Operator Theory:
Advances and Applications, Vol. 176, 255–294

Algorithms to Solve Hierarchically Semi-separable Systems

Zhifeng Sheng, Patrick Dewilde and Shivkumar Chandrasekaran

Abstract. 'Hierarchical Semi-separable' matrices (HSS matrices) form an important class of structured matrices for which matrix transformation algorithms that are linear in the number of equations (and a function of other structural parameters) can be given. In particular, a system of linear equations $Ax = b$ can be solved with linear complexity in the size of the matrix, the overall complexity being linearly dependent on the defining data. Also, LU and ULV factorization can be executed 'efficiently', meaning with a complexity linear in the size of the matrix. This paper gives a survey of the main results, including a proof for the formulas for LU-factorization that were originally given in the thesis of Lyon [1], the derivation of an explicit algorithm for ULV factorization and related Moore-Penrose inversion, a complexity analysis and a short account of the connection between the HSS and the SSS (sequentially semi-separable) case. A direct consequence of the computational theory is that from a mathematical point of view the HSS structure is 'closed' for a number operations. The HSS complexity of a Moore-Penrose inverse equals the HSS complexity of the original, for a sum and a product of operators the HSS complexity is no more than the sum of the individual complexities.

Mathematics Subject Classification (2000). Primary 65F; Secondary 05

Keywords. Hierarchically Semi-separable system, structured matrices, direct solver, ULV factorization

1. Introduction

The term 'semi-separable systems' originated in the work of Gohberg, Kailath and Koltracht [2] where these authors remarked that if an integral kernel is approximated by an outer sum, then the system could be solved with a number of operations essentially determined by the order of the approximation rather than by a power of the number of input and output data. In the same period, Greengard and Rokhlin [3, 4] proposed the 'multipole method' where an integral kernel such as a Green's function is approximated by an outer product resulting in a matrix in which large sub-matrices have low rank. These two theories evolved in

parallel in the system theoretical literature and the numerical literature. In the system theoretical literature it was realized that an extension of the semi-separable model (sometimes called 'quasi-separability') brings the theory into the realm of time-varying systems, with its rich theory of state realization, interpolation, model order reduction, factorization and embedding [5]. In particular, it was shown in [6] that, based on this theory, a numerically backward stable solver of low complexity can be derived realizing a URV factorization of an operator T, in which U and V are low unitary matrices of state dimensions at most as large as those of T and R is causal, outer and also of state dimensions at most equal those of T. Subsequently, this approach has been refined by a number of authors, a.o. [7, 8, 9].

Although the SSS theory leads to very satisfactory results when applicable, it also became apparent in the late nineties that it is insufficient to cover major physical situations in which it would be very helpful to have system solvers of low complexity – in wiew of the often very large size of the matrices involved. Is it possible to extend the framework of SSS systems so that its major properties remain valid, in particular the fact that the class is closed under system inversion? The HSS theory, pioneered by Chandrasekaran and Gu [10] provides an answer to this question. It is based on a different state space model than the SSS theory, namely a hierarchical rather than a sequential one, but it handles the transition operators very much in the same taste. Based on this, a theory that parallels the basic time-varying theory of [5] can be developed, and remarkably, many results carry over. In the remainder of this paper we recall and derive some major results concerning system inversion, and discuss some further perspectives. The remainder sections of this introduction are devoted to a brief summary of the construction of SSS systems which lay at the basis of the HSS theory. In the numerical literature, the efforts have been concentrated on 'smooth' matrices, i.e., matrices in which large sub matrices can be approximated by low rank matrices thanks to the fact that their entries are derived from smooth kernels [11, 12]. Both the SSS and HSS structures are more constrained than the 'H-matrices' considered by Hackbusch a.o., but they do have the desirable property that they are closed under inversion and fit naturally in a state space framework. In the sequel we explore in particular the state space structure of HSS systems, other structures such as hierarchical multi-band decomposition have also been considered [13] but are beyond the present scope.

Our basic context is that of block matrices or operators $T = [T_{i,j}]$ with rows of dimensions $\ldots, m_{-1}, m_0, m_1, \ldots$ and column dimensions $\ldots, n_{-1}, n_0, n_1, \ldots$. Any of these dimensions may be zero, resulting in an empty row or column (matrix calculus can easily be extended to cover this case, the main rule being that the product of a matrix of dimensions $m \times 0$ with a matrix of dimensions $0 \times n$ results in a zero matrix of dimensions $m \times n$). Concentrating on an upper block matrix (i.e., when $T_{i,j} = 0$ for $i > j$), we define the *the degree of semi-separability* of T as the sequence of ranks $[\delta_i]$ of the matrices H_i where H_i is the sub-matrix corresponding to the row indices $\ldots, n_{i-2}, n_{i-1}$ and the column indices $m_i, m_{i+1}, \ldots$. H_i is called the ith Hankel operator of the matrix T. In case of infinite-dimensional operators,

we say that the system is *locally finite* if all H_i have finite dimensions. Corresponding to the local dimension δ_i there are minimal factorizations $H_i = \mathcal{C}_i\mathcal{O}_i$ into what are called the i^{th} *controllability matrix* $\mathcal{C}_i$ and *observability matrix* $\mathcal{O}_i$, of dimensions $(\sum_{k=i-1}^{-\infty} m_k) \times \delta_i$ and $\delta_i \times (\sum_{k=i}^{\infty} n_k)$. Connected to such a system of factorizations there is an indexed realization $\{A_i, B_i, C_i, D_i\}$ of dimensions $\{\delta_i \times \delta_{i+1}, m_i \times \delta_{i+1}, \delta_i \times n_i, m_i \times n_i\}$ constituting a local set of 'small' matrices with the characteristic property of semi-separable realizations for which it holds that

$$\left\{ \begin{array}{ccc} \mathcal{C}_i = \begin{bmatrix} \vdots \\ B_{i-2}A_{i-1} \\ B_{i-1} \end{bmatrix} & , & \mathcal{O}_i = \begin{bmatrix} C_i & A_iC_{i+1} & A_iA_{i+1}C_{i+2} & \cdots \end{bmatrix} \\ T_{i,j} = D_i & \text{for} & i = j \\ T_{i,j} = B_iA_{i+1}\cdots A_{j-1}C_j & \text{for} & i < j. \end{array} \right. \tag{1.1}$$

The vector-matrix multiplication $y = uT$ can be represented by local state space computations

$$\left\{ \begin{array}{rcl} x_{i+1} & = & x_iA_i + u_iB_i \\ y_i & = & x_iC_i + u_iD_i. \end{array} \right. \tag{1.2}$$

The goal of most semi-separable computational theory (as done in [5]) is to perform computations with a complexity linear in the overall dimensions of the matrix, and some function of the degree δ_i, preferably linear, but that is often not achievable (there is still quite some work to do on this topic even in the SSS theory!). The above briefly mentioned realization theory leads to nice representations of the original operator. To this end we only need to introduce a shift operator Z with the characteristic property $Z_{i,i+1} = I$, zero elsewhere, where the dimension of the unit matrix is context dependent, and global representations for the realization as block diagonal operators $\{A = \text{diag}[A_i], B = \text{diag}[B_i], C = \text{diag}[C_i], D = \text{diag}[D_i]\}$. The lower triangular part can of course be dealt with in the same manner as the upper, resulting in the general semi-separable representation of an operator as (the superscript 'H' indicates Hermitian conjugation)

$$T = B_\ell Z^H (I - A_\ell Z^H)^{-1} C_\ell + D + B_u Z (I - A_u Z)^{-1} C_u \tag{1.3}$$

in which the indices refer to the lower, respect. upper semi-separable decomposition. In general we assume that the inverses in this formula do exist and have reasonable bounds, if that is not the case one has to resort to different techniques that go beyond the present exposition. In the finite-dimensional case the matrix $(I - AZ)$ takes the special form when the indexing runs from 0 to n (for orientation the $0,0$ element is boxed in):

$$(I - AZ) = \begin{bmatrix} \boxed{I} & A_0 & & & \\ & I & A_1 & & \\ & & \ddots & \ddots & \\ & & & I & A_n \\ & & & & I \end{bmatrix} \tag{1.4}$$

one may think that this matrix is always invertible, but that is numerically not true, how to deal with numerical instability in this context is also still open territory.

The SSS theory (alias time-varying system theory) has produced many results paralleling the classical LTI theory and translating these results to a matrix context, (see [5] for a detailed account):

- *System inversion*: $T = URV$ in which the unitary matrices U, V and the outer matrix R (outer means: upper and upper invertible) are all semi-separable of degree at most the degree of T;
- *System approximation and model reduction*: sweeping generalizations of classical interpolation theory of the types Nevanlinna-Pick, Caratheodory-Fejer and even Schur-Takagi, resulting in a complete model reduction theory of the 'AAK-type' but now for operators and matrices;
- *Cholesky and spectral factorization*: $T = FF^*$ when T is a positive operator, in which F is semi-separable of the same degree sequence as T – a theory closely related to Kalman filtering;
- and many more results in embedding theory and minimal algebraic realization theory.

2. Hierarchical semi-separable systems

The Hierarchical Semi-Separable representation of a matrix (or operator) A is a layered representation of the multi-resolution type, indexed by the hierarchical level. At the top level 1, it is a 2×2 block matrix representation of the form (notice the redefinition of the symbol A):

$$A = \begin{bmatrix} A_{1;1,1} & A_{1;1,2} \\ A_{1;2,1} & A_{1;2,2} \end{bmatrix} \tag{2.1}$$

in which we implicitely assume that the ranks of the off-diagonal blocks are low so that they can be represented by an 'economical' factorization ('H' indicates Hermitian transposition, for real matrices just transposition), as follows:

$$A = \begin{bmatrix} D_{1;1} & U_{1;1}B_{1;1,2}V_{1;2}^H \\ U_{1;2}B_{1;2,1}V_{1;1}^H & D_{1;2} \end{bmatrix}. \tag{2.2}$$

The second hierarchical level is based on a further but similar decomposition of the diagonal blocks, respect. $D_{1;1}$ and $D_{1;2}$:

$$\begin{aligned} D_{1;1} &= \begin{bmatrix} D_{2:1} & U_{2;1}B_{2;1,2}V_{2;2}^H \\ U_{2;2}B_{2;2,1}V_{2;1}^H & D_{2;2} \end{bmatrix} \\ D_{1;2} &= \begin{bmatrix} D_{2;3} & U_{2;3}B_{2;3,4}V_{2;4}^H \\ U_{2;4}B_{2;4,3}V_{2;3}^H & D_{2;4} \end{bmatrix} \end{aligned} \tag{2.3}$$

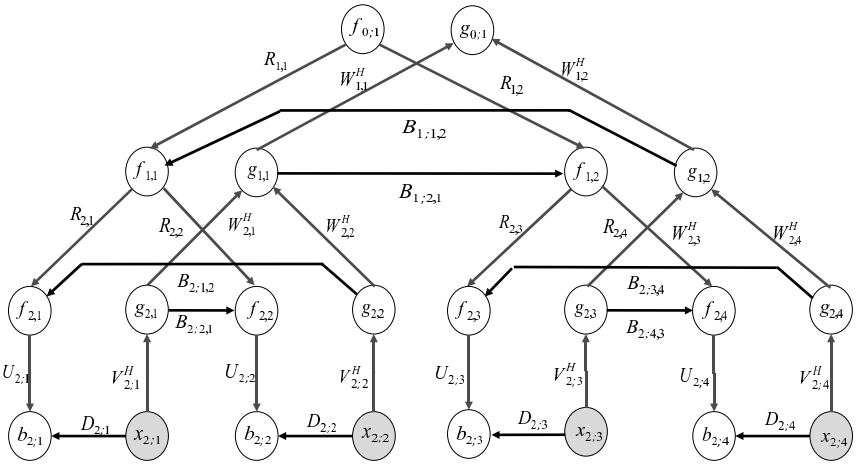

FIGURE 1. HSS Data-flow diagram for a two level hierarchy representing operator-vector multiplication, arrows indicate matrix-vector multiplication of sub-data, nodes correspond to states and are summing incoming data (the top levels f_0 and g_0 are empty).

for which we have the further *level compatibility* assumption (the 'span operator' refers to the column vectors of the subsequent matrix)

$$\text{span}(U_{1;1}) \subset \text{span}\left(\begin{bmatrix} U_{2;1} \\ 0 \end{bmatrix}\right) \oplus \text{span}\left(\begin{bmatrix} 0 \\ U_{2;2} \end{bmatrix}\right), \tag{2.4}$$

$$\text{span}(V_{1;1}) \subset \text{span}\left(\begin{bmatrix} V_{2;1} \\ 0 \end{bmatrix}\right) \oplus \text{span}\left(\begin{bmatrix} 0 \\ V_{2;2} \end{bmatrix}\right) \text{ etc.} \tag{2.5}$$

This spanning property is characteristic for the HSS structure, it allows for a substantial improvement on the numerical complexity for, e.g., matrix-vector multiplication as a multiplication with the higher level structures always can be done using lower level operations, using the *translation operators*

$$U_{1;i} = \begin{bmatrix} U_{2;2i-1}R_{2;2i-1} \\ U_{2;2i}R_{2;2i} \end{bmatrix}, \quad i = 1, 2, \tag{2.6}$$

$$V_{1;i} = \begin{bmatrix} V_{2;2i-1}W_{2;2i-1} \\ V_{2;2i}W_{2;2i} \end{bmatrix}, \quad i = 1, 2. \tag{2.7}$$

Notice the use of indices: at a given level i rows respect. columns are subdivided in blocks indexed by $1, \ldots, i$. Hence the ordered index $(i; k, \ell)$ indicates a block at level i in the position (k, ℓ) in the original matrix. The same kind of subdivision can be used for column vectors, row vectors and bases thereof (as are generally represented in the matrices U and V).

In [14] it is shown how this multilevel structure leads to efficient matrix-vector multiplication and a set of equations that can be solved efficiently as well. For the

sake of completeness we review this result briefly. Let us assume that we want to solve the system $Tx = b$ and that T has an HSS representation with deepest hierarchical level K. We begin by accounting for the matrix-vector multiplication Tx. At the leave node $(K;i)$ we can compute

$$g_{K;i} = V^H_{K;i} x_{K;i}.$$

If $(k;i)$ is not a leaf node, we can infer, using the hierarchical relations

$$g_{k;i} = V^H_{k;i} x_{k;i} = W^H_{k+1;2i-1} g_{k+1;2i-1} + W^H_{k+1;2i} g_{k+1;2i}.$$

These operations update a 'hierarchical state' $g_{k;i}$ upward in the tree. To compute the result of the multiplication, a new collection of state variables $\{f_{k;i}\}$ is introduced for which it holds that

$$b_{k;i} = T_{k;i,i} + U_{k;i} f_{k;i}$$

and which can now be computed recursively downward by the equations

$$\begin{bmatrix} f_{k+1;2i-1} \\ f_{k+1;2i} \end{bmatrix} = \begin{bmatrix} B_{k+1;2i-1,2i} g_{k+1;2i} + R_{k+1;2i-1} f_{k,i} \\ B_{k+1;2i,2i-1} g_{k+1;2i-1} + R_{k+1;2i} f_{k;i} \end{bmatrix},$$

the starting point being $f_{0;} = []$, an empty matrix. At the leaf level we can now compute (at least in principle – as we do not know x) the outputs from

$$b_{K;i} = D_{K;i} x_{K;i} + U_{K;i} f_{K;i}.$$

The next step is to represent the multiplication recursions in a compact form using matrix notation and without indices. We fix the maximum order K as before. We define diagonal matrices containing the numerical information, in breadth first order:

$$\mathbf{D} = \mathrm{diag}[D_{K;i}]_{i=1,\ldots,K},\ \mathbf{W} = \mathrm{diag}[(W_{1;i})_{i=1,2}, (W_{2;i})_{i=1\cdots 4}, \ldots],\ \text{etc.}$$

Next, we need two shift operators relevant for the present situation, much as the shift operator Z in time-varying system theory explained above. The first one is the shift-down operator $Z_\downarrow$ on a tree. It maps a node in the tree on its children and is a nilpotent operator. The other one is the level exchange operator $Z_\leftrightarrow$. At each level it is a permutation that exchanges children of the same node. Finally, we need the leaf projection operator $\mathbf{P}_{\mathrm{leaf}}$ which on a state vector which assembles in breadth first order all the values $f_{k;i}$ produces the values of the leaf nodes (again in breadth first order). The state equations representing the efficient multiplication can now be written as

$$\left\{ \begin{array}{rcccc} \mathbf{g} & = & \mathbf{P}^H_{\mathrm{leaf}} \mathbf{V}^H \mathbf{x} & + & Z^H_\downarrow \mathbf{W}^H \mathbf{g} \\ \mathbf{f} & = & \mathbf{R} Z_\downarrow \mathbf{f} & + & \mathbf{B} Z_\leftrightarrow \mathbf{g} \end{array} \right. \tag{2.8}$$

while the 'output' equation is given by

$$\mathbf{b} \ = \ \mathbf{D}\mathbf{x} \ + \ \mathbf{U}\mathbf{P}_{\mathrm{leaf}}\mathbf{f}\ . \tag{2.9}$$

This is the resulting HSS state space representation that parallels the classical SSS state space formulation reviewed above. Written in terms of the hidden state space quantities we find

$$\begin{bmatrix} (I - Z_{\downarrow}^{H}\mathbf{W}^{H}) & 0 \\ -BZ_{\leftrightarrow}\mathbf{g} & (I - RZ_{\downarrow}) \end{bmatrix} \begin{bmatrix} \mathbf{g} \\ \mathbf{f} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{\text{leaf}}^{H}\mathbf{V}^{H} \\ 0 \end{bmatrix} \mathbf{x}. \tag{2.10}$$

The state quantities can always be eliminated in the present context as $(I - \mathbf{W}Z_{\downarrow})$ and $(I - \mathbf{R}Z_{\downarrow})$ are invertible operators due to the fact that $Z_{\downarrow}$ is nilpotent. We obtain as a representation for the original operator

$$T\mathbf{x} = (\mathbf{D} + \mathbf{U}\mathbf{P}_{\text{leaf}}(I - \mathbf{R}Z_{\downarrow})^{-1}\mathbf{B}Z_{\leftrightarrow}(I - Z_{\downarrow}^{H}\mathbf{W}^{H})^{-1}\mathbf{P}_{\text{leaf}}^{H}\mathbf{V}^{H})\mathbf{x} = \mathbf{b}. \tag{2.11}$$

3. Matrix operations based on HSS representation

In this section we describe a number of basic matrix operations based on the HSS representation. Matrix operations using the HSS representation are normally much more efficient than operations with plain matrices. Many matrix operations can be done with a computational complexity (or sequential order of basic operations) linear with the dimension of the matrix. These fast algorithms to be described are either collected from other publications [14, 10, 1, 15] or new. We will handle a somewhat informal notation to construct new block diagonals. Suppose, e.g., that $\mathbf{R}_A$ and $\mathbf{R}_B$ are conformal block diagonal matrices from the description given in the preceding section, then the construction operator inter$[\mathbf{R}_A|\mathbf{R}_B]$ will represent a diagonal operator in which the diagonal entries of the two constituents are block-column-wise intertwined:

$$\text{inter}[\mathbf{R}_A|\mathbf{R}_B] = \text{diag}\left[\begin{bmatrix} R_{A;1;1} & R_{B;1;1} \end{bmatrix}, \begin{bmatrix} R_{A;1;2} & R_{B;1;2} \end{bmatrix}, \begin{bmatrix} R_{A;2;1} & R_{B;2;1} \end{bmatrix}, \dots\right].$$

Block-row intertwining

$$\text{inter}[\mathbf{W}_A|\mathbf{W}_B] = \text{diag}\left[\begin{bmatrix} W_{A;1;1} \\ W_{B;1;1} \end{bmatrix}, \begin{bmatrix} W_{A;1;2} \\ W_{B;1;2} \end{bmatrix}, \begin{bmatrix} W_{A;2;1} \\ W_{B;2;1} \end{bmatrix}, \dots\right].$$

matrix intertwining is defined likewise.

3.1. HSS addition

Matrix addition can be done efficiently with HSS representations. The addition algorithm for Sequentially semi-separable representation has been presented in [16]. The addition algorithm for HSS representation which has been studied in [1] is quite similar.

3.1.1. Addition with two commensurately partitioned HSS matrices.

When adding two HSS commensurately partitioned matrices together, the sum will be an HSS matrix with the same partitioning. Let $C = A + B$ where A is

defined by sequences U_A, V_A, D_A, R_A, W_A and B_A ; B is defined by sequences U_B, V_B, D_B, R_B, W_B and B_B.

$$\begin{cases} \mathbf{R}_C = \text{inter}\left[\begin{array}{c|c} \mathbf{R}_A & 0 \\ \hline 0 & \mathbf{R}_B \end{array}\right] \\ \mathbf{W}_C = \text{inter}\left[\begin{array}{c|c} \mathbf{W}_A & 0 \\ \hline 0 & \mathbf{W}_B \end{array}\right] \\ \mathbf{B}_C = \text{inter}\left[\begin{array}{c|c} \mathbf{B}_A & 0 \\ \hline 0 & \mathbf{B}_B \end{array}\right] \\ \mathbf{U}_C = \text{inter}\left[\begin{array}{c|c} \mathbf{U}_A & \mathbf{U}_B \end{array}\right] \\ \mathbf{V}_C = \text{inter}\left[\begin{array}{c|c} \mathbf{V}_A & \mathbf{V}_B \end{array}\right] \\ \mathbf{D}_C = \mathbf{D}_A + \mathbf{D}_B. \end{cases} \tag{3.1}$$

The addition can be done in time proportional to the number of entries in the representation. Note that the computed representation of the sum may not be efficient, in the sense that the HSS complexity of the sum increases additively. It is quite possible that the HSS representation is not minimal as well, as is the case when $A = B$. In order to get an efficient HSS representation, we could do *fast model reduction* (described in [15]) or *compression* (to be presented later) on the resulting HSS representation. However, these operations might be too costly to be applied frequently, one could do *model reduction* or *compression* after a number of additions.

3.1.2. Adaptive HSS addition. When two HSS matrices of the same dimensions do not have the same depth, leaf-split or leaf-merge operations described in [15] are needed to make these two HSS representations compatible. Note that we have two choices: we can either split the shallower HSS tree to make it compatible with the deeper one, or we can do leaf-merge on the deeper tree to make it compatible with shallower one. From the point of view of computation complexity, leaf-merge is almost always preferred since it amounts to several matrix multiplications with small matrices (ideally); leaf-split needs several factorization operations which are more costly than matrix multiplications. However, this does not imply leaf-merge should always be used if possible. Keeping in mind the fact that the efficiency of the HSS representation also comes from a deeper HSS tree with smaller translation matrices, the HSS tree should be kept deep enough to capture the low rank off-diagonal blocks. On the other hand, it is obviously impossible to always apply leaf-merge or leaf-split, because one HSS tree may have both a deeper branch and a shallower one than the other HSS tree does.

3.1.3. HSS addition with rank-m matrices. The sum of a level-n hierarchically semi-separable matrix A and a rank-m matrix UBV^H is another level-n hierarchically semi-separable matrix $A' = A + UBV^H$. A rank-m matrix has an almost trivial HSS representation conformal to any hierarchical scheme. With such a representation the HSS addition described in Section 3.1.1 is applicable.

In order to add two matrices together, the rank-m matrix should be represented in a form compatible with the HSS matrix. That is, the rank-m matrix

will have to be partitioned recursively according to the partitioning of the HSS matrix A.

Let us first denote U as $U_{0;1}$, V as $V_{0;1}$, UBV^H as $D_{0;1}$. We partition U and V according to the partition of matrix A as follows:
for $k = 0, 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^k$:

$$U_{k;i} = \begin{bmatrix} U_{k+1;2i-1} \\ U_{k+1;2i} \end{bmatrix} \quad V_{k;i} = \begin{bmatrix} V_{k+1;2i-1} \\ V_{k+1;2i} \end{bmatrix}$$

Then at the first level of the partition:

$$U_{0;1}BV_{0;1}^H = \begin{bmatrix} U_{1;1}BV_{1;1}^H & U_{1;1}BV_{1;2}^H \\ U_{1;2}BV_{1;1}^H & U_{1;2}BV_{1;2}^H \end{bmatrix}$$

and following levels are given by:

Theorem 1. *The level-n HSS representation of the rank-m matrix UBV^H is: for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^k$ and $\langle i \rangle = i + 1$ for odd i, $\langle i \rangle = i - 1$ for even i:*

$$\begin{cases} \widehat{D}_{k;i} = U_{k;i}BV_{k;i}^H & \widehat{R}_{k;i} = I \\ \widehat{W}_{k;i} = I & \widehat{B}_{k;i,\langle i \rangle} = B \\ \widehat{U}_{k;i} = U_{k;i} & \widehat{V}_{k;i} = V_{k;i} \end{cases} \tag{3.2}$$

$D_{k;i}$ are again rank-m matrices, assuming recursive correctness of this constructive method, $D_{k;i}$ can also be partitioned and represented recursively.

Other ways of constructing HSS representations for rank-m matrices are possible. One is to firstly form a one-level HSS representation for the rank-m matrix and then use the leaf-split algorithm [15] to compute its HSS representation according to certain partitioning. In principle, this method leads to an efficient HSS tree in the sense that its column bases and row bases are irredundant. However, this method needs much more computations. If m is reasonably small, the method described in this section is recommended.

3.1.4. HSS addition with rank-m matrices with hierarchically semi-separable bases.

In HSS representations, the column bases and row bases of the HSS nodes are not explicitly stored. This means when we compute $\widehat{A} = A + UBV^H$, U and V are probably not explicitly stored, instead, they are implicitly stored with the formulas (2.6) and (2.7).

We can of course compute these row bases and column bases and then construct an HSS representation for UBV^H with the method described in the last subsection. This is not recommended because computing U and V may be costly and not memory efficient.

Theorem 2. *Suppose U and V are defined in HSS form, the HSS representation of UBV^H is given by the following formulas:*

for $k = 2, 3, \ldots, n$; $i \in 1, 2, \ldots, 2^k$; *and* $\langle i \rangle = i + 1$ *for odd* i, $\langle i \rangle = i - 1$ *for even* i:

$$\begin{cases} \widehat{W}_{1;1} = I & \widehat{W}_{1;2} = I & \widehat{R}_{1;1} = I \\ \widehat{R}_{1;2} = I & \widehat{B}_{1;1,2} = B & \widehat{B}_{1;2,1} = B \\ \widehat{W}_{k;i} = W_{k-1;\lceil \frac{i}{2} \rceil} & \widehat{R}_{k;i} = R_{k-1;\lceil \frac{i}{2} \rceil} & \widehat{B}_{k;i,\langle i \rangle} = R_{k-1;\lceil \frac{i}{2} \rceil} \widehat{B}_{k-1;\lceil \frac{i}{2} \rceil, \langle \lceil \frac{i}{2} \rceil \rangle} W^H_{k-1;\lceil \frac{i}{2} \rceil} \\ \widehat{U}_{n;i} = U_{n;i} R_{n;i} & \widehat{V}_{n;i} = V_{n;i} W_{n;i} & \widehat{D}_{n;i} = U_{n;i} \widehat{B}_{n;\lceil \frac{i}{2} \rceil, \langle \lceil \frac{i}{2} \rceil \rangle} V^H_{n;i}. \end{cases} \tag{3.3}$$

After having the HSS representation of UBV^H, the sum can be computed easily using the HSS addition algorithm described in Section 3.1.1.

3.2. HSS matrix-matrix multiplication

Matrix-matrix multiplication can also be done in time linear with the dimensions of the matrices. The product $C = AB$ is another hierarchically semi-separable matrix.

A is a HSS matrix whose HSS representation is defined by sequences U_A, V_A, D_A, R_A, W_A, and B_A.

B is a HSS matrix whose HSS representation is defined by sequences U_B, V_B, D_B, R_B, W_B, and B_B.

3.2.1. Multiplication of two commensurately partitioned HSS matrices. When two HSS matrices are compatible, that is, they are commensurately partitioned, we can get the HSS representation of the product with the following algorithm. The algorithm was originally given with proof in Lyon's thesis [1].

The notations F and G to be used in following paragraphs represent the intermediate variables representing intermediate states in computing the HSS representation of C. They can be computed using the recursive formulas (3.4) to (3.7).

$F_{k;2i-1}$ represents the F intermediate variable propagated to the left children; similarly, $F_{k;2i}$ represents the intermediate F propagated to the right children. $G_{k;2i-1}$ represents the intermediate variable G coming from the left children; while $G_{k;2i}$ represents the intermediate variable G coming from the right ones. At last, $G_{n;i}$ represents the variable G calculated at leaves.

We first define the intermediate variables recursively via:

Definition 1. *For the multiplication of two level-n HSS matrices the upsweep recursion is defined as:*
for $i \in 1, 2, \ldots, 2^n$*:*

$$G_{n;i} = V^H_{A;n;i} U_{B;n;i} \tag{3.4}$$

for $k = n, \ldots, 2, 1$ *and* $i \in 1, 2, \ldots, 2^k$*:*

$$G_{k-1;i} = W^H_{A;k;2i-1} G_{k;2i-1} R_{B;k;2i-1} + W^H_{A;k;2i} G_{k;2i} R_{B;k;2i}. \tag{3.5}$$

Definition 2. *For the multiplication of two level-n HSS matrices the downsweep recursion is defined as:*
for $(i,j) = (1,2)$ *or* $(2,1)$*:*

$$F_{1;i} = B_{A;1;i,j} G_{1;j} B_{B;j,i} \tag{3.6}$$

for $i \in 1,2,\ldots,2^k$*,* $j = i+1$ *for odd* i*,* $j = i-1$ *for even* i *and* $k = 2,\ldots,n$*:*

$$F_{k;i} = B_{A;k;i,j} G_{k;j} B_{B;k;j,i} + R_{A;k;i} F_{k-1;\lceil \frac{i}{2} \rceil} W^H_{B;k,i}. \tag{3.7}$$

Theorem 3. *The HSS representation of the product is:*
for $i \in 1,2,\ldots,2^n$

$$\begin{cases} \widehat{D}_{n;i} = D_{A;n;i} D_{B;n;i} + U_{A;n;i} F_{n;i} V^H_{B;n;i} \\ \widehat{U}_{n;i} = \left[\begin{array}{c|c} U_{A;n;i} & D_{A;n;i} U_{B;n;i} \end{array}\right] \\ \widehat{V} = \left[\begin{array}{c|c} D^H_{B;n;i} V_{A;n;i} & V_{B;n;i} \end{array}\right] \end{cases} \tag{3.8}$$

for $k = 1,2,\ldots,n$*;* $i \in 1,2,\ldots,2^k$ *and* $j = i+1$ *for odd* i*,* $j = i-1$ *for even* i*:*

$$\begin{cases} \widehat{R}_{k;i} = \left[\begin{array}{c|c} R_{A;k;i} & B_{A;k;i,j} G_{k;j} R_{B;k;j} \\ \hline 0 & R_{B;k;i} \end{array}\right] \\ \widehat{W}_{k;i} = \left[\begin{array}{c|c} W_{A;k;i} & 0 \\ \hline B^H_{B;j,i} G^H_{k;j} W_{A;k;j} & W_{B;k;i} \end{array}\right] \\ \widehat{B}_{k;i,j} = \left[\begin{array}{c|c} B_{A;k;i,j} & R_{A;k;i} F_{k-1;\lceil \frac{i}{2} \rceil} W^H_{B;k;j} \\ \hline 0 & B_{B;k;i,j} \end{array}\right]. \end{cases} \tag{3.9}$$

Once again, the complexity of the HSS representation increases additively. *Model reduction* or *compression* may be needed to bring down the complexity. Note that, the algorithm above is given without proof. For a detailed proof and analysis, we refer to [1].

3.2.2. Adaptive HSS Matrix-Matrix multiplication. Adaptive multiplication is needed when two HSS matrices are not completely compatible, then leaf-split and leaf-merge are needed to make them compatible. The comment given in Section (3.1.2) for adaptive addition also applies here.

3.2.3. HSS Matrix-Matrix multiplication with rank-m matrices. A is a level-n HSS matrix whose HSS representation is defined by sequences U_A, V_A, D_A, R_A, W_A, and B_A. UBV^H is a rank-m matrix. The product $AUBV^H$ is also a level-n HSS matrix.

As we mentioned in Section 3.1.3, a rank-m matrix is a hierarchically semi-separable matrix and can be represented with a HSS representation. Then we can easily construct the HSS representation for the rank-m matrix and then perform the HSS Matrix-Matrix multiplication. This is the most straightforward way. However, making use of the fact that the translation matrices (R, W) of the rank-m matrix are identity matrices, the Matrix-Matrix multiplication algorithm can be simplified by substituting the R_B and W_B matrices in Section 3.2.1 with I matrices.

Again, because the complexity has been increased additively, *compression* or *Model reduction* could be helpful.

3.3. HSS matrix transpose

The transpose of a level-n HSS matrix will again be a level-n HSS matrix. Suppose the HSS matrix A is given by sequences B, R, W, U, V, D; it is quite easy to verify that

Theorem 4. *the HSS representation of the transpose A^H is given by the sequences: for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^k$ and $j = i + 1$ for odd i, $j = i - 1$ for even i:*

$$\begin{cases} \widehat{D}_{k;i} = D^H_{k;i} & \widehat{U}_{k;i} = V_{k;i} \quad \widehat{V}_{k;i} = U_{k;i} \\ \widehat{W}_{k;i} = R_{k;i} & \widehat{R}_{k;i} = W_{k;i} \quad \widehat{B}_{k;i,j} = B^H_{k;j,i}. \end{cases} \tag{3.10}$$

3.4. Generic inversion based on the state space representation

A state space representation for the inverse with the same state complexity can generically be given. We assume the existence of the inverse, the same hierarchical partitioning of the input and output vectors $\mathbf{x}$ and $\mathbf{b}$, and as generic conditions the invertibility of the direct operators $\mathbf{D}$ and $S = (I + \mathbf{B}Z_{\leftrightarrow}\mathbf{P}^H_{\text{leaf}}\mathbf{V}^H\mathbf{D}^{-1}\mathbf{U}\mathbf{P}_{\text{leaf}})$, the latter being a (very) sparse perturbation of the unit operator with a local (that is leaf based) inversion operator. Let $\mathbf{L} = \mathbf{P}^H_{\text{leaf}}\mathbf{V}^H\mathbf{D}^{-1}\mathbf{U}\mathbf{P}_{\text{leaf}}$, then we find

Theorem 5. *Under generic conditions, the inverse system T^{-1} has the following state space representation*

$$\begin{aligned} \begin{bmatrix} \mathbf{g} \\ \mathbf{f} \end{bmatrix} \;=\; & \left\{ \begin{bmatrix} I & \\ & 0 \end{bmatrix} - \begin{bmatrix} \mathbf{L} \\ -I \end{bmatrix} S^{-1} \begin{bmatrix} \mathbf{B}Z_{\leftrightarrow} & I \end{bmatrix} \right\} \\ & \cdot \left\{ \begin{bmatrix} Z^H_{\downarrow}\mathbf{W}^H & \\ & \mathbf{R}Z_{\downarrow} \end{bmatrix} \begin{bmatrix} \mathbf{g} \\ \mathbf{f} \end{bmatrix} + \begin{bmatrix} \mathbf{P}^H_{\text{leaf}}\mathbf{V}^H\mathbf{D}^{-1}\mathbf{b} \\ 0 \end{bmatrix} \right\} \end{aligned} \tag{3.11}$$

and the output equation

$$\mathbf{x} = -\mathbf{D}^{-1}\mathbf{U}\mathbf{P}_{\text{leaf}}\mathbf{f} + \mathbf{D}^{-1}\mathbf{b}. \tag{3.12}$$

The proof of the theorem follows from inversion of the output equation which involves the invertibility of the operator $\mathbf{D}$, and replacing the unknown $\mathbf{x}$ in the state equations, followed by a segregation of the terms that are directly dependent on the states and those that are dependent on the shifted states leading to the matrix $\begin{bmatrix} I & \mathbf{L} \\ -\mathbf{B}Z_{\leftrightarrow} & I \end{bmatrix}$ whose inverse is easily computed as the first factor in the right-hand side of the equation above. It should be remarked that this factor only involves operations at the leaf level of the hierarchy tree so that the given state space model can be efficiently executed (actually the inversion can be done using the original hierarchy tree much as is the case for the inversion of upper SSS systems).

Having the theorem, we can derive a closed formula for T^{-1} assuming the generic invertiblity conditions.

$$\begin{aligned} T^{-1} &= \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{U}\mathbf{P}_{\text{leaf}} \cdot \\ &\quad \cdot \left[I - \mathbf{R}Z_{\downarrow} + \mathbf{B}Z_{\leftrightarrow}(I - Z_{\downarrow}^{H}\mathbf{W}^{H})^{-1}\mathbf{P}_{\text{leaf}}^{H}\mathbf{D}^{-1}\mathbf{U}\mathbf{P}_{\text{leaf}} \right]^{-1} \cdot \\ &\quad \cdot \mathbf{B}Z_{\leftrightarrow}(I - Z_{\downarrow}^{H}\mathbf{W}^{H})^{-1}\mathbf{P}_{\text{leaf}}^{H}\mathbf{V}^{H}\mathbf{D}^{-1}. \end{aligned} \tag{3.13}$$

The equation given is a compact diagonal representation of T^{-1}, it also proves that the inverse of an invertible HSS matrix is again a HSS matrix of comparable complexity.

3.5. LU decomposition of HSS matrix

The formulas to compute the L and U factors of a square invertible matrix $T = LU$ in HSS form were originally given without proof in the thesis of Lyon [1] (they were checked computationally and evaluated in the thesis). Here we reproduce the formulas and give proof. The assumptions needed for the existence of the factorization are the same as is in the non-hierarchical case: a hierarchical tree that is n deep, the 2^n (block-) pivots have to be invertible.

The 'generic' situation (which occurs at each level in the HSS LU factorization) is a specialization of the classical Schur inversion theorem as follows:
we are given a matrix with the following 'generic' block decomposition

$$T = \begin{bmatrix} D_A & U_1 B_{12} V_2^H \\ U_2 B_{21} V_1^H & D_B \end{bmatrix} \tag{3.14}$$

in which D_A is a square invertible matrix, D_B is square (but not necessarily invertible), and T is invertible as well. Suppose we dispose of an LU factorization of the 11-block entry $D_A = L_A U_A$ and let us define two new quantities (which in the further proceedings will acquire an important meaning)

$$G_1 = V_1^H D_A^{-1} U_1, \quad F_2 = B_{21} G_1 B_{12}. \tag{3.15}$$

Then the first block step in a LU factorizaton of T is given by

$$T = \begin{bmatrix} L_A & \\ U_2 B_{21} V_1^H U_A^{-1} & I \end{bmatrix} \begin{bmatrix} I & \\ & D_B - U_2 F_2 V_2^H \end{bmatrix} \begin{bmatrix} U_A & L_A^{-1} U_1 B_{12} V_2^H \\ & I \end{bmatrix}. \tag{3.16}$$

The block entry $D_B - U_2 F_2 V_2^H$ is the classical 'Schur-complement' of D_A in the given matrix and it will be invertible if the matrix T is, as we assumed. At this point the first block column of the 'L' factor and the first block row of the 'U' matrix are known (the remainder will follow from an LU-decomposition of the Schur complement $D_B - U_2 F_2 V_2^H$). We see that the 21-entry in L and the 12-entry in U inherit the low rank of the originals with the same U_2, respect. V_2^H entry. In fact, more is true, the hierarchical relations in the first block column of L, respect. block row of U remain valid because $L_A = D_A U_A^{-1}$, respect. $U_A = L_A^{-1} D_A$, with modified row basis, respect. column basis. In the actual HSS computation the

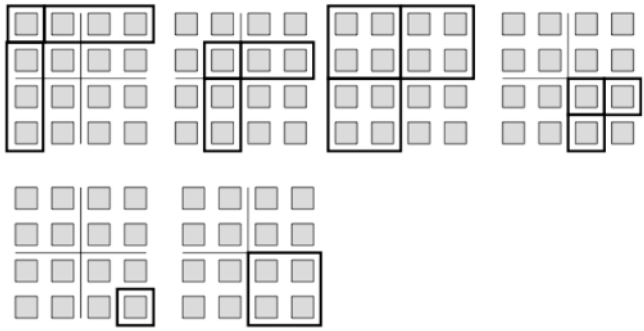

FIGURE 2. Recursive positioning of the LU first blocks in the HSS post-ordered LU factorization.

Schur complement will not be computed directly – it is lazily evaluated in what is called 'post-order traverse', meaning that each node (k, i) is evaluated only after evaluation of nodes $(k, \ell), \ell < k$ at the same level and its sub nodes $(k+1, 2i-1)$ and $(k+1, 2i)$.

This basic step can be interpreted as a specialization of the LU-factorization algorithm for sequentially separable systems, which reduces here to just two steps. In the first step the F_1 matrix is empty, the LU-factorization of $D_A = L_A U_A$ is done and the $V_1^{LH} = V_1^H U_A^{-1}$, respect. $U_1^U = L_A^{-1} U_1$ are computed. In the second step (in this case there are only two steps), G_1 is computed as $G_1 = V_1^{LH} U_1^U$, with $F_2 = B_{21} G_1 B_{12}$ and finally the Schur complement $D_B - U_2 F_2 V_2^H$ is evaluated (the sequential algorithm would be more complicated if more terms are available).

The HSS LU factorization is executed lazily in post-order traverse (w.r. to the hierarchical ordering of the blocks in the matrix), whereby previously obtained results are used as much as possible. For a tree that is 2 levels deep it goes as in Figure 2.

The collection of information needed to update the Schur complement at each stage of the algorithm is accomplished by an 'upward' movement, represented by the G matrices. Once a certain node (k, i) is reached, the $G_{k,i}$ equals the actual $V_1^H D_A^{-1} U_1$ pertaining to that node and hence subsumes all the data that is needed from previous steps to update the remaining Schur complement. However, the next 'lazy' step in the evaluation does not involve the whole matrix D_B, but only the at that point relevant top left corner matrix, the next candidate for reduction in the ongoing elimination – and determination of the next pivot. This restriction to the relevant top entry is accomplished by the matrix F, which takes information from the G's that are relevant at that level and specializes them to compute the contributions to the Schur-complement update of that specific matrix. Before formulating the algorithm precisely, we make this strategy that leads to efficient computations more precise.

Definition 3. *G propagates the quantity $V_1^H D_A^{-1} U_1$.*

Definition 4. *F propagates the quantity $B_{21} V_1^H D_A^{-1} U_1 B_{12}$ in equation* (3.16).

Updating G. The update situation involves exclusively the upward collection of the $G_{k,i}$. We assume that at some point in the recursion the matrices $G_{k,2i-1}$ and $G_{k,2i}$ are known, the objective is to compute $G_{k-1,i}$. The relevant observation here is that only this recursive data and data from the original matrix are needed to achieve the result. In matrix terms the situation is as follows:

$$\left[\begin{array}{cc|c} D_\ell & U_\ell \widehat{B}_u V_r^H & U_\ell R_\ell[\cdots] \\ U_r \widehat{B}_\ell V_\ell^H & D_r & U_r R_r[\cdots] \\ \hline [\cdots] W_\ell V_\ell^H & [\cdots] W_r^H V_r^H & D_B \end{array}\right] \tag{3.17}$$

where $\widehat{B}_u$ stands for $B_{k;2i-1,2i}$, $\widehat{B}_\ell$ stands for $B_{k;2i,2i-1}$, the subscript 'ℓ' stands for the left branch in the hierarchy for which $G_\ell = G_{k,2i-1} = V_\ell^H D_\ell^{-1} U_\ell$ is known, while the subscript 'r' stands for the right branch, for which $G_r = G_{k,2i} = V_r^H C_r^{-1} U_r$ is known with $C_r = D_r - U_r \widehat{B}_\ell V_\ell^H D_\ell^{-1} U_\ell \widehat{B}_u V_r^H$ the Schur complement of the first block in the left top corner submatrix, the objective being to compute $G = G_{k-1,i}$ given by

$$G = V^H D^{-1} U = \left[\begin{array}{cc} W_\ell^H V_\ell^H & W_r^H V_r^H \end{array}\right] \left[\begin{array}{cc} D_\ell & U_\ell \widehat{B}_u V_r^H \\ U_r \widehat{B}_\ell V_\ell^H & D_r \end{array}\right]^{-1} \left[\begin{array}{c} U_\ell R_\ell \\ U_r R_r \end{array}\right] \tag{3.18}$$

(note that the entries indicated by '$[\cdots]$' in equation (3.17) are irrelevant for this computation, they are taken care of in the F-downdate explained further on, while the $\widehat{B}_u$ and $\widehat{B}_\ell$ subsume the B-data at this level, which are also not relevant at this point of the computation). Computing the inverse of the new Schur complement produces:

$$\begin{aligned} G = &\left[W_\ell^H V_\ell^H \quad W_r^H V_r^H\right] \cdot \\ &\cdot \begin{bmatrix} D_\ell^{-1} + D_\ell^{-1} U_\ell \widehat{B}_u V_r^H C_r^{-1} U_r \widehat{B}_\ell V_\ell^H D_\ell^{-1} & -D_\ell^{-1} U_\ell \widehat{B}_u V_r^H C_r^{-1} \\ -C_r^{-1} U_r \widehat{B}_\ell V_\ell^H D_\ell^{-1} & C_r^{-1} \end{bmatrix} \begin{bmatrix} U_\ell R_\ell \\ U_r R_r \end{bmatrix} \\ G = &\left[W_\ell^H \quad W_r^H\right] \cdot \\ &\cdot \begin{bmatrix} V_\ell^H D_\ell^{-1} U_\ell + V_\ell^H D_\ell^{-1} U_\ell \widehat{B}_u V_r^H C_r^{-1} U_r \widehat{B}_\ell V_\ell^H D_\ell^{-1} U_\ell & -V_\ell^H D_\ell^{-1} U_\ell \widehat{B}_u V_r^H C_r^{-1} U_r \\ -V_r^H C_r^{-1} U_r \widehat{B}_\ell V_\ell^H D_\ell^{-1} U_\ell & V_r^H C_r^{-1} U_r \end{bmatrix} \cdot \\ &\cdot \begin{bmatrix} R_\ell \\ R_r \end{bmatrix} \end{aligned}$$

where $G_\ell = V_\ell^H D_\ell^{-1} U_\ell$ and $G_r = V_r^H C_r^{-1} U_r$ have been introduced. Hence

$$G = \left[\begin{array}{cc} W_\ell^H & W_r^H \end{array}\right] \left[\begin{array}{cc} G_\ell + G_\ell \widehat{B}_u G_r \widehat{B}_\ell G_\ell & -G_\ell \widehat{B}_u G_r \\ -G_r \widehat{B}_\ell G_\ell & G_r \end{array}\right] \left[\begin{array}{c} R_\ell \\ R_r \end{array}\right]. \tag{3.19}$$

Downdating F. The downdate situation can be subsumed as follows. We assume that we have arrived at a stage where the LU factorization has progressed just beyond the (hierarchical) diagonal block D_ℓ in the original matrix, the last block for which the Schur complement data G_ℓ has been updated. The hierarchical diagonal block preceding D_ℓ is subsumed as D_A, covering all the indices preceding those of D_ℓ. For this block, the corresponding G_A is also assumed to be known – these are the recursive assumptions. Let us assume moreover that the next (hierarchical) block to be processed in the post-order is D_r. The relations in the off diagonal entries, using higher level indices as needed are given in the matrix
Let us denote:

$$\left[\begin{array}{cc|cc} D_A & U_A B_u W_\ell^H V_\ell^H & U_A B_u W_r^H V_r^H & \cdots \\ U_\ell R_\ell B_\ell V_A^H & D_\ell & U_\ell B_u' V_r^H & \cdots \\ \hline U_r R_r B_\ell V_A^H & U_r B_\ell' V_\ell^H & D_r & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}\right] = \left[\begin{array}{c|cc} A_{11} & A_{12} & \cdots \\ \hline A_{21} & A_{22} & \cdots \\ \vdots & \vdots & \ddots \end{array}\right].$$

The recursive assumptions, expressed in the data of this matrix are the knowledge of $G_A = V_A^H D_A^{-1} V_A^H$ and $G_\ell = V_\ell^H C_\ell^{-1} U_\ell$ in which C_ℓ is the Schur complement of D_A for the diagonal block D_ℓ. then

$$\begin{aligned} A_{21}A_{11}^{-1}A_{12} = & \left[\begin{array}{cc} U_r R_r B_\ell V_A^H & U_r B_\ell V_\ell^H \end{array}\right] . \\ & \cdot \left[\begin{array}{cc} D_A & U_A B_u W_\ell^H V_\ell^H \\ U_\ell R_\ell B_\ell V_A^H & D_\ell \end{array}\right]^{-1} \left[\begin{array}{c} U_A B_u W_r^H V_r^H \\ U_\ell B_u V_r^H \end{array}\right]. \end{aligned} \tag{3.20}$$

With the definition of F and the Schur inverse algorithm, we can rewrite the above formula as:

$$\begin{aligned} & A_{21}A_{11}^{-1}A_{12} = U_r F_r V_r^H \\ & = \left[\begin{array}{cc} U_r R_r B_\ell V_A^H & U_r B_\ell V_\ell^H \end{array}\right] . \\ & \quad \cdot \left[\begin{array}{cc} D_A^{-1} + D_A^{-1} U_A B_u W_\ell^H V_\ell^H C_\ell^{-1} U_\ell R_\ell B_\ell V_A^H D_A^{-1} & -D_A^{-1} U_A B_u W_\ell^H V_\ell^H C_\ell^{-1} \\ -C_\ell^{-1} U_\ell R_\ell B_\ell V_A^H D_A^{-1} & C_\ell^{-1} \end{array}\right] . \\ & \quad \cdot \left[\begin{array}{c} U_A B_u W_r^H V_r^H \\ U_\ell B_u V_r^H \end{array}\right] \\ & = U_r \left[\begin{array}{cc} R_r B_\ell V_A^H & B_\ell V_\ell^H \end{array}\right] . \\ & \quad \cdot \left[\begin{array}{cc} D_A^{-1} + D_A^{-1} U_A B_u W_\ell^H V_\ell^H C_\ell^{-1} U_\ell R_\ell B_\ell V_A^H D_A^{-1} & -D_A^{-1} U_A B_u W_\ell^H V_\ell^H C_\ell^{-1} \\ -C_\ell^{-1} U_\ell R_\ell B_\ell V_A^H D_A^{-1} & C_\ell^{-1} \end{array}\right] . \\ & \quad \cdot \left[\begin{array}{c} U_A B_u W_r^H \\ U_\ell B_u \end{array}\right] V_r^H \\ & = U_r \left[\begin{array}{cc} R_r & B_\ell \end{array}\right] . \\ & \quad \cdot \left[\begin{array}{cc} B_\ell V_A^H D_A^{-1} U_A B_u + Y & -B_\ell V_A^H D_A^{-1} U_A B_u W_\ell^H V_\ell^H C_\ell^{-1} U_\ell \\ -V_\ell^H C_\ell^{-1} U_\ell R_\ell B_\ell V_A^H D_A^{-1} U_A B_u & V_\ell^H C_\ell^{-1} U_\ell \end{array}\right] . \\ & \quad \cdot \left[\begin{array}{c} W_r^H \\ B_u \end{array}\right] V_r^H \end{aligned}$$

where

$$Y = B_\ell V_A^H D_A^{-1} U_A B_u W_\ell^H V_\ell^H C_\ell^{-1} U_\ell R_\ell B_\ell V_A^H D_A^{-1} U_A B_u.$$

As defined, F_r should represent the above term (excluding U_r and V_r). Assuming G_ℓ and $F = F_A$ given, we find

$$G_\ell = V_\ell^H C_\ell^{-1} U_\ell, F = B_\ell V_A^H D_A^{-1} U_A B_u.$$

Finally the update formula for F_r becomes:

$$F_r = \begin{bmatrix} R_r & B_\ell \end{bmatrix} \begin{bmatrix} F + F W_\ell^H G_\ell R_\ell F & -F W_\ell^H G_\ell \\ -G_\ell R_\ell F & G_\ell \end{bmatrix} \begin{bmatrix} W_r^H \\ B_u \end{bmatrix}. \tag{3.21}$$

And F_r again satisfies the definition.

The update formula for F_ℓ can be easily derived from the definition of F. To preserve the definition of F on the left branch, the F from the parent has to be pre-multiplied with R_ℓ and post-multiplied with W_ℓ^H. Thus the update formulas for G and F have been explained and proven.

Modifying B matrices and computing block pivots. To compute the Schur complement $D_{k;i} - U_{k;i} F_{k;i} V_{k;i}^H$ efficiently, we only need to update the B matrices and modify the block pivots. Here we assume that we are moving one level up in the recursion and that the levels below have already been computed. Let

$$\begin{aligned}
S_{k-1;i} &= D_{k-1;i} - U_{k-1;i} F_{k-1;i} V_{k-1;i}^H \\
S_{k-1;i} &= \begin{bmatrix} D_{k;2i-1} & U_{k;2i-1} B_{k;2i-1,2i} V_{k;2i}^H \\ U_{k;2i} B_{k;2i,2i-1} V_{k;2i-1}^H & D_{k;2i} \end{bmatrix} \\
&\quad - \begin{bmatrix} U_{k;2i-1} R_{k;2i-1} \\ U_{k;2i} R_{k;2i} \end{bmatrix} F_{k-1;i} \begin{bmatrix} W_{k;2i-1}^H V_{k;2i-1}^H & W_{k;2i}^H V_{k;2i}^H \end{bmatrix} \\
S_{k-1;i} &= \begin{bmatrix} D_{k;2i-1} & U_{k;2i-1} B_{k;2i-1,2i} V_{k;2i}^H \\ U_{k;2i} B_{k;2i,2i-1} V_{k;2i-1}^H & D_{k;2i} \end{bmatrix} \\
&\quad - \begin{bmatrix} U_{k;2i-1} R_{k;2i-1} F_{k-1;i} W_{k;2i-1}^H V_{k;2i-1}^H & U_{k;2i-1} R_{k;2i-1} F_{k-1;i} W_{k;2i}^H V_{k;2i}^H \\ U_{k;2i} R_{k;2i} F_{k-1;i} W_{k;2i-1}^H V_{k;2i-1}^H & U_{k;2i} R_{k;2i} F_{k-1;i} W_{k;2i}^H V_{k;2i}^H \end{bmatrix} \\
S_{k-1;i} &= \begin{bmatrix} \bar{D}_{k;2i-1} & Y_{k;2i-1,2i} \\ Y_{k;2i,2i-1} & \bar{D}_{k;2i} \end{bmatrix}
\end{aligned}$$

where

$$\begin{aligned}
Y_{k;2i-1,2i} &= U_{k;2i-1}(B_{k;2i-1,2i} - R_{k;2i-1} F_{k-1;i} W_{k;2i}^H) V_{k;2i}^H = U_{k;2i-1} \widehat{B}_{k;2i-1,2i} V_{k;2i}^H \\
Y_{k;2i,2i-1} &= U_{k;2i}(B_{k;2i,2i-1} - R_{k;2i} F_{k-1;i} W_{k;2i-1}^H) V_{k;2i-1}^H = U_{k;2i} \widehat{B}_{k;2i,2i-1} V_{k;2i-1}^H.
\end{aligned}$$

Hence

$$\bar{D}_{k;i} = D_{k;i} - U_{k;i} F_{k;i} V_{k;i}^H \tag{3.22}$$

$$\widehat{B}_{k;i,j} = B_{k;i,j} - R_{k;i} F_{k-1;\lceil \frac{i}{2} \rceil} W_{k;j}^H \tag{3.23}$$

and the F for the left branches:

$$F_{k;2i-1} = R_{k;2i-1} F_{k-1;i} W_{k;2i-1}^H. \tag{3.24}$$

Construction formulas for the L and the U matrices. We are now ready to formulate the LU-factorization relations and procedure.

Theorem 6. *Let a level-n HSS matrix T be given by the sequences R, W, B, U, V and D and assume that the pivot condition for existence of the LU-factorization is satisfied. Then the following relations hold:*
for $i \in 1, 2, \ldots, 2^n$:

$$G_{n;i} = V_{n;i}^H (D_{n;i} - U_{n;i} F_{n;i} V_{n;i}^H)^{-1} U_{n;i} \tag{3.25}$$

for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^k$ and $j = i + 1$ for odd i, $j = i - 1$ for even i, let

$$\widehat{B}_{k;i,j} = B_{k;i,j} - R_{k;i} F_{k-1;\lceil \frac{i}{2} \rceil} W_{k;j}^H \tag{3.26}$$

for $k = 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^{k-1}$:

$$\begin{aligned} G_{k-1;i} = & \\ & \begin{bmatrix} W_{k;2i-1}^H & W_{k;2i}^H \end{bmatrix} \cdot \\ & \cdot \begin{bmatrix} G_{k;2i-1} + G_{k;2i-1} \widehat{B}_{k;2i-1,2i} G_{k;2i} \widehat{B}_{k;2i,2i-1} G_{k;2i-1} & -G_{k;2i-1} \widehat{B}_{k;2i-1,2i} G_{k;2i} \\ -G_{k;2i} \widehat{B}_{k;2i,2i-1} G_{k;2i-1} & G_{k;2i} \end{bmatrix} \cdot \\ & \cdot \begin{bmatrix} R_{k;2i-1} \\ R_{2i} \end{bmatrix}. \end{aligned} \tag{3.27}$$

Initial value for F is:

$$F_{0;1} = \phi \tag{3.28}$$

left branches F_ℓ are given as:
for $k = 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^{k-1}$:

$$F_{k;2i-1} = R_{k;2i-1} F_{k-1;i} W_{k;2i-1}^H \tag{3.29}$$

right branches F_r are given as:
for $k = 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^{k-1}$:

$$\begin{aligned} F_{k;2i} \quad = \quad & \begin{bmatrix} R_{k;2i} & B_{k;2i,2i-1} \end{bmatrix} \cdot \\ & \cdot \begin{bmatrix} F_{k-1;i} + F_{k-1;i} W_{k;2i-1}^H G_{k;2i-1} R_{k;2i-1} F_{k-1;i} & -F_{k-1;i} W_{k;2i-1}^H G_{k;2i-1} \\ -G_{k;2i-1} R_{k;2i-1} F_{k;i} & G_{k;2i-1} \end{bmatrix} \cdot \\ & \cdot \begin{bmatrix} W_{k;2i}^H \\ B_{k;2i-1,2i} \end{bmatrix}. \end{aligned} \tag{3.30}$$

The (block) pivots are given by

$$\bar{D}_{n;i} = D_{n;i} - U_{n;i} F_{n;i} V_{n;i}^H. \tag{3.31}$$

Let now the pivots be LU-factored (these are elementary blocks that are not further decomposed): for $i \in 1, 2, \ldots, 2^n$:

$$\bar{L}_{n;i} \bar{U}_{n;i} = \bar{D}_{n;i} = D_{n;i} - U_{n;i} F_{n;i} V_{n;i}^H \tag{3.32}$$

be a *LU* decomposition at each leaf. Then based on the information generated, the L and U factors are defined as follows:

Theorem 7. *The level-n HSS representation of the L factor will be given as:*
at a non-leaf node:

for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^{k-1}$ *and* $j = 1, 2, \ldots, 2^k$*:*

$$\begin{cases} \widehat{R}_{k;j} = R_{k;j} & \widehat{W}_{k;2i-1} = W_{k;2i-1} \\ \widehat{W}^H_{k;2i} = W^H_{k;2i} - W^H_{k;2i-1} G_{k;2i-1} B_{k;2i-1,2i} & \\ \widehat{B}_{k;2i,2i-1} = \widehat{B}_{k;2i,2i-1} & \widehat{B}_{k;2i-1,2i} = 0 \end{cases} \tag{3.33}$$

at a leaf:

for $i \in 1, 2, \ldots, 2^n$*:*

$$\widehat{U}_{n;i} = U_{n;i} \quad \widehat{V}_{n;i} = \bar{U}^{-H}_{n;i} V_{n;i} \quad \widehat{D} = \bar{L}_{n;i}. \tag{3.34}$$

Theorem 8. *The level-n HSS representation of the U factor will be given as:*
at a non-leaf node:

for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^{k-1}$ *and* $j = 1, 2, \ldots, 2^k$*:*

$$\begin{cases} \widehat{R}_{k;2i-1} = R_{k;2i-1} & \widehat{R}_{k;2i} = R_{k;2i} - \widehat{B}_{k;2i,2i-1} G_{k;2i-1} R_{k;2i-1} & \widehat{W}_{k;j} = W_{k;j} \\ \widehat{B}_{k;2i,2i-1} = 0 & \widehat{B}_{k;2i-1,2i} = \widehat{B}_{k;2i-1,2i} & \end{cases} \tag{3.35}$$

at a leaf:

for $i \in 1, 2, \ldots, 2^n$*:*

$$\widehat{U}_{n;i} = \bar{L}^{-1}_{n;i} U_{n;i} \quad \widehat{V}_{n;i} = V_{n;i} \quad \widehat{D}_{n;i} = \bar{U}_{n;i}. \tag{3.36}$$

Proof for the traverse. We start with the proof of theorem 6. Given the updating operations on G and downdating operations on F accounted for in the introductory part of this section, it remains to verify that there exists a recursive order to compute all the quantities indicated. Initialization results in the $F_{k,1} = \phi$ for all $k = 1 \cdots n$. In particular, $F_{n,1}$ is now known, and $G_{n,1}$ can be computed. This in turn allows for the computation of $F_{n,2}$ thanks to the F_r downdate formula at level $(k-1, 1)$. Now $G_{n,2}$ can be computed, and next $G_{n-1,1}$ – the first left bottom node is dealt with. We now dispose of enough information to compute $F_{n-1,2}$, since $G_{n-1,1}$ and $F_{n-2,1} = \phi$ are known (this being the beginning of the next step).

The general dependencies in the formulas are as follows. At a leaf: $G_{n;i}$ depends on $F_{n;i}$; at a non-leaf node: $G_{k-1,i}$ is dependent on $G_{k,2i-1}$ and $G_{k,2i}$; $F_{k;2i-1}$ is dependent on $F_{k-1,i}$ and $F_{k,2i}$ is dependent on both $F_{k-1,i}$ and $G_{k,2i-1}$. Considering the closure of data dependencies, the full dependencies at a node are given in Figure 3. With the F matrices on the root initialized, the order in which all the F and G quantities can be computed on a node is $F_{k-1;i} \to F_{k;2i-1} \to G_{k;2i-1} \to F_{k;2i} \to G_{k;2i} \to G_{k-1;i}$, or equivalently *parent→left children→right children→parent*. That is: with a post-order traverse on the binary tree (note that: the F on the root is initialized), all unknown Fs and Gs can be filled in.

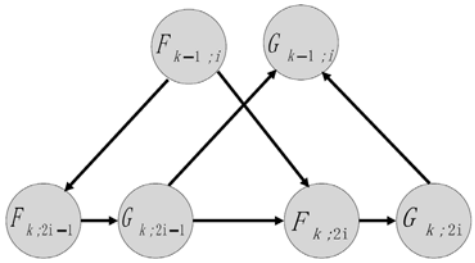

FIGURE 3. The dependencies of the intermediate variables on one no-leaf node.

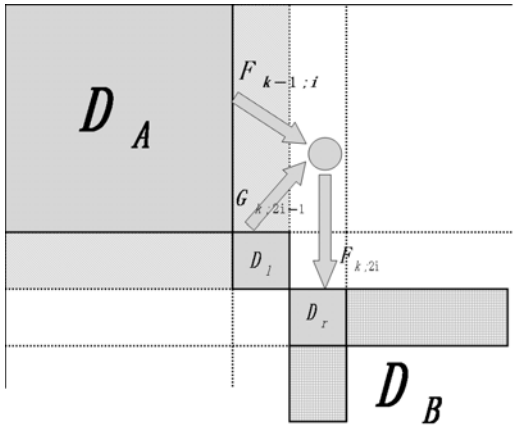

FIGURE 4. The computation of $F_{k;2i}$ with the help of $F_{k-1;i}$ and $G_{k;2i-1}$.

Proof of the formulas for L and U factors. Let now the pivots be LU-factored (these are elementary blocks that are not further decomposed). We may assume that at each step the Schur complements have been computed and updated. To get the **L** and **U** factors recursively as in formula (3.16), it is obvious that for each leaf of the L factor, $\bar{D} = \widehat{\mathbf{L}}$, $\bar{U} = U$, $\bar{V}^H = V_l^H \widehat{\mathbf{U}}^{-1}$; for each leaf of the U factor, $\bar{D} = \widehat{\mathbf{U}}$, $\bar{U} = \widehat{\mathbf{L}}^{-1} U$, $\bar{V} = V$.

For all left branches, the blocks are updated by modifying B matrices with formula (3.26) to compute the Schur complement $\bar{D}_{k;i} = D_{k;i} - U_{n;i} F_{n;i} V_{n;i}^H$. But for the right branches, updating B matrices with formula (3.26) is not enough because $F_{k-1;i}$ only subsumes the information from its parent. Its left sibling has to be taken into consideration for the update of the Schur complement.

Assuming the correct update has been done for the $\mathbf{D}_A$ block and D_ℓ block (see Figure 4), we may also assume that the Schur complement of D_ℓ has been computed. Hence, we only need to update D_r and the blocks indicated by grids in

Figure 4. That is for the block

$$\begin{bmatrix} \widehat{D}_\ell & U_\ell \widehat{B}_u V_r^H & U_\ell R_\ell B_u \mathbf{V}_B^H & \cdots \\ U_r \widehat{B}_\ell V_\ell^H & D_r & U_r R_r B_u \mathbf{V}_B^H & \cdots \\ \mathbf{U}_B B_\ell W_\ell^H V_\ell^H & \mathbf{U}_B B_\ell W_r^H V_r^H & \mathbf{D}_B & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Hence, only the blocks $\begin{bmatrix} U_r R_r B_u \mathbf{V}_B^H & \cdots \end{bmatrix}$ and $\begin{bmatrix} \mathbf{U}_B B_\ell W_r^H V_r^H \\ \vdots \end{bmatrix}$ have to be updated, other parts of the computation are taken care of by the recursive algorithm. Now, the Schur complement of $\widehat{D}_\ell$ has to be determined. That is:

$$\begin{aligned} S = & \begin{bmatrix} D_r & U_r R_r B_u \mathbf{V}_B^H & \cdots \\ \mathbf{U}_B B_\ell W_r^H V_r^H & \mathbf{D}_B & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \\ & - \begin{bmatrix} U_r \widehat{B}_\ell V_\ell^H \\ \mathbf{U}_B B_\ell W_\ell^H V_\ell^H \\ \vdots \end{bmatrix} \widehat{D}_\ell^{-1} \begin{bmatrix} U_\ell \widehat{B}_u V_r^H & U_\ell R_\ell B_u \mathbf{V}_B^H & \cdots \end{bmatrix} \\ S = & \begin{bmatrix} D_r - U_r \widehat{B}_\ell V_\ell^H \widehat{D}_\ell^{-1} U_\ell \widehat{B}_u V_r^H & U_r (R_r - \widehat{B}_\ell V_\ell^H D_\ell^{-1} U_\ell R_\ell) B_u \mathbf{V}_B^H & \cdots \\ \mathbf{U}_B B_\ell (W_r^H - W_\ell^H V_\ell^H D_\ell^{-1} U_\ell \widehat{B}_u) V_r^H & \mathbf{D}_B - \mathbf{U}_B B_\ell W_\ell^H V_\ell^H D_\ell^{-1} U_\ell R_\ell B_u \mathbf{V}_B^H & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \end{aligned}$$

Since $G_\ell = V_\ell^H D_\ell^{-1} U_\ell$,

$$S = \begin{bmatrix} D_r - U_r \widehat{B}_l V_\ell^H \widehat{D}_\ell^{-1} U_\ell \widehat{B}_u V_r^H & U_r (R_r - \widehat{B}_\ell G_\ell R_\ell) B_u \mathbf{V}_B^H & \cdots \\ \mathbf{U}_B B_\ell (W_r^H - W_\ell^H G_\ell \widehat{B}_u) V_r^H & \mathbf{D}_B - \mathbf{U}_B B_\ell W_\ell^H V_\ell^H D_\ell^{-1} U_\ell R_\ell B_u \mathbf{V}_B^H & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}.$$

Hence the update of the blocks $\begin{bmatrix} U_r R_r B_u \mathbf{V}_B^H \\ \vdots \end{bmatrix}$ and $\begin{bmatrix} \mathbf{U}_B^H B_\ell W_r^H V_r^H & \cdots \end{bmatrix}$ is given by $\widehat{R}_r = R_r - \widehat{B}_\ell G_\ell R_\ell$ and $\widehat{W}_r^H = W_r^H - W_\ell^H G_\ell \widehat{B}_u$. These prove the update formulas for $\widehat{R}_r$ and $\widehat{W}_r$.

Finally, all the update formulas have been explained, and the whole algorithm consists in recursively applying these formulas which actually compute and update the Schur complement recursively. This will be possible iff the pivot condition is satisfied.

4. Explicit ULV factorization

The LU factorization, however important, has only limited applicability. A backward stable algorithm that can always be applied is 'ULV-factorization'. It factors an arbitrary matrix in three factors, a unitary matrix U, a (generalized) lower triangular L (a non-singular triangular matrix embedded in a possibly larger zero

matrix) and another unitary matrix V. In the present section we show that the ULV-factorization for an HSS matrix of order n can be obtained in a special form. Both U and V are again HSS, and the lower triangular factor L has a special HSS form that is extremely sparse (many transfer matrices are zero). The ULV-factorization of A leads directly to the Moore-Penrose inverse for A. One trims the L factor to its purely triangular part, and the U and V factors to the corresponding relevant columns and rows to obtain the so called 'economic ULV factorization' $A = U_e L_e V_e$, the Moore-Penrose inverse then being given as $A^\dagger = V_e^H L_e^{-1} U_e^H$. The determination of the inverse of a lower triangular HSS factor is treated in the following section and gives rise to an HSS matrix of the same order and complexity. In this paper we follow the implicit ULV factorization method presented in [17], and show that the implicit method can be made explicit with some non-trivial modifications. The Moore-Penrose system can be then be solved with the explicit L factor. Alternatively one could follow the method presented in [18] which has similar flavors, but uses a slightly different approach.

For the sake of definiteness and without impairing generality, we assume here that the HSS matrix A has full row rank, and its n-level HSS representation is defined by the sequences U, V, D, B, R, W. Similar to the implicit ULV factorization method, the explicit method involves an upsweep recursion (or equivalently a post-order traverse). We start with the left-most leaf. First, we treat the case in which the HSS representation, which will be recursively reduced, has reached the situation given in equation (4.1). The second block row in that equation has a central purely triangular block $A_{k;i}$ of dimension $\delta_{k;i}$, the goal will be to reduce the matrix further by treating the next block row. Through the steps described in the following treatment this situation will be reached recursively by converting subtrees to leaves, so that the central compression step always happens at the level of a leaf.

4.1. Treatment of a leaf

The situation to be treated in this part of the recursion has the form

$$A = \begin{bmatrix} \ddots & [\vdots] V_{k;i}^{(1)H} & [\vdots] V_{k;i}^{(2)H} & \iddots \\ 0 & A_{k;i} & 0 & 0 \\ U_{k;i}[\cdots] & D_{k;i}^{(1)} & D_{k;i}^{(2)} & U_{k;i}[\cdots] \\ \iddots & [\vdots] V_{k;i}^{(1)H} & [\vdots] V_{k;i}^{(2)H} & \ddots \end{bmatrix}. \tag{4.1}$$

It is assumed at this point that $A_{k;i}$ is already lower triangular and invertible with dimension $\delta_{k;i}$. The next block row stands in line for treatment. The compression step attacks $U_{k;i}$. If $U_{k;i}$ has more rows than columns, it can be compressed by applying QL factorization on it:

$$U_{k;i} = Q_{k;i} \begin{bmatrix} 0 \\ \widehat{U}_{k;i} \end{bmatrix} \tag{4.2}$$

where $\widehat{U}_{k;i}$ is square and has l rows. To keep consistence in the rows, we must apply $Q_{k;i}^H$ to $D_{k;i}$:

$$\widehat{D}_{k;i} = Q_{k;i}^H D_{k;i}. \tag{4.3}$$

Assume that $D_{k;i}$ has m columns. We can partition $D_{k;i}$ as:

$$\begin{matrix} \delta_{k;i} & m-\delta_{k;i} \\ \left[\begin{array}{cc} \widehat{D}_{k;i}^{(1)} & \widehat{D}_{k;i}^{(2)} \end{array}\right] \end{matrix} = \widehat{D}_{k;i} \tag{4.4}$$

Since $A_{k;i}$ is already lower-triangular matrix, to proceed we only have to process the block $\widehat{D}_{k;i}^{(2)}$ so as to obtain a larger upper-triangular reduced block. Hence we LQ factorize $\widehat{D}_{k;i}^{(2)}$ as:

$$\widehat{D}_{k;i}^{(2)} = \left[\begin{array}{cc} \widehat{D}_{k;i;0,0}^{(2)} & 0 \\ \widehat{D}_{k;i;1,0}^{(2)} & \widehat{D}_{k;i;1,1}^{(2)} \end{array}\right] w_{k;i}, \tag{4.5}$$

where $\widehat{D}_{k;i;0,0}^{(2)}$ is lower triangular and has n columns;$\widehat{D}_{k;i;1,0}^{(2)}$ and $\widehat{D}_{k;i;1,1}^{(2)}$ have l rows. Now to adjust the columns, we must apply $w_{k;i}$ on $V_{k;i}$. Let

$$\begin{matrix} \delta_{k;i} \\ m-\delta_{k;i} \end{matrix} \left[\begin{array}{c} V_{k;i}^{(1)} \\ V_{k;i}^{(2)} \end{array}\right] = V_{k;i}. \tag{4.6}$$

Apply $w_{k;i}$ on $V_{k;i}^{(2)}$ as

$$\widehat{V}_{k;i}^{(2)} = w_{k;i} V_{k;i}^{(2)} \tag{4.7}$$

let:

$$\left[\begin{array}{c} \widehat{D}_{k;i}^{(1,1)} \\ \widehat{D}_{k;i}^{(1,2)} \end{array}\right] = \widehat{D}_{k;i}^{(1)}, \tag{4.8}$$

where $D_{k;i}^{(1,2)}$ has l rows. After these operations, the HSS representation has become

$$\bar{A} = \left[\begin{array}{ccccc} \ddots & [\vdots]V_{k;i}^{(1)H} & [\vdots]\widehat{V}_{k;i}^{(21)H} & [\vdots]\widehat{V}_{k;i}^{(22)H} & \cdot^{\cdot} \\ 0 & A_{k;i} & 0 & 0 & 0 \\ 0 & \widehat{D}_{k;i}^{(11)} & \widehat{D}_{k;i;0,0}^{(2)} & 0 & 0 \\ \widehat{U}_{k;i}[\cdots] & \widehat{D}_{k;i}^{(1,2)} & \widehat{D}_{k;i;1,0}^{(2)} & \widehat{D}_{k;i;1,1}^{(2)} & \widehat{U}_{k;i}[\cdots] \\ \cdot^{\cdot} & [\vdots]V_{k;i}^{(1)H} & [\vdots]\widehat{V}_{k;i}^{(21)H} & [\vdots]\widehat{V}_{k;i}^{(22)H} & \ddots \end{array}\right]. \tag{4.9}$$

The compressed leaf will be returned as:

$$\bar{D}_{k;i} = \left[\begin{array}{ccc} \widehat{D}_{k;i}^{(1,2)} & \widehat{D}_{k;i;1,0}^{(2)} & \widehat{D}_{k;i;1,1}^{(2)} \end{array}\right] \tag{4.10}$$

$$\bar{U}_{k;i} = \widehat{U}_{k;i} \tag{4.11}$$

$$\bar{V}_{k;i} = \left[\begin{array}{c} V_{k;i}^{(1)} \\ \widehat{V}_{k;i}^{(2)} \end{array}\right]. \tag{4.12}$$

With

$$\widehat{A}_{k;i} = \begin{bmatrix} A_{k;i} & 0 \\ \widehat{D}^{(1,1)}_{k;i} & \widehat{D}^{(2)}_{k;i;0,0} \end{bmatrix} \tag{4.13}$$

representing the reduced row slices, and

$$\widehat{\delta}_{k;i} = \delta_{k;i} + n. \tag{4.14}$$

Now, the commented HSS representation is exactly the same as the original, except the leaf has become smaller. When $U_{k;i}$ has more columns than rows, nothing can be done to compress in this way. Then a new arrangement has to be created by merging two leaves into a new, integrated leave. This process is treated in the next paragraph.

4.2. Merge

The behavior of this part of the algorithm on a leaf has been specified. If no leaf is available for processing, one can be created by merging. Assume that we are at the node $k;i$, the algorithm works in a post-order traverse way, it proceeds by first calling itself on the left children and then on the right children. When the algorithm comes to the present stage, both the left and the right child are already compressed leaves. They can then be merged by the following explicit procedure.

Before the merge, the HSS representation is, in an obvious notation: Let

$$\begin{aligned} Y^{(1)}_{k+1;2i-1;2i} &= U_{k+1;2i-1} B_{k+1;2i-1;2i} V^{(1)H}_{k+1;2i} \\ Y^{(2)}_{k+1;2i-1;2i} &= U_{k+1;2i-1} B_{k+1;2i-1;2i} V^{(2)H}_{k+1;2i} \\ Y^{(1)}_{k+1;2i;2i-1} &= U_{k+1;2i} B_{k+1;2i;2i-1} V^{(1)H}_{k+1;2i-1} \\ Y^{(2)}_{k+1;2i;2i-1} &= U_{k+1;2i} B_{k+1;2i;2i-1} V^{(2)H}_{k+1;2i-1} \end{aligned}$$

thus $D_{k;i}$ can be represented as:

$$D_{k;i} = \begin{bmatrix} A_{k+1;2i-1} & 0 & 0 & 0 \\ D^{(1)}_{k+1;2i-1} & D^{(2)}_{k+1;2i-1} & Y^{(1)}_{k+1;2i-1;2i} & Y^{(2)}_{k+1;2i-1;2i} \\ 0 & 0 & A_{k+1;2i} & 0 \\ Y^{(1)}_{k+1;2i;2i-1} & Y^{(2)}_{k+1;2i;2i-1} & D^{(1)}_{k+1;2i} & D^{(2)}_{k+1;2i} \end{bmatrix}. \tag{4.15}$$

Next, the rows and columns are moved to put all reduced rows on the top-left. After the reordering, the HSS representation becomes:

$$\widehat{D}_{k;i} = \begin{bmatrix} A_{k+1;2i-1} & 0 & 0 & 0 \\ 0 & A_{k+1;2i} & 0 & 0 \\ D^{(1)}_{k+1;2i-1} & Y^{(1)}_{k+1;2i-1;2i} & D^{(2)}_{k+1;2i-1} & Y^{(2)}_{k+1;2i-1;2i} \\ Y^{(1)}_{k+1;2i;2i-1} & D^{(1)}_{k+1;2i} & Y^{(2)}_{k+1;2i;2i-1} & D^{(2)}_{k+1;2i} \end{bmatrix} \tag{4.16}$$

and the merged leaf now has:

$$\bar{D}_{k;i} = \begin{bmatrix} D^{(1)}_{k+1;2i-1} & Y^{(1)}_{k+1;2i-1;2i} & D^{(2)}_{k+1;2i-1} & Y^{(2)}_{k+1;2i-1;2i} \\ Y^{(1)}_{k+1;2i;2i-1} & D^{(1)}_{k+1;2i} & Y^{(2)}_{k+1;2i;2i-1} & D^{(2)}_{k+1;2i} \end{bmatrix} \tag{4.17}$$

$$\bar{U}_{k;i} = \begin{bmatrix} U_{k+1;2i-1}R_{k+1;2i-1} \\ U_{k+1;2i}R_{k+1;2i} \end{bmatrix}, \bar{V}_{k;i} = \begin{bmatrix} V^{(1)}_{k+1;2i-1}W_{k+1;2i-1} \\ V^{(1)}_{k+1;2i}W_{k+1;2i} \\ V^{(2)}_{k+1;2i-1}W_{k+1;2i-1} \\ V^{(2)}_{k+1;2i}W_{k+1;2i-1} \end{bmatrix}. \tag{4.18}$$

With the intermediate block

$$A_{k;i} = \begin{bmatrix} A_{k+1;2i-1} & 0 \\ 0 & A_{k+1;2i} \end{bmatrix} \tag{4.19}$$

and

$$\delta_{k;i} = \delta_{k+1;2i-1} + \delta_{k+1;2i}. \tag{4.20}$$

Note that now the node has been reduced to a leaf, and the actual HSS system has two fewer leaves. The compression algorithm can then be called on this leaf with $A_{k;i}$ and $\delta_{k;i}$.

4.3. Formal algorithm

Having the above three procedures, we now describe the algorithm formally. Similar to the implicit ULV factorization method, this algorithm is a tree-based recursive algorithm. It involves a post-order traverse of the binary tree of the HSS representation. Let T be the root of the HSS representation.

```
Function: post-order-traverse
Input: an actual HSS node or leaf T;
Output: a compressed HSS leaf)
 1. (node, left-children, right-children) = T;
 2. left-leaf = post-order-traverse left-child;
 3. right-leaf = post-order-traverse right-child;
 4. if left-child is compressible then
    left-leaf = compress left-leaf;
    else
    do nothing;
 5. if right-child is compressible then
    right-leaf = compress right-leaf;
    else
    do nothing;
 6. return compress (Merge(node,left-leaf,right-leaf));
```

```
Function : Explicit-ULV-Factorization
Input: a HSS representation T; Output: the factor L in
sparse matrix format
 1. actual-T = T;
 2. Leaf = post-order-traverse actual-T;
 3. return Leaf.A_{0;1}
```

Once the whole HSS tree is compressed as a leaf and the leaf is further compressed, the L factor has been computed as $L = A_{0;1}$.

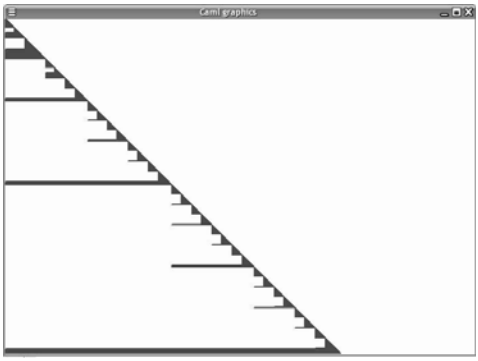

FIGURE 5. The sparsity pattern of L factor of the explicit ULV factorization.

4.4. Results

We show the result of the procedure applied to an HSS matrix A of dimensions 500×700 with full row rank. Its HSS representation is 5 levels deep and balanced. We apply the explicit ULV factorization algorithm on it. Then the sparsity pattern of the L factor will be as in Figure 5. L has 500 rows and 700 columns. Its sparsity is 3.08% (The sparsity depends on the HSS complexity, the lower the complexity is, the sparser the L factor is.). With the assumption that A has full row rank, The non-zero block of L is square and invertible.

4.5. Remarks

- Assume A has full column rank, the algorithm above can be modified to produce the URV factorization (by compressing $V_{k;i}$ instead of $U_{k;i}$).
- The explicit factor shall be kept in sparse matrix form.
- the U and V factors are kept in an implicit form. This is convenient because they can be easily applied to b and x when solve the system $Ax = b$.
- The complexity is higher than the implicit ULV factorization method, but it shall still be linear. It can also be easily be seen that the HSS complexity of the result is the same as of the original (with many transfer matrices reduced to zero).

5. Inverse of triangular HSS matrix

In this section, we will show how a triangular HSS matrix can be inverted efficiently. We shall only present our fast inverse algorithm on upper triangular HSS matrices, since the algorithm for lower triangular matrices is dual and similar. With the combination of the LU factorization algorithm, the inverse algorithm for triangular systems and the matrix-matrix multiplication algorithm, the HSS inverse of a square invertible HSS matrix, of which all block pivots are invertible, can be computed.

Let the level-n HSS representation of the upper triangular matrix A be given by the sequence of R, W, B, U, V and D (where the D's are upper triangular). Assuming all D matrices invertible, the level-n HSS representation of the inverse of A is given by $\widehat{R}$, $\widehat{W}$, $\widehat{B}$, $\widehat{U}$, $\widehat{V}$ and $\widehat{D}$ (where $\widehat{D}$s are again upper triangular) with the formulas given below. We use the following (trivial) fact recursively.

Lemma 1. *The inverse of* $D_{k-1;\lceil \frac{i}{2} \rceil}$ (*i is an odd number*) *is given by*

$$\begin{aligned} D^{-1}_{k-1;\lceil \frac{i}{2} \rceil} &= \begin{bmatrix} D_{k;i} & U_{k;i}B_{k;i,i+1}V^H_{k;i+1} \\ 0 & D_{k;i+1} \end{bmatrix}^{-1} \\ &= \begin{bmatrix} D^{-1}_{k;i} & -D^{-1}_{k;i}U_{k;i}B_{k;i,i+1}V^H_{k;i+1}D^{-1}_{k;i+1} \\ 0 & D^{-1}_{k;i+1} \end{bmatrix}. \end{aligned} \tag{5.1}$$

We have

$$\widehat{U}_{k;i} = \begin{bmatrix} D_{k+1;2i-1} & U_{k+1;2i-1}B_{k+1;2i-1,2i}V^H_{k+1;2i} \\ 0 & D_{k+1;2i} \end{bmatrix}^{-1} U_{k;i}$$

$$\widehat{U}_{k;i} = \begin{bmatrix} D_{k+1;2i-1} & U_{k+1;2i-1}B_{k+1;2i-1,2i}V^H_{k+1;2i} \\ 0 & D_{k+1;2i} \end{bmatrix}^{-1} \cdot \begin{bmatrix} U_{k+1;2i-1}R_{k+1;2i-1} \\ U_{k+1;2i}R_{k+1;2i} \end{bmatrix}$$

$$\widehat{U}_{k;i} = \begin{bmatrix} D^{-1}_{k+1;2i-1} & -D^{-1}_{k+1;2i-1}U_{k+1;2i-1}B_{k+1;2i-1,2i}V^H_{k+1;2i}D^{-1}_{k+1;2i} \\ 0 & D^{-1}_{k+1;2i} \end{bmatrix} \cdot \begin{bmatrix} U_{k+1;2i-1}R_{k+1;2i-1} \\ U_{k+1;2i}R_{k+1;2i} \end{bmatrix}$$

$$\widehat{U}_{k;i} = \begin{bmatrix} D^{-1}_{k+1;2i-1}U_{k+1;2i-1}(R_{k+1;2i-1} - B_{k+1;2i-1,2i}V^H_{k+1;2i}D^{-1}_{k+1;2i}U_{k+1;2i}R_{k+1;2i}) \\ D^{-1}_{k+1;2i}U_{k+1;2i}R_{k+1;2i} \end{bmatrix}.$$

Assuming that $\widehat{U}_{k+1;2i-1}$ and $\widehat{U}_{k+1;2i}$ have been updated as $D^{-1}_{k+1;2i-1}\ U_{k+1;2i-1}$ and $D^{-1}_{k+1;2i}U_{k+1;2i}$ the update for $\widehat{U}_{k;i}$ follows from the update $R_{k+1;2i-1}$ as

$$\widehat{R}_{k+1;2i-1} = R_{k+1;2i-1} - B_{k+1;2i-1,2i}V^H_{k+1;2i}\bar{U}^{-1}_{k+1;2i}U_{k+1;2i}R_{k+1;2i}. \tag{5.2}$$

The formulas for $\widehat{V}_{k;i+1}$ become

$$\begin{aligned} \widehat{V}^H_{k;i+1} &= V^H_{k;i+1}\begin{bmatrix} D_{k+1;2i+1} & U_{k+1;2i+1}B_{k+1;2i+1,2i+2}V^H_{k+1;2i+2} \\ 0 & D_{k+1;2i+2} \end{bmatrix}^{-1} \\ \widehat{V}^H_{k;i+1} &= \begin{bmatrix} W^H_{k+1;2i+1}V^H_{k+1;2i+1} & W^H_{k+1;2i+2}V^H_{k+1;2i+2} \end{bmatrix} \cdot \\ &\quad \cdot \begin{bmatrix} D_{k+1;2i+1} & U_{k+1;2i+1}B_{k+1;2i+1,2i+2}V^H_{k+1;2i+2} \\ 0 & D_{k+1;2i+2} \end{bmatrix}^{-1} \\ \widehat{V}^H_{k;i+1} &= \begin{bmatrix} W^H_{k+1;2i+1}V^H_{k+1;2i+1} & W^H_{k+1;2i+2}V^H_{k+1;2i+2} \end{bmatrix} \cdot \\ &\quad \cdot \begin{bmatrix} D^{-1}_{k+1;2i+1} & -D^{-1}_{k+1;2i+1}U_{k+1;2i+1}B_{k+1;2i+1,2i+2}V^H_{k+1;2i+2}D^{-1}_{k+1;2i+2} \\ 0 & D^{-1}_{k+1;2i+2} \end{bmatrix}. \end{aligned}$$

Let

$$\widehat{W}^H_{k+1;2i+2} = W^H_{k+1;2i+2} - W^H_{k+1;2i+1} V^H_{k+1;2i+1} D^{-1}_{k+1;2i+1} U_{k+1;2i+1} B_{k+1;2i+1,2i+2}$$

then

$$\widehat{V}^H_{k;i+1} = \left[\begin{array}{cc} W^H_{k+1;2i+1} V^H_{k+1;2i+1} D^{-1}_{k+1;2i+1} & \widehat{W}^H_{k+1;2i+2} V^H_{k+1;2i+2} D^{-1}_{k+1;2i+2} \end{array} \right].$$

Assuming now that $\widehat{V}^H_{k+1;2i+1}$ and $\widehat{V}^H_{k+1;2i+2}$ have been updated as

$$V^H_{k+1;2i+1} D^{-1}_{k+1;2i+1} \qquad \text{and} \qquad V^H_{k+1;2i+2} D^{-1}_{k+1;2i+2},$$

the update for $\widehat{V}_{k;i+1}$ follows from

$$\widehat{W}^H_{k+1;2i+2} = W^H_{k+1;2i+2} - W^H_{k+1;2i+1} V^H_{k+1;2i+1} D^{-1}_{k+1;2i+1} U_{k+1;2i+1} B_{k+1;2i+1,2i+2} \tag{5.3}$$

next the update for $-\widehat{U}_{k;i} B_{k;i,j} \widehat{V}^H_{k;j}$ follows from

$$\widehat{B}_{k;i,j} = -B_{k;i,j}. \tag{5.4}$$

Let the intermediate G be defined as $G_{k;i} = V^H_{k;i} D^{-1}_{k;i} U_{k;i}$, then the above update formulas can be written as

$$\left\{ \begin{array}{l} \widehat{W}^H_{k;2i} = W^H_{k;2i} - W^H_{k;2i-1} G_{k;2i-1} B_{k;2i-1,2i} \\ \widehat{W}_{k;2i-1} = W_{k;2i-1} \\ \widehat{R}_{k;2i-1} = R_{k;2i-1} - B_{k;2i-1,2i} G_{k;2i} R_{k;2i} \\ \widehat{R}_{k;2i} = R_{k;2i} \\ \widehat{B}_{k;i,j} = -B_{k;i,j}. \end{array} \right. \tag{5.5}$$

The recursive formula for the intermediate variable G are as follows. According to the definition of $G_{k-1;i}$:

$$\begin{aligned} G_{k-1;i} &= \left[\begin{array}{cc} W^H_{k;2i-1} V^H_{k;2i-1} & W^H_{k;2i-1} V^H_{k;2i} \end{array} \right] \cdot \\ &\quad \cdot \left[\begin{array}{cc} D_{k;2i-1} & U_{k;2i-1} B_{k;2i-1,2i} V^H_{k;2i} \\ 0 & D_{k;2i} \end{array} \right]^{-1} \left[\begin{array}{c} U_{k;2i-1} R_{k;2i-1} \\ U_{k;2i} R_{k;2i} \end{array} \right] \\ G_{k-1;i} &= \left[\begin{array}{cc} W^H_{k;2i-1} V^H_{k;2i-1} & W^H_{k;2i-1} V^H_{k;2i} \end{array} \right] \cdot \\ &\quad \cdot \left[\begin{array}{cc} D^{-1}_{k;2i-1} & -D^{-1}_{k;2i-1} U_{k;2i-1} B_{k;2i-1,2i} V^H_{k;2i} D^{-1}_{k;2i} \\ 0 & D^{-1}_{k;2i} \end{array} \right] \left[\begin{array}{c} U_{k;2i-1} R_{k;2i-1} \\ U_{k;2i} R_{k;2i} \end{array} \right] \\ G_{k-1;i} &= \left[\begin{array}{cc} W^H_{k;2i-1} & W^H_{k;2i-1} \end{array} \right] \cdot \\ &\quad \cdot \left[\begin{array}{cc} V^H_{k;2i-1} D^{-1}_{k;2i-1} U_{k;2i-1} & -V^H_{k;2i-1} D^{-1}_{k;2i-1} U_{k;2i-1} B_{k;2i-1,2i} V^H_{k;2i} D^{-1}_{k;2i} U_{k;2i} \\ 0 & V^H_{k;2i} D^{-1}_{k;2i} U_{k;2i} \end{array} \right] \cdot \\ &\quad \cdot \left[\begin{array}{c} R_{k;2i-1} \\ R_{k;2i} \end{array} \right] \\ G_{k-1;i} &= \left[\begin{array}{cc} W^H_{k;2i-1} & W^H_{k;2i-1} \end{array} \right] \left[\begin{array}{cc} G_{k;2i-1} & -G_{k;2i-1} B_{k;2i-1,2i} G_{k;2i} \\ 0 & G_{k;2i} \end{array} \right] \left[\begin{array}{c} R_{k;2i-1} \\ R_{k;2i} \end{array} \right]. \end{aligned}$$

Summarizing

Definition 5. *Let the intermediate variable G be defined as*
for $k = 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^{k-1}$:

$$G_{k;i} = V_{k;i}^H D_{k;i}^{-1} U_{k;i}. \tag{5.6}$$

The upsweep recursion for G is:

$$G_{k-1;i} = \begin{bmatrix} W_{k;2i-1}^H & W_{k;2i}^H \end{bmatrix} \begin{bmatrix} G_{k;2i-1} & -G_{k;2i-1}B_{k;2i-1,2i}G_{k;2i} \\ 0 & G_{k;2i} \end{bmatrix} \begin{bmatrix} R_{k;2i-1} \\ R_{k;2i} \end{bmatrix} \tag{5.7}$$

and hence

Theorem 9. *The level-n HSS representation of the inverse of the upper triangular HSS matrix is given by the following sequence of operations*
for $k = 1, 2, \ldots, n$; $j \in 1, 2, \ldots, 2^k$ and $i \in 1, 2, \ldots, 2^{k-1}$:

$$\begin{cases} \widehat{W}_{k;2i}^H = W_{k;2i}^H - W_{k;2i-1}^H G_{k;2i-1}B_{k;2i-1,2i} & \widehat{W}_{k;2i-1} = W_{k;2i-1} \\ \widehat{R}_{k;2i-1} = R_{k;2i-1} - B_{k;2i-1,2i}G_{k;2i}R_{k;2i} & \widehat{R}_{k;2i} = R_{k;2i} \\ \widehat{B}_{k;2i-1,2i} = -B_{k;2i-1,2i} & \widehat{B}_{k;2i,2i-1} = 0 \\ \widehat{U}_{k;j} = D_{k;j}^{-1} U_{k;j} & \widehat{V}_{k;j}^H = V_{k;j}^H D_{k;j}^{-1} \\ \widehat{D}_{k;j}^H = D_{k;j}^{-1}. \end{cases} \tag{5.8}$$

6. Ancillary operations

In this section, we will discuss various ancillary operations that help to (re-) construct an HSS representation in various circumstances. These operations will help to reduce the HSS complexity or to keep the column base and row base dependencies of the HSS representation.

6.1. Column (row) base insertion

When the off-diagonal blocks have to be changed at the nodes at a higher level, column bases and row bases may have to be changed. To keep the column and row base dependencies, new column (row) bases may have to be added to the lower levels. We might be able to generate these bases from the column (row) bases of the lower level nodes, but this is not guaranteed. Taking a conservative approach we insert column (row) bases into the lower level and then do *compression* to reduce the complexity of HSS representation.

The algorithm combines two sub-algorithms (downsweep base insertion and then a compression). The *compression* procedure is used to eliminate redundant bases and reduce the HSS complexity. *Compression* does not have to be done after every downsweep column (row) base insertion. To save the computation cost, we may do one step of *compression* after a number of steps of bases insertion.

We will present row base insertion in details, while column base insertion is dual and hence similar.

6.1.1. Downsweep row base insertion. Suppose that we need to add a row base represented by a conformal matrix v to an HSS node A without changing the matrix it represents (the column dimension of v should of course be conformal to the row dimension of A.) Let the original HSS node be represented as

$$\begin{bmatrix} D_{1;1} & U_{1;1}B_{1;1,2}V_{1;2}^H \\ U_{1;2}B_{1;2,1}V_{1;1}^H & D_{1;2} \end{bmatrix}.$$

The algorithm works in a downsweep fashion modifying the nodes and leaves in the HSS tree.

- **Row base insertion at a non-leaf node**

  $v_{k;i}$ is split according to the column partition of A at this node:
  for $k = 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^k$:

$$v_{k;i} = \begin{bmatrix} v_{k+1;2i-1} \\ v_{k+1;2i} \end{bmatrix}$$

  $v_{k+1;2i-1}$ is inserted to the left child, $v_{k+1;2i}$ to the right child recursively. $v_{k+1;2i-1}$ can be generated from $D_{k+1;2i-1}$, and $v_{k+1;2i}$ from $D_{k+1;2i}$. The translation matrices of this node must be modified to make sure that the base insertion does not change the matrix it represents as follows

  for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^k$, and $j = i + 1$ for odd i, $j = i - 1$ for even i:

$$\begin{cases} \widehat{W}_{k;i} = \begin{bmatrix} W_{k;i} & 0 \\ 0 & I \end{bmatrix} \\ \widehat{B}_{k;i,j} = \begin{bmatrix} B_{k;i,j} & 0 \end{bmatrix} \end{cases}. \tag{6.1}$$

- **Row base insertion at a leaf**

  a leaf is reached by recursion, $v_{n;i}$ has to be inserted to the leaf, hence for $k = 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^k$:

$$\widehat{V}_{n;i} = \begin{bmatrix} V_{n;i} & v_{n;i} \end{bmatrix}. \tag{6.2}$$

6.1.2. Compression. After applying downsweep base insertion to A, the row bases v required by the upper level can be generated from A. But the HSS representation we get may have become redundant.

Since only the row base has been modified, we only have to factor $V_{n;i}$ matrices as

for $k = 1, 2, \ldots, n$ and $i \in 1, 2, \ldots, 2^k$:

$$V_{n;i} = \widehat{V}_{n;i} w_{n;i}. \tag{6.3}$$

This should be done by a rank revealing QR or QL factorization, then $\widehat{V}_{n;i}$ will be column orthonormal (and it will surely be column independent). The factor $w_{n;i}$ will then be propagated to the upper level, where the translation matrices $B_{n;i,j}$ and $W_{n;i}$ will be modified by the factor w as follows

for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^k$, and $j = i+1$ for odd i, $j = i-1$ for even i:

$$\begin{cases} \widehat{B}_{k;i,j} = B_{k;i,j} w_{k;j}^H \\ \bar{W}_{k;i} = w_{k;i} W_{k;i} \\ \widehat{R}_{k;i} = R_{k;i}. \end{cases} \tag{6.4}$$

Then we do compression on the higher level. Since only row bases have been modified, we only have to factor

$$\begin{bmatrix} \bar{W}_{k;2i-1} \\ \bar{W}_{k;2i} \end{bmatrix} = \begin{bmatrix} \widehat{W}_{k;2i-1} \\ \widehat{W}_{k;2i} \end{bmatrix} w_{k-1;i}. \tag{6.5}$$

Note that $\bar{W}_{k;2i-1}$ and $\bar{W}_{k;2i}$ have been modified by the $w_{k;2i-1}$ and $w_{k;2i}$ factors coming from its children with the formulas (6.4), the factorization should again be rank-revealing. The $w_{k-1;i}$ factor will then be propagated further to the upper level, and the algorithm proceeds recursively.

After applying the whole algorithm to the a HSS node, the new row base v^H will be inserted by appending it to the original row base. Suppose the row base of the original node is given by V^H, the modified node becomes $\begin{bmatrix} V & v \end{bmatrix}^H$. Note that base insertion does not change the HSS matrix, it only modifies its HSS representation.

6.1.3. Column base insertion. The algorithm for column base insertion is similar and dual to the one for row base insertion. Modifications will now be done on U, R instead of V, W. And the B matrices will be modified as $\widehat{B}_{k;i,j} = \begin{bmatrix} B_{k;i,j} \\ 0 \end{bmatrix}$ instead of $\begin{bmatrix} B_{k;i,j} & 0 \end{bmatrix}$. After applying the row bases insertion to a HSS node, the new column bases will be appended after its original column bases. The *compression* algorithm for column base insertion should also be modified accordingly.

6.2. Append a matrix to a HSS matrix

This algorithm appends a thin slice C to a HSS matrix A. This operation is central in the Moore-Penrose HSS inversion treated in [18]. We establish that the result of this operation will still be HSS matrix whose HSS representation can be computed easily. Obviously, we may append the matrix to the top of the HSS matrix, to the left of the HSS matrix, to the right of the HSS matrix or to the bottom of the HSS matrix. Here we just present the method to append matrix to the left of the HSS matrix. Others can be easily derived mutatis mutandis.

6.2.1. Append a rank-k matrix to a HSS matrix. Suppose

$$\widehat{A} = \begin{bmatrix} C & A \end{bmatrix}. \tag{6.6}$$

Matrix B should have the same number of rows as A does, A is HSS matrix whose HSS representation is defined by sequences U_A, V_A, D_A, R_A, W_A and B_A.

Instead of trying to absorb the C matrix into the HSS representation of A matrix, we rewrite the formula (6.6) as:

$$\widehat{A} = \begin{bmatrix} - & - \\ C & A \end{bmatrix} \tag{6.7}$$

where $-$ is a dummy matrix which has no rows. $\widehat{A}$ is an HSS matrix which has one more level than A does.

We then assume that $C = UBV^H$. That is: C is a rank-k matrix. The decomposition of C can be computed by a URV factorization or SVD factorization (in practice, we normally have its decomposition already available).

Then the column base U of C shall be inserted to the HSS representation of A so that U can be generated from the HSS representation of A. This can be done in many different ways. The most straightforward is to insert the column base using the algorithm described in Section 6.1 and followed by a step of *compression* depending on how many columns U has. Suppose that after column bases insertion, the HSS representation of A becomes $\bar{A}$. (Note that: column bases insertion does not change the HSS matrix, it only changes the HSS representation.)

Then $\widehat{A}$ will be represented as

$$\widehat{A} = \begin{bmatrix} - & - \\ UBV^H & \bar{A} \end{bmatrix}. \tag{6.8}$$

It is easy to check that the HSS representation of $\widehat{A}$ will be given as
at the top node

$$\begin{cases} B_{1;1,2} = \emptyset & B_{1;2,1} = B & W_{1;1} = | \\ W_{1;2} = \emptyset & R_{1;1} = \emptyset & R_{1;2} = | \end{cases} \tag{6.9}$$

at the left branch:

$$D_{1;1} = - \quad U_{1;1} = \emptyset \quad V_{1;1} = V \tag{6.10}$$

at the right branch:

$$D_{1;2} = \bar{A} \tag{6.11}$$

where $|$ and $-$ represent dummy matrices with no columns respect. no rows. $\emptyset$ represents the dummy matrix without column or or row. The other dimensions of all these should be correct such that the HSS representation is still valid.

6.2.2. Matrix-append when bases are semi-separable. In practice, we almost never compute U and V, since these computations are costly and break the idea of HSS representation. For instance, when a matrix UBV^H needs to be appended to a HSS matrix A, U and V are not explicit stored. They are defined by the formulas (2.6) and (2.7).

In this case, the formulas in the last subsection will have to be modified accordingly, The left branch of $\widehat{A}$ will not be of just one level. Instead, the left child will be a sub-HSS tree defined by the following sequences:
at the root:

$$\begin{cases} B_{1;1,2} = \emptyset & B_{1;2,1} = B & W_{1;1} = | \\ W_{1;2} = \emptyset & R_{1;1} = \emptyset & R_{1;2} = | \end{cases} \tag{6.12}$$

at non-leaf nodes: for $k = 2, 3, \ldots, n$ and $i \in 1, 2, \ldots, 2^{k-1}$:

$$\begin{cases} \widehat{R}_{k;2i-1} = | & \widehat{R}_{k;2i} = | & \widehat{B}_{k;2i-1,2i} = - \\ \widehat{B}_{k;2i,2i-1} = - & \widehat{W}_{k;2i-1} = W_{k;2i-1} & \widehat{W}_{k;2i} = W_{k;2i} \end{cases} \tag{6.13}$$

at the leaves:

$$\widehat{U}_{n;i} = \emptyset \quad \widehat{V}_{n;i} = V_{n;i} \quad \widehat{D}_{n;i} = - \tag{6.14}$$

note that since the column base U is also in a hierarchically semi-separable form, inserting it into A will be somewhat different than that in Section (6.1). The modified formulas for inserting a column base U to A are given by

for $k = 1, 2, \ldots, n$; $i \in 1, 2, \ldots, 2^{k-1}$; $j = i + 1$ for odd i and $j = i - 1$ for even i:

$$\begin{cases} \widehat{B}_{k;i,j} = \begin{bmatrix} B_{A;k;i,j} \\ 0 \end{bmatrix} \\ \widehat{R}_{k;i} = \begin{bmatrix} R_{A;k;i} & 0 \\ 0 & R_{k;i} \end{bmatrix} \\ \widehat{U}_{k;i} = \begin{bmatrix} U_{A;k;i} & U_{k;i} \end{bmatrix}. \end{cases} \tag{6.15}$$

7. Complexity analysis

From the algorithms given, the time complexity of the elementary operations can easily be evaluated together with their effect on the *representation complexity* of the resulting HSS structure. The same matrix can be represented by many different HSS representations, in which some are better than others in terms of computation complexity and space complexity. The *HSS representation complexity* should be defined in such a way that operations on the HSS representation with higher *HSS representation complexity* cost more time and memory than those on HSS representations with lower *HSS representation complexity*. Many indicators can be used. Here, we use a rough measure for the *HSS representation complexity* as follows

Definition 6. *HSS complexity: the total number of free entries in the HSS representation.*

Definition 7. *Free entries: free entries are the entries which can be changed without restriction* (*For instance, the number of free entries in $n \times n$ diagonal matrix will be n, that in $n \times n$ triangular matrix will be $n(n-1)/2\ldots$ etc.*).

The *HSS complexity* actually indicates the least possible memory needed to store the HSS representation. It also implies the computation complexity, assuming each free entry is accessed once or a small number of times during operations (we may have to account for intermediary representations as well).

Since most of the algorithms given are not so complicated and some have been studied in the literature, we shall limit ourselves to listing a summarizing table for the existing HSS algorithms (including some from the literature). We assume that n is the dimension of the HSS matrix and k is the maximum rank

TABLE 1. Computation complexity analysis table

Operation	**Numerical Complexity**	**Resulting representation complexity**
Matrix-Vector Multiplication [10]	$C_{\text{Matrix}\times\text{Vector}}(n) = O(nk^2)$	A vector of dim. n
Matrix-Matrix Multiplication [10]	$C_{\text{Matrix}\times\text{Matrix}}(n) = O(nk^3)$	Addition
Construct HSS for rank-k matrix	$C_{k\text{-construction}}(n) = O(nk)$	proportional to k
Bases insertion	$C_{\text{Bases-insert}}(n) = O(n)$	Increase by the size of V
Matrix-Append	$C_{\text{Matrix-append}}(n) = O(n)$	Increase by one level
Matrix addition [14]	$C_{\text{Addition}}(n) = O(nk^2)$	Increase additively
Compression	$C_{\text{Compression}}(n) = O(nk^3)$	Does not increase
Model reduction [15]	$C_{\text{Model-reduction}}(n) = O(nk^3)$	Decreases
LU Decomposition [1]	$C_{\text{LU}}(n) = O(nk^3)$	Does not change
Fast solve [10] [14]	$C_{\text{Solve}}(n) = O(nk^3)$	A vector of dim. n
Inverse	$C_{\text{Inverse}}(n) = Onk^3$	Does not change
Transpose	$C_{\text{Transpose}}(n) = O(nk)$	Does not change

of the translation matrices (more accurate formulas can be derived when more detailed information on local rank is available). Table 1 gives a measure of the numerical complexity in terms of n and k, as well as an indication of the HSS complexity of the resulting structure.

We see that in all cases the complexity is linear in the original size of the matrix, and a to be expected power of the size of the translation matrices. Of course, a much more detailed analysis is possible but falls beyond the scope of this paper.

8. Connection between SSS, HSS and the time varying notation

In the earlier papers on SSS [19, 16], efficient algorithms have been developed. Although different algorithms have to be used corresponding to these two seemingly different representations, we would like to show that they are not so different, and we will show how they can be converted to each other. By converting between

these two representations, we can take advantages of the fast algorithms for these two different representations.

8.1. From SSS to HSS

In [16], the SSS representation for A is defined as follows: let A be an $N \times N$ matrix satisfying the SSS matrix structure. Then there exist n positive integers $m_1, \ldots, m_n$ with $N = m_1 + \cdots + m_n$ to block-partition A as $A = A_{i,j}$, where $A_{ij} \in C^{m_i \times m_j}$ satisfies

$$A_{ij} = \begin{cases} D_i & \text{if } i = j \\ U_i W_{i+1} \ldots W_{j-1} V_j^H & \text{if } j > i \\ P_i R_{i-1} \ldots R_{j+1} Q_j^H & \text{if } j < i. \end{cases} \tag{8.1}$$

For simplicity, we consider casual operators. For $n = 4$, the matrix A has the form

$$A = \begin{bmatrix} D_1 & U_1 V_2^H & U_1 W_2 V_3^H & U_1 W_2 W_3 V_4^H \\ 0 & D_2 & U_2 V_3^H & U_2 W_3 V_4^H \\ 0 & 0 & D_3 & U_3 V_4^H \\ 0 & 0 & 0 & D_4 \end{bmatrix}. \tag{8.2}$$

Let us first split the SSS matrix as following

$$A = \left[\begin{array}{cc|cc} D_1 & U_1 V_2^H & U_1 W_2 V_3^H & U_1 W_2 W_3 V_4^H \\ 0 & D_2 & U_2 V_3^H & U_2 W_3 V_4^H \\ \hline 0 & 0 & D_3 & U_3 V_4^H \\ 0 & 0 & 0 & D_4 \end{array}\right]. \tag{8.3}$$

The top-left block goes to the left branch of the HSS representation, while the right-bottom block goes to the right branch. The root is defined by setting:

$$\begin{cases} \widehat{B}_{1;1,2} = I & \widehat{B}_{1;2,1} = 0 & \widehat{W}_{1;1} = I \\ \widehat{W}_{1;2} = W_2^H & \widehat{R}_{1;1} = W_3 & \widehat{R}_{1;2} = I. \end{cases} \tag{8.4}$$

Then we construct the left branch with a similar partitioning.

$$\left[\begin{array}{c|c} D_1 & U_1 V_2^H \\ \hline 0 & D_2 \end{array}\right] \tag{8.5}$$

hence

$$\widehat{D}_{2;1} = D_1 \quad \widehat{U}_{2;1} = U_1 \quad \widehat{V}_{2;1} = V_1 \tag{8.6}$$

while for the right child

$$\widehat{D}_{2;2} = D_2 \quad \widehat{U}_{2;2} = U_2 \quad \widehat{V}_{2;2} = V_2. \tag{8.7}$$

In order to keep the HSS representation valid, R and W matrices on the left node should be set properly. That is

$$\begin{cases} \widehat{R}_{2;1} = W_2 & \widehat{R}_{2;2} = I & \widehat{W}_{2;1} = I \\ \widehat{W}_{2;2} = W_1^H & \widehat{B}_{2;2,1} = 0 & B_{2;1,2} = I \end{cases} \tag{8.8}$$

and similarly for the right branch with partitioning as in (8.5)

$$\widehat{D}_{2;3} = D_3 \quad \widehat{U}_{2;3} = U_3 \quad \widehat{V}_{2;3} = V_3, \tag{8.9}$$

$$\widehat{D}_{2;4} = D_4 \quad \widehat{U}_{2;3} = U_4 \quad \widehat{V}_{2;4} = V_4. \tag{8.10}$$

In order to keep the HSS representation valid, R and W matrices on the right node should be set properly. That is

$$\begin{cases} \widehat{R}_{2;3} = W_4 & \widehat{R}_{2;4} = I & \widehat{W}_{2;3} = I \\ \widehat{W}_{2;4} = W_3^H & \widehat{B}_{2;4,3} = 0 & \widehat{B}_{2;3,4} = I. \end{cases} \tag{8.11}$$

Finally the HSS representation can be written as:

$$A = \begin{bmatrix} \widehat{D}_{2;1} & \widehat{U}_{2;1}\widehat{B}_{2;1,2}\widehat{V}_{2;2}^H & \widehat{U}_{2;1}R_{2;1}\widehat{B}_{1;1,2}\widehat{W}_{2;3}^H\widehat{V}_{2;3}^H & \widehat{U}_{2;1}\widehat{R}_{2;1}\widehat{B}_{1;1,2}\widehat{W}_{2;4}^H\widehat{V}_{2;4}^H \\ 0 & \widehat{D}_{2;2} & \widehat{U}_{2;2}\widehat{R}_{2;2}\widehat{B}_{1;1,2}\widehat{W}_{2;3}^H\widehat{V}_{2;3}^H & \widehat{U}_{2;2}\widehat{R}_{2;2}\widehat{B}_{1;1,2}\widehat{W}_{2;4}^H\widehat{V}_{2;4}^H \\ 0 & 0 & \widehat{D}_{2;3} & \widehat{U}_{2;3}\widehat{B}_{2;3,4}\widehat{V}_{2;4}^H \\ 0 & 0 & 0 & \widehat{D}_{2;4} \end{bmatrix} \tag{8.12}$$

with all the translation matrices set in equation (8.4) to (8.11).

The general transformation is then as follows. First we must partition the SSS matrix according to a certain hierarchical partitioning. Then for a current HSS node at k level which should contain the SSS blocks A_{xy} where $i \le x, y \le j$ $(1 \le i < j \le n)$ and assuming the HSS block is further partitioned at block h $(i < h < j)$ the translation matrices of the current node can be chosen as

$$\begin{cases} \widehat{B}_{k;2i-1,2i} = I & \widehat{B}_{k;2i,2i-1} = 0 & \widehat{W}_{k;2i-1} = I \\ \widehat{W}_{k;2i} = \prod_{x=h}^{i} W_x^H & \widehat{R}_{k;2i-1} = \prod_{x=h+1}^{j} W_x & \widehat{R}_{k;2i} = I \end{cases} \tag{8.13}$$

note that undefined W_x matrices are set equal I (the dimension of I is defined according to context). If $i = h$ or $h + 1 = j$, then one (or two) HSS leaf (leaves) have to be constructed by letting

$$\widehat{D}_{k;i} = D_h \quad \widehat{U}_{k;i} = U_h \quad \widehat{V}_{k;i} = V_h. \tag{8.14}$$

After the HSS node of the current level has been constructed, the same algorithm is applied recursively to construct the HSS node for SSS blocks $A_{xy}, i \le x, y \le h$ and for SSS block $A_{xy}, h+1 \le x, y \le j$ (the recursion stops when a leaf is constructed.).

Observing the fact that all $\widehat{B}_{k;2i,2i-1}$ matrices are zeros matrices and $W_{k;2i-1}$, $R_{k;2i-1}$ are identity matrices, modifications can be done to get a more efficient HSS representation.

8.2. From HSS to SSS

In this section, we shall consider HSS as recursive SSS using the concise time-varying notation of [5]. We shall first illustrate the algorithm by an example on 8×8 HSS representation. Different partitioning are possible, e.g., those illustrated in Figure 6.

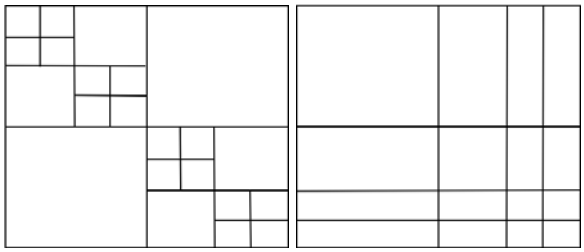

FIGURE 6. HSS partitioning (on the left), SSS partitioning (on the right).

We shall only consider the upper triangular case, as that is the standard case in time-varying system theory. The 4-level balanced HSS representation can be expanded as:

$$A = \begin{bmatrix} D_{1;1} & U_{1;1}B_{1;1,2}W_{2;3}^H V_{2;3}^H & U_{1;1}B_{1;1,2}W_{2;4}^H W_{3;7}^H V_{3;7}^H & U_{1;1}B_{1;1,2}W_{2;4}^H W_{3;8}^H V_{3;8}^H \\ 0 & D_{2,3} & U_{2;3}B_{2;3,4}W_{3;7}^H V_{3;7}^H & U_{2;3}B_{2;3,4}W_{3;8}^H V_{3;8}^H \\ 0 & 0 & D_{3;7} & U_{3;7}B_{3;7,8}V_{3;8}^H \\ 0 & 0 & 0 & D_{3;8}. \end{bmatrix} \tag{8.15}$$

This has to be converted to the time-varying representation for $k = 4$:

$$A = \begin{bmatrix} D_1 & B_1C_2 & B_1A_2C_3 & B_1A_2A_3C_4 \\ 0 & D_2 & B_2C_3 & B_2A_3C_4 \\ 0 & 0 & D_3 & B_3C_4 \\ 0 & 0 & 0 & D_4. \end{bmatrix} \tag{8.16}$$

Representing the time-varying realization matrices as $T_k = \left[\begin{array}{c|c} A_k & C_k \\ \hline B_k & D_k \end{array}\right]$) we obtain

$$T_1 = \left[\begin{array}{c|c} . & . \\ \hline U_{1;1}B_{1;1,2} & D_{1;1} \end{array}\right], T_2 = \left[\begin{array}{c|c} W_{2;4}^H & W_{2;3}^H V_{2;3}^H \\ \hline U_{2;3}B_{2;3,4} & D_{2;3} \end{array}\right] \tag{8.17}$$

$$T_3 = \left[\begin{array}{c|c} W_{3;8}^H & W_{3;7}^H V_{3;7}^H \\ \hline U_{3;7}B_{3;7,8} & D_{3;7} \end{array}\right], T_4 = \left[\begin{array}{c|c} . & V_{3;8}^H \\ \hline . & D_{3;8}. \end{array}\right]. \tag{8.18}$$

More generally, it is easy to see that the realization at k step is given by

$$T_k = \left[\begin{array}{c|c} A_k & C_k \\ \hline B_k & D_k \end{array}\right] = \left[\begin{array}{c|c} W_{k;2^k}^H & W_{k;2^k-1}^H V_{k;2^k-1}^H \\ \hline U_{k;2^k-1}B_{k;2^k-1,2^k} & D_{k;2^k-1}. \end{array}\right]. \tag{8.19}$$

According to the reconfigured partitioning, we see that for step k (indexing the current node) all right children belong to the further steps, while all left children go to $D_{k;2^k-1}$ in the realization of the current step. $W_{k;2^k-1}$, $W_{k;2^k}$ and $B_{k;2^k-1,2^k}$ are the translation matrices of the current node. $U_{k;2^k-1}$ and $V_{k;2^k-1}$ form the column base and row base of the current node, yet they are not explicitly stored.

Note that, according to the HSS definition, they should be generated(recursively) from the left children.

The conversion algorithm should start from the root node and proceed recursively. After constructing the realization on the current step, the algorithm proceeds by setting the right child as the current node and the algorithm goes recursively until it reaches the right bottom where no more right child exist. Then the realization of the last step will be given as:

$$\left[\begin{array}{c|c} \cdot & V^H_{k-1;2^{k-1}} \\ \hline \cdot & D_{k-1;2^{k-1}} \end{array}\right] \tag{8.20}$$

since a leaf does not have a right child.

To show how a HSS tree can be split as time-varying steps, we shall show the partition on an HSS binary tree shown in Figure 7.

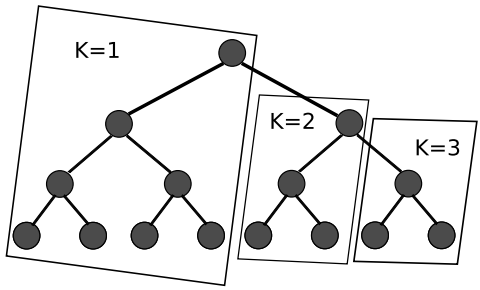

FIGURE 7. Binary tree partitioning.

$D_{k;2^k-1}$ is a potentially large HSS block. Another level of time-varying notation can be used to represent this $D_{k;2^k-1}$ whose realization may again contain sub-blocks represented by the time-varying notation. Since $U_{k;2^k-1}$, $V_{k;2^k-1}$ are not explicitly stored and can be derived locally from the current step, no efficiency is lost by applying recursive time-varying notation.

Here are a number of remarks on the recursive time-varying notation for HSS:

1. $D_{k;2^k-1}$ in the realization is an HSS block which can either be represented in HSS form or by time-varying notation. This suggests a possibly hybrid notation consisting of HSS representations and recursive time-varying notations.
2. $U_{k;2^k-1}$ and $V_{k;2^k-1}$ form HSS bases generated from $D_{k;2^k-1}$. For this recursive time-varying notation, they should not be explicitly stored and can be derived locally in the current step.
3. It is possible to represent general HSS matrices (not just block upper-triangular matrices) with the recursive time-varying notation.
4. All fast HSS algorithms can be interpreted in a recursive time-varying fashion.
5. Some algorithms applied on time-varying notation described in [5] can be extended to the recursive time-varying notation (HSS representation).

9. Final remarks

Although the HSS theory is not yet developed to the same full extent as the sequentially semi-separable theory, the results obtained so far show that the HSS structure has indeed a number of very attractive properties that make it a welcome addition to the theory of structured matrices. Fundamental operations such as matrix-vector multiplication, matrix-matrix multiplication and matrix inversion (including the Moore-Penrose case accounted for in [18]) can all be excuted with a computational complexity linear in the size of the matrix, and additional efficiency induced by the translation operators. A representation in terms of global diagonal and shift operators is also available, very much in the taste of the more restrictive multi-scale theory. These formulas have not yet been exploited fully. The connection with time-varying system theory is also very strong, and it should be possible in the future to transfer a number of its results to the HSS representation, in particular model reduction, interpolation and Hankel norm approximation (i.e., model reduction).

References

[1] W. Lyons, "Fast algorithms with applications to pdes," *PHD thesis*, June 2005.

[2] I. Gohberg, T. Kailath, and I. Koltracht, "Linear complexity algorithms for semiseparable matrices," *Integral Equations and Operator Theory*, vol. 8, pp. 780–804, 1985.

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

[4] V. Rokhlin, "Applications of volume integrals to the solution of pde's," *J. Comp. Phys.*, vol. 86, pp. 414–439, 1990.

[5] P. Dewilde and A.-J. van der Veen, *Time-varying Systems and Computations*. Kluwer, 1998.

[6] A. van der Veen, "Time-varying lossless systems and the inversion of large structured matrices," *Archiv f. Elektronik u. Übertragungstechnik*, vol. 49, no. 5/6, pp. 372–382, Sept. 1995.

[7] Y. Eidelman and I. Gohberg, "On a new class of structured matrices," *Notes distributed at the 1999 AMS-IMS-SIAM Summer Research Conference*, vol. Structured Matrices in Operator Theory, Numerical Analysis, Control, Signal and Image Processing, 1999.

[8] S. Chandrasekaran, M. Gu, and T. Pals, "A fast and stable solver for smooth recursively semi-separable systems," in *SIAM Annual Conference, San Diego and SIAM Conference of Linear Algebra in Controls, Signals and Systems, Boston*, 2001.

[9] P. Dewilde and A.-J. van der Veen, "Inner-outer factorization and the inversion of locally finite systems of equations," *Linear Algebra and its Applications*, vol. 313, pp. 53–100, 2000.

[10] S. Chandrasekaran, M. Gu, and T. Pals, "Fast and stable algorithms for hierarchically semi-separable representations," in *Technical Report*. University of California at Santa Barbara, April 2004.

[11] W. Hackbusch, "A sparse arithmetic based on $\mathcal{H}$-matrices. part i: Introduction to $\mathcal{H}$-matrices," *Computing*, vol. 64, pp. 21–47, 2000.

[12] T. Pals, "Multipole for scattering computations: Spectral discretization, stabilization, fast solvers," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of California, Santa Barbara, 2004.

[13] P. Dewilde, K. Diepold, and W. Bamberger, "A semi-separable approach to a tridiagonal hierarchy of matrices with application to image flow analysis," in *Proceedings MTNS*, 2004.

[14] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, "A fast solver for hss representations via sparse matrices," in *Technical Report*. Delft University of Technology, August 2005.

[15] S. Chandrasekaran, Z. Sheng, P. Dewilde, M. Gu, and K. Doshi, "Hierarchically semi-separable representation and dataflow diagrams," vol. Technical document, Nov 2005.

[16] S. Chandrasekaran, P. Dewilde, W. Lyons, T. Pals, and A.-J. van der Veen, "Fast stable solver for sequentially semi-separable linear systems of equations," October 2002.

[17] S. Chandrasekaran, M. Gu, and T. Pals, "A fast ulv decomposition solver for hierachically semiseparable representations," 2004.

[18] P. Dewilde and S. Chandrasekaran, "A hierarchical semi-separable Moore-Penrose equation solver," *Operator Theory: Advances and Applications*, vol. 167, pp. 69–85, Nov 2006, Birkhäuser Verlag.

[19] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, T. Pals, A.-J. van der Veen, and J. Xia, "A fast backward stable solver for sequentially semi-separable matrices," September 2005.

Zhifeng Sheng and Patrick Dewilde
Circuits and Systems Group
Faculty of EEMCS
Delft University of Technology The Netherlands
e-mail: `Z.Sheng@ewi.tudelft.nl`
e-mail: `p.dewilde@ewi.tudelft.nl`

Shivkumar Chandrasekaran
Room 3109, Engineering I
Department of Electrical and Computer Engineering
University of California Santa Barbara
CA 93106-9560, USA
e-mail: `Shiv@ece.ucsb.edu`