SIAM J. MATRIX ANAL. APPL.
Vol. 33, No. 2, pp. 388–410

ON THE COMPLEXITY OF SOME HIERARCHICAL STRUCTURED MATRIX ALGORITHMS*

JIANLIN XIA†

Abstract. In recent years, hierarchical structured matrices have been widely used in fast solutions of integral equations, PDEs, structured matrix (such as Toeplitz) problems, companion eigenproblems, etc. In this paper, we systematically study the complexity of some hierarchical structured matrix algorithms, in terms of hierarchically semiseparable (HSS) matrices. Several important aspects are considered. We perform detailed complexity analysis for some typical HSS algorithms, with the aid of certain graph techniques. This analysis helps us provide some significant improvements to classical HSS methods. One improvement is to propose more efficient HSS construction and solution algorithms. Another improvement is a recompression procedure which reorthonormalizes some HSS generators and converts noncompact HSS forms to compact ones. A precise theoretical justification of the compactness is also given. The third improvement is to relax the rank requirement in HSS operations. Unlike many classical HSS methods where the appropriate off-diagonal (numerical) ranks are often required to be bounded, we allow the ranks to increase. Certain general rank patterns are proposed, so that similar performance can be achieved with the maximum rank unbounded. These improvements significantly enhance both the efficiency and the applicability of HSS methods. Numerical examples from some applications are included to support the analysis.

Key words. rank structure, hierarchically semiseparable (HSS) matrix, HSS construction and ULV factorization, recompression, relaxation of rank requirement

AMS subject classifications. 65F05, 65F30

DOI. 10.1137/110827788

1. Introduction.

In recent years, rank structured matrices have attracted much attention and are widely used in fast solutions of many numerical problems, including certain PDEs, integral equations, special structured (such as Toeplitz) matrices, companion eigenvalue problems, etc. See [1, 4, 6, 15, 16, 19, 24, 22, 36, 37, 40] for a partial list of references. It is observed that some dense intermediate matrices in these problems have a low-rank property, or their off-diagonal blocks have small ranks or numerical ranks. For example, in the direct solution of elliptic PDEs, the Schur complements in the factorization have this property [2, 11, 33, 35, 41]. Many rank structured matrix representations have been proposed to make use of this property. They provide data-sparse representations or approximations for dense matrices. An important class of rank structured representations are hierarchical structured matrices, such as $\mathcal{H}$-matrices [7, 26, 27, 29], $\mathcal{H}^2$-matrices [5, 28, 30], and hierarchically semiseparable (HSS) matrices [9, 14, 42]. With these structures, a matrix is hierarchically partitioned into blocks at multiple levels. Hierarchical structured representations and operations enable one to reuse and share information across different levels so as to achieve high efficiency.

In this paper, we focus on HSS representations, which have a nice binary tree structure called the HSS tree. HSS forms have been used to develop highly efficient structured solvers and effective preconditioners for both dense and sparse linear sys-

*Received by the editors March 17, 2011; accepted for publication (in revised form) by N. Mastronardi February 21, 2012; published electronically May 17, 2012. This work was supported in part by NSF grants DMS-1115572 and CHE-0957024.
http://www.siam.org/journals/simax/33-2/82778.html
†Department of Mathematics, Purdue University, West Lafayette, IN 47907 (xiaj@math.purdue.edu).

tems [9, 14, 33, 39, 43]. For example, based on HSS methods and the multifrontal method [18, 32], a structured sparse direct solver is proposed in [41] for solving two-dimensional (2D) Poisson equations and 2D elasticity equations with nearly linear complexity and storage.

Classical HSS methods for dense matrices rely on an assumption that appropriate off-diagonal blocks (called HSS blocks) of the dense matrix have small bounded (numerical) ranks. Let n be the order of the matrix and let r be the maximum rank (called HSS rank) of all the HSS blocks. Then the matrix can be converted into an HSS matrix in $O(rn^2)$ flops or less, and a linear system with this HSS coefficient matrix can be solved in $O(r^2n)$ flops with the aid of ULV-type factorizations [9, 14, 42]. Many other fast HSS algorithms are also developed.

However, existing studies on HSS matrices generally do not include detailed complexity analysis. In fact, different HSS algorithms with the same order of complexity may have significantly different prefactors in the flop counts. This indicates the possibility of improving standard HSS algorithms. In particular, HSS construction algorithms as in [42] involve HSS blocks across different levels which makes the flop count nontrivial. Here, we provide a graph technique where an HSS tree is converted into a Pascal triangle so as to count the number of HSS blocks across different levels during the HSS construction. Then, systematic flop counts for a sequence of HSS algorithms can be conveniently conducted. We can also make some useful observations. For example, the HSS construction scheme in [42] for a general dense matrix with certain block sizes costs $6rn^2$ flops, and then it needs $42r^2n$ to compute a ULV factorization and $37rn$ to solve the ULV system. In contrast, while a triangular HSS factorization of the dense matrix costs $11rn^2$ flops, it needs only $10rn$ to solve a triangular HSS system. Such a comparison provides a way to choose different algorithms in different circumstances. These counts can also help study the performance of generalizations of HSS methods such as the sparse structured direct solver in [41].

We then provide several useful strategies for improving existing HSS methods and for developing new fast ones. Based on different internal operations and different ways of accessing matrix data, we can get more efficient HSS construction and factorization algorithms. For example, a modified ULV factorization scheme can improve the cost from $42r^2n$ (as in [14]) to $\frac{74}{3}r^2n$. We also show some HSS methods which specifically takes advantage of symmetry, such as a ULV factorization scheme with $9r^2n$ cost.

As another improvement, a procedure for reorthonormalizing and recompressing noncompact HSS forms is proposed. It is useful in improving the efficiency and reliability of HSS forms arising from indirect construction methods such as those based on randomized sampling [31, 34] or strong rank-revealing LU factorizations [25], and those in sparse factorizations [41]. The recompression procedure also provides a precise theoretical justification of the compactness of HSS representations.

Furthermore, we enhance the flexibility and applicability of HSS methods by relaxing the requirement of bounded HSS ranks. In practical problems such as direct solutions of Helmholtz equations and Toeplitz linear systems, related off-diagonal ranks depend on the size (n) of appropriate dense matrices. In these two cases, the maximum HSS rank is actually $O(\log n)$ [20, 37] under certain conditions. In this paper, instead of requiring the HSS rank to be bounded by a constant, we allow the ranks of the HSS blocks to increase following certain function patterns of the block sizes so that the HSS ranks depend on n. We show that with these rank patterns, we can still get the same or similar order of complexity as before. This allows us to apply regular HSS methods to broader classes of applications. It is also possible to

generalize such rank relaxation techniques to other structured methods.

The remaining sections are organized as follows. In section 2, we briefly review HSS structures. Section 3 presents systematic flop counts for some HSS algorithms with the aid of some graph techniques. Section 4 shows some useful improvements to existing HSS algorithms and demonstrates the performance with numerical examples. An HSS recompression procedure is given in section 5. Techniques for relaxing classical rank requirements in HSS methods are presented in section 6. We draw some concluding remarks in section 7.

2. Review of HSS structures. We first briefly review some concepts of HSS structures. The following notation is used in this paper:

1. Assume $\mathcal{T}$ is a full binary tree in its postordering with $2k-1$ nodes labeled as $i = 1, 2, \ldots, 2k-1$. That is, each node i of $\mathcal{T}$ is either a leaf, or is a nonleaf node with the left and right children c_1 and c_2, respectively, which are postordered so that $c_1 < c_2 < i$. Use $\mathrm{sib}(i)$ and $\mathrm{par}(i)$ to denote the sibling and the parent of a node i, respectively. The root node of $\mathcal{T}$ is $2k-1$, which is written as $\mathrm{root}(\mathcal{T})$.
2. Let A be an $n \times n$ real matrix and let $\mathcal{I} = \{1, 2, \ldots, n\}$ be the set of all row/column indices. Assume $t_i \subset \mathcal{I}$ is a subset of $\mathcal{I}$ with contiguous indices. We let $\mathcal{I} = t_i^c \cup t_i \cup t_i^r$, where all indices in t_i^c are smaller than those in t_i, and all indices in t_i^r are larger than those in t_i.
3. $A|_{t_i \times t_j}$ is the submatrix of A with row index set t_i and column index set t_j.
4. By the compression of a block $A|_{t_i \times t_j}$, we mean a rank-revealing QR (RRQR) factorization or a truncated SVD of $A|_{t_i \times t_j}$. For simplicity, the former is used mostly in this paper, and sometimes, we write $A|_{t_i \times t_j} \approx U_i A|_{\hat{t}_i \times t_j}$, which means that the second factor is still stored in $A|_{t_i \times t_j}$ with row index set $\hat{t}_i$.
5. $\mathrm{diag}(\ldots)$ represents a block diagonal matrix formed by the blocks in $(\ldots)$.

An HSS representation is generally defined recursively [9, 14, 42]. Here, we use a postordered HSS form defined in [42].

DEFINITION 2.1. *Assume A is an $n \times n$ matrix and $\mathcal{I} = \{1, 2, \ldots, n\}$. Let $\mathcal{T}$ be a full binary tree with $2k-1$ nodes labeled as $i = 1, 2, \ldots, 2k-1$, and let $t_i \subset \mathcal{T}$ be an index set associated with each node i of $\mathcal{T}$. We say $\mathcal{T}$ is a* postordered HSS tree *and A is in an* HSS form *if the following conditions hold:*

1. *$\mathcal{T}$ is a postordered full binary tree (with the root $2k-1$ at level 0).*
2. *There is one index set t_i associated with each node i of $\mathcal{T}$ which is defined hierarchically. For each nonleaf node i, $t_{c_1} \cup t_{c_2} = t_i$, $t_{c_1} \cap t_{c_2} = \emptyset$, and $t_{2k-1} = \mathcal{I}$.*
3. *There exist matrices D_i, U_i, V_i, R_i, W_i, B_i (called* HSS generators*) associated with each node i satisfying the following relations for $i = 1, 2, \ldots, 2k-1$:*

$$D_i \equiv A|_{t_i \times t_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^T \\ U_{c_2} B_{c_2} V_{c_1}^T & D_{c_2} \end{pmatrix}, \tag{2.1}$$

$$U_i = \begin{pmatrix} U_{c_1} & \\ & U_{c_2} \end{pmatrix} \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1} & \\ & V_{c_2} \end{pmatrix} \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix}. \tag{2.2}$$

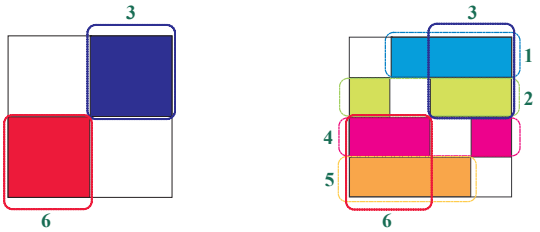
Here, $D_{2k-1} \equiv A$, and U_{2k-1}, V_{2k-1}, R_{2k-1}, W_{2k-1}, B_{2k-1} are empty matrices.

For a nonleaf node i, the generators D_i, U_i, V_i are recursively defined and are not explicitly stored. The HSS form conveniently represents the rank structure of certain off-diagonal blocks as defined next.

DEFINITION 2.2. *Let*

$$A_i^- = \begin{pmatrix} A|_{t_i \times t_i^c} & A|_{t_i \times t_i^r} \end{pmatrix}, \quad A_i^| = \begin{pmatrix} A|_{t_i^c \times t_i} \\ A|_{t_i^r \times t_i} \end{pmatrix}. \tag{2.3}$$

The blocks A_i^- and $A_i^|$ are called the HSS block row and column associated with node i, respectively. (See Figure 2.1.*) The maximum (numerical) rank of all HSS blocks is called the* HSS rank *of A.*



(i) One level of HSS blocks. (ii) Two levels of HSS blocks.

FIG. 2.1. *HSS (off-diagonal) block rows A_i^-, $i = 1, 2, \ldots, 6$.*

Clearly, the columns of U_i form a basis for the column space of A_i^-, and the rows of V_i^T form a basis for the row space of $A_i^|$. Thus, U_i and V_i are called basis matrices in this paper, or cluster bases in [5]. They are often made to have orthonormal columns for good stability.

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

$$A = \begin{pmatrix} D_1 & U_1 B_1 V_2^T & U_1 R_1 B_3 W_4^T V_4^T & U_1 R_1 B_3 W_5^T V_5^T \\ U_2 B_2 V_1^T & D_2 & U_2 R_2 B_3 W_4^T V_4^T & U_2 R_2 B_3 W_5^T V_5^T \\ U_4 R_4 B_6 W_1^T V_1^T & U_4 R_4 B_6 W_2^T V_2^T & D_4 & U_4 B_4 V_5^T \\ U_5 R_5 B_6 W_1^T V_1^T & U_5 R_5 B_6 W_2^T V_2^T & U_5 B_5 V_4^T & D_5 \end{pmatrix}. \tag{2.4}$$

See Figure 2.2. If A is symmetric, we can set [42]

$$D_i = D_i^T, \quad V_i = U_i, \quad B_{\mathrm{sib}(i)} = B_i^T. \tag{2.5}$$

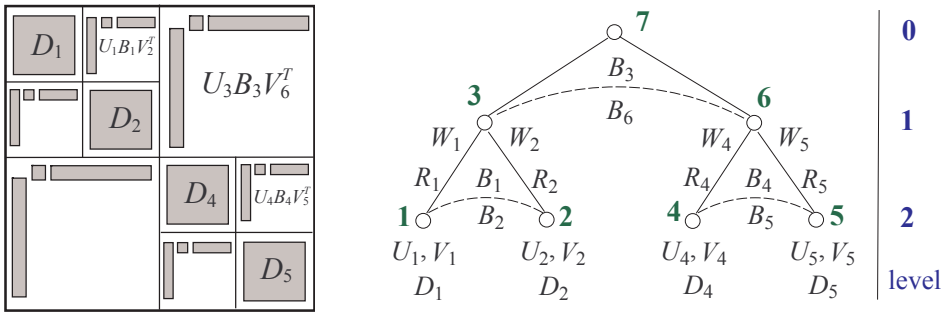


FIG. 2.2. *HSS matrix pattern for* (2.4) *and the corresponding HSS tree.*

3. Complexity of hierarchical construction and solution schemes. In hierarchical compression schemes for computing structured matrix representations and their factorizations, it often involves simultaneous compression of blocks across different levels [14, 28, 42, 43]. Here, in the context of HSS representations, we present an innovative way of counting the detailed complexity of such compression with the aid of some graph results. These counts are hardly done in [14, 42, 43] and most other HSS literature. For convenience, the flop counts of some basic matrix operations are listed in Table 3.1. They can be found, say, in [17, 23].

TABLE 3.1
Flop counts of some basic matrix operations, with low order terms dropped. The RRQR factorization we use is based on the modified Gram–Schmidt algorithm with column pivoting as in [23].

Operation	Flops
QR factorization of an $m \times q$ tall matrix ($m > q$)	$2q^2\left(m - \frac{q}{3}\right)$
Product of the Q factor and an $m \times r$ matrix	$2rq(2m - q)$
Product of an $m \times q$ matrix and a $q \times r$ matrix	$2mqr$
LU factorization of an $m \times m$ matrix	$\frac{2}{3}m^3$
Solution of an order m triangular system $Lx = b$	m^2
RRQR factorization of an $m \times q$ matrix with rank r	$4mqr - 2r^2(m+q) + \frac{4}{3}r^3$

3.1. An analytical result for binary trees. In hierarchical compression schemes, later compression steps are usually designed to take advantage of compressed forms from previous steps so as to improve the efficiency. Here, we briefly rephrase the major steps of the HSS construction algorithm in [42] with more systematic notation in [43] and then show the detailed complexity analysis. This is done with the aid of the following definition given in [43].

DEFINITION 3.1. *The* visited set $\mathcal{V}_i$ *associated with a node* i *of a postordered binary tree* $\mathcal{T}$ *is*

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

where $\mathrm{pred}(i)$ *is the* set of predecessors *associated with node* i *given by*

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

See Figure 3.3(b) below for an example. Clearly, $\mathcal{V}_i$ is the set of visited nodes (before i) whose siblings have not been visited. As pointed out in [43], the set $\mathcal{V}_i$ essentially corresponds to a stack which is often used in the postordering traversal of binary trees such as elimination trees in the multifrontal method [18, 32] and HSS trees in HSS methods. This stack is obtained as follows. For $i = 1, 2, \ldots$, if i is a left node, push i onto the stack. Otherwise, pop a node from the top of the stack. Then before the visit of i, the nodes in the stack form $\mathcal{V}_i$. For convenience, we denote the elements of $\mathcal{V}_i$ by

$$\mathcal{V}_i = \{j_1, j_2, \ldots, j_s \mid j_1 < j_2 < \cdots < j_{s_i}\}, \tag{3.1}$$

where s_i is the cardinality of $\mathcal{V}_i$ (later, we sometimes write s_i as s to simplify the subscripts when no confusion is caused).

The use of $\mathcal{V}_i$ helps clearly describe the HSS construction algorithm, and s_i will be involved in the flop count. According to the definition, each $j \in \mathcal{V}_i$ corresponds

to a right node $\mathrm{sib}(j) \in \mathrm{pred}(i)$ and also a right edge in $\mathbf{p}_{\mathrm{root}(\mathcal{T})\to i}$, the path from $\mathrm{root}(\mathcal{T})$ to i. (A right edge is an edge connecting a nonleaf node with its right child.) Then it is not hard to verify the following lemma.

LEMMA 3.2. *For $\mathcal{V}_i$ in Definition* 3.1 *with cardinality s_i as in* (3.1)*, we have the following:*

1. $\mathrm{pred}(i) = \{v \mid \mathrm{sib}(v) \in \mathcal{V}_i\} \cup \mathrm{pred}(j_1)$*, where j_1 is given in* (3.1)*.*
2. $s_i = s_{c_1} = s_{c_2} - 1$ *for c_1 and c_2, the left and right children of i, respectively.*
3. s_i *is equal to the number of right edges in* $\mathbf{p}_{\mathrm{root}(\mathcal{T})\to i}$*.*

Based on this lemma, we have the following result.

THEOREM 3.3. *Assume the HSS tree $\mathcal{T}$ is a perfect binary tree with the root at level* 0*. Let s_i be the cardinality of $\mathcal{V}_i$ as in* (3.1)*. Then at tree level l, a node i has possible s_i value $j \in \{1, 2, \ldots, l\}$, and the number of nodes i with s_i being the same value $s_i \equiv j$ is $f_j^l = \binom{l}{j}$ for $j = 1, 2, \ldots, l$.*

Proof. For a node i at level l, there are at most l right edges in $\mathbf{p}_{\mathrm{root}(\mathcal{T})\to i}$. Then the third statement in Lemma 3.2 indicates that the possible values of s_i are $\{1, 2, \ldots, l\}$.

To find f_j^l for a fixed $j = 1, 2, \ldots, l$, we rearrange $\mathcal{T}$ into a new graph $\mathcal{G}$ by merging certain nodes. That is, at each level l, all the nodes with the same s_i value j (Figure 3.1) are merged into one single node, which is the jth node (from the left) at level l of $\mathcal{G}$. According to Lemma 3.2, this can be performed by merging all nodes i with the same number of right edges in $\mathbf{p}_{\mathrm{root}(\mathcal{T})\to i}$. Then there are l nodes at level l of $\mathcal{G}$. Note that all paths in $\mathcal{G}$ from its root (the top node) to the jth node at level l have the same number of right paths. Thus, $\mathbf{p}_{\mathrm{root}(\mathcal{T})\to i}$ in $\mathcal{T}$ is transformed into a path in $\mathcal{G}$. See Figure 3.2(a).

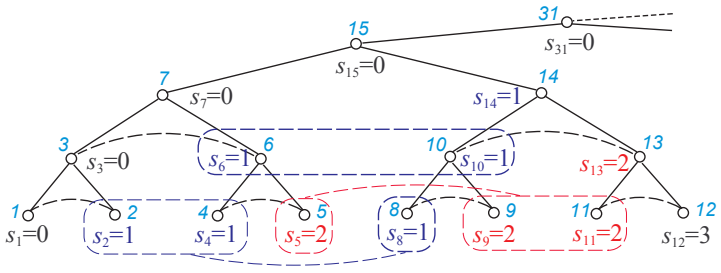
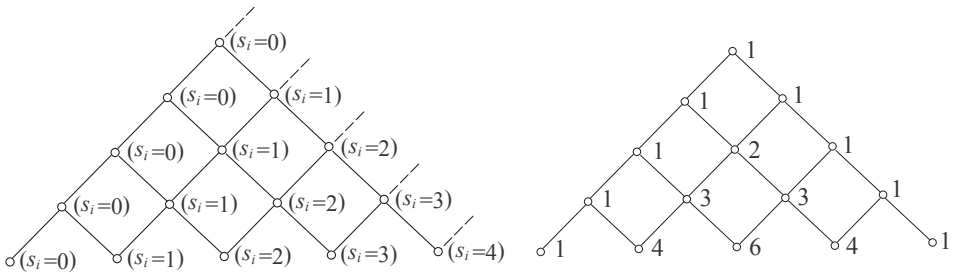


FIG. 3.1. *s_i values for the nodes i of an HSS tree, and the nodes to be merged.*

Therefore, f_j^l is simply the number of top-down paths from the root to the jth node at level l of the new graph. It is known that this number is the binomial coefficient $\binom{l}{j}$, with the aid of the corresponding Pascal triangle as illustrated in Figure 3.2(b). ∎

3.2. Complexity of HSS construction. Here, we show systematic complexity analysis for HSS constructions using the results in the previous subsection. We reformulate the major steps of the algorithm in [42] with clearer notation. The algorithm traverses the HSS tree $\mathcal{T}$ following the postordering and hierarchically compresses the HSS blocks A_i^- and $A_i^|$ in (2.3), where $A|_{t_i \times t_i^c}$ and $A|_{t_i^c \times t_i}$ have been partially compressed in previous steps. By ignoring appropriate U, V basis matrices, we need only to compress certain blocks Ψ_i and Φ_i as in Figure 3.3 instead of $A|_{t_i \times t_i^c}$ and $A|_{t_i^c \times t_i}$, respectively. This is explained as follows. For simplicity, we mainly consider A_i^-. We assume r is the maximum numerical rank of all the HSS blocks of A for a

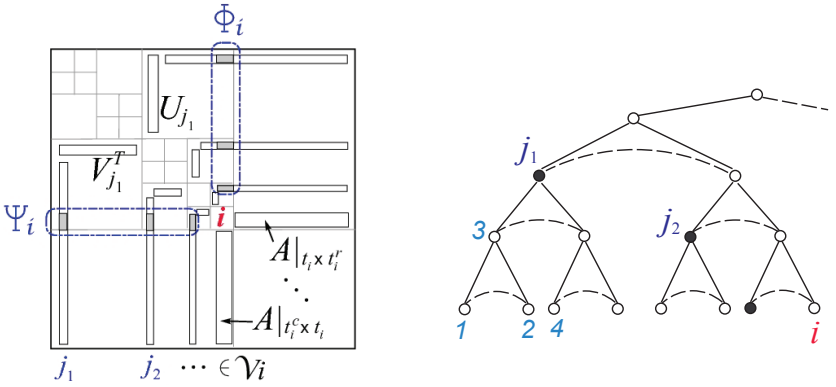
given relative tolerance τ. (Note that, for different τ, the numerical rank r may vary. Sometimes, we may also specify a fixed r when we compress an HSS block or truncate off-diagonal singular values. In this paper, for convenience, we use a single symbol r.)



(a) $\mathcal{G}$ after merging the nodes in Figure 3.1.

(b) A sample Pascal triangle (when Figure 3.1 has 31 nodes).

FIG. 3.2. *Merging nodes in $\mathcal{T}$ so as to find f_j^l with the aid of a Pascal triangle.*



(a) Block compression in HSS construction.

(b) $\mathcal{V}_i = \{j_1, j_2, \dots\}$ (solid nodes) in $\mathcal{T}$.

FIG. 3.3. *How previously compressed blocks participate in later compression with the assistance of the visited set $\mathcal{V}_i$.*

For node $i = 1$, compute the following compression or RRQR factorization:

$$A_i^- = A|_{t_1 \times t_1^r} \approx U_1 A^{(1)}|_{\hat{t}_1 \times t_1^r}, \quad A_i^| = A|_{t_1^r \times t_1}^T \approx V_1 A^{(1)}|_{t_1^r \times \hat{t}_1}^T.$$

For node $i > 1$, the compression is done hierarchically. If i is a leaf, by recursion, $A|_{t_i \times t_i^c}$ is in a compressed form $A|_{t_i \times t_i^c} \approx \Psi_i \tilde{V}_i^T$, where $\tilde{V}_i$ is given by existing V basis matrices, and Ψ_i is the contribution from the previous $i-1$ compression steps, associated with nodes $j_1, j_2, \dots, j_s$ in (3.1) due to the hierarchy (Figure 3.3):

$$\Psi_i = \begin{pmatrix} A^{(i-1)}|_{t_i \times \hat{t}_{j_1}} & \cdots & A^{(i-1)}|_{t_i \times \hat{t}_{j_s}} \end{pmatrix}. \tag{3.2}$$

Thus, the compression of the HSS block row $A_i^- \approx \begin{pmatrix} \Psi_i \tilde{V}_i^T & A|_{t_i \times t_i^r} \end{pmatrix}$ can be effectively done on $\begin{pmatrix} \Psi_i & A|_{t_i \times t_i^r} \end{pmatrix}$, where we ignore a matrix $\mathrm{diag}(\tilde{V}_i, I)$ which has orthonormal columns. Compute an RRQR factorization

$$\begin{pmatrix} \Psi_i & A|_{t_i \times t_i^r} \end{pmatrix} \approx U_i \begin{pmatrix} \begin{pmatrix} A^{(i)}|_{\hat{t}_i \times \hat{t}_{j_1}} & \cdots & A^{(i)}|_{\hat{t}_i \times \hat{t}_{j_s}} \end{pmatrix} & A|_{\hat{t}_i \times t_i^r} \end{pmatrix}, \tag{3.3}$$

where the column partition of $\begin{pmatrix} A^{(i)}|_{\hat{t}_i\times\hat{t}_{j_1}} & \cdots & A^{(i)}|_{\hat{t}_i\times\hat{t}_{j_s}} \end{pmatrix}$ follows that of Ψ_i in (3.2). The corresponding HSS block column is similarly compressed to get V_i.

If i is a nonleaf node, suppose c_1 and c_2 are the left and right children of i, respectively. Let

$$\Psi_i = \begin{pmatrix} \Psi_{c_1} \\ \Psi_{c_2} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} A^{(i-1)}|_{\hat{t}_{c_1}\times\hat{t}_{j_1}} & \cdots & A^{(i-1)}|_{\hat{t}_{c_1}\times\hat{t}_{j_s}} \end{pmatrix} \\ \begin{pmatrix} A^{(i-1)}|_{\hat{t}_{c_2}\times\hat{t}_{j_1}} & \cdots & A^{(i-1)}|_{\hat{t}_{c_2}\times\hat{t}_{j_s}} \end{pmatrix} \end{pmatrix}. \tag{3.4}$$

Compute the compression

$$\begin{pmatrix} \Psi_{c_1} & A^{(i-1)}|_{\hat{t}_{c_1}\times t_i^r} \\ \Psi_{c_2} & A^{(i-1)}|_{\hat{t}_{c_2}\times t_i^r} \end{pmatrix} \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} A^{(i)}|_{\hat{t}_i\times\hat{t}_{j_1}} & \cdots & A^{(i)}|_{\hat{t}_i\times\hat{t}_{j_s}} \end{pmatrix} & A^{(i)}|_{\hat{t}_i\times t_i^r} \end{pmatrix}, \tag{3.5}$$

where the column partition of $\begin{pmatrix} A^{(i)}|_{\hat{t}_i\times\hat{t}_{j_1}} & \cdots & A^{(i)}|_{\hat{t}_i\times\hat{t}_{j_s}} \end{pmatrix}$ follows that of Ψ_i in (3.4). Similarly, we compress the associated HSS block column to get $\begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix}$. In addition, we set

$$B_{c_1} = A^{(i-1)}|_{\hat{t}_{c_1}\times\hat{t}_{c_2}}, \quad B_{c_2} = A^{(i-1)}|_{\hat{t}_{c_2}\times\hat{t}_{c_1}}. \tag{3.6}$$

Note $c_2 = i - 1$.

The algorithm is summarized in Algorithm 1.

ALGORITHM 1 (*HSS construction*).

`for` nodes (separators) $i = 1, 2, \ldots, \text{root}(\mathcal{T})$,
  `if` i is a leaf
    – (*Row compression*) Form $\begin{pmatrix} \Psi_i & A|_{t_i\times t_i^r} \end{pmatrix}$ with (3.2) and compute U_i as in (3.3)
    – (*Column compression*) Compute V_i similarly
  `else`
    – (*Row compression*) Form $\begin{pmatrix} \Psi_{c_1} & A^{(i-1)}|_{\hat{t}_{c_1}\times t_i^r} \\ \Psi_{c_2} & A^{(i-1)}|_{\hat{t}_{c_2}\times t_i^r} \end{pmatrix}$ with (3.4) and compute $\begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix}$ as in (3.5)
    – (*Column compression*) Compute $\begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix}$ similarly
    – Form B_{c_1} and B_{c_2} as in (3.6) for the children c_1, c_2 of i
  `end`
`end`

To simplify the complexity count, we assume that the HSS tree $\mathcal{T}$ is a perfect binary tree with $2k-1$ nodes (k leaves) and $L \approx \log_2 k$ levels, the numerical rank of any HSS block is $r \ll n$, and any bottom/leaf level HSS block row dimension is $m = O(r)$. To avoid unnecessary compression for full rank blocks at the leaf level, we let $m > r$. Also let n_i be the column size of $A|_{t_i\times t_i^r}$.

We count the costs levelwise. Two useful formulas are needed:

$$\sum_{i \text{ at level } l} s_i = \sum_{j=1}^{l} j\binom{l}{j} = \frac{1}{2} l 2^l, \tag{3.7}$$

$$\sum_{i \text{ at level } l} n_i = \sum_{j=1}^{2^l} \left(k - j\frac{k}{2^l}\right) m = \frac{1}{2} km(2^l - 1). \tag{3.8}$$

The formula (3.7) is a direct result of Theorem 3.3, and (3.8) is based on the fact that each level l has 2^l nodes with n_i values $(k - j\frac{k}{2^l})m,\ j = 1, 2, 3, \ldots, 2^l$.

First, we count the costs associated with all leaves. Clearly, $\begin{pmatrix} \Psi_i & A|_{t_i \times t_i^r} \end{pmatrix}$ in (3.3) has size $m \times (rs_i + n_i)$. From Table 3.1, (3.3) costs $4m(s_i r + n_i)r - 2r^2(m + (s_i r + n_i)) + \frac{4}{3}r^3$ flops for each leaf i. The total for all leaves is then

$$\begin{aligned}
\mathcal{C}_1 &= \sum_{i:\ \text{leaf}} \left(4m(s_i r + n_i)r - 2r^2(m + (s_i r + n_i)) + \frac{4}{3}r^3 \right) \\
&= (4mr^2 - 2r^3) \sum_{i:\ \text{leaf}} s_i + (4mr - 2r^2) \sum_{i:\ \text{leaf}} n_i + \left(\frac{4}{3}r^3 - 2mr^2 \right) 2^L \\
&\approx 2rk^2m^2 - r^2k^2m,
\end{aligned}$$

where (3.7) and (3.8) are used together with $k \approx 2^L$, and some low order terms are dropped since $r, m \ll n$.

Next, consider nonleaf nodes. The left-hand side of (3.5) has size $2r \times (rs_i + n_i)$. Thus, (3.5) costs $8r^2(rs_i + n_i) - 2r^2(2r + (rs_i + n_i)) + \frac{4}{3}r^3$ for each nonleaf node i. The total block-row compression cost for all nonleaf nodes can be similarly counted:

$$\mathcal{C}_2 = \sum_{l=1}^{L-1} \sum_{i \text{ at level } l} \left(8r^2(rs_i + n_i) - 2r^2(2r + (rs_i + n_i)) + \frac{4}{3}r^3 \right) \approx 3r^2k^2m.$$

The cost for all block-column compression or the computation of V, W generators is also $\mathcal{C}_1 + \mathcal{C}_2$. Thus, the total HSS construction cost is

$$\mathcal{C}_{\text{constr}} = 2\left(\mathcal{C}_1 + \mathcal{C}_2\right) \approx 2\left(2rk^2m^2 - r^2k^2m + 3r^2k^2m\right) = 4\left(1 + \frac{r}{m}\right) rn^2.$$

For example, if $m = 2r$, we have

$$\mathcal{C}_{\text{constr}} \approx 6rn^2.$$

Note that the storage of this HSS matrix can be easily verified to be

$$\mathcal{S}_{\text{mem}} \approx (m^2 + 2mr + 2r^2)\frac{n}{m}.$$

If $m = 2r$, we have $\mathcal{S}_{\text{mem}} \approx 5rn$.

3.3. Complexity of HSS factorization and solution. For a given HSS form, we can quickly compute ULV-type factorizations and solutions [14, 42]. The fundamental idea of ULV HSS factorizations can be illustrated in terms of a block 2×2 HSS form

$$\begin{pmatrix} D_j & U_j B_j V_i^T \\ U_i B_i V_j^T & D_i \end{pmatrix} \begin{matrix} m \\ m \end{matrix}, \tag{3.9}$$

where $j = \text{sib}(i)$, the generators U_i and V_i have sizes $m \times r$, and B_i has size $r \times r$. Compute a QL factorization of U_i and update D_i as $\hat{D}_i$:

$$U_i = Q_i \begin{pmatrix} 0 \\ \tilde{U}_i \end{pmatrix} \begin{matrix} m - r \\ r \end{matrix}, \quad \hat{D}_i = Q_i^T D_i. \tag{3.10}$$

U_j is similarly factorized and D_j is updated. We have

$$\begin{pmatrix} Q_j^T & \\ & Q_i^T \end{pmatrix} \begin{pmatrix} D_j & U_j B_j V_i^T \\ U_i B_i V_j^T & D_i \end{pmatrix} = \begin{pmatrix} \hat{D}j & \begin{pmatrix} 0 \\ \tilde{U}_j \end{pmatrix} B_j V_i^T \\ \begin{pmatrix} 0 \\ \tilde{U}_i \end{pmatrix} B_i V_j^T & \hat{D}_i \end{pmatrix}. \tag{3.11}$$

Then compute a QR factorization of $\hat{D}_i^T$ and update V_i as

$$\hat{D}_i^T = W_i \tilde{D}_i^T, \quad \tilde{V}_i = W_i^T V_i. \tag{3.12}$$

$\hat{D}_j^T$ is similarly factorized and V_j is updated. We have

$$\begin{aligned} &\begin{pmatrix} Q_j^T & \\ & Q_i^T \end{pmatrix} \begin{pmatrix} D_j & U_j B_j V_i^T \\ U_i B_i V_j^T & D_i \end{pmatrix} \begin{pmatrix} W_j & \\ & W_i \end{pmatrix} \\ &= \begin{pmatrix} \begin{pmatrix} \tilde{D}_{j;1,1} & 0 \\ \tilde{D}_{j;2,1} & \tilde{D}_{j;2,2} \end{pmatrix} & \begin{pmatrix} 0 \\ \tilde{U}_j \end{pmatrix} B_j \begin{pmatrix} \tilde{V}_{i;1}^T & \tilde{V}_{i;2}^T \end{pmatrix} \\ \begin{pmatrix} 0 \\ \tilde{U}_i \end{pmatrix} B_i \begin{pmatrix} \tilde{V}_{j;1}^T & \tilde{V}_{j;2}^T \end{pmatrix} & \begin{pmatrix} \tilde{D}_{i;1,1} & 0 \\ \tilde{D}_{i;2,1} & \tilde{D}_{i;2,2} \end{pmatrix} \end{pmatrix}, \end{aligned} \tag{3.13}$$

where $\tilde{D}_i$, $\tilde{V}_i$, $\tilde{D}_j$, and $\tilde{V}_j$ are partitioned conformably. Then the blocks $\tilde{D}_{j;1,1}$ and $\hat{D}_{i;1,1}$ can be eliminated, and the rest of the blocks are merged into a new matrix. That is, nodes j and i are eliminated, and their parent node p becomes a leaf with generators

$$D_p = \begin{pmatrix} \tilde{D}_{j;2,2} & \tilde{U}_j B_j \tilde{V}_{i;2}^T \\ \tilde{U}_i B_i \tilde{V}_{j;2}^T & \tilde{D}_{i;2,2} \end{pmatrix}, \quad U_p = \begin{pmatrix} \tilde{U}_j R_j \\ \tilde{U}_i R_i \end{pmatrix}, \quad V_p = \begin{pmatrix} \tilde{V}_{j;2} W_j \\ \tilde{V}_{i;2} W_i \end{pmatrix}. \tag{3.14}$$

After this step, we have a smaller HSS form and the elimination continues recursively. This is summarized in Algorithm 2.

ALGORITHM 2 (*HSS ULV factorization*).

`for` nodes (separators) $i = 1, 2, \ldots, \text{root}(\mathcal{T}) - 1$
  `if` i is a leaf
    – (*Introducing zeros into off-diagonal blocks*) Compute a QL factorization of U_i and update D_i as in (3.10)
    – (*Introducing zeros into diagonal blocks*) Compute a QR factorization of $\hat{D}_i^T$ and update V_i as in (3.12)
    – (Partial elimination) Eliminate $\tilde{D}_{i;1,1}$ in (3.13)
  `else`
    – (*Merging*) Form D_i, U_i, V_i as in (3.14) (with j, i, and p in (3.14) replaced by c_1, c_2, and i, respectively, where c_1, c_2 are the children of i)
  `end`
`end`

To get the complexity of this ULV factorization, we list the individual costs of the operations associated with each node in Table 3.2, where Table 3.1 is used.

Then it can be easily verified that the total ULV factorization cost is about

$$\mathcal{C}_{\text{fact}} \approx \frac{4}{3} m^2 n + 6rmn + \frac{148}{3} kr^3. \tag{3.15}$$

TABLE 3.2
Operations associated with each leaf of $\mathcal{T}$ in the HSS factorization scheme described in this section, where some low order terms are dropped. For a nonleaf node, replace m in the table by $2r$.

Operation	Flops	Times needed
QR factorization of a size $m \times r$ matrix	$2r^2\left(m - \frac{r}{3}\right)$	$\times 1$
Product of this Q factor and an $m \times m$ matrix	$2rm(2m - r)$	$\times 1$
QR factorization of a size $m \times m$ matrix	$\frac{2}{3}m^3$	$\times 1$
Product of this Q factor and an $m \times r$ matrix	$2rm^2$	$\times 1$
Product of two $r \times r$ matrices in the merge process	$2r^3$	$\times 4$

When $m = 2r$, we have $\mathcal{C}_{\text{fact}} \approx 42r^2n$, which is slightly tighter than the bound $46r^2n$ in [14]. In such a situation, it can be similarly shown that the ULV factors can be used to solve a system with one right-hand side in $\mathcal{C}_{\text{sol}} = 37rn$ flops. (Later, the complexity of other algorithms is similarly obtained and the details are omitted.)

Another HSS solution method is to directly factorize a dense matrix into triangular HSS factors. The case of symmetric positive definite matrices is discussed in [43]. The case of general nonsymmetric matrices can be considered in a way similar to the HSS construction algorithm. The costs of the dense-to-triangular-HSS factorization and the triangular HSS solution by substitution are

$$\mathcal{C}_{\text{fact}} \approx 6rn^2 + mn^2 + 6kr^2n \quad \text{and} \quad \mathcal{C}_{\text{sol}} \approx mn + 4rn + 8kr^2,$$

respectively.

The complexity of these algorithms with $m = 2r$ is summarized in Table 3.3.

TABLE 3.3
Complexity of some HSS algorithms (with the low order terms dropped) when $m = 2r$, where all leaf level HSS block rows have row sizes m, and the HSS rank of the matrix is r.

Operation	Complexity
HSS construction (bottom-up)	$6rn^2$
ULV HSS factorization	$42r^2n$
ULV HSS solution	$37rn$
Dense-to-triangular-HSS factorization	$11rn^2$
Triangular HSS solution	$10rn$

4. Selected improvements.

4.1. HSS construction. One improvement to the HSS construction algorithm is to use a top-down construction procedure. For node i with children c_1 and c_2, the HSS block associated with i is given by

$$\begin{pmatrix} A|_{t_i \times t_i^c} & A|_{t_i \times t_i^r} \end{pmatrix} = \begin{pmatrix} A|_{t_{c_1} \times t_{c_1}^c} & A|_{t_{c_1} \times t_{c_2}^r} \\ A|_{t_{c_2} \times t_{c_1}^c} & A|_{t_{c_2} \times t_{c_2}^r} \end{pmatrix}. \tag{4.1}$$

Thus, $\begin{pmatrix} A|_{t_i \times t_i^c} & A|_{t_i \times t_i^r} \end{pmatrix}$ is first compressed; then its compressed form participates in the compression of the HSS blocks associated with c_1 and c_2 which include the subblocks on the right-hand side of (4.1).

The cost can be similarly counted with (3.7)–(3.8). When $m = 2r$, the complexity is $4rn^2$ flops, which is less than the bottom-up construction in the previous section. However, this top-down procedure accesses the matrix entries globally and may not be suitable for parallelization.

4.2. ULV HSS factorization. For a given HSS matrix, an improvement to the ULV factorization algorithm is to replace the full QR factorization of $\hat{D}_i^T$ in (3.12) by a partial one, as in [42]. That is, partition $\hat{D}_i$ in (3.11) as $\hat{D}_i = \begin{pmatrix} \hat{D}_{i;1,1} & \hat{D}_{i;1,2} \\ \hat{D}_{i;2,1} & \hat{D}_{i;2,2} \end{pmatrix} \begin{matrix} m-r \\ r \end{matrix}$ and compute an LQ factorization

$$\begin{pmatrix} \hat{D}_{i;1,1} & \hat{D}_{i;1,2} \end{pmatrix} = \begin{pmatrix} \tilde{D}_{1;1,1} & 0 \end{pmatrix} W_i^T.$$

Also let

$$\begin{pmatrix} \tilde{D}_{i;2,1} & \tilde{D}_{i;2,2} \end{pmatrix} = \begin{pmatrix} \hat{D}_{i;2,1} & \hat{D}_{i;2,2} \end{pmatrix} W_i.$$

Then we also get a form (3.13). It can be verified that the cost of such a factorization is $\mathcal{C}_{\text{fact}} \approx \frac{4}{3}km^3 + \frac{128}{3}kr^3 + 6km^2r$ flops. With $m = 2r$, we have $\mathcal{C}_{\text{fact}} \approx \frac{116}{3}r^2n$.

A further improvement is to replace the QR factorization in (3.12) by an LU one and the matrix multiplication in (3.12) by a triangular solution. That is, compute LU factorizations $D_i = \tilde{D}_i T_i$, and let $\tilde{V}_i = T_i^{-T} V_i$. This is done similarly for j. Then the following operation also yields a form as on the right-hand side of (3.13):

$$\begin{pmatrix} Q_j^T & \\ & Q_i^T \end{pmatrix} \begin{pmatrix} D_j & U_j B_j V_i^T \\ U_i B_i V_j^T & D_i \end{pmatrix} \begin{pmatrix} T_j^{-1} & \\ & T_j^{-1} \end{pmatrix}.$$

We can verify that the complexity is

$$\mathcal{C}_{\text{fact}} \approx \frac{2}{3}m^2n + 5rmn + 24kr^3. \tag{4.2}$$

The saving over (3.15) is $\frac{2}{3}m^2n + rmn + \frac{100}{3}kr^3$ flops. When $m = 2r$, we have $\mathcal{C}_{\text{fact}} \approx \frac{74}{3}r^2n$, as compared with $42r^2n$ of (3.15).

4.3. Symmetry. Significant improvements can be made specifically for symmetric matrices. If A is symmetric, in the HSS construction, only either the block row compression or block column compression is needed. A straightforward implementation of the bottom-up construction in section 3.2 as in [42] costs $3rn^2$ flops, and a top-down procedure costs $2rn^2$, where $m = 2r$ is assumed.

For a symmetric positive definite HSS matrix, the general ULV factorization algorithm in [14] does not specifically take advantage of the symmetry. A modified ULV scheme is proposed in [42], where after introducing zeros into the off-diagonal blocks just like (3.11), the diagonal blocks $\hat{D}_i$ are partially eliminated with Cholesky factorizations. This version costs $\frac{113}{3}r^2n$ flops when $m = 2r$.

It can be further improved. Before the step (3.11), compute a full Cholesky factorization $D_i = L_i L_i^T$. In this way, we can avoid the dense matrix product $Q_i^T D_i$ in (3.11). That is, by letting $\hat{U}_i = L_i^{-1} U_i$ and computing a QR factorization $\hat{U}_i = Q_i \tilde{U}_i$, we have

$$\begin{aligned} &\begin{pmatrix} Q_1^T L_1^{-1} & \\ & Q_2^T L_2^{-1} \end{pmatrix} \begin{pmatrix} D_1 & U_1 B_1 U_2^T \\ U_2 B_2 U_1^T & D_2 \end{pmatrix} \begin{pmatrix} L_1^{-T} Q_1 & \\ & L_2^{-T} Q_2 \end{pmatrix} \\ &= \begin{pmatrix} \begin{pmatrix} I & \\ & I \end{pmatrix} & \begin{pmatrix} 0 & \\ & \tilde{U}_1 B_1 \tilde{U}_2^T \end{pmatrix} \\ \begin{pmatrix} 0 & \\ & \tilde{U}_2 B_1^T \tilde{U}_1^T \end{pmatrix} & \begin{pmatrix} I & \\ & I \end{pmatrix} \end{pmatrix}. \end{aligned}$$

Then $\tilde{U}_1 B_1 \tilde{U}_2^T$ participates in later eliminations. Here, there are no explicit matrix multiplications involving Q_i, and the partial diagonal elimination is straightforward.

In such a scheme, the total complexity is

$$\mathcal{C}_{\text{fact}} = \frac{1}{3}m^2 n + rmn + \frac{28}{3}kr^3 + 2r^2 n.$$

If $m = 2r$, this is only about $9r^2 n$. Table 4.1 lists the complexity of some symmetric HSS algorithms.

TABLE 4.1

Complexity of some symmetric HSS algorithms (with the low order terms dropped) for symmetric positive definite problems when $m = 2r$, where all leaf level HSS block rows have row sizes m and the HSS rank of the matrix is r.

Operation	Complexity
HSS construction (bottom-up version)	$3rn^2$
HSS construction (top-down version)	$2rn^2$
Dense to triangular HSS factorization	$\frac{11}{2}rn^2$
ULV HSS factorization	$9r^2 n$

4.4. Numerical experiments. We demonstrate the performance of some important HSS algorithms for a linear system

$$Ax = b, \text{ with } A = \alpha I + B = \left(\alpha + \sqrt{|x_i - x_j|}\right)_{n\times n}, \tag{4.3}$$

where the points $x_i = \cos((2i+1)\pi/2n)$ are the zeros of the nth Chebyshev polynomial, and α is chosen to make A positive definite. B is illustrated to have a small numerical HSS rank r in [9, 13]. Thus, A has the low-rank property. The right-hand side b is obtained by $b = Ax^*$ with the exact solution x^* being random. The preliminary implementation is in MATLAB and the flop counts, timing, and accuracies are shown in Tables 4.2, 4.3, and 4.4, respectively.

TABLE 4.2

Complexity (floating point operations or flops) of some HSS algorithms applied to a symmetric system (4.3)*, where all leaf level HSS block rows have row sizes $m = 30$, and the relative tolerance in the compression is $\tau = 10^{-8}$.*

n		256	512	1024	2048	4096
Direct HSS	Construction	$2.26E6$	$1.11E7$	$4.87E7$	$1.98E8$	$7.88E8$
	ULV solution	$4.55E5$	$1.08E6$	$2.24E6$	$4.44E6$	$8.67E6$
Triangular HSS	Factorization	$4.34E6$	$2.09E7$	$9.28E7$	$3.82E8$	$1.57E9$
	Solution	$2.74E4$	$6.20E4$	$1.30E5$	$2.62E5$	$5.19E5$

n		8192	16384	32768	65536	131072
Direct HSS	Construction	$3.09E9$	$1.18E10$	$4.47E10$	$1.67E11$	$6.17E11$
	ULV solution	$1.64E7$	$3.03E7$	$5.53E7$	$9.96E7$	$1.78E8$
Triangular HSS	Factorization	$6.29E9$	$2.52E10$	$9.90E10$	$3.89E11$	$1.52E12$
	Solution	$1.01E6$	$1.94E6$	$3.69E6$	$6.94E6$	$1.30E7$

The flop counts are consistent with the theoretical counts. For example, the direct HSS construction (symmetric version of Algorithm 1) and the ULV factorization (symmetric version of Algorithm 2) have roughly $O(n^2)$ and $O(n)$ complexities, respectively. Also, the rows in Table 4.3 for the timing follow similar patterns. For

TABLE 4.3

Preliminary timing in seconds (MATLAB) corresponding to Table 4.3. *(Note that the difference between the solution timings of the two methods is small. This is because the solution time is small for such sizes and is mainly for memory access which is similar for both methods.)*

n		256	512	1024	2048	4096
Direct HSS	Construction	$1.09E-1$	$1.15E-1$	$4.18E-1$	$1.95E0$	$8.27E0$
	ULV solution	$4.20E-2$	$1.74E-2$	$6.43E-2$	$1.02E-1$	$1.74E-1$
Triangular HSS	Factorization	$1.10E-1$	$1.57E-1$	$5.31E-1$	$2.48E0$	$1.17E1$
	Solution	$6.90E-2$	$1.26E-2$	$5.04E-2$	$7.71E-2$	$1.35E-1$

n		8192	16384	32768	65536	131072
Direct HSS	Construction	$4.00E1$	$1.73E2$	$7.34E2$	$2.88E3$	$1.11E4$
	ULV solution	$3.18E-1$	$6.33E-1$	$1.24E0$	$2.48E0$	$5.71E0$
Triangular HSS	Factorization	$5.28E1$	$2.14E2$	$1.09E3$	$4.22E3$	$1.79E4$
	Solution	$2.49E-1$	$4.75E-1$	$9.71E-1$	$2.13E0$	$4.85E0$

TABLE 4.4

Accuracy corresponding to Table 4.3, *where the error is* $\frac{\|x-x^*\|_2}{\|x^*\|_2}$ *and the residual is* $\frac{\|Ax-b\|_2}{\|b\|_2}$.

n		256	512	1024	2048	4096
Direct ULV HSS	Error	$1.41E-9$	$1.92E-9$	$2.19E-9$	$1.69E-9$	$1.70E-9$
	Residual	$1.29E-9$	$1.14E-9$	$2.17E-9$	$1.61E-9$	$1.66E-9$
Triangular HSS	Error	$1.27E-9$	$2.32E-9$	$7.34e-10$	$1.77E-9$	$1.80E-9$
	Residual	$1.25E-9$	$1.27E-9$	$7.09e-10$	$1.64E-9$	$1.64E-9$

n		8192	16384	32768	65536	131072
Direct ULV HSS	Error	$6.33E-9$	$3.07E-9$	$2.62E-9$	$1.36E-9$	$2.37E-9$
	Residual	$3.88E-9$	$1.76E-9$	$1.92E-9$	$1.36E-9$	$2.32E-9$
Triangular HSS	Error	$2.33E-9$	$3.88E-9$	$2.86E-9$	$1.89E-9$	$1.88E-9$
	Residual	$1.99E-9$	$2.15E-9$	$1.64E-9$	$1.49E-9$	$1.83E-9$

example, the ULV solution time roughly doubles when n doubles. In addition, both methods give satisfactory accuracies, as in Table 4.4.

From another point of view, we can compare the HSS rank with its estimate based on the actual costs. The flop count of the bottom-up symmetric HSS construction is $\mathcal{C}_{\text{constr}} \approx 2rn^2 + 2r^2n^2/m$, which indicates that r can be estimated by $\tilde{r} = \frac{m}{2}(\sqrt{1+2\frac{\mathcal{C}}{mn}} - 1)$ with the actual cost $\mathcal{C}$. The estimate $\tilde{r}$ can be viewed as an average rank value, while r is an upper bound. These results are shown in Table 4.5.

TABLE 4.5

Estimated HSS rank $\tilde{r}$, *as compared with the actual HSS rank* r *(an upper bound), based on the complexity of the symmetric HSS construction algorithm applied to* (4.3).

n	256	512	1024	2048	4096	8192	16384
HSS rank r	15	16	17	18	19	20	20
Estimated $\tilde{r}$	13	15	16	16	16	16	15

5. HSS recompression. An important improvement to HSS methods is to recompress noncompact HSS representations. During the operations of HSS matrices we may get HSS forms where the U, V generators are not orthogonal or which are not compact enough even though the actual HSS rank is small. Such HSS matrices arise in sparse factorizations [41] and other construction schemes such as those based on randomized sampling [31, 34] or strong rank-revealing LU factorizations [25].

As an example, we consider adding two HSS matrices A and C with the same HSS tree structure and block partition. Assume that their generators are $D_i(A)$, $U_i(A)$, $\ldots$, and $D_i(C)$, $U_i(C)$, $\ldots$, respectively. Then, obviously, the sum $A+C$ is an HSS matrix with generators

$$\begin{aligned} &D_i(A+C) = D_i(A) + D_i(C), && B_i(A+C) = \operatorname{diag}\left(B_i(A), B_i(C)\right), \\ &U_i(A+C) = \begin{pmatrix} U_i(A) & U_i(C) \end{pmatrix}, && R_i(A+C) = \operatorname{diag}\left(R_i(A), R_i(C)\right), \\ &V_i(A+C) = \begin{pmatrix} V_i(A) & V_i(C) \end{pmatrix}, && W_i(A+C) = \operatorname{diag}\left(W_i(A), W_i(C)\right). \end{aligned} \tag{5.1}$$

We see that the sizes of the HSS generators associated with the off-diagonal blocks of $A+C$ increase additively. However, the HSS rank of $A+C$ may be much smaller, and the HSS form is not sufficiently compact. To enhance the effectiveness of the HSS representation we can use recompression techniques to recover a compact HSS form.

In general, we consider recompressing an HSS matrix A represented by generators D_i, U_i, $\ldots$, B_i. The details are explained as follows. The next proposition forms the foundation of the recompression technique here. It shows a nested representation for the HSS block row A_i^- and column $A_i^|$ associated with a node i as in (2.3). Here for convenience, we permute A_i^- and $A_i^|$ and rewrite them as

(5.2)
$$\mathcal{A}_i^- \equiv A_i^- \Pi_i = \begin{pmatrix} A|_{t_i \times t_j} & A|_{t_i \times (\mathcal{I} \setminus (t_i \cup t_j))} \end{pmatrix}, \quad \mathcal{A}_i^| \equiv \Pi_i^T A_i^| = \begin{pmatrix} A|_{t_j \times t_i} \\ A|_{(\mathcal{I} \setminus (t_i \cup t_j)) \times t_i} \end{pmatrix}.$$

That is, Π_i permutes $A|_{t_i \times t_j}$ to the front of $\mathcal{A}_i^-$, and Π_i^T permutes $A|_{t_j \times t_i}$ to the top of $\mathcal{A}_i^-$.

PROPOSITION 5.1. *Assume A is an HSS matrix as defined in Definition* 2.1. *For a node $i < \operatorname{root}(\mathcal{T})$ with sibling j and parent p, the corresponding HSS blocks $\mathcal{A}_i^-$ and $\mathcal{A}_i^|$ in* (5.2) *have the following forms:*

$$\mathcal{A}_i^- = U_i S_i \hat{V}_i^T, \text{ with } S_i = \begin{pmatrix} B_i & R_i S_p \end{pmatrix}, \ \hat{V}_i = \operatorname{diag}\left(V_j, \ \Pi_p \hat{V}_p\right), \tag{5.3}$$

$$\mathcal{A}_i^| = \hat{U}_i T_i V_i^T, \text{ with } T_i = \begin{pmatrix} B_j \\ T_p^T W_j^T \end{pmatrix}, \quad \hat{U}_i = \operatorname{diag}\left(U_j, \ \Pi_p \hat{U}_p\right), \tag{5.4}$$

where Π_p is a permutation matrix as defined in (5.2), *$\hat{V}_i$ and $\hat{U}_i$ have orthonormal columns, and S_p, T_p, $\hat{U}_p$, $\hat{V}_p$ are empty matrices when $p = \operatorname{root}(\mathcal{T})$.*

Proof. We prove (5.3) by induction, and (5.4) can be proved similarly. The result is obvious for a node i with $\operatorname{par}(i) = 2k-1$. Assuming (5.3) holds for a nonleaf node $i < 2k-1$, we show it also holds for the children c_1 and c_2 of i. Clearly, $A|_{t_{c_1} \times t_{c_2}} = U_{c_1} B_{c_1} V_{c_2}^T$. Since

$$\mathcal{A}_i^- = \begin{pmatrix} A|_{t_{c_1} \times (\mathcal{I} \setminus (t_{c_1} \cup t_{c_2}))} \\ A|_{t_{c_2} \times (\mathcal{I} \setminus (t_{c_1} \cup t_{c_2}))} \end{pmatrix} \Pi_i, \quad U_i S_i \hat{V}_i^T = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix} S_i \hat{V}_i^T,$$

we have $A|_{t_{c_1} \times (\mathcal{I} \setminus (t_{c_1} \cup t_{c_2}))} = U_{c_1} R_{c_1} S_i^T \hat{V}_i^T \Pi_i^T$. Thus,

$$\begin{aligned} \mathcal{A}_{c_1}^- &= \begin{pmatrix} A|_{t_{c_1} \times t_{c_2}} & A|_{t_{c_1} \times (\mathcal{I} \setminus (t_{c_1} \cup t_{c_2}))} \end{pmatrix} = \begin{pmatrix} U_{c_1} B_{c_1} V_{c_2}^T & U_{c_1} R_{c_1} S_i \hat{V}_i^T \Pi_i^T \end{pmatrix} \\ &= U_{c_1} \begin{pmatrix} B_{c_1} & R_{c_1} S_i \end{pmatrix} \operatorname{diag}\left(V_{c_2}^T, (\Pi_i \hat{V}_i)^T\right) \\ &\equiv U_{c_1} S_{c_1} \hat{V}_{c_1}^T. \end{aligned}$$

Therefore, (5.3) holds for c_1. Similarly, it holds for c_2. ∎

This proposition also justifies the precise definition of the compactness of an HSS form traditionally used in HSS methods.

DEFINITION 5.2. *An HSS matrix A as defined in Definition* 2.1 *is said to be in a* compact form *if its HSS rank is small, and the column sizes of all U_i and V_i are close to the ranks of A_i^- and $A_i^|$, respectively. Also, we say A is in a* proper form *if it is compact and all U_i and V_i have orthonormal columns.*

According to this proposition, we design a recompression scheme with two stages. In the first stage or a forward stage, we compress U_i and V_i and make the columns of each to be orthonormal, which is done in a postordering traversal of the HSS tree. In the second stage or a backward stage, we compress S_i and T_i in (5.3)–(5.4) to further make the HSS form a compact one, which is done in a reverse-postordering traversal. This procedure is more systematic than a preliminary one in our technical report [12]. We use tilded notation $\tilde{R}$, $\tilde{W}$, etc. to denote the generators after compression, and hatted notation $\hat{R}$, $\hat{W}$, etc. to denote intermediate temporary forms of the generators.

5.1. Forward stage. In the first stage, we traverse the tree bottom-up for nodes $i = 1, 2, \ldots$ The generators U_i and V_i are compressed and R_i, W_i, and B_i are updated. If i is a leaf, compute QR factorizations

$$U_i = \tilde{U}_i F_i, \quad V_i = \tilde{V}_i G_i, \tag{5.5}$$

where $\tilde{U}_i$ and $\tilde{V}_i$ are the new generators. That is, after this compression step, U_i and V_i are updated to $\tilde{U}_i$ and $\tilde{V}_i$, respectively. The matrices F_i and G_i are passed to generators R_i and W_i, respectively, as in

$$\hat{R}_i = F_i R_i, \quad \hat{W}_i = G_i W_i. \tag{5.6}$$

If i is a nonleaf node, U_i and V_i are compressed indirectly. For example, U_i is implicitly given by

$$U_i = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix} = \begin{pmatrix} \tilde{U}_{c_1} \hat{R}_{c_1} \\ \tilde{U}_{c_2} \hat{R}_{c_2} \end{pmatrix} = \begin{pmatrix} \tilde{U}_{c_1} & \\ & \tilde{U}_{c_2} \end{pmatrix} \begin{pmatrix} \hat{R}_{c_1} \\ \hat{R}_{c_2} \end{pmatrix},$$

where $\tilde{U}_{c_1}$ and $\tilde{U}_{c_2}$ are compressed by recursion, and have orthonormal columns. Thus it suffices to compute QR factorizations

$$\begin{pmatrix} \hat{R}_{c_1} \\ \hat{R}_{c_2} \end{pmatrix} = \begin{pmatrix} \tilde{R}_{c_1} \\ \tilde{R}_{c_2} \end{pmatrix} F_i, \quad \begin{pmatrix} \hat{W}_{c_1} \\ \hat{W}_{c_2} \end{pmatrix} = \begin{pmatrix} \tilde{W}_{c_1} \\ \tilde{W}_{c_2} \end{pmatrix} G_i. \tag{5.7}$$

This gives the new generators $\tilde{R}_{c_1}$, $\tilde{R}_{c_2}$, $\tilde{W}_{c_1}$, $\tilde{W}_{c2}$. Then use (5.6) to update R_i, W_i.

If i is also a right node, update

$$\tilde{B}_i = F_i B_i G_j^T, \quad \tilde{B}_j = F_j B_j F_i^T, \tag{5.8}$$

where $j = \mathrm{sib}(i)$, and F_j and G_j are available from the jth compression step (associated with node j which has been visited before).

Repeat these steps along the postordering traversal of $\mathcal{T}$. At the end of this stage, we have A in a new HSS form with generators $\tilde{D}_i$, $\tilde{U}_i$, $\ldots$, $\tilde{B}_i$. After this stage, all $\tilde{U}_i$ and $\tilde{V}_i$ generators have orthonormal columns (and full column ranks).

5.2. Backward stage. This is a top-down stage, or the reverse-postordering traversal of the nodes $i = \text{root}(\mathcal{T})-1, \text{root}(\mathcal{T})-2, \ldots, 1$. For convenience, we still use D_i, U_i, $\ldots$, B_i to denote the generators of A after the forward stage and use tilded notation for the new generators. Here, we compress $\mathcal{A}_i^-$ and $\mathcal{A}_j^|$ via the compression of S_i and T_j in Proposition 5.1, respectively, where $j = \text{sib}\,(i)$.

If a node i satisfies $\text{par}(i) = \text{root}(\mathcal{T})$, we compute an SVD for $S_i = B_i$:

$$S_i = P_i \tilde{S}_i Q_i^T. \tag{5.9}$$

Then we update $R_{c_1}, W_{c_1}, R_{c_2}, W_{c_2}$ for the children c_1, c_2 of i with

$$\begin{pmatrix} \hat{R}_{c_1} \\ \hat{R}_{c_2} \end{pmatrix} = \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} P_i, \quad \begin{pmatrix} \hat{W}_{c_1} \\ \hat{W}_{c_2} \end{pmatrix} = \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix} Q_i. \tag{5.10}$$

This implicitly updates U_i, V_i. Accordingly, set $\tilde{B}_i = \tilde{S}_i$ and $\tilde{T}_i = B_j$, where $j = \text{sib}\,(i)$.

If i satisfies $\text{par}(i) \neq \text{root}(\mathcal{T})$, according to Proposition 5.1, we compress S_i and T_j. Let $p = \text{par}(i),\ j = \text{sib}(i)$. The proof in Theorem 5.3 shows that the compression of S_i and T_j can be done via

$$\hat{S}_i = \begin{pmatrix} B_i & \hat{R}_i \tilde{S}_p \end{pmatrix} \quad \text{and} \quad \hat{T}_j^T = \begin{pmatrix} B_i^T & \hat{W}_i \tilde{T}_p \end{pmatrix}, \tag{5.11}$$

respectively, where $\hat{R}_i,\ \tilde{S}_p,\ \hat{W}_i$, and $\tilde{T}_p$ are results from the recompression step associated with p by recursion (with i set to be p in (5.10), (5.12), and (5.13)). Compute SVDs

$$\hat{S}_i = P_i \tilde{S}_i X_i^T \equiv P_i \tilde{S}_i \begin{pmatrix} X_{i,1}^T & X_{i,2}^T \end{pmatrix}, \tag{5.12}$$

$$\hat{T}_j^T = Q_j \tilde{T}_j Y_i^T \equiv Q_j \tilde{T}_j \begin{pmatrix} Y_{j,1}^T & Y_{j,2}^T \end{pmatrix}, \tag{5.13}$$

where X_i^T and Y_i^T are partitioned according to the column partitions in (5.11). The fact that $\mathcal{A}_i^-$ and $\mathcal{A}_j^|$ share a common block $A|_{t_i \times t_j} = U_i B_i V_j^T$ yields

$$B_i = P_i \tilde{S}_i X_{i,1}^T = Y_{j,1} \tilde{T}_j^T Q_j^T.$$

Thus, we set

$$\tilde{B}_i (\equiv P_i^T B_i Q_j) = P_i^T Y_{j,1} \tilde{T}_j^T = \tilde{S}_i X_{i,1}^T Q_j, \tag{5.14}$$

$$\tilde{R}_i = P_i^T \hat{R}_i,\ \tilde{W}_j = Q_j^T \hat{W}_j. \tag{5.15}$$

Next, if i is also a leaf node, we update U_i and V_i as

$$\tilde{U}_i = U_i P_i, \quad \tilde{V}_i = V_i Q_i, \tag{5.16}$$

respectively. Otherwise, we compute the update (5.10).

The recompression is then done recursively. When it finishes, A is in a compact HSS form with generators $\tilde{D}_i$, $\tilde{U}_i$, $\ldots$, $\tilde{B}_i$.

5.3. Algorithm and analysis. We summarized the procedure as follows.

ALGORITHM 3 (*HSS recompression*).

`for` nodes (separators) $i = 1, 2, \ldots, \text{root}(\mathcal{T}) - 1$ (*Forward stage*)

```
  if i is a leaf
    – Compress U_i and V_i as in (5.5) and update R_i and W_i as in (5.6)
  else
    – Compute QR factorizations in (5.7) and update R_i and W_i as in (5.6)
  end
  if i is a right node, update B_i and B_j as in (5.8) for j = sib(i)
end
Set D_i, U_i, etc. to be D̃_i, Ũ_i, etc., respectively
for nodes (separators) i = root(T) − 1, root(T) − 2, ..., 1 (Backward stage)
  if par(i) = root(T)
    – Compute an SVD for B_i as in (5.9) and update R_{c_1}, W_{c_1}, R_{c_2}, W_{c_2} as in
      (5.10) for the children c_1, c_2 of i
    – Set B̃_i = S̃_i and T̃_i = B_j
  else
    – Form Ŝ_i and T̂_j^T as in (5.11) and compute SVDs as in (5.12) and (5.13)
    – Compute B̃_i as in (5.14) and R̃_i and W̃_j as in (5.15)
  end
  if i is a leaf, update U_i and V_i as in (5.16)
end
Set D_i, U_i, etc. to be D̃_i, Ũ_i, etc., respectively.
```

Next, we briefly discuss the effectiveness of this recompression procedure and the complexity.

THEOREM 5.3. *After the two stages of recompression, the new HSS representation of A is in a proper form.*

Proof. After the first stage, all $\tilde{U}_i$ and $\tilde{V}_i$ have orthonormal columns due to (5.5) and (5.7). This property is also preserved in the second stage as in (5.10) and (5.16). We just need to show that the second stage produces a compact HSS form. Let the new generators after the second stage be $\tilde{D}_i$, $\tilde{U}_i$, ..., $\tilde{B}_i$.

We show that the second stage recursively yields S_p in the following form for a node $p = \mathrm{par}(i)$:

$$S_p = P_p \tilde{S}_p Z_p^T, \quad Z_p \equiv \mathrm{diag}\left(I, Z_{\mathrm{par}(p)}\right) X_p, \tag{5.17}$$

where Z_p has orthonormal columns, and $Z_p = Q_p$ if $\mathrm{par}(p) = \mathrm{root}(\mathcal{T})$, or by (5.12) otherwise. We show the result also holds for i. That is,

$$\begin{aligned} S_i &= \begin{pmatrix} B_i & R_i S_p \end{pmatrix} = \begin{pmatrix} B_i & R_i P_p \tilde{S}_p Z_p^T \end{pmatrix} \\ &= \begin{pmatrix} B_i & \hat{R}_i \tilde{S}_p \end{pmatrix} \mathrm{diag}\left(I, Z_p^T\right) = \hat{S}_i \mathrm{diag}\left(I, Z_p^T\right) \\ &= P_i \tilde{S}_i X_i^T \mathrm{diag}\left(I, Z_p^T\right) = P_i \tilde{S}_i Z_i^T, \end{aligned} \tag{5.18}$$

where (5.12) is used, and $Z_i = \mathrm{diag}\,(I, Z_p)\, X_i$. This verifies the recursion and confirms that the compression of S_i can be done by compressing $\hat{S}_i$ in (5.11). Thus,

$$\mathcal{A}_i^- = U_i S_i \hat{V}_i^T = U_i (P_i \tilde{S}_i Z_i^T) \hat{V}_i^T = (U_i P_i) \tilde{S}_i (Z_i^T \hat{V}_i^T) = \tilde{U}_i \tilde{S}_i (\hat{V}_i Z_i)^T,$$

where both $\tilde{U}_i$ and $\hat{V}_i Z_i$ have orthonormal columns, and $\tilde{S}_i$ has full rank. Therefore, the column size of $\tilde{U}_i$ is the rank of $\mathcal{A}_i^-$, and $\mathcal{A}_i^-$ is in a compact compressed form. A similar result holds for $\mathcal{A}_i^{\downarrow}$. ∎

The complexity of the algorithm can be conveniently counted. For simplicity, we make assumptions about the generator sizes and ranks as indicated in Table 5.1. Then it can be verified that the total cost of the forward and backward stages is

$$\mathcal{C}_{\text{recom}} = r_1\left(8mr_0 + 12r_0^2 + 12r_0r_1 - \frac{8}{3}r_1^2 - 4mr_1\right)\frac{n}{m} + r\left(4mr_1 + 24r_1^2 + 28rr_1 + \frac{20}{3}r^2\right)\frac{n}{m}.$$

TABLE 5.1
Generator sizes before and after the two stages of recompression, where r_1 is the rank of each U_i and V_i initially, and r is the HSS rank satisfying $r < r_1$.

Generator	Initial size	After first stage	After second stage
U_i, V_i	$m \times r_0$	$m \times r_1$	$m \times r$
R_i, W_i, B_i	$r_0 \times r_0$	$r_1 \times r_1$	$r \times r$

As an example, we consider recompressing an HSS matrix where $r_0 = 2r_1 = 4r$ and $m = 2r$. See Table 5.2 for the recompression cost and the comparison of ULV operation costs before and after the recompression.

TABLE 5.2
Costs of HSS operations for an HSS matrix before and after recompression, where the HSS rank is r, and $m = 2r$, $r_0 = 2r_1 = 4r$ (before recompression, as in Table 5.1).

	HSS recompression	ULV factorization	ULV solution
Before recompression	\	$736r^2n$	$148rn$
After recompression	$413r^2n$	$46r^2n$	$37rn$

6. Relaxation of rank requirements in HSS operations. In this section, we propose another useful improvement to classical HSS methods from a different point of view.

Existing HSS algorithms often assume the HSS ranks to be bounded [39, 41, 42, etc.]. Sometimes, the bound may be pessimistic (see, e.g., r as a bound for $\tilde{r}$ in Table 4.5). In [14], a special rank pattern for HSS blocks is used. A concept of rank functions is also given in [8]. Here, we relax the HSS rank requirement by proposing several general rank patterns. For simplicity, assume all HSS block rows corresponding to level l of the HSS tree have the same row size m_{L-l}, where the bottom/leaf HSS block rows have row dimensions m_0 (constant) so that there are totally L levels with

$$m_0 \approx n/2^L, \quad m_{L-l} = m_0 2^{L-l}. \tag{6.1}$$

Also assume r_0 is maximum rank of bottom level HSS blocks. At level l, we allow the numerical ranks of the HSS blocks to be bounded by

$$r_{L-l} = (L-l+1)r_0 = \left(\log_2 \frac{m_{L-l}}{m_0} + 1\right) r_0.$$

That is, r_{L-l} increases as m_{L-l} increases (and as l decreases).

For HSS construction algorithms such as the one described in section 3.2, the cost associated with each tree node at level l is $O(r_{L-l}^2 n)$ flops. There are 2^l nodes at level

l. Using (6.1) and the fact that $\sum_{l=0}^{L}(L-l+1)^2 2^l \approx 12 \times 2^L$, we have the total cost for the HSS construction:

$$\mathcal{C}_{\text{constr}} = \sum_{l=0}^{L} O(r_{L-l}^2 n)2^l = O\left(r_0^2 n \sum_{l=0}^{L}(L-l+1)^2 2^l\right) = O(r_0^2 2^L n) = O(n^2).$$

Thus, after relaxing the rank requirement, we do not increase the order of the complexity.

Similarly, for ULV HSS factorizations, the cost associated with each tree node is $O({r_{L-l}}^3)$ flops. Using $\sum_{l=0}^{L}(L-l+1)^3 2^l \approx 52 \times 2^L$, we have the total cost

$$\mathcal{C}_{\text{fact}} = \sum_{l=0}^{L} O(r_{L-l}^3)2^l = O\left(r_0^3 \sum_{l=0}^{L}(L-l+1)^3 2^l\right) = O(n).$$

The assumption (6.2) indicates $r_{L-l} = O(r_0 \log_2 m_{L-l})$, and the HSS rank is bounded by $O(\log_2 n)$. More generally, we can allow r_l to be

$$r_{L-l} = O\left((r_0 \log_2 m_{L-l})^p\right) = O\left(r_0^p (L-l+1)^p\right), \tag{6.2}$$

where p is a positive integer. For example, when $p = 2$ we have

$$\mathcal{C}_{\text{fact}} = \sum_{l=0}^{L} r_0^6 (L-l+1)^6 2^l \approx O(r_0^6 2^L) = O(n).$$

Similarly, the storage remains $O(n)$ under the new rank condition.

We can further relax the rank requirements so that r_{L-l} increases as $O(m_{L-l}^{1/p})$, where p is a positive integer. The derivations are similar. For example, for $p = 3$ and 2, the HSS construction costs are now

$$\mathcal{C}_{\text{constr}} = \sum_{l=0}^{L} O\left(m_{L-l}^{1/3}\right)^2 n2^l = \sum_{l=0}^{L} O\left(\left(m_0 2^{L-l}\right)^{2/3}\right) n2^l = O(n^2) \text{ and}$$

$$\mathcal{C}_{\text{constr}} = \sum_{l=0}^{L} O\left(m_{L-l}^{1/2}\right)^2 n2^l = \sum_{l=0}^{L} O\left(m_0 2^{L-l}\right) n2^l = O(n^2 \log_2 n),$$

respectively.

In addition, we can generalize the rank pattern used in [14]. That is, assume

$$r_{L-l} = \alpha^{L-l} r_0.$$

$|\alpha| = \sqrt{2}$ is discussed in [14]. Here, we estimate the counts for a general α. For example,

$$\mathcal{C}_{\text{constr}} = \sum_{l=0}^{L} O(r_{L-l}^2 n)2^l = n r_0^2 \sum_{l=0}^{L} O\left(\alpha^{2(L-l)} \times 2^l\right) = n r_0^2 \alpha^{2L} O\left(\sum_{l=0}^{L}\left(\frac{2}{\alpha^2}\right)^l\right)$$

$$= \begin{cases} O(n^2) & \text{if } |\alpha| < \sqrt{2}, \\ O\left(n^2 \log n\right) & \text{if } |\alpha| = \sqrt{2}, \\ O\left(n^{1+\log_2 \alpha^2}\right) & \text{if } |\alpha| > \sqrt{2}. \end{cases}$$

Therefore, it is desirable to choose $|\alpha| \le \sqrt{2}$.

All of these results are summarized in the following theorem.

THEOREM 6.1. *Suppose an order n matrix A is recursively partitioned into $L = O(\log_2 \frac{n}{m_0})$ levels of HSS blocks following a perfect binary tree, so that the bottom level HSS block row size is m_0. Let $m_{L-l} = O(m_0 2^{L-l})$ be the row dimensions of the HSS block rows at level l, and let r_l be their maximum numerical rank.*

1. *When $r_l = O((\log_2 m_l)^p)$ (so that $\max_l r_{L-l} = O((\log_2 n)^p)$) with an integer $p \geq 0$, an HSS form of A can be constructed in $\mathcal{C}_{\text{constr}}$ flops, a ULV factorization of the HSS form needs $\mathcal{C}_{\text{fact}}$ flops, and the HSS form or its ULV factor needs memory size $\mathcal{S}_{\text{mem}}$, where*

$$\mathcal{C}_{\text{constr}} = O(n^2), \quad \mathcal{C}_{\text{fact}} = O(n), \quad \mathcal{S}_{\text{mem}} = O(n).$$

   (These are the same counts as in the classical situation [14, 42, *etc.*] *where $r_l = O(1)$ or $p = 0$.)*
2. *When $r_{L-l} = O(m_{L-l}^{1/p})$ (so that $\max_l r_{L-l} = O(n^{1/p})$) with an integer $p > 0$, we have the following counts:*
   (a) *If $p > 3$, then the counts are*

$$\mathcal{C}_{\text{constr}} = O(n^2), \quad \mathcal{C}_{\text{fact}} = O(n), \quad \mathcal{S}_{\text{mem}} = O(n).$$

   (b) *If $p = 3$, then the counts are*

$$\mathcal{C}_{\text{constr}} = O(n^2), \quad \mathcal{C}_{\text{fact}} = O(n \log_2 n), \quad \mathcal{S}_{\text{mem}} = O(n).$$

   (c) *If $p = 2$, then the counts are*

$$\mathcal{C}_{\text{constr}} = O(n^2 \log_2 n), \quad \mathcal{C}_{\text{fact}} = O(n^{3/2}), \quad \mathcal{S}_{\text{mem}} = O(n \log_2 n).$$

3. *When $r_{L-l} = \alpha^{L-l} r_0$ with $\alpha > 0$, we have the following counts:*
   (a) *If $|\alpha| < \sqrt[3]{2}$,*

$$\mathcal{C}_{\text{constr}} = O(n^2), \quad \mathcal{C}_{\text{fact}} = O(n), \quad \mathcal{S}_{\text{mem}} = O(n).$$

   (b) *If $|\alpha| = \sqrt[3]{2}$,*

$$\mathcal{C}_{\text{constr}} = O(n^2), \quad \mathcal{C}_{\text{fact}} = O(n \log_2 n), \quad \mathcal{S}_{\text{mem}} = O(n).$$

   (c) *If $\sqrt[3]{2} < |\alpha| < \sqrt{2}$,*

$$\mathcal{C}_{\text{constr}} = O(n^2), \quad \mathcal{C}_{\text{fact}} = O(n^{\log_2 \alpha^3}), \quad \mathcal{S}_{\text{mem}} = O(n).$$

   (d) *If $|\alpha| = \sqrt{2}$,*

$$\mathcal{C}_{\text{constr}} = O(n^2 \log_2 n), \quad \mathcal{C}_{\text{fact}} = O(n^{3/2}), \quad \mathcal{S}_{\text{mem}} = O(n \log_2 n).$$

These results indicate that we can allow flexible rank patterns to achieve similar results, and enhance the applicability of HSS methods. Moreover, notice that the memory sizes (and also the solution costs) are both nearly linear in n. It is possible to generalize them to other structured matrices such as sequentially semiseparable matrices [10], quasiseparable matrices [3, 19], and $\mathcal{H}^2$ matrices. Similar results can also be derived for structured sparse solvers based on HSS matrices as in [41].

As a numerical example, we consider some ill-conditioned Toeplitz matrices, which are converted into Cauchy-like matrices with the aid of displacement structures (see, e.g., [21, 38]). It is shown that these Cauchy-like matrices have the low-rank property

TABLE 6.1
Complexity of two HSS algorithms applied to some Cauchy-like matrices in Toeplitz linear system solutions.

n	256	512	1024	2048	4096	8192	16384	32768	65536
HSS construction ($\frac{\text{cost}}{n^2}$)	40.54	52.9	59.4	59.9	60.1	60.1	59.3	58.9	58.7
ULV HSS solution ($\frac{\text{cost}}{10^4 n}$)	2.0	2.0	2.2	1.9	1.9	2.0	2.0	2.0	2.0

and the HSS rank is bounded by $O(\log n)$ [15, 37]. It can also be shown that the ranks satisfy part 1 of Theorem 6.1 with $p = 1$. Here, we just use the low-rank property and ignore any further special structures so as to test the performance of regular HSS algorithms. See Table 6.1, which verifies that with the relaxed rank bounds, the order of the complexity is still similar to that with the classical rank bound, although the prefactor or constant in the count may be larger.

7. Conclusions. This paper gives systematic complexity analysis for some HSS algorithms. Detailed flop counts are conducted. This helps the development of several strategies for improving existing HSS methods and for developing new fast ones, including modified HSS construction and solution and new HSS recompression. We also relax classical rank requirements to allow more flexibility in using HSS methods. The analysis is verified by some numerical examples. The ideas are also useful in sparse structured methods.

Acknowledgments. The author is very grateful to the anonymous referees for their valuable suggestions. The author would also like to thank Professors Shivkumar Chandrasekaran, Ming Gu, and Dan Jiao for some useful discussions.

REFERENCES

[1] M. BEBENDORF, *Why finite element discretizations can be factored by triangular hierarchical matrices*, SIAM J. Numer. Anal., 45 (2007), pp. 1472–1494.
[2] M. BEBENDORF, *Efficient inversion of Galerkin matrices of general second-order elliptic differential operators with nonsmooth coefficients*, Math. Comp., 74 (2005), pp. 1179–1199.
[3] T. BELLA, Y. EIDELMAN, I. GOHBERG, AND V. OLSHEVSKY, *Computations with quasiseparable polynomials and matrices*, Theoret. Comput. Sci., 409 (2008), pp. 158–179.
[4] D. A. BINI, P. BOITO, Y. EIDELMAN, L. GEMIGNANI, AND I. GOHBERG, *A fast implicit QR eigenvalue algorithm for companion matrices*, Linear Algebra Appl., 432 (2010), pp. 2006–2031.
[5] S. BÖRM, *Construction of data-sparse $\mathcal{H}^2$-matrices by hierarchical compression*, SIAM J. Sci. Comput., 31 (2009), pp. 1820–1839.
[6] S. BÖRM AND L. GRASEDYCK, *Hybrid cross approximation of integral operators*, Numer. Math., 101 (2005), pp. 221–249.
[7] S. BÖRM, L. GRASEDYCK, AND W. HACKBUSCH, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem, 27 (2003), pp. 405–422.
[8] W. CHAI AND D. JIAO, *An $\mathcal{H}^2$-matrix-based integral-equation solver of reduced complexity and controlled accuracy for solving electrodynamic problems*, IEEE Trans. Antennas and Propagation, 57 (2009), pp. 3147–3159.
[9] S. CHANDRASEKARAN, P. DEWILDE, M. GU, W. LYONS, AND T. PALS, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.
[10] S. CHANDRASEKARAN, P. DEWILDE, M. GU, T. PALS, X. SUN, A.-J. VAN DER VEEN, AND D. WHITE, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.
[11] S. CHANDRASEKARAN, P. DEWILDE, M. GU, AND N. SOMASUNDERAM, *On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2261–2290.
[12] S. CHANDRASEKARAN, M. GU, X. S. LI, AND J. XIA, *Some fast algorithms for hierarchically semiseparable matrices*, LBNL Technical Report LBNL-62897, Berkeley, CA, 2007.

[13] S. Chandrasekaran, M. Gu, and W. Lyons, *A fast adaptive solver for hierarchically semiseparable representations*, Calcolo, 42 (2005), pp. 171–185.
[14] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.
[15] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu, *A superfast algorithm for Toeplitz systems of linear equations*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1247–1266.
[16] S. Delvaux and M. Van Barel, *A QR-based solver for rank structured matrices*, SIAM. J. Matrix Anal. Appl., 30 (2008), pp. 464–490.
[17] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
[18] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.
[19] Y. Eidelman, I. Gohberg, and V. Olshevsky, *The QR iteration method for Hermitian quasiseparable matrices of an arbitrary order*, Linear Algebra Appl., 404 (2005), pp. 305–324.
[20] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: Hierarchical matrix representation*, Comm. Pure Appl. Math., 64 (2011), pp. 697–735.
[21] M. Fiedler, *Hankel and Loewner matrices*, Linear Algebra Appl., 58 (1984), pp. 75–95.
[22] I. Gohberg, T. Kailath, and I. Koltracht, *Linear complexity algorithms for semiseparable matrices*, Integral Equations Operator Theory, 8 (1985), pp. 780–804.
[23] G. H. Golub and C. V. Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[24] L. Grasedyck, R. Kriemann, and S. Le Borne, *Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems*, Comput. Vis. Sci., 11 (2008), pp. 273–291.
[25] M. Gu and J. Xia, *A numerically stable and superfast algorithm for solving Toeplitz systems of linear equations*, in Proceedings of the SIAM Conference on Applied Linear Algebra (LA09), Monterey, CA, 2009.
[26] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part* I: *Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[27] W. Hackbusch, *Hierarchische Matrizen: Algorithmen und Analysis*, Springer, Berlin, 2009.
[28] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.
[29] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part* II: *Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.
[30] W. Hackbusch, B. Khoromskij, and S. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, H. Bungartz, R. H. W. Hoppe, and C. Zenger, eds., Springer, Berlin, 2000, pp. 9–29.
[31] 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.
[32] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Rev., 34 (1992), pp. 82–109.
[33] W. Lyons, *Fast Algorithms with Applications to PDEs*, Ph.D. thesis, University of California, Santa Barbara, CA, 2005.
[34] 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.
[35] P. G. Martinsson, *A fast direct solver for a class of elliptic partial differential equations*, J. Sci. Comput., 38 (2009), pp. 316–330.
[36] 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.
[37] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A fast algorithm for the inversion of general Toeplitz matrices*, Comput. Math. Appl., 50 (2005), pp. 741–752.
[38] V. Pan, *On computations with dense structured matrices*, Math. Comp., 55 (1990), pp. 179–190.
[39] Z. Sheng, P. Dewilde, and S. Chandrasekaran, *Algorithms to Solve Hierarchically Semiseparable Systems*, Oper. Theory Adv. Appl. 176, Birkhäuser, Basel, 2007, pp. 255–294.
[40] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, *A bibliography on semiseparable matrices*, Calcolo, 42 (2005), pp. 249–270.
[41] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.
[42] 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.
[43] J. Xia and M. Gu, *Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2899–2920.