$ *are called the incoming skeleton and the outgoing skeleton of box* τ*, respectively. We enforce that these are "nested," which is to say that if the children of* τ *are* $\{\alpha, \beta\}$*, then*

$$\tilde{I}_\tau^{\text{in}} \subseteq \tilde{I}_\alpha^{\text{in}} \cup \tilde{I}_\beta^{\text{in}} \quad \textit{and} \quad \tilde{I}_\tau^{\text{out}} \subseteq \tilde{I}_\alpha^{\text{out}} \cup \tilde{I}_\beta^{\text{out}}.$$

Our terminology using the terms "incoming" and "outgoing" skeletons follows [30] and related literature on generalized fast multipole methods (FMMs). The idea is that the "incoming" skeleton is where a patch receives information communicated *from* other patches and then broadcasts information *to* the other patches on its "outgoing" skeleton. In other words, the "incoming" and "outgoing" representations are analogous to "local" and "multipole" expansions in an FMM, respectively.

Remark 5. The straightforward way to build a basis matrix $\boldsymbol{\mathcal{U}}_\tau$ for a node τ is to explicitly form the corresponding neutered row block $\mathbf{A}(I_\tau, I_\tau^{\text{c}})$ and then compress it (perform column pivoted Gram–Schmidt on its *columns* to form an ON-basis $\boldsymbol{\mathcal{U}}_\tau$, or perform column pivoted Gram–Schmidt on its *rows* to form the ID). However, suppose that we can somehow construct a smaller matrix $\boldsymbol{\mathcal{Y}}_\tau$ of size $n_\tau \times \ell$ with the property that the columns of $\boldsymbol{\mathcal{Y}}_\tau$ span the columns of $\mathbf{A}(I_\tau, I_\tau^{\text{c}})$. Then it would be sufficient to process the columns of $\boldsymbol{\mathcal{Y}}_\tau$ to build a basis for $\mathbf{A}(I_\tau, I_\tau^{\text{c}})$. For instance, if we orthonormalize the columns of $\boldsymbol{\mathcal{Y}}_\tau$ to form a basis matrix $\boldsymbol{\mathcal{U}}_\tau = \texttt{qr}(\boldsymbol{\mathcal{Y}}_\tau)$, then the columns of $\boldsymbol{\mathcal{U}}_\tau$ will necessarily form an ON-basis for the columns of $\mathbf{A}(I_\tau, I_\tau^{\text{c}})$. The key point here is that one can often find such a matrix $\boldsymbol{\mathcal{Y}}_\tau$ with a small number ℓ of columns. In [29] we use a representation theorem from potential theory to find such a matrix $\boldsymbol{\mathcal{Y}}_\tau$ when $\mathbf{A}$ results from discretizing a boundary integral equation (BIE). In [27] we do this via randomized sampling, so that $\boldsymbol{\mathcal{Y}}_\tau = \mathbf{A}(I_\tau, I_\tau^{\text{c}})\,\boldsymbol{\Omega}$ for some Gaussian random matrix $\boldsymbol{\Omega}$. The main point of [27] is that this can be done by applying all of $\mathbf{A}$ to a *single* random matrix with $N \times \ell$ columns, where $\ell \approx k$. In the current paper, we use a similar strategy, but we now require the application of $\mathbf{A}$ to a set of $O(k)$ random vector for each level.

4. An algorithm for compressing an HODLR matrix. In this section, we present a randomized algorithm for constructing the HODLR representation of a given matrix $\mathbf{A}$ of size $N \times N$, for which a tree structure (as defined in section 3.1) has been provided. The algorithm consists of a single sweep through the levels in the

tree of nodes, starting from the root (the entire domain), and processing each level of successively smaller blocks, one level at a time. The algorithm presented here is directly inspired by the "peeling algorithm" of [24]. We assume that an upper bound k on the ranks of the off-diagonal blocks has been given in advance. We then pick an oversampling parameter p (cf. section 2.5), say, $p = 10$, and set $r = k + p$.

In presenting the algorithm, we let $\mathbf{G}_\tau$ denote a random matrix of size $n_\tau \times r$ drawn from a Gaussian distribution. Blocked matrices are drawn with the blocks to be processed set in red type. To minimize clutter, blocks of the matrix that will not play a part in the current step are marked with a star (*) and may or may not be zero.

Processing level 0 *(the root of the tree).* Let $\{\alpha, \beta\}$ denote the children of the root node (for the tree in Figure 1, $\alpha = 2$ and $\beta = 3$). Our objective is now to find low rank factorizations of the off-diagonal blocks

$$\mathbf{A} = \begin{bmatrix} * & \mathbf{A}_{\alpha,\beta} \\ \mathbf{A}_{\alpha,\beta} & * \end{bmatrix}.$$

To this end, we build two random matrices, each of size $N \times r$, and defined by

$$\mathbf{\Omega}_1 = \begin{bmatrix} \mathbf{G}_\alpha \\ \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{\Omega}_2 = \begin{bmatrix} \mathbf{0} \\ \mathbf{G}_\beta \end{bmatrix}.$$

Then we construct the matrices of samples

$$\mathbf{Y}_1 = \mathbf{A}\mathbf{\Omega_2} = \begin{bmatrix} \mathbf{A}_{\alpha,\beta}\mathbf{G}_\beta \\ * \end{bmatrix} \quad \text{and} \quad \mathbf{Y}_2 = \mathbf{A}\mathbf{\Omega_1} = \begin{bmatrix} * \\ \mathbf{A}_{\beta,\alpha}\mathbf{G}_\alpha \end{bmatrix}.$$

Supported by the results on randomized sampling described in section 2.5, we now know that it is almost certain that the columns in the top block of $\mathbf{Y}_1$ will span the columns of $\mathbf{A}_{\alpha,\beta}$. By orthonormalizing the columns of $\mathbf{Y}_1(I_\alpha, :)$, we therefore obtain an ON-basis for the column space of $\mathbf{A}_{\alpha,\beta}$. In other words, we set

$$\mathcal{U}_\alpha = \mathtt{qr}(\mathbf{Y}_1(I_\alpha, :)) \quad \text{and} \quad \mathcal{U}_\beta = \mathtt{qr}(\mathbf{Y}_2(I_\beta, :)),$$

and then we know that $\mathcal{U}_\alpha$ and $\mathcal{U}_\beta$ will serve as the relevant basis matrices in the HODLR representation of $\mathbf{A}$. To compute $\mathcal{V}_\alpha$, $\mathcal{V}_\beta$, $\mathbf{B}_{\alpha,\beta}$, and $\mathbf{B}_{\beta,\alpha}$, we now need to form the matrices $\mathcal{U}_\alpha^*\mathbf{A}_{\alpha,\beta}$ and $\mathcal{U}_\beta^*\mathbf{A}_{\beta,\alpha}$. To do this through our "black-box" matrix-matrix multiplier, we form the new test matrices

$$\mathbf{\Omega}_1 = \begin{bmatrix} \mathcal{U}_\alpha \\ \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{\Omega}_2 = \begin{bmatrix} \mathbf{0} \\ \mathcal{U}_\beta \end{bmatrix}.$$

Then we construct the matrices of samples

$$\mathbf{Z}_1 = \mathbf{A}^*\mathbf{\Omega_2} = \begin{bmatrix} \mathbf{A}_{\beta,\alpha}^*\mathcal{U}_\beta \\ * \end{bmatrix} \quad \text{and} \quad \mathbf{Z}_2 = \mathbf{A}^*\mathbf{\Omega_1} = \begin{bmatrix} * \\ \mathbf{A}_{\alpha,\beta}^*\mathcal{U}_\alpha \end{bmatrix}.$$

All that remains now is to compute QR factorizations

$$[\mathcal{V}_\alpha, \mathbf{B}_{\beta,\alpha}] = \mathtt{qr}(\mathbf{Z}_1(I_\alpha, :)) \quad \text{and} \quad [\mathcal{V}_\beta, \mathbf{B}_{\alpha,\beta}] = \mathtt{qr}(\mathbf{Z}_2(I_\beta, :)). \tag{7}$$

Remark 6. At a slight increase in cost, one can obtain *diagonal* sibling interaction matrices $\mathbf{B}_{\alpha,\beta}$ and $\mathbf{B}_{\beta,\alpha}$. We would then replace the QR factorization in (7) by a full SVD

$$[\mathcal{V}_\alpha, \mathbf{B}_{\beta,\alpha}, \hat{\mathbf{U}}_\beta] = \mathtt{svd}(\mathbf{Z}_1(I_\alpha,:)) \quad \text{and} \quad [\mathcal{V}_\beta, \mathbf{B}_{\alpha,\beta}, \hat{\mathbf{U}}_\alpha] = \mathtt{svd}(\mathbf{Z}_2(I_\beta,:)). \tag{8}$$

This step then requires an update to the long basis matrices for the column space:

$$\mathcal{U}_\alpha \leftarrow \mathcal{U}_\alpha \hat{\mathbf{U}}_\alpha \quad \text{and} \quad \mathcal{U}_\beta \leftarrow \mathcal{U}_\beta \hat{\mathbf{U}}_\beta. \tag{9}$$

Processing level 1. Now that all off-diagonal blocks on level 1 have been computed, we use this information to compress the blocks on level 2. Let $\{\alpha, \beta, \gamma, \delta\}$ denote the boxes on level 2 (for the tree in Figure 1, $\alpha = 4$, $\beta = 5$, $\gamma = 6$, $\delta = 7$). Our objective is now to construct low rank approximations to the following off-diagonal blocks:

$$\mathbf{A} = \begin{array}{|cc|cc|}\hline * & \mathbf{A}_{\alpha,\beta} & \multicolumn{2}{c|}{\multirow{2}{*}{$*$}} \\ \mathbf{A}_{\beta,\alpha} & * & & \\ \hline \multicolumn{2}{|c|}{\multirow{2}{*}{$*$}} & * & \mathbf{A}_{\gamma,\delta} \\ \multicolumn{2}{|c|}{} & \mathbf{A}_{\delta,\gamma} & * \\ \hline \end{array}$$

First, observe that

$$\mathbf{A} - \mathbf{A}^{(1)} = \begin{array}{|cc|cc|}\hline * & \mathbf{A}_{\alpha,\beta} & \multicolumn{2}{c|}{\multirow{2}{*}{$\mathbf{0}$}} \\ \mathbf{A}_{\beta,\alpha} & * & & \\ \hline \multicolumn{2}{|c|}{\multirow{2}{*}{$\mathbf{0}$}} & * & \mathbf{A}_{\gamma,\delta} \\ \multicolumn{2}{|c|}{} & \mathbf{A}_{\delta,\gamma} & * \\ \hline \end{array}$$

We then define two random test matrices, each of size $N \times r$, via

$$\mathbf{\Omega}_1 = \begin{bmatrix} \mathbf{G}_\alpha \\ \mathbf{0} \\ \mathbf{G}_\gamma \\ \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{\Omega}_2 = \begin{bmatrix} \mathbf{0} \\ \mathbf{G}_\beta \\ \mathbf{0} \\ \mathbf{G}_\delta \end{bmatrix}.$$

We compute the sample matrices via

$$\mathbf{Y}_1 = \mathbf{A}\mathbf{\Omega_2} - \mathbf{A}^{(1)}\mathbf{\Omega_2} = \begin{bmatrix} \mathbf{A}_{\alpha,\beta}\mathbf{G}_\beta \\ * \\ \mathbf{A}_{\gamma,\delta}\mathbf{G}_\delta \\ * \end{bmatrix} \quad \text{and} \quad \mathbf{Y}_2 = \mathbf{A}\mathbf{\Omega_1} - \mathbf{A}^{(1)}\mathbf{\Omega_1} = \begin{bmatrix} * \\ \mathbf{A}_{\beta,\alpha}\mathbf{G}_\alpha \\ * \\ \mathbf{A}_{\delta,\gamma}\mathbf{G}_\gamma \end{bmatrix}. \tag{10}$$

In evaluating $\mathbf{Y}_1$ and $\mathbf{Y}_2$, we use the black-box multiplier to form $\mathbf{A\Omega_2}$ and $\mathbf{A\Omega_1}$, and the compressed representation of $\mathbf{A}^{(1)}$ obtained on the previous level to form $\mathbf{A}^{(1)}\mathbf{\Omega_2}$ and $\mathbf{A}^{(1)}\mathbf{\Omega_1}$. We get ON-bases for the column spaces of the four sibling interaction matrices by orthonormalizing the pertinent blocks of $\mathbf{Y}_1$ and $\mathbf{Y}_2$:

$$\mathcal{U}_\alpha = \mathtt{qr}(\mathbf{Y}_1(I_\alpha,:)), \quad \mathcal{U}_\beta = \mathtt{qr}(\mathbf{Y}_2(I_\beta,:)), \quad \mathcal{U}_\gamma = \mathtt{qr}(\mathbf{Y}_1(I_\gamma,:)), \quad \mathcal{U}_\delta = \mathtt{qr}(\mathbf{Y}_2(I_\delta,:)).$$

It remains to construct the ON-bases for the corresponding row spaces and the compressed sibling interaction matrices. To this end, we form two new test matrices, both of size $N \times r$, via

$$\mathbf{\Omega}_1 = \begin{bmatrix} \mathcal{U}_\alpha \\ \mathbf{0} \\ \mathcal{U}_\gamma \\ \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{\Omega}_2 = \begin{bmatrix} \mathbf{0} \\ \mathcal{U}_\beta \\ \mathbf{0} \\ \mathcal{U}_\delta \end{bmatrix}.$$

Then the sample matrices are computed via

$$\mathbf{Z}_1 = \mathbf{A}^*\mathbf{\Omega_2} - \left(\mathbf{A}^{(1)}\right)^*\mathbf{\Omega_2} = \begin{bmatrix} \mathbf{A}^*_{\beta\alpha}\mathcal{U}_\beta \\ * \\ \mathbf{A}^*_{\delta,\gamma}\mathcal{U}_\delta \\ * \end{bmatrix} \quad \text{and} \quad \mathbf{Z}_2 = \mathbf{A}^*\mathbf{\Omega_1} - \left(\mathbf{A}^{(1)}\right)^*\mathbf{\Omega_1} = \begin{bmatrix} * \\ \mathbf{A}^*_{\alpha,\beta}\mathcal{U}_\alpha \\ * \\ \mathbf{A}^*_{\gamma,\delta}\mathcal{U}_\gamma \end{bmatrix}.$$

We obtain diagonal compressed sibling interaction matrices by taking a sequence of dense SVDs of the relevant subblocks (cf. (8)),

$$\begin{aligned} [\mathcal{V}_\alpha, \mathbf{B}_{\beta\alpha}, \hat{\mathbf{U}}_\beta] &= \mathtt{svd}(\mathbf{Z}_1(I_\alpha, :)), \\ [\mathcal{V}_\beta, \mathbf{B}_{\alpha,\beta}, \hat{\mathbf{U}}_\alpha] &= \mathtt{svd}(\mathbf{Z}_2(I_\beta, :)), \\ [\mathcal{V}_\gamma, \mathbf{B}_{\delta,\gamma}, \hat{\mathbf{U}}_\delta] &= \mathtt{svd}(\mathbf{Z}_1(I_\gamma, :)), \\ [\mathcal{V}_\delta, \mathbf{B}_{\gamma,\delta}, \hat{\mathbf{U}}_\gamma] &= \mathtt{svd}(\mathbf{Z}_2(I_\delta, :)). \end{aligned}$$

Finally we update the bases for the column spaces (cf. (9)),

$$\mathcal{U}_\alpha \leftarrow \mathcal{U}_\alpha\hat{\mathbf{U}}_\alpha, \qquad \mathcal{U}_\beta \leftarrow \mathcal{U}_\beta\hat{\mathbf{U}}_\beta, \qquad \mathcal{U}_\gamma \leftarrow \mathcal{U}_\gamma\hat{\mathbf{U}}_\gamma, \qquad \mathcal{U}_\delta \leftarrow \mathcal{U}_\delta\hat{\mathbf{U}}_\delta.$$

Processing levels 2 *through* $L-1$. The processing of every level proceeds in a manner completely analogous to the processing of level 1. The relevant formulas are given in Figure 4.

Processing the leaves. Once all L levels have been traversed using the procedure described, compressed representations of all off-diagonal blocks will have been computed. At this point, all that remains is to extract the diagonal blocks. We illustrate the process for a simplistic example of a tree with only $L = 2$ levels (beyond the root). Letting $\{\alpha, \beta, \gamma, \delta\}$ denote the leaf nodes, our task is then to extract the diagonal blocks $\mathbf{D}_\alpha$, $\mathbf{D}_\beta$, $\mathbf{D}_\gamma$, $\mathbf{D}_\delta$:

$$\mathbf{A} = \begin{array}{|cc|cc|}\hline \mathbf{D}_\alpha & * & \multicolumn{2}{c|}{*} \\ * & \mathbf{D}_\beta & \multicolumn{2}{c|}{} \\ \hline \multicolumn{2}{|c|}{*} & \mathbf{D}_\gamma & * \\ \multicolumn{2}{|c|}{} & * & \mathbf{D}_\delta \\ \hline \end{array} = \begin{array}{|cc|cc|}\hline \mathbf{D}_\alpha & \mathbf{0} & \multicolumn{2}{c|}{\mathbf{0}} \\ \mathbf{0} & \mathbf{D}_\beta & \multicolumn{2}{c|}{} \\ \hline \multicolumn{2}{|c|}{\mathbf{0}} & \mathbf{D}_\gamma & \mathbf{0} \\ \multicolumn{2}{|c|}{} & \mathbf{0} & \mathbf{D}_\delta \\ \hline \end{array} + \mathbf{A}^{(2)}.$$

Since the diagonal blocks are not rank-deficient, we will extract them directly, without using randomized sampling. To describe the process, we assume at first (for simplicity) that every diagonal block has the same size, $m \times m$. We then choose a test matrix of size $N \times m$,

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{I}_m \\ \mathbf{I}_m \\ \mathbf{I}_m \\ \mathbf{I}_m \end{bmatrix},$$

and trivially extract the diagonal blocks via the sampling

$$\mathbf{Y} = \mathbf{A\Omega} - \mathbf{A}^{(2)}\mathbf{\Omega} = \begin{bmatrix} \mathbf{D}_\alpha \\ \mathbf{D}_\beta \\ \mathbf{D}_\gamma \\ \mathbf{D}_\delta \end{bmatrix}.$$

The diagonal blocks can then be read off directly from $\mathbf{Y}$.

Build compressed representations of all off-diagonal blocks.
loop over levels $\ell = 0 : (L-1)$

Build the random matrices $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_2$.
$\mathbf{\Omega}_1 = \texttt{zeros}(n, r)$
$\mathbf{\Omega}_2 = \texttt{zeros}(n, r)$
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
$\mathbf{\Omega}_1(I_\alpha, :) = \texttt{randn}(n_\alpha, r)$
$\mathbf{\Omega}_2(I_\beta, :) = \texttt{randn}(n_\beta, r)$
end loop

Apply $\mathbf{A}$ to build the samples for the incoming basis matrices.
$\mathbf{Y}_1 = \mathbf{A}\mathbf{\Omega}_2 - \mathbf{A}^{(\ell)}\mathbf{\Omega}_2$
$\mathbf{Y}_2 = \mathbf{A}\mathbf{\Omega}_1 - \mathbf{A}^{(\ell)}\mathbf{\Omega}_1$

Orthonormalize the sample matrices to build the incoming basis matrices.
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
$\mathcal{U}_\alpha = \texttt{qr}(\mathbf{Y}_1(I_\alpha, :))$.
$\mathcal{U}_\beta = \texttt{qr}(\mathbf{Y}_2(I_\beta, :))$.
$\mathbf{\Omega}_1(I_\alpha, :) = \mathcal{U}_\alpha$
$\mathbf{\Omega}_2(I_\beta, :) = \mathcal{U}_\beta$
end loop

Apply $\mathbf{A}^*$ to build the samples for the outgoing basis matrices.
$\mathbf{Z}_1 = \mathbf{A}^*\mathbf{\Omega}_2 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{\Omega}_2$
$\mathbf{Z}_2 = \mathbf{A}^*\mathbf{\Omega}_1 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{\Omega}_1$

Take local SVDs to build incoming basis matrices and sibling interaction matrices. We determine the actual rank and update the $\mathcal{U}_*$ basis matrices accordingly.
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
$[\mathcal{V}_\alpha, \mathbf{B}_{\beta,\alpha}, \hat{\mathbf{U}}_\beta] = \texttt{svd}(\mathbf{Z}_1(I_\alpha, :), \varepsilon)$.
$[\mathcal{V}_\beta, \mathbf{B}_{\alpha,\beta}, \hat{\mathbf{V}}_\alpha] = \texttt{svd}(\mathbf{Z}_2(I_\beta, :), \varepsilon)$.
$\mathcal{U}_\beta \leftarrow \mathcal{U}_\beta \hat{\mathbf{U}}_\beta$.
$\mathcal{U}_\alpha \leftarrow \mathcal{U}_\alpha \hat{\mathbf{U}}_\alpha$.
end loop
end loop

Extract the diagonal matrices.
$n_{\max} = \texttt{max}\{n_\tau : \tau \text{ is a leaf}\}$
$\mathbf{\Omega} = \texttt{zeros}(N, n_{\max})$
loop over leaf boxes τ
$\mathbf{\Omega}(I_\tau, 1 : n_\tau) = \texttt{eye}(n_\tau)$.
end loop
$\mathbf{Y} = \mathbf{A}\mathbf{\Omega} - \mathbf{A}^{(L)}\mathbf{\Omega}$
loop over leaf boxes τ
$\mathbf{D}_\tau = \mathbf{Y}(I_\tau, 1 : n_\tau)$.
end loop

FIG. 4. *Randomized compression of an HODLR matrix.*

For the general case where the leaves may be of different sizes we simply pad a few zero columns after the identity matrix for any "smaller" leaves. To be precise, set $m = \max\{n_\tau : \tau \text{ is a leaf}\}$, and then form a test matrix $\mathbf{\Omega}$ of size $N \times m$ so that for every leaf τ,

$$\mathbf{\Omega}(I_\tau, :) = \big[\mathbf{I}_{n_\tau} \ \texttt{zeros}(n_\tau, m - n_\tau)\big].$$

The entire algorithm is summarized in Figure 4.

4.1. Asymptotic complexity. Let L denote the number of levels in the tree. We find that $L \sim \log N$. Let T_{mult} denote the time required to apply $\mathbf{A}$ or $\mathbf{A}^*$ to a vector, let $T_{\mathbf{A}^{(\ell)}}$ denote the time required to apply $\mathbf{A}^{(\ell)}$ or $(\mathbf{A}^{(\ell)})^*$ to a vector, and let T_{flop} denote the time required for a flop. Then the cost of processing level ℓ is

$$T_\ell \sim T_{\text{mult}} \times k + T_{\mathbf{A}^{(\ell)}} \times k + T_{\text{flop}} \times 2^\ell \, k^2 \, \frac{N}{2^\ell}, \tag{11}$$

since on level ℓ there are 2^ℓ blocks to be processed, and each "long" matrix at this level has height $2^{-\ell} N$ and width $r = k + p = O(k)$. Further, we find that the cost of applying $\mathbf{A}^{(\ell)}$ to a single vector is

$$T_{\mathbf{A}^{(\ell)}} \sim T_{\text{flop}} \times \sum_{j=0}^{\ell} 2^j \, k \, \frac{N}{2^j} \sim T_{\text{flop}} \times \ell \, k \, N. \tag{12}$$

Combining (11) and (12) and summing from $\ell = 0$ to $\ell = L$, we find (using that $L \sim \log N$)

$$T_{\text{compress}} \sim T_{\text{mult}} \times k \, \log N + T_{\text{flop}} \times k^2 \, N \, \big(\log N\big)^2. \tag{13}$$

5. An algorithm for compressing an HBS matrix. The algorithm for computing a compressed representation of an HBS matrix is a slight variation of the algorithm for an HODLR matrix described in section 4. The key difference is that the long basis matrices $\boldsymbol{\mathcal{U}}_\alpha$ and $\boldsymbol{\mathcal{V}}_\alpha$ associated with any node now must satisfy a more rigorous requirement. We will accomplish this objective by constructing, for every node α, two new "long" sampling matrices $\boldsymbol{\mathcal{Y}}_\alpha$ and $\boldsymbol{\mathcal{Z}}_\alpha$, each of size $n_\alpha \times r$, that help transmit information from the higher levels to the node α. The presentation will start in section 5.1 with a description of the modification to the scheme of section 4 required to enforce the more rigorous requirement. In this initial description, we will assume that all four "long" matrices associated with a node ($\boldsymbol{\mathcal{U}}_\tau$, $\boldsymbol{\mathcal{V}}_\tau$, $\boldsymbol{\mathcal{Y}}_\tau$, $\boldsymbol{\mathcal{Z}}_\tau$) are stored explicitly, resulting in an $O(k\,N\,\log N)$ memory requirement, just like for the HODLR algorithm. In section 5.2 we show that while these "long" matrices do need to be temporarily built and processed, they can be stored *implicitly*, which will allow the algorithm to use only $O(k\,N)$ memory. Finally, section 5.3 will describe how to construct a representation using IDs in all low-rank approximations.

5.1. A basic scheme for compressing an HBS matrix. Throughout this section, let α denote a node with a parent τ that is not the root, and with a sibling β. We will first describe the process for building the long basis matrices $\{\boldsymbol{\mathcal{U}}_\tau\}_\tau$. To this end, recall that the difference in requirements on the long basis matrices in the two frameworks is as follows:

HODLR framework: The columns of $\boldsymbol{\mathcal{U}}_\alpha$ need to span the columns of $\mathbf{A}(I_\alpha, I_\beta)$.

HBS framework: The columns of $\boldsymbol{\mathcal{U}}_\alpha$ need to span the columns of $\mathbf{A}(I_\alpha, I_\alpha^{\text{c}})$.

The assertion that the requirements on a basis in the HBS framework is more stringent follows from the fact that $I_\beta \subseteq I_\alpha^c$, and that I_β is in general much smaller than I_α^c; cf. Figure 3. Now note that

$$\mathbf{A}(I_\alpha, I_\alpha^c) = \big[\mathbf{A}(I_\alpha, I_\beta)\ \mathbf{A}(I_\alpha, I_\tau^c)\big]\,\mathbf{P}. \tag{14}$$

In (14), the matrix $\mathbf{P}$ is a permutation matrix whose effect is to reorder the columns. For purposes of constructing a basis for the column space, the matrix $\mathbf{P}$ can be ignored. The idea is now to introduce a new sampling matrix $\boldsymbol{\mathcal{Y}}_\alpha$ of size $n_\alpha \times r$ that encodes all the information that needs to be transmitted from the parent τ. Specifically, we ask that:

The columns of $\boldsymbol{\mathcal{Y}}_\alpha$ span the columns of $\mathbf{A}(I_\alpha, I_\tau^c)$ (to within precision ε).

Then, when processing box α, we will sample $\mathbf{A}(I_\alpha, I_\beta)$ using a Gaussian matrix $\mathbf{G}_\beta$ of size $n_\beta \times r$ just as in the HODLR algorithm. In the end, we will build $\boldsymbol{\mathcal{U}}_\alpha$ by combining the two sets of samples

$$[\boldsymbol{\mathcal{U}}_\alpha, \mathbf{D}_\alpha, \sim] = \mathtt{svd}\big(\big[\mathbf{A}(I_\alpha, I_\beta)\mathbf{G}_\beta,\ \boldsymbol{\mathcal{Y}}_\alpha\big], r\big).$$

In other words, we take a matrix $\big[\mathbf{A}(I_\alpha, I_\beta)\mathbf{G}_\beta,\ \boldsymbol{\mathcal{Y}}_\alpha\big]$ of size $n_\alpha \times 2r$ and extract its leading r singular components (the trailing r components are ignored). All that remains is to build the sample matrices $\boldsymbol{\mathcal{Y}}_\gamma$ and $\boldsymbol{\mathcal{Y}}_\delta$ that transmit information to the children $\{\gamma, \delta\}$ of α. To be precise, let J_γ and J_δ denote the local *relative* index vectors, so that

$$I_\gamma = I_\alpha(J_\gamma) \quad \text{and} \quad I_\delta = I_\alpha(J_\delta).$$

Then set

$$\boldsymbol{\mathcal{Y}}_\gamma = \boldsymbol{\mathcal{U}}(J_\gamma, :)\mathbf{D}_\alpha \quad \text{and} \quad \boldsymbol{\mathcal{Y}}_\delta = \boldsymbol{\mathcal{U}}(J_\delta, :)\mathbf{D}_\alpha.$$

The process for building the long basis matrices $\{\boldsymbol{\mathcal{V}}_\alpha\}_\alpha$ is entirely analogous to the process described for building the $\{\mathbf{U}_\alpha\}_\alpha$ matrices. We first recall that the difference between the HODLR and the HBS frameworks is as follows:

HODLR framework: The columns of $\boldsymbol{\mathcal{V}}_\alpha$ need to span the rows of $\mathbf{A}(I_\beta, I_\alpha)$.
HBS framework: The columns of $\boldsymbol{\mathcal{V}}_\alpha$ need to span the rows of $\mathbf{A}(I_\alpha^c, I_\alpha)$.

With $\mathbf{P}$ again denoting a permutation matrix, we have

$$\mathbf{A}(I_\alpha^c, I_\alpha) = \mathbf{P}\begin{bmatrix} \mathbf{A}(I_\beta, I_\alpha) \\ \mathbf{A}(I_\tau^c, I_\alpha) \end{bmatrix}.$$

It follows that the role that was played by $\boldsymbol{\mathcal{Y}}_\alpha$ in the construction of $\boldsymbol{\mathcal{U}}_\alpha$ is now played by a sampling matrix $\boldsymbol{\mathcal{Z}}_\alpha$ of size $n_\alpha \times r$ that satisfies the following:

The columns of $\boldsymbol{\mathcal{Z}}_\alpha$ span the rows of $\mathbf{A}(I_\tau^c, I_\alpha)$ (to within precision ε).

The algorithm for computing the HBS representation of a matrix is given in Figure 5.

loop over levels $\ell = 0 : (L-1)$

Build the random matrices $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_2$.
$\mathbf{\Omega}_1 = \texttt{zeros}(n, r)$
$\mathbf{\Omega}_2 = \texttt{zeros}(n, r)$
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
$\mathbf{\Omega}_1(I_\alpha, :) = \texttt{randn}(n_\alpha, r)$
$\mathbf{\Omega}_2(I_\beta, :) = \texttt{randn}(n_\beta, r)$
end loop

Apply $\mathbf{A}$ to build the samples for the incoming basis matrices.
$\mathbf{Y}_1 = \mathbf{A}\mathbf{\Omega}_2 - \mathbf{A}^{(\ell)}\mathbf{\Omega}_2$
$\mathbf{Y}_2 = \mathbf{A}\mathbf{\Omega}_1 - \mathbf{A}^{(\ell)}\mathbf{\Omega}_1$

Orthonormalize the sample matrices to build the incoming basis matrices.
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
if (τ is the root)
$\mathbf{Y}_1^{\text{loc}} = \mathbf{Y}_1(I_\alpha, :)$
$\mathbf{Y}_2^{\text{loc}} = \mathbf{Y}_2(I_\beta, :)$
else
$\mathbf{Y}_1^{\text{loc}} = [\mathbf{Y}_1(I_\alpha, :),\ \boldsymbol{\mathcal{Y}}_\tau(J_\alpha, :)]$
$\mathbf{Y}_2^{\text{loc}} = [\mathbf{Y}_2(I_\beta, :),\ \boldsymbol{\mathcal{Y}}_\tau(J_\beta, :)]$
end if
$[\boldsymbol{\mathcal{U}}_\alpha, \mathbf{y}_\alpha, \sim] = \texttt{svd}(\mathbf{Y}_1^{\text{loc}}, r)$.
$[\boldsymbol{\mathcal{U}}_\beta, \mathbf{y}_\beta, \sim] = \texttt{svd}(\mathbf{Y}_2^{\text{loc}}, r)$.
$\boldsymbol{\mathcal{Y}}_\alpha = \boldsymbol{\mathcal{U}}_\alpha\, \texttt{diag}(\mathbf{y}_\alpha)$.
$\boldsymbol{\mathcal{Y}}_\beta = \boldsymbol{\mathcal{U}}_\beta\, \texttt{diag}(\mathbf{y}_\beta)$.
$\mathbf{\Omega}_1(I_\alpha, :) = \boldsymbol{\mathcal{U}}_\alpha$
$\mathbf{\Omega}_2(I_\beta, :) = \boldsymbol{\mathcal{U}}_\beta$
$\mathbf{U}_\tau = \begin{bmatrix} \boldsymbol{\mathcal{U}}_\alpha^* \boldsymbol{\mathcal{U}}_\tau(J_\alpha, :) \\ \boldsymbol{\mathcal{U}}_\beta^* \boldsymbol{\mathcal{U}}_\tau(J_\beta, :) \end{bmatrix}$.
end loop

Apply $\mathbf{A}^*$ to build the samples for the outgoing basis matrices.
$\mathbf{Z}_1 = \mathbf{A}^*\mathbf{\Omega}_2 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{\Omega}_2$
$\mathbf{Z}_2 = \mathbf{A}^*\mathbf{\Omega}_1 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{\Omega}_1$

Take local SVDs to build incoming basis matrices and sibling interaction matrices.
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
if (τ is the root)
$\mathbf{Z}_1^{\text{loc}} = \mathbf{Z}_1(I_\alpha, :)$
$\mathbf{Z}_2^{\text{loc}} = \mathbf{Z}_2(I_\beta, :)$
else
$\mathbf{Z}_1^{\text{loc}} = [\mathbf{Z}_1(I_\alpha, :),\ \boldsymbol{\mathcal{Z}}_\tau(J_\alpha, :)]$
$\mathbf{Z}_2^{\text{loc}} = [\mathbf{Z}_2(I_\beta, :),\ \boldsymbol{\mathcal{Z}}_\tau(J_\beta, :)]$
end if
$[\boldsymbol{\mathcal{V}}_\alpha, \mathbf{b}_{21}, \mathbf{X}_1] = \texttt{svd}(\mathbf{Z}_1^{\text{loc}}, r)$.
$[\boldsymbol{\mathcal{V}}_\beta, \mathbf{b}_{12}, \mathbf{X}_2] = \texttt{svd}(\mathbf{Z}_2^{\text{loc}}, r)$.
$\boldsymbol{\mathcal{Z}}_\alpha = \boldsymbol{\mathcal{V}}_\alpha \texttt{diag}(\mathbf{b}_{21})$
$\boldsymbol{\mathcal{Z}}_\beta = \boldsymbol{\mathcal{V}}_\beta \texttt{diag}(\mathbf{b}_{12})$
$\mathbf{B}_{\alpha,\beta} = \mathbf{X}_1(1:r, :)\texttt{diag}(\mathbf{b}_{12})$
$\mathbf{B}_{\beta,\alpha} = \mathbf{X}_2(1:r, :)\texttt{diag}(\mathbf{b}_{21})$
$\mathbf{V}_\tau = \begin{bmatrix} \boldsymbol{\mathcal{V}}_\alpha^* \boldsymbol{\mathcal{V}}_\tau(J_\alpha, :) \\ \boldsymbol{\mathcal{V}}_\beta^* \boldsymbol{\mathcal{V}}_\tau(J_\beta, :) \end{bmatrix}$.
end loop
end loop

Extract the diagonal matrices.
$n_{\max} = \texttt{max}\,\{n_\tau :\ \tau \text{ is a leaf}\}$
$\mathbf{\Omega} = \texttt{zeros}(N, n_{\max})$
loop over leaf boxes τ
$\mathbf{\Omega}(I_\tau, 1 : n_\tau) = \texttt{eye}(n_\tau)$.
end loop
$\mathbf{Y} = \mathbf{A}\mathbf{\Omega} - \mathbf{A}^{(L)}\mathbf{\Omega}$
loop over leaf boxes τ
$\mathbf{D}_\tau = \mathbf{Y}(I_\tau, 1 : n_\tau)$.
end loop

FIG. 5. *A basic scheme for compressing an HBS matrix.*

5.2. A storage efficient scheme for compressing an HBS matrix. The scheme described in section 5.1 assumes that all "long" basis and spanning matrices ($\boldsymbol{\mathcal{U}}_\tau$, $\boldsymbol{\mathcal{V}}_\tau$, $\boldsymbol{\mathcal{Y}}_\tau$, $\boldsymbol{\mathcal{Z}}_\tau$) are stored explicitly, resulting in an $O(k\,N\,\log N)$ storage requirement. We will now demonstrate that, in fact, only $O(k\,N)$ is required.

First, observe that in the HBS framework, we only need to keep at hand the long basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$ for nodes τ on the level ℓ that is currently being processed. In the HODLR compression algorithm in Figure 4, we needed the long basis matrices associated with nodes on coarser levels in order to apply $\mathbf{A}^{(\ell)}$, but in the HBS framework, all we need in order to apply $\mathbf{A}^{(\ell)}$ is the long basis matrices $\{\boldsymbol{\mathcal{U}}_\tau, \boldsymbol{\mathcal{V}}_\tau\}$ *on the level currently being processed*, and then only the short basis matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ for any box τ on a level coarser than ℓ; cf. the algorithm in Figure 13.

Next, observe that the long "spanning" matrices $\boldsymbol{\mathcal{Y}}_\tau$ and $\boldsymbol{\mathcal{Z}}_\tau$ that were introduced

loop over levels $\ell = 0 : (L-1)$

Build the random matrices $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_2$.
$\mathbf{\Omega}_1 = \texttt{zeros}(n, r)$
$\mathbf{\Omega}_2 = \texttt{zeros}(n, r)$
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
$\mathbf{\Omega}_1(I_\alpha, :) = \texttt{randn}(n_\alpha, r)$
$\mathbf{\Omega}_2(I_\beta, :) = \texttt{randn}(n_\beta, r)$
end loop

Apply $\mathbf{A}$ to build the samples for the incoming basis matrices.
$\mathbf{Y}_1 = \mathbf{A}\mathbf{\Omega}_2 - \mathbf{A}^{(\ell)}\mathbf{\Omega}_2$
$\mathbf{Y}_2 = \mathbf{A}\mathbf{\Omega}_1 - \mathbf{A}^{(\ell)}\mathbf{\Omega}_1$

Orthonormalize the sample matrices to build the incoming basis matrices.
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
if (τ is the root)
$\mathbf{Y}_1^{\text{loc}} = \mathbf{Y}_1(I_\alpha, :)$
$\mathbf{Y}_2^{\text{loc}} = \mathbf{Y}_2(I_\beta, :)$
else
$\mathbf{Y}_1^{\text{loc}} = [\mathbf{Y}_1(I_\alpha, :),\ \boldsymbol{\mathcal{U}}_\tau(J_\alpha, :)\texttt{diag}(\mathbf{y}_\tau)]$
$\mathbf{Y}_2^{\text{loc}} = [\mathbf{Y}_2(I_\beta, :),\ \boldsymbol{\mathcal{U}}_\tau(J_\beta, :)\texttt{diag}(\mathbf{y}_\tau)]$
end if
$[\boldsymbol{\mathcal{U}}_\alpha, \mathbf{y}_\alpha, \sim] = \texttt{svd}(\mathbf{Y}_1^{\text{loc}}, r)$.
$[\boldsymbol{\mathcal{U}}_\beta, \mathbf{y}_\beta, \sim] = \texttt{svd}(\mathbf{Y}_2^{\text{loc}}, r)$.
$\mathbf{\Omega}_1(I_\alpha, :) = \boldsymbol{\mathcal{U}}_\alpha$
$\mathbf{\Omega}_2(I_\beta, :) = \boldsymbol{\mathcal{U}}_\beta$
$\mathbf{U}_\tau = \begin{bmatrix} \boldsymbol{\mathcal{U}}_\alpha^* \boldsymbol{\mathcal{U}}_\tau(J_\alpha, :) \\ \boldsymbol{\mathcal{U}}_\beta^* \boldsymbol{\mathcal{U}}_\tau(J_\beta, :) \end{bmatrix}$.
Delete $\boldsymbol{\mathcal{U}}_\tau$.
end loop

Apply $\mathbf{A}^*$ to build the samples for the outgoing basis matrices.
$\mathbf{Z}_1 = \mathbf{A}^*\mathbf{\Omega}_2 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{\Omega}_2$
$\mathbf{Z}_2 = \mathbf{A}^*\mathbf{\Omega}_1 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{\Omega}_1$

Take local SVDs to build incoming basis matrices and sibling interaction matrices.
loop over boxes τ on level ℓ
Let $\{\alpha, \beta\}$ denote the children of box τ.
if (τ is the root)
$\mathbf{Z}_1^{\text{loc}} = \mathbf{Z}_1(I_\alpha, :)$
$\mathbf{Z}_2^{\text{loc}} = \mathbf{Z}_2(I_\beta, :)$
else
$\mathbf{Z}_1^{\text{loc}} = [\mathbf{Z}_1(I_\alpha, :),\ \boldsymbol{\mathcal{V}}_\tau(J_\alpha, :)\texttt{diag}(\mathbf{z}_\tau)]$
$\mathbf{Z}_2^{\text{loc}} = [\mathbf{Z}_2(I_\beta, :),\ \boldsymbol{\mathcal{V}}_\tau(J_\beta, :)\texttt{diag}(\mathbf{z}_\tau)]$
end if
$[\boldsymbol{\mathcal{V}}_\alpha, \mathbf{b}_{21}, \mathbf{X}_1] = \texttt{svd}(\mathbf{Z}_1^{\text{loc}}, r)$.
$[\boldsymbol{\mathcal{V}}_\beta, \mathbf{b}_{12}, \mathbf{X}_2] = \texttt{svd}(\mathbf{Z}_2^{\text{loc}}, r)$.
$\mathbf{z}_\alpha = \mathbf{b}_{21}$
$\mathbf{z}_\beta = \mathbf{b}_{12}$
$\mathbf{B}_{\alpha,\beta} = \mathbf{X}_1(1 : r, :)\texttt{diag}(\mathbf{b}_{12})$
$\mathbf{B}_{\beta,\alpha} = \mathbf{X}_2(1 : r, :)\texttt{diag}(\mathbf{b}_{21})$
$\mathbf{V}_\tau = \begin{bmatrix} \boldsymbol{\mathcal{V}}_\alpha^* \boldsymbol{\mathcal{V}}_\tau(J_\alpha, :) \\ \boldsymbol{\mathcal{V}}_\beta^* \boldsymbol{\mathcal{V}}_\tau(J_\beta, :) \end{bmatrix}$.
Delete $\boldsymbol{\mathcal{V}}_\tau$.
end loop
end loop

Extract the diagonal matrices.
$n_{\max} = \texttt{max}\{n_\tau : \tau \text{ is a leaf}\}$
$\mathbf{\Omega} = \texttt{zeros}(N, n_{\max})$
loop over leaf boxes τ
$\mathbf{\Omega}(I_\tau, 1 : n_\tau) = \texttt{eye}(n_\tau)$.
end loop
$\mathbf{Y} = \mathbf{A}\mathbf{\Omega} - \mathbf{A}^{(L)}\mathbf{\Omega}$
loop over leaf boxes τ
$\mathbf{D}_\tau = \mathbf{Y}(I_\tau, 1 : n_\tau)$.
end loop

FIG. 6. *A storage efficient algorithm for compressing an HBS matrix.*

in section 5.1 do not need to be stored explicitly either. The reason is that these matrices can be expressed in terms of the long basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$. In fact, in the algorithm in Figure 4, we *compute* $\boldsymbol{\mathcal{Y}}_\tau$ and $\boldsymbol{\mathcal{Z}}_\tau$ via the relations

$$\boldsymbol{\mathcal{Y}}_\tau = \boldsymbol{\mathcal{U}}_\tau\, \texttt{diag}(\mathbf{y}_\tau) \quad \text{and} \quad \boldsymbol{\mathcal{Z}}_\tau = \boldsymbol{\mathcal{V}}_\tau\, \texttt{diag}(\mathbf{z}_\tau).$$

Since the long basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$ are available during the processing of level ℓ, we only need to store the short vectors $\mathbf{y}_\tau$ and $\mathbf{z}_\tau$, and can then construct $\boldsymbol{\mathcal{Y}}_\tau$ and $\boldsymbol{\mathcal{Z}}_\tau$ when they are actually needed.

The memory efficient algorithm we described in this section is summarized in Figure 6.

5.3. Adaptive rank determination and conversion to the HBSID format. The schemes presented in sections 5.1 and 5.2 do not adaptively determine the ranks of the off-diagonal blocks being compressed. Instead, every block is factored using a preset uniform rank $r = k + p$ that must be picked to be larger than any actual numerical rank encountered. It is possible to incorporate adaptive rank determination into the scheme, but we found it easier to perform this step in a second sweep that travels through the tree in the opposite direction—from smaller boxes to larger. In this second sweep, we also convert the standard HBS format to the HBSID format, which leads to a slight improvement in storage requirements and improves interpretability of the sibling interaction matrices, as discussed in section 3.3 and Definition 2.

The conversion to the HBSID is a "postprocessing" step, so in what follows, we assume that the compression algorithm in Figure 6 has already been executed so that both the HBS basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$ and the sample matrices $\boldsymbol{\mathcal{Y}}_\tau$ and $\boldsymbol{\mathcal{Z}}_\tau$ are available for every node (these are stored *implicitly* in terms of the short basis matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$, as described in section 5.2).

The first step is to sweep over all leaves τ in the tree. For each leaf, we now have available spanning matrices $\boldsymbol{\mathcal{Y}}_\tau$ and $\boldsymbol{\mathcal{Z}}_\tau$ whose columns span the columns of $\mathbf{A}(I_\tau, I_\tau^{\mathrm{c}})$ and $\mathbf{A}(I_\tau^{\mathrm{c}}, I_\tau)^*$, respectively. In order to find a set of spanning rows $\tilde{I}_\tau^{\mathrm{in}}$ of $\mathbf{A}(I_\tau, I_\tau^{\mathrm{c}})$ and a set of spanning columns $\tilde{I}_\tau^{\mathrm{out}}$ of $\mathbf{A}(I_\tau^{\mathrm{c}}, I_\tau)^*$, all we need to do is compute IDs of the small matrices $\boldsymbol{\mathcal{Y}}_\tau$ and $\boldsymbol{\mathcal{Z}}_\tau$ (cf. Remark 5):

$$[\mathbf{T}_{\mathrm{in}}, J_{\mathrm{in}}] = \mathtt{id}(\boldsymbol{\mathcal{Y}}_\tau^*, \varepsilon) \quad \text{and} \quad [\mathbf{T}_{\mathrm{out}}, J_{\mathrm{out}}] = \mathtt{id}(\boldsymbol{\mathcal{Z}}_\tau^*, \varepsilon). \tag{15}$$

In (15), we give the computational tolerance ε as an input parameter. This reveals the "true" ε-ranks k_{in} and k_{out}. Then the skeleton index vectors for τ are given by

$$\tilde{I}_\tau^{\mathrm{in}} = I_\tau(J_{\mathrm{in}}(1 : k_{\mathrm{in}})) \quad \text{and} \quad \tilde{I}_\tau^{\mathrm{out}} = I_\tau(J_{\mathrm{out}}(1 : k_{\mathrm{out}})).$$

Now define the *subsampled* basis matrices $\mathbf{U}_\tau^{\mathrm{samp}}$ and $\mathbf{V}_\tau^{\mathrm{samp}}$ via

$$\mathbf{U}_\tau^{\mathrm{samp}} = \mathbf{U}(J_{\mathrm{in}}(1 : k_{\mathrm{in}}), :) \quad \text{and} \quad \mathbf{V}_\tau^{\mathrm{samp}} = \mathbf{V}(J_{\mathrm{out}}(1 : k_{\mathrm{out}}), :). \tag{16}$$

Once all leaves have been processed, we can determine the sibling interaction matrices $\mathbf{B}_{\alpha,\beta}^{\mathrm{skel}}$ in the HBSID representation. Let $\{\alpha, \beta\}$ denote a sibling pair consisting of two leaves. First, observe that, by definition,

$$\mathbf{B}_{\alpha,\beta}^{\mathrm{skel}} = \mathbf{A}(\tilde{I}_\alpha^{\mathrm{in}}, \tilde{I}_\beta^{\mathrm{out}}). \tag{17}$$

Next, recall that

$$\mathbf{A}(I_\alpha, I_\beta) = \mathbf{U}_\alpha \mathbf{B}_{\alpha,\beta} \mathbf{V}_\beta^*. \tag{18}$$

Combining (16), (17), and (18), we find that

$$\mathbf{B}_{\alpha,\beta}^{\mathrm{skel}} = \mathbf{U}_\alpha^{\mathrm{samp}} \, \mathbf{B}_{\alpha,\beta} \, (\mathbf{V}_\beta^{\mathrm{samp}})^*.$$

Once all leaves have been processed, we next proceed to the parent nodes. We do this by transversing the tree, going from smaller to larger boxes. When a box τ is processed, its children $\{\alpha, \beta\}$ have already been processed. The key observation is now that if we set $\hat{I}_\tau^{\mathrm{in}} = \tilde{I}_\alpha^{\mathrm{in}} \cup \tilde{I}_\beta^{\mathrm{in}}$ and $\hat{I}_\tau^{\mathrm{out}} = \tilde{I}_\alpha^{\mathrm{out}} \cup \tilde{I}_\beta^{\mathrm{out}}$, then these index vectors form skeletons for τ. These skeletons are inefficient, but by simply compressing the

Execute the compression algorithm described in Figure 6.
loop over leaves τ
  $\mathbf{Y}_{\text{tmp}} = \mathbf{U}_\tau \texttt{diag}(\mathbf{y}_\tau)$
  $\mathbf{Z}_{\text{tmp}} = \mathbf{V}_\tau \texttt{diag}(\mathbf{z}_\tau)$
  $[\mathbf{T}_{\text{in}}, J_{\text{in}}] = \texttt{id}(\mathbf{Y}^*_{\text{tmp}}, \varepsilon)$
  $[\mathbf{T}_{\text{out}}, J_{\text{out}}] = \texttt{id}(\mathbf{Z}^*_{\text{tmp}}, \varepsilon)$
  $\mathbf{U}^{\text{samp}}_\tau = \mathbf{U}_\tau(J_{\text{in}}(1:k_{\text{in}}),:)$.
  $\mathbf{V}^{\text{samp}}_\tau = \mathbf{V}_\tau(J_{\text{out}}(1:k_{\text{out}}),:)$.
end loop

loop over levels $\ell = (L-1):(-1):1$
  loop over boxes τ on level ℓ
    Let $\{\alpha, \beta\}$ denote the children of box τ.
    $\mathbf{U}_{\text{tmp}} = \begin{bmatrix} \mathbf{U}^{\text{samp}}_\alpha & \mathbf{0} \\ \mathbf{0} & \mathbf{U}^{\text{samp}}_\beta \end{bmatrix} \mathbf{U}_\tau$.
    $\mathbf{V}_{\text{tmp}} = \begin{bmatrix} \mathbf{V}^{\text{samp}}_\alpha & \mathbf{0} \\ \mathbf{0} & \mathbf{V}^{\text{samp}}_\beta \end{bmatrix} \mathbf{V}_\tau$.
    $\mathbf{Y}_{\text{tmp}} = \mathbf{U}_{\text{tmp}} \texttt{diag}(\mathbf{y}_\tau)$
    $\mathbf{Z}_{\text{tmp}} = \mathbf{V}_{\text{tmp}} \texttt{diag}(\mathbf{z}_\tau)$
    $[\mathbf{T}_{\text{in}}, J_{\text{in}}] = \texttt{id}(\mathbf{Y}^*_{\text{tmp}}, \varepsilon)$
    $[\mathbf{T}_{\text{out}}, J_{\text{out}}] = \texttt{id}(\mathbf{Z}^*_{\text{tmp}}, \varepsilon)$
    $\mathbf{U}^{\text{samp}}_\tau = \mathbf{U}_{\text{tmp}}(J_{\text{in}}(1:k_{\text{in}}),:)$.
    $\mathbf{V}^{\text{samp}}_\tau = \mathbf{V}_{\text{tmp}}(J_{\text{out}}(1:k_{\text{out}}),:)$.
    $\mathbf{B}^{\text{skel}}_{\alpha,\beta} = \mathbf{U}^{\text{samp}}_\alpha \mathbf{B}_{\alpha,\beta} (\mathbf{V}^{\text{samp}}_\beta)^*$.
    $\mathbf{B}^{\text{skel}}_{\beta,\alpha} = \mathbf{U}^{\text{samp}}_\beta \mathbf{B}_{\beta,\alpha} (\mathbf{V}^{\text{samp}}_\alpha)^*$.
  end loop
end loop
Let $\{\alpha, \beta\}$ denote the children of the root.
$\mathbf{B}^{\text{skel}}_{\alpha,\beta} = \mathbf{U}^{\text{samp}}_\alpha \mathbf{B}_{\alpha,\beta} (\mathbf{V}^{\text{samp}}_\beta)^*$.
$\mathbf{B}^{\text{skel}}_{\beta,\alpha} = \mathbf{U}^{\text{samp}}_\beta \mathbf{B}_{\beta,\alpha} (\mathbf{V}^{\text{samp}}_\alpha)^*$.

FIG. 7. *An algorithm for computing the HBSID representation of a given matrix* $\mathbf{A}$. *This algorithm adaptively determines the ranks of the off-diagonal blocks of* $\mathbf{A}$.

corresponding rows of $\boldsymbol{\mathcal{Y}}_\tau$ and $\boldsymbol{\mathcal{Z}}_\tau$, we can build the skeletons and the interpolation matrices associated with τ. Due to the self-similarity between levels in the HBS representation, the compression is entirely analogous to the compression of a leaf, with the only difference being that the role played by the basis matrices $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$ for a leaf are now played by the subsampled matrices $\mathbf{U}^{\text{samp}}_\tau$ and $\mathbf{V}^{\text{samp}}_\tau$ which represent the restriction of $\boldsymbol{\mathcal{U}}_\tau$ and $\boldsymbol{\mathcal{V}}_\tau$ to the index rows and columns indicated by the index vectors $\hat{I}^{\text{in}}_\tau$ and $\hat{I}^{\text{out}}_\tau$, respectively. The process is summarized in Figure 7.

5.4. Asymptotic complexity. The asymptotic complexity for the HBSID algorithm is very similar to that for the HODLR algorithm. But, since $\mathbf{A}^{(\ell)}$ is now applied using nested basis matrices, we find that

$$T_{\mathbf{A}^{(\ell)}} \sim T_{\text{flop}} \times \left(2^\ell \, k \, \frac{N}{2^\ell} + \sum_{j=0}^{\ell-1} 2^j \, k^2 \right) \sim T_{\text{flop}} \times k \, N.$$

In other words, the complexity of applying $T_{\mathbf{A}^{(\ell)}}$ is now less by a factor of $O(\ell)$, so

$$T_{\text{compress}} \sim T_{\text{mult}} \times k \, \log N + T_{\text{flop}} \times k^2 \, N \, \log N. \tag{19}$$

Comparing (19) to (13), we see that the HBS method beats the HODLR algorithm by a factor of $\log N$.

6. Numerical experiments. In this section, we present results from numerical experiments that substantiate claims on asymptotic complexity made in sections 4.1 and 5.4, and we demonstrate that the practical execution time is very competitive (in other words, that the scaling constants suppressed in the asymptotic analysis are moderate). We investigate three different test problems: In section 6.1 we apply the randomized compression schemes to a discretized boundary integral operator for which other compression techniques are already available. This allows us to benchmark the new algorithms and verify their accuracy. In section 6.2 we demonstrate how the proposed scheme can be used to form a compressed representation of a product of two compressed matrices, thus illustrating how our method can be used to avoid the expensive algorithms currently available for directly multiplying two rank-structured matrices. In section 6.3 we apply the scheme to compress large dense matrices that arise in the classical "nested dissection" or "multifrontal" direct solvers for the sparse matrices arising from finite element or finite difference discretization of elliptic PDEs. The ability to efficiently manipulate such matrices allows for the construction of $O(N)$ complexity direct solvers for the associated sparse linear systems.

For each test problem, we compare two different techniques for computing a data-sparse representation of $\mathbf{A}$: (1) The technique for computing an HODLR-representation in Figure 4. (2) The technique for computing an HBSID-representation in Figure 7. We report the following quantities:

N	Number of DOFs (so that $\mathbf{A}$ is of size $N \times N$).
ε	Requested local precision.
r	Number of random vectors used at each level (r must be larger than maximal rank).
k	Largest *actual* ε-rank encountered during the compression.
N_{matvec}	Number of applications of $\mathbf{A}$ required (so that $N_{\text{matvec}} = (L+1) \times r \sim \log(N) \times r$).
T_{compress}	Time required for compression (in seconds).
T_{net}	Time required for compression, excluding time to apply $\mathbf{A}$ and $\mathbf{A}^*$ (in seconds).
T_{app}	Time required for applying the compressed matrix to a vector (in seconds).
M	Amount of memory required to store $\mathbf{A}$ (in MB).

The reason that we report the time T_{net} (which does not count time spent in the black-box matrix-vector multiplier) is to validate our claims (13) and (19) regarding the asymptotic complexity of the method. To summarize, our predictions are, for the HODLR algorithm,

$$T_{\text{net}} \sim N\,(\log N)^2, \quad T_{\text{app}} \sim N\,\log N, \quad M \sim N\,\log(N),$$

and, for the HBSID algorithm,

$$T_{\text{net}} \sim N\,\log N, \quad T_{\text{app}} \sim N, \quad M \sim N.$$

We also provide a randomized estimate E of the compression error, computed as follows: We drew ten vectors $\{\boldsymbol{\omega}_i\}_{i=1}^{10}$ from a uniform distribution on the unit sphere in $\mathbb{R}^N$. Then E is defined via

$$E = \max_{1 \le i \le 10} \frac{||\mathbf{A}\boldsymbol{\omega}_i - \mathbf{A}_{\text{compressed}}\boldsymbol{\omega}_i||}{||\mathbf{A}\boldsymbol{\omega}_i||}.$$

All experiments were run through MATLAB on an office desktop with 32GB of RAM and an Intel i7-3820 CPU with 4 cores running at 3.6GHz. The technical report [28] upon which this publication is based provides one additional numerical example and tables giving additional details on the examples included here.

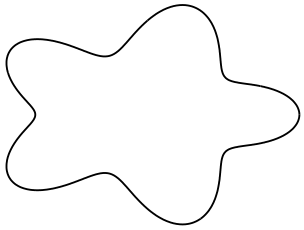

FIG. 8. *Contour* Γ *on which the BIE* (20) *in section* 6.1 *is defined.*

6.1. Compressing a boundary integral equation. Our first numerical example concerns compression of a discretized version of the boundary integral equation (BIE)

$$\frac{1}{2}q(\boldsymbol{x}) + \int_\Gamma \frac{(\boldsymbol{x}-\boldsymbol{y})\cdot\boldsymbol{n}(\boldsymbol{y})}{4\pi|\boldsymbol{x}-\boldsymbol{y}|^2}\, q(\boldsymbol{y})\, ds(\boldsymbol{y}) = f(\boldsymbol{x}), \quad \boldsymbol{x}\in\Gamma, \tag{20}$$

where Γ is the simple curve shown in Figure 8, and where $\boldsymbol{n}(\boldsymbol{y})$ is the outward pointing unit normal of Γ at $\boldsymbol{y}$. The BIE (20) is a standard integral equation formulation of the Laplace equation with boundary condition f on the domain interior to Γ. We discretize the BIE (20) using the Nyström method on N equispaced points on Γ, with the trapezoidal rule as the quadrature. Note that the kernel in (20) is smooth, so the trapezoidal rule has exponential convergence. This problem is artificial in that only about a couple of hundred points are needed to attain full double precision accuracy in the discretization, meaning that the examples we present are *vastly* overresolved. We include it for benchmarking purposes to verify the scaling of the proposed method.

To be precise, the matrix $\mathbf{A}$ used in this numerical experiment is itself an HBS representation of the matrix resulting from discretization of (20), computed using the technique of [29]. To minimize the risk of spurious effects due to both the "exact" and the computed $\mathbf{A}$ being HBS representations, we used a higher precision in computing the reference $\mathbf{A}$, and also a shifted tree structure to avoid having the compressed blocks of our reference $\mathbf{A}$ align with the compressed blocks constructed by the randomized sampling algorithms.

For this experiment, we benchmark the proposed algorithms against (a) the $O(N)$ randomized algorithm described in [27], and (b) the compression technique based on potential theory of [29], run at the same precision as the proposed schemes. The results of the experiments are given in Figure 9.

6.2. Operator multiplication. We next apply the proposed schemes to compute the Neumann-to-Dirichlet operator for the boundary value problem

$$\begin{cases} -\Delta u(\boldsymbol{x}) = 0, & \boldsymbol{x}\in\Omega, \\ \partial_{\boldsymbol{n}} u(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x}\in\Gamma, \end{cases} \tag{21}$$

where Γ is again the contour shown in Figure 8, where Ω is the domain exterior to Γ, and where $\boldsymbol{n}$ is the unit normal vector pointing in the outward direction from Γ. With u the solution of (21), let f denote the restriction of u to Γ, and let T denote the linear operator $T: g \mapsto f$, known as the *Neumann-to-Dirichlet* (NtD) operator. It is well known (see Remark 7) that T can be built explicitly as the product

$$T = S\left(\tfrac{1}{2}I + D^*\right)^{-1}, \tag{22}$$

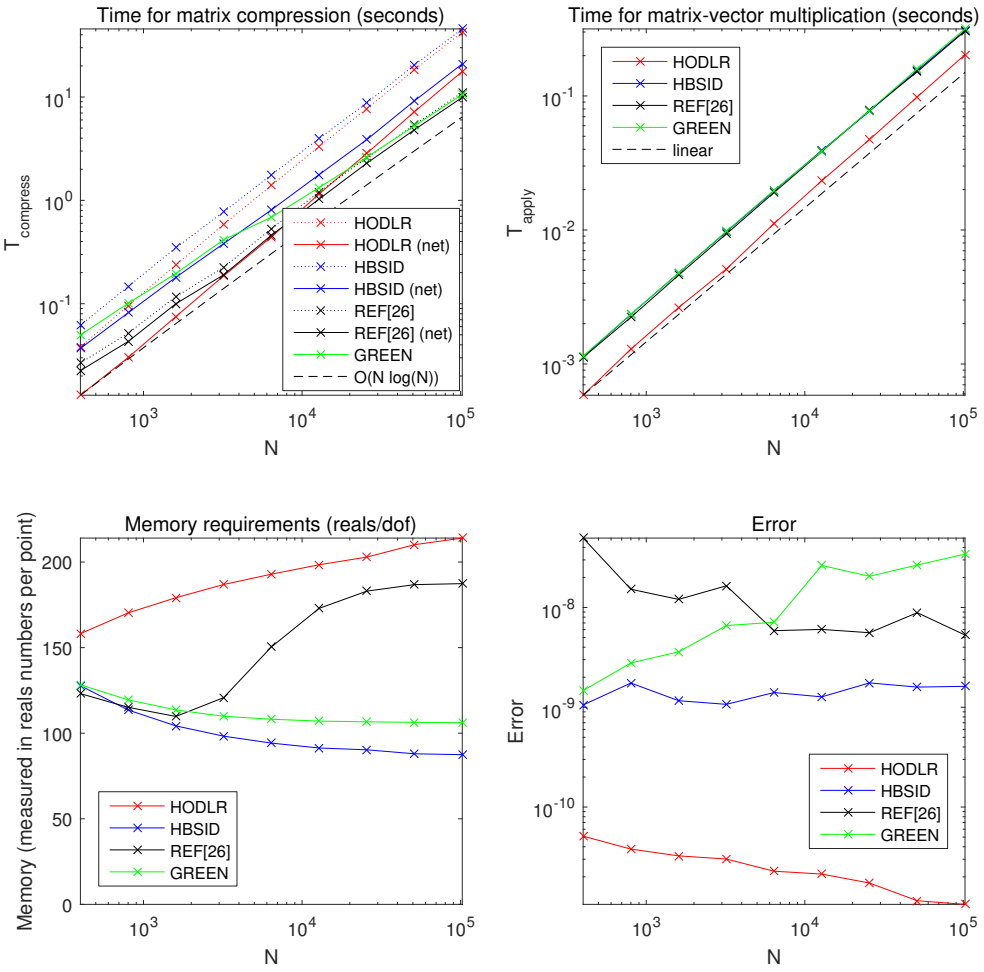

FIG. 9. *Results from compressing the double layer potential on a simple contour in the plane, as described in section* 6.1. *Here* $\varepsilon = 10^{-9}$ *and* $r = 35$.

where S is the *single-layer operator* $[Sq](\boldsymbol{x}) = \int_\Gamma -\frac{1}{2\pi} \log|\boldsymbol{x}-\boldsymbol{y}|\, q(\boldsymbol{y})\, ds(\boldsymbol{y})$, and where D^* is the adjoint of the double-layer operator $[D^*q](\boldsymbol{x}) = \int_\Gamma \frac{\boldsymbol{n}(\boldsymbol{x})\cdot(\boldsymbol{x}-\boldsymbol{y})}{2\pi|\boldsymbol{x}-\boldsymbol{y}|^2}\, q(\boldsymbol{y})\, ds(\boldsymbol{y})$. We form discrete approximations $\mathbf{S}$ and $\mathbf{D}^*$ to S and D^*, and we compute $(\frac{1}{2}\mathbf{I} + \mathbf{D}^*)^{-1}$ using the techniques in [15], with sixth order Kapur–Rokhlin quadrature used to discretize the singular integral operator S. Then we can evaluate a discrete approximation $\mathbf{T}$ to T via

$$\mathbf{T} = \mathbf{S}\left(\tfrac{1}{2}\mathbf{I} + \mathbf{D}^*\right)^{-1}.$$

For this example, we evaluated an additional error metric by testing the computed NtD operator to an exact solution u_{exact} to (21). The function u_{exact} is given as the potential from a collection of five randomly placed charges inside Γ, and then $\mathbf{f}_{\text{exact}}$ and $\mathbf{g}_{\text{exact}}$ are simply the evaluations of u_{exact} and its normal derivative on the quadrature nodes on Γ. Then the new error measure is given by

$$E_{\text{pot}} = \frac{\|\mathbf{f}_{\text{exact}} - \mathbf{T}_{\text{approx}}\,\mathbf{g}_{\text{exact}}\|_{\max}}{\|\mathbf{f}_{\text{exact}}\|_{\max}},$$

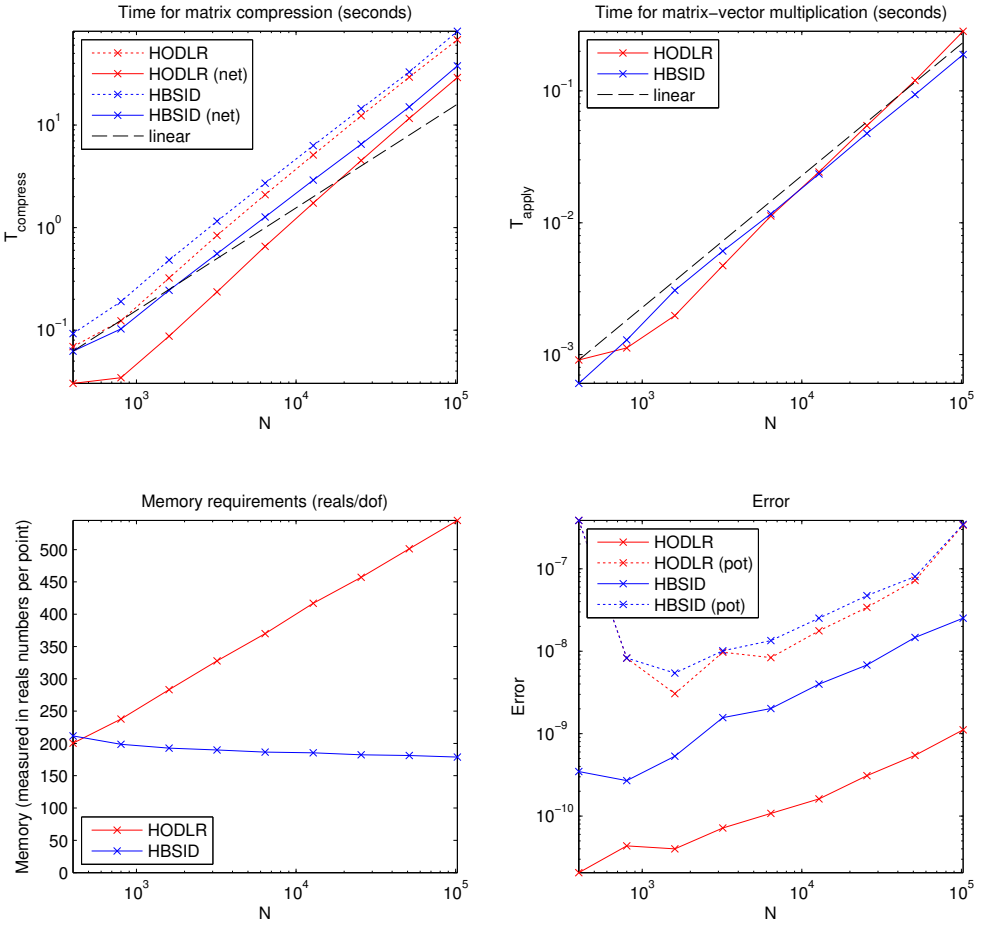

FIG. 10. *Results from compressing the NtD operator for a simple contour in the plane. This operator is given as a product of two compressed operators, as described in section* 6.2. *Here* $\varepsilon = 10^{-9}$ *and* $r = 60$.

where $\|\cdot\|_{\max}$ is the maximum norm, and where $\mathbf{T}_{\text{approx}}$ is the compressed representation of T determined by the randomized sampling scheme proposed. The results are given in Figure 10.

Remark 7. The formula (22) for the NtD operator is derived as follows: We first look for a solution to (21) of the form $u = Sq$. Then it can be shown that q must satisfy $(1/2)q + D^*q = g$. Solving for q and using $f = Sq$, we obtain (22).

6.3. Compression of frontal matrices in nested dissection. Our final example applies the proposed compression schemes to the problem of constructing $O(N)$ direct solvers for the sparse linear systems arising upon the discretization of elliptic PDEs via finite difference or finite element methods. The idea is to build a solver on the classical "nested dissection" scheme of George [13, 11, 10]. In standard implementations, the problem of this direct solver is that it requires the inversion or LU factorization of a set of successively larger dense matrices. However, it has recently been demonstrated that while these matrices are dense, they have an internal

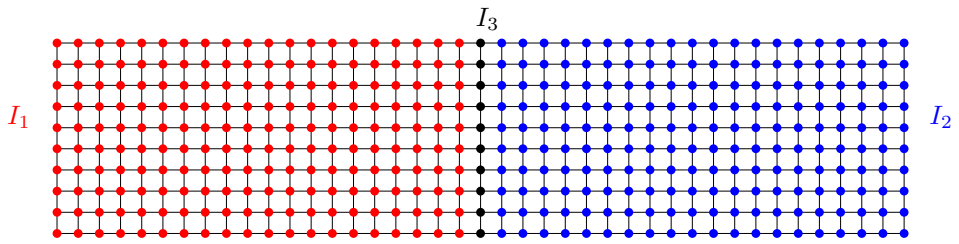

FIG. 11. *Geometry of problem described in section* 6.3. *We consider a grid conduction problem on the grid shown. As* N *is increased, the* width *of the grid is fixed at* 41 *nodes, while the* height *of the grid equals* N.

structure that allows for linear or close to linear time matrix algebra to be executed [23, 34, 26, 14]. In this paper, we test the proposed compression scheme on a set of matrices whose behavior is directly analogous to the matrices encountered in the algorithms of [23, 34, 26, 14]. To be precise, let $\mathbf{B}$ denote the sparse coefficient matrix associated with a grid conduction problem on the grid shown in Figure 11. Each bar has a conductivity that is drawn at random from a uniform distribution on the interval $[1, 2]$. Let I_1, I_2, I_3 denote three index vectors that mark the three regions shown in Figure 11, set $\mathbf{B}_{ij} = \mathbf{B}(I_i, I_j)$ for $i, j = 1, 2, 3$, and then define the $N \times N$ matrix $\mathbf{A}$ via (cf. Remark 8)

$$\mathbf{A} = \mathbf{B}_{33} - \mathbf{B}_{31}\mathbf{B}_{11}^{-1}\mathbf{B}_{13} - \mathbf{B}_{32}\mathbf{B}_{22}^{-1}\mathbf{B}_{23}. \tag{23}$$

In our numerical experiments, the black-box application of $\mathbf{A}$, as defined by (23), was executed using the sparse matrix built-in routines in MATLAB, which relies on UMFPACK [9] for the sparse solves implicit in the application of $\mathbf{B}_{11}^{-1}$ and $\mathbf{B}_{22}^{-1}$. The results are given in Figure 12.

Remark 8. To illustrate the significance of the matrix $\mathbf{A}$ defined by (23) to the LU factorization of a matrix such as $\mathbf{B}$, observe first that the blocks $\mathbf{B}_{12} = \mathbf{B}_{21} = \mathbf{0}$, so that (up to a permutation of the rows and columns)

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{0} & \mathbf{B}_{13} \\ \mathbf{0} & \mathbf{B}_{22} & \mathbf{B}_{23} \\ \mathbf{B}_{31} & \mathbf{B}_{32} & \mathbf{B}_{33} \end{bmatrix}.$$

Next suppose that we can somehow determine the LU factorizations of $\mathbf{B}_{11}$ and $\mathbf{B}_{22}$,

$$\mathbf{B}_{11} = \mathbf{L}_{11}\mathbf{U}_{11} \quad \text{and} \quad \mathbf{B}_{22} = \mathbf{L}_{22}\mathbf{U}_{22}.$$

Then the LU factorization of $\mathbf{B}$ is given by

$$\mathbf{B} = \begin{bmatrix} \mathbf{L}_{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_{22} & \mathbf{0} \\ \mathbf{B}_{31}\mathbf{U}_{11}^{-1} & \mathbf{B}_{32}\mathbf{U}_{22}^{-1} & \mathbf{L}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{11} & \mathbf{0} & \mathbf{L}_{11}^{-1}\mathbf{B}_{13} \\ \mathbf{0} & \mathbf{U}_{22} & \mathbf{L}_{22}^{-1}\mathbf{B}_{23} \\ \mathbf{0} & \mathbf{0} & \mathbf{U}_{33} \end{bmatrix},$$

where $\mathbf{L}_{33}$ and $\mathbf{U}_{33}$ are defined as the LU factors of

$$\mathbf{L}_{33}\mathbf{U}_{33} = \underbrace{\mathbf{B}_{33} - \mathbf{B}_{31}\mathbf{U}_{11}^{-1}\mathbf{L}_{11}^{-1}\mathbf{B}_{13} - \mathbf{B}_{32}\mathbf{U}_{22}^{-1}\mathbf{L}_{22}^{-1}\mathbf{B}_{23}}_{=:\mathbf{A}}. \tag{24}$$

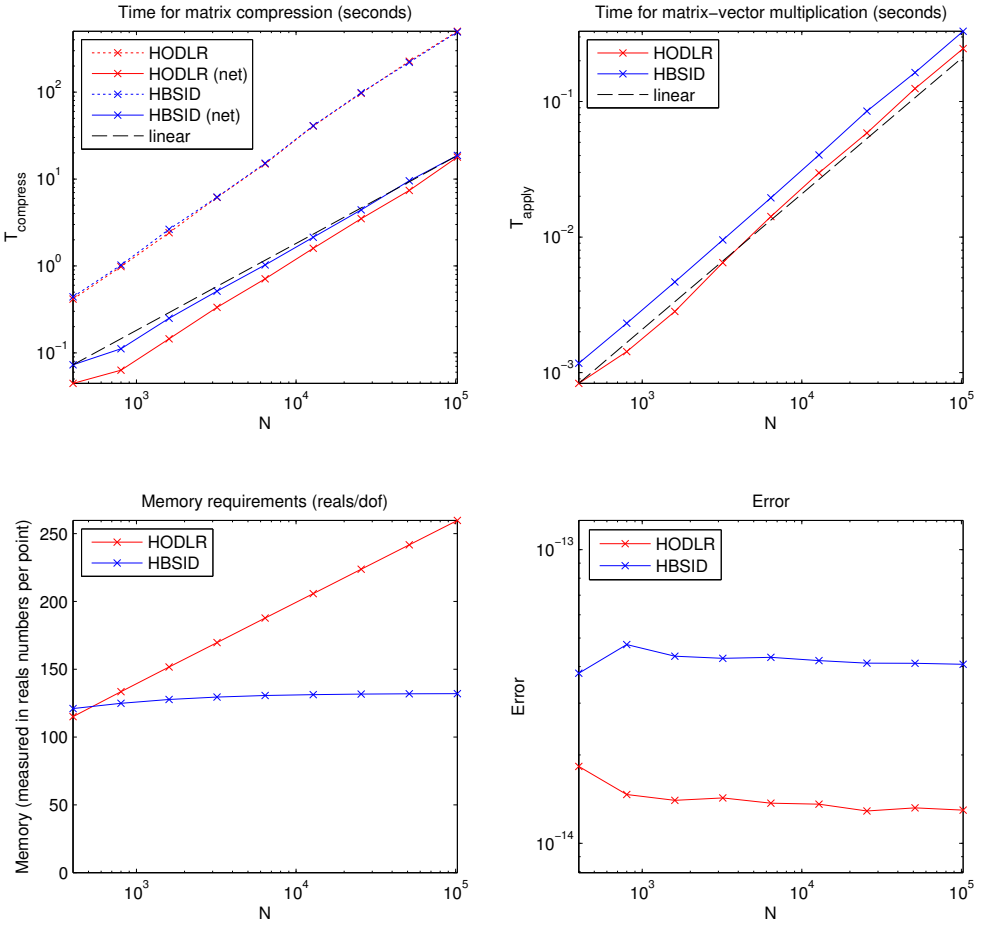

FIG. 12. *Results from the example involving compression of a "frontal matrix" in the nested dissection algorithm, as described in section* 6.3. *Here* $\varepsilon = 10^{-9}$ *and* $r = 25$.

Observe that since the matrices $\mathbf{L}_{11}$, $\mathbf{U}_{11}$, $\mathbf{L}_{22}$, $\mathbf{U}_{22}$ are all triangular, their inverses are inexpensive to apply. To summarize, if one can cheaply evaluate the LU factorization (24), then the task of LU-factoring $\mathbf{B}$ directly reduces to the task of LU-factoring the two matrices $\mathbf{B}_{11}$ and $\mathbf{B}_{22}$, which are of roughly half the size of $\mathbf{B}$. The idea of nested dissection is now to apply this observation recursively to factor $\mathbf{B}_{11}$ and $\mathbf{B}_{22}$. The problem of this scheme has been that in order to evaluate $\mathbf{L}_{33}$ and $\mathbf{U}_{33}$ in (24), one must factorize the *dense* matrix $\mathbf{A}$.

6.4. Summary of observations from numerical experiments.

- The numerical examples support the claims on asymptotic scaling made in sections 4.1 and 5.4. The experiments indicate that while the algorithms for compression to HODLR format have slightly higher asymptotic complexity, the break-even point where HBSID wins is higher than the range of problem sizes tested and appears to be in the range of N between 200 000 and 300 000. The high practical efficiency of HODLR (despite the less favorable asymptotic scaling) is due to the numerical ranks being lower when this format is used.

Build outgoing expansions on level m.
loop over all nodes τ on level m
  $\hat{\mathbf{q}}_\tau = \boldsymbol{\mathcal{V}}_\tau^* \, \mathbf{q}(I_\tau)$
end loop

Build outgoing expansions on levels coarser than m (upward pass).
loop over levels $\ell = (m-1) : (-1) : 1$
  loop over boxes τ on level ℓ
    Let $\{\alpha, \beta\}$ denote the children of τ.
    $\hat{\mathbf{q}}_\tau = \mathbf{V}_\tau^* \begin{bmatrix} \hat{\mathbf{q}}_\alpha \\ \hat{\mathbf{q}}_\beta \end{bmatrix}$.
  end loop
end loop

Build incoming expansions for the children of the root.
Let $\{\alpha, \beta\}$ denote the children of the root node.
$\hat{\mathbf{u}}_\alpha = \mathbf{B}_{\alpha,\beta} \, \hat{\mathbf{q}}_\beta$.
$\hat{\mathbf{u}}_\beta = \mathbf{B}_{\beta,\alpha} \, \hat{\mathbf{q}}_\alpha$.

Build incoming expansions on levels coarser than m (downward pass).
loop over levels $\ell = (m-1) : (-1) : 1$
  loop over boxes τ on level ℓ
    Let $\{\alpha, \beta\}$ denote the children of τ.
    $\hat{\mathbf{u}}_\alpha = \mathbf{B}_{\alpha,\beta} \, \hat{\mathbf{q}}_\beta + \mathbf{U}_\tau(J_\alpha, :) \, \hat{\mathbf{u}}_\tau$.
    $\hat{\mathbf{u}}_\beta = \mathbf{B}_{\beta,\alpha} \, \hat{\mathbf{q}}_\alpha + \mathbf{U}_\tau(J_\beta, :) \, \hat{\mathbf{u}}_\tau$.
  end loop
end loop

Build incoming expansions on level m.
loop over boxes τ on level m
  $\mathbf{u}(I_\tau) = \boldsymbol{\mathcal{U}}_\tau \, \hat{\mathbf{u}}_\tau$
end loop

FIG. 13. *Application of $\mathbf{A}^{(m)}$ in the HBS framework. Given a vector $\mathbf{q}$ of charges, build the vector $\mathbf{u} = \mathbf{A}^{(m)} \, \mathbf{q}$ of potentials.*

- Excellent approximation errors are obtained in every case. In three of the four examples, aggregation of errors over levels is almost imperceptible, and is modest for the fourth example.
- The computational time is in all cases dominated by the time spent in the external black-box multiplier. As a consequence, the primary route by which the proposed algorithm could be improved would be to lessen the number of matrix-vector multiplications required.
- For modest problem sizes, the HODLR algorithm is very fast and easy to use. However, as problems grow large, the memory requirements of the HODLR format become slightly problematic, and the HBSID algorithm gains a more pronounced advantage.
- The example in section 6.1 indicates that when matrix entries can be evaluated explicitly, the previously published algorithm [27] is substantially faster than the techniques presented here, primarily due to the fact that it requires far fewer matrix-vector multiplications.

7. Conclusions. This paper describes two randomized algorithms for constructing a data-sparse representation of a rank-structured matrix $\mathbf{A}$ of size $N \times N$. These algorithms require as inputs (1) a method for applying $\mathbf{A}$ and $\mathbf{A}^*$ to vectors, (2) the hierarchical partitioning of the index set that specifies the structure of the off-diagonal blocks, and (3) an upper bound k on the numerical ranks of the off-diagonal blocks. Numerical experiments demonstrate that the method is computationally efficient, stable, and highly accurate. The execution time is dominated by the time required to apply $\mathbf{A}$ and $\mathbf{A}^*$ to L sets of $O(k)$ vectors, where L is the number of levels in the hierarchical tree (so that, typically, $L \sim \log N$).

REFERENCES

[1] S. Ambikasaran and E. Darve, *An $o(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices*, J. Sci. Comput., 57 (2013), pp. 477–501.

[2] M. Bebendorf, *Approximation of boundary element matrices*, Numer. Math., 86 (2000), pp. 565–589.

[3] M. Bebendorf, *Hierarchical Matrices. A Means to Efficiently Solve Elliptic Boundary Value Problems*, Lect. Notes Comput. Sci. Eng. 63, Springer-Verlag, Berlin, 2008.

[4] S. Börm, *Efficient Numerical Methods for Non-local Operators. $\mathcal{H}^2$-Matrix Compression, Algorithms and Analysis*, EMS Tracts in Math. 14, European Mathematical Society (EMS), Zürich, Switzerland, 2010.

[5] S. Börm and L. Grasedyck, *Hybrid cross approximation of integral operators*, Numer. Math., 101 (2005), pp. 221–249.

[6] T. F. Chan, *Rank revealing QR factorizations*, Linear Algebra Appl., 88–89 (1987), pp. 67–82.

[7] S. Chandrasekaran, M. Gu, and W. Lyons, *A fast adaptive solver for hierarchically semi-separable representations*, Calcolo, 42 (2005), pp. 171–185.

[8] H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404, doi:10.1137/030602678.

[9] T. A. Davis, *Algorithm* 832: *UMFPACK V*4.3*—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Software, 30 (2004), pp. 196–199.

[10] T. A. Davis, *Direct Methods for Sparse Linear Systems*, Fundamentals of Algorithms 2, SIAM, Philadelphia, 2006.

[11] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*, 2nd ed., The Clarendon Press, Oxford University Press, New York, 1989.

[12] K. Frederix and M. V. Barel, *Solving a large dense linear system by adaptive cross approximation*, J. Comput. Appl. Math., 234 (2010), pp. 3181–3195.

[13] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363, doi:10.1137/0710032.

[14] A. Gillman and P. G. Martinsson, *A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method*, SIAM J. Sci. Comput., 36 (2014), pp. A2023–A2046, doi:10.1137/130918988.

[15] A. Gillman, P. Young, and P. G. Martinsson, *A direct solver $o(n)$ complexity for integral equations on one-dimensional domains*, Front. Math. China, 7 (2012), pp. 217–247, doi:10.1007/s11464-012-0188-3.

[16] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 1996.

[17] 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, doi:10.1137/0917055.

[18] W. Hackbusch, *A sparse matrix arithmetic based on H-matrices.* I: *Introduction to H-matrices*, Computing, 62 (1999), pp. 89–108.

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

[20] W. Hackbusch, B. Khoromskij, and S. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, Springer, Berlin, 2002, pp. 9–29.

[21] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288, doi:10.1137/090771806.

[22] D. Huybrechs, *Multiscale and Hybrid Methods for the Solution of Oscillatory Integral Equations*, Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 2006.

[23] S. Le Borne, L. Grasedyck, and R. Kriemann, *Domain-decomposition based $\mathcal{H}$-LU preconditioners*, in Domain Decomposition Methods in Science and Engineering XVI, Lect. Notes Comput. Sci. Eng. 55, Springer, Berlin, 2007, pp. 667–674.

[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] P. G. Martinsson, *Rapid Factorization of Structured Matrices via Randomized Sampling*, preprint, arXiv:0806.2339 [math.NA], 2008.

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

[27] 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, doi:10.1137/100786617.

[28] P. G. Martinsson, *Compressing Rank-Structured Matrices via Randomized Sampling*, preprint, arXiv:1503.07152 [math.NA], 2015.

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

[30] P. G. Martinsson and V. Rokhlin, *An accelerated kernel-independent fast multipole method in one dimension*, SIAM J. Sci. Comput., 29 (2007), pp. 1160–1178, doi:10.1137/060662253.

[31] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A Randomized Algorithm for the Approximation of Matrices*, Tech. report, Yale CS research report YALEU/DCS/RR-1361, Computer Science Department, Yale University, New Haven, CT, 2006.

[32] 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.

[33] J. Xia, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227, doi:10.1137/12087116X.

[34] 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 (2010), pp. 1382–1411, doi:10.1137/09074543X.

[35] 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.