Electronic Transactions on Numerical Analysis.
Volume 33, pp. 163-188, 2009.

ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

TRANSFORMING A HIERARCHICAL INTO A UNITARY-WEIGHT REPRESENTATION*

STEVEN DELVAUX†, KATRIJN FREDERIX‡, AND MARC VAN BAREL‡

Abstract. In this paper, we consider a class of hierarchically rank structured matrices that includes some of the hierarchical matrices occurring in the literature, such as hierarchically semiseparable (HSS) and certain $\mathcal{H}^2$-matrices. We describe a fast ($O(r^3 n \log(n))$) and stable algorithm to transform this hierarchical representation into a so-called unitary-weight representation, as introduced in an earlier work of the authors. This reduction allows the use of fast and stable unitary-weight routines (or by the same means, fast and stable routines for sequentially semiseparable (SSS) and quasiseparable representations used by other authors in the literature), leading, e.g, to direct methods for linear system solution and for the computation of all the eigenvalues of the given hierarchically rank structured matrix.

Key words. hierarchically semiseparable (HSS) matrix, $\mathcal{H}^2$-matrix, low rank submatrix, tree, QR factorization, unitary-weight representation

AMS subject classifications. 65F30, 15A03

1. Introduction.

1.1. Hierarchically rank structured matrices in the literature. In the literature, several types of hierarchically rank structured matrices have been investigated. A first example is the class of $\mathcal{H}$-*matrices*, which has been studied, e.g., in [13, 15]. Loosely speaking, a matrix is called an $\mathcal{H}$-matrix if it can be partitioned into a set of disjoint blocks of low rank. This idea can be used to approximate the matrices arising in the discretization of certain integral equations. The idea of partitioning such matrices into disjoint blocks of low rank also appears in the so-called *mosaic skeleton method* in [20, 21]. A typical example [13, 15] of the partition in low-rank blocks occurring in $\mathcal{H}$-matrices is shown in Figure 1.1(a).

In many cases, additional speed-up can be achieved by forcing the different low-rank blocks into which the $\mathcal{H}$-matrix is partitioned to be related to each other. An often-encountered condition in this respect is that the row and column spaces of the generators of the low-rank blocks must be compatible, in the sense that the low-rank blocks must form huge horizontal and vertical low-rank 'shafts' in the matrix. A graphical illustration is given in Figure 1.1(b); this figure shows some of the horizontal low-rank shafts by means of bold boxes. Rk r denotes that the rank of the shafts is at most r.

The precise way in which the huge low-rank shafts as in Figure 1.1(b) are enforced will be recalled in Section 2. We note that some examples of hierarchically rank structured matrices that are explicitly based on this principle are the classes of $\mathcal{H}^2$-*matrices* [14, 16]

*Received January 31, 2008. Accepted May 14, 2009. Published online on December 11, 2009. Recommended by Ahmed Salam. The research was partially supported by the Research Council K.U. Leuven, project OT/05/40 (Large rank structured matrix computations), CoE EF/05/006 Optimization in Engineering (OPTEC), by the Fund for Scientific Research–Flanders (Belgium), G.0455.0 (RHPH: Riemann-Hilbert problems, random matrices and Padé-Hermite approximation), G.0423.05 (RAM: Rational modelling: optimal conditioning and stable algorithms), and by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization). The scientific responsibility rests with its authors.

†Department of Mathematics, Katholieke Universiteit Leuven, Celestijnenlaan 200B, B-3001 Leuven (Heverlee), Belgium. (`Steven.Delvaux@wis.kuleuven.be`). The first author is a Postdoctoral Fellow of the Fund for Scientific Research, Flanders (Belgium). The work of this author is supported by the Onderzoeksfonds K.U. Leuven/Research Fund K.U. Leuven.

‡Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven (Heverlee), Belgium. ({`Katrijn.Frederix,Marc.VanBarel`}`@cs.kuleuven.be`).

ETNA
Kent State University
http://etna.math.kent.edu

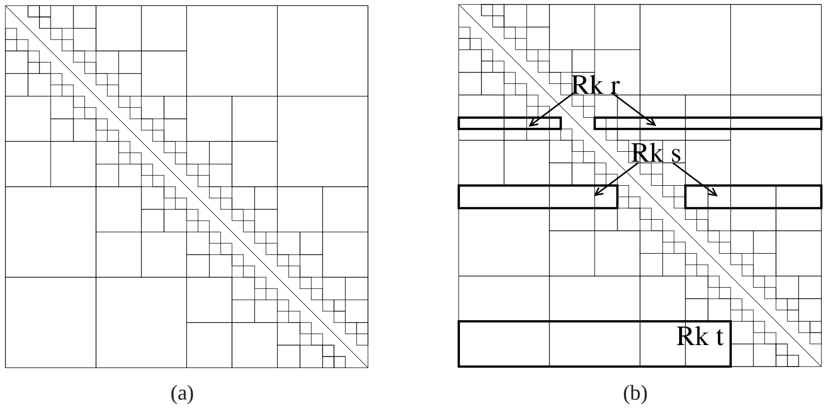


FIGURE 1.1. *(a) Example of an $\mathcal{H}$-matrix. Each of the indicated blocks is of low rank. The closer to the main diagonal they are, the more difficult it is for the elements to be approximated by low-rank blocks. (b) Example of an $\mathcal{H}^2$-matrix hierarchical structure. The row and column space generators of the different low-rank blocks are now related in such a way that huge horizontal and vertical low-rank shafts are formed. The figure shows some of the horizontal low-rank shafts by means of the bold boxes (but there are many others which are not shown in the figure). Rk r indicates that the rank of the shaft is at most r.*

and *hierarchically semiseparable matrices* (HSS matrices) [2, 4], all of which have been introduced very recently.

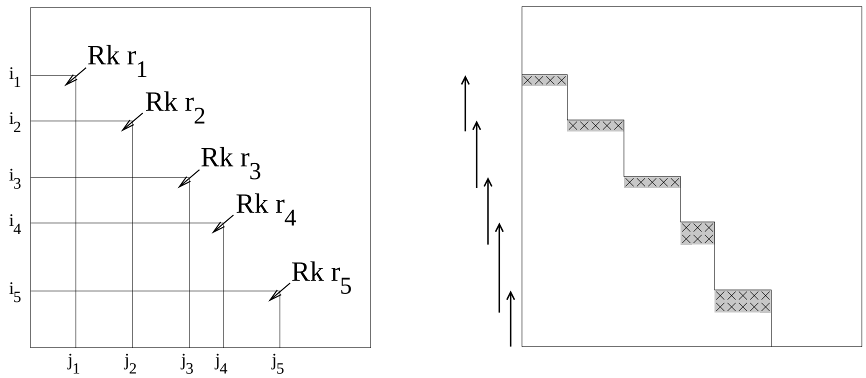
Historically, the hierarchically rank structured matrices of the last paragraph were first used in the 1980s in the *Fast Multipole Method* [1, 12]. This method can be interpreted as a fast way to compute the matrix-vector multiplication with a hierarchically rank structured matrix as in Figure 1.1(b). In addition, the method describes how one can approximate in this format a matrix whose (i, j)th entry is given by the evaluation of an appropriate bivariate function $f(\mathbf{x}_i, \mathbf{x}_j)$ in a set of points $\mathbf{x}_i \in \Omega \subset \mathbb{R}^d$, $i = 1, \ldots, n$, for some fixed dimension $d \in \{1, 2, 3\}$. These approximations are of an analytical flavor, and are based on separable expansions of the form $f(\mathbf{x}, \mathbf{y}) \approx \sum_{j=1}^{r} g_j(\mathbf{x}) h_j(\mathbf{y})$. The point is to find such separable expansions on several subdomains of the domain $\Omega \times \Omega$. Here, the number of terms r is related to the rank of the low-rank blocks in the hierarchical structure.

The interpretation of the Fast Multipole Method in terms of hierarchically rank structured matrices in the general higher-dimensional case $d > 1$ is given in [19]. The classes of $\mathcal{H}^2$- and HSS matrices mentioned above can then be viewed as an underlying matrix framework to describe the Fast Multipole Method.

Apart from matrix-vector multiplication, there are also situations where one is interested in the solution of a linear system with a hierarchically rank structured coefficient matrix. Such solution methods were originally iterative; see, e.g., [13, 15]. Recently, it was shown in [2, 4] how to provide fast and stable direct solvers for HSS-type matrices. This may be a very important contribution in view of the fast and stable manipulation of these matrices.

Recently, hierarchically rank structured matrices were also considered as a tool for the numerical approximation of (Fourier transformed) Toeplitz matrices [18]. The low-rank shafts involved in the approximation of these matrices are termed *neutered block rows* and *neutered block columns* by these authors, and they derive an $O(n \log^2(n))$ method to approximate a general Toeplitz matrix in this format. These authors also present a fast method for solving the hierarchically rank structured linear system [17], but this method is probably unstable in the general case. It seems, however, that a combination of the approximation techniques in [18] with the solution methods in [2, 4] might lead to a superfast and stable

ETNA
Kent State University
http://etna.math.kent.edu



(a) (b)

FIGURE 1.2. *(a) Rank structured matrix having* 5 *structure blocks. (b) Unitary-weight representation.*

solver for Toeplitz matrices.

1.2. Rank structured matrices. Another class of matrices that is often used in the literature is the class of *rank structured matrices* [8]. These are matrices with low-rank blocks that are contiguous and include either the lower left or the upper right corner of the matrix. We call these low-rank blocks *structure blocks*. In contrast to the blocks in the hierarchical rank structure (hierarchical blocks), which are small and disjoint, these structure blocks are large and are allowed to intersect each other.

Each structure block can be characterized by a 3-tuple $\mathcal{B}_k = (i_k, j_k, r_k)$, with i_k the row index, j_k the column index, and r_k the rank upper bound. A collection of structure blocks is a rank structure and a matrix $A \in \mathbf{C}^{m\times n}$ satisfies the rank structure if for each k, $\operatorname{rank} A(i_k : m, 1 : j_k) \leq r_k$[1]. A graphical illustration of a rank structure with five structure blocks is shown in Figure 1.2(a); the meaning of Figure 1.2(b) is explained further in Section 4.1. By symmetry considerations it will be sufficient to describe our algorithm only for those structure blocks in the *lower* triangular part of A, but it is useful to keep in mind that the block *upper* triangular part of A also will be rank structured, i.e., that the matrix A^T also will satisfy the definition of rank structure.

1.3. Rank structure induced by hierarchical rank structure. The aim of this paper is to 'embed' the hierarchically rank structured matrices of Section 1.1 into the larger class of rank structured matrices described in Section 1.2. To see what this means, recall that the hierarchically rank structured matrices of Section 1.1 consist of a number of *small, disjoint* low-rank blocks (hierarchical blocks). To get these into the format of Section 1.2, we have to combine these hierarchical blocks into huge, non-disjoint low-rank blocks that start from the bottom left corner element of the matrix (structure blocks). One can achieve this by constructing structure blocks via *tilings* of the given hierarchical blocks or shafts. To see what this means, the reader could already have a quick glimpse at Figure 1.3(a). The figure is explained in more detail below.

It is clear that this tiling procedure only requires the hierarchical blocks in the *lower* triangular part of A. Hence, from now on, we will be allowed to 'decouple' the hierarchical

[1]This MATLAB-like notation is interpreted as follows: $i : m = [i, i+1, i+2, \ldots, m]$ and $A(i : m, 1 : j)$ denotes the submatrix of A with rows labeled by $i : m$ and columns labeled by $1 : j$. Note that this submatrix lies in the lower left corner of A.

ETNA
Kent State University
http://etna.math.kent.edu

structure by neglecting its upper triangular part. Moreover, in order for this tiling procedure to lead to structure blocks with reasonably small ranks, it is clear that the region around the bottom left corner element of the matrix should be well-approximated by a low-rank block. A typical example of a hierarchical rank structure for which this is the case is shown in Figure 1.1(a); the point here is that the off-diagonal regions can be well-approximated by low-rank hierarchical blocks. A counterexample is shown in Figure 2.3. See Section 2 for the precise assumptions that we will impose on the hierarchical rank structure.

Let us now give a rough description of the expected rank upper bounds of the structure blocks induced by this tiling procedure for a typical class of $\mathcal{H}$- or $\mathcal{H}^2$-matrices. Consider a matrix $H \in \mathbb{C}^{n\times n}$ ($n = 2^\alpha$), which is partitioned into disjoint low-rank blocks of size $n/2^k$ (k is the corresponding level, $k = 1, \dots, \alpha$) as shown in Figure 1.1(a). It is assumed that all blocks are of the same rank r. For the matrix H in the case of the $\mathcal{H}$-matrix, no relation is defined between the hierarchical blocks, while in the case of the $\mathcal{H}^2$-matrices, the hierarchical blocks are organized in shafts as in Figure 1.1(b). The rank of the structure blocks of each level k will then be $O(k^2 r)$ in case of the $\mathcal{H}$-matrices and $O(kr)$ in case of the $\mathcal{H}^2$-matrices. This means that the rank of the structure blocks increases compared to the rank of the hierarchical blocks by a factor $\log^2(n)$ (when k becomes $\alpha = \log(n)$) for a typical class of $\mathcal{H}$-matrices, and by a factor $\log(n)$ in the case of the $\mathcal{H}^2$-matrices. We will work with $\mathcal{H}^2$-matrices in the rest of this paper.

We illustrate how these rank bounds are obtained for an $\mathcal{H}^2$-matrix with $r = 1$ in Figure 1.3. Figure 1.3(a) shows a structure with rank-one hierarchical blocks (indicated by the number '1' in the middle of each block). It also shows an example of a structure block (surrounded by the outermost bold box). The rank of this structure block is obtained by partitioning it into a tiling of horizontal and vertical shafts in a minimal way; we find here a tiling with four shafts, and hence the given structure block is of rank at most four. In Figure 1.3(b), the corresponding rank upper bounds of *all* the different structure blocks are shown; e.g., the structure block in Figure 1.3(a) has the value '4' at the position indicated by the arrow. The other values should be interpreted in the same way.

Note that the rank structure in Figure 1.3(b) includes a lot of 'inner' structure blocks, where inner means that the structure block is fully contained in another structure block. For practical reasons [5, 8], we focus only on the *outermost* structure blocks, i.e., the structure blocks that are closest to the main diagonal. Note that the rank of these outermost structure blocks is typically 4 ($\approx \log n$) around the middle and < 4 close to the borders of the matrix.

1.4. Outline of the paper. The above observations show that the hierarchically rank structured matrices of Section 1.1 often can be embedded in the larger class of rank structured matrices described in Section 1.2, with rank upper bounds that increase by a moderate factor of about $\log n$. This opens the door for practical algorithms to achieve this embedding. In this paper, we will present such an embedding algorithm. We will do this by transforming the parameters of the hierarchically rank structured matrix representation (cf. Section 2) into those for a *unitary-weight representation* [5]. Figure 1.2(b) shows an example of a unitary-weight representation; the basic ideas of this representation are recalled in Section 4.1. The embedding algorithm requires about $O(r^3 n \log(n))$ operations.

When the unitary-weight representation has been computed, one can then make use of a variety of fast and stable routines for working with rank structured matrices, including methods for linear system solution [7] and the computation of all the eigenvalues of the given hierarchically rank structured matrix [6]. The reduction to a unitary-weight representation is not restrictive, since this representation can be easily transformed [5] into other kinds of representations for rank structured matrices, namely the *block quasiseparable* (also called *sequentially semiseparable*) representations introduced in [8] and subsequently used by several

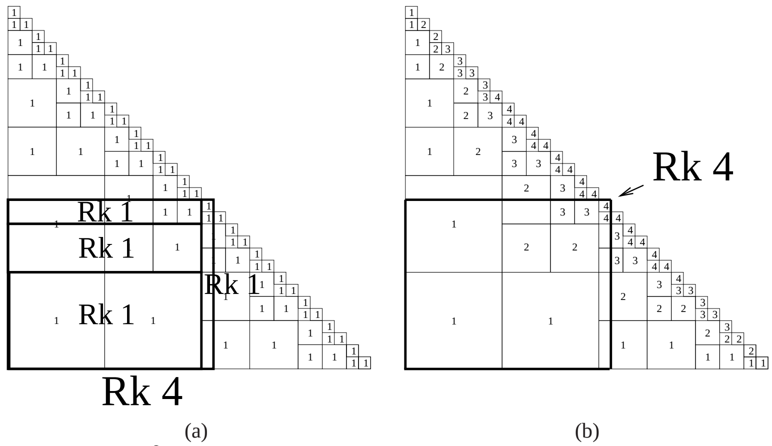


FIGURE 1.3. *(a) A $\mathcal{H}^2$ hierarchically rank structured matrix with rank-one hierarchical blocks. The the indicated structure block can be realized as a tiling of 4 rank-one shafts (3 horizontal and 1 vertical) and hence it is of rank at most 4. (b) For the matrix in Figure 1.3(a), the figure shows the rank of each structure block, written at the top right corner of the structure block. Note that typically the rank increases by a factor of about 4.*

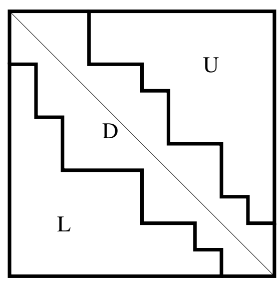


FIGURE 2.1. *A partition of the matrix H in three parts: the block lower, block upper, and block diagonal part. These three parts are denoted in the figure by L, U, and D, respectively.*

authors in the literature. Thus, once the hierarchically rank structured matrix has been embedded in the class of rank structured matrices, one can make use of virtually all the existing routines to perform matrix operations with rank structured matrices in a fast and accurate way [3, 9–11].

The remainder of this paper is structured as follows. Section 2 defines the hierarchical structures that are considered in this paper. Section 3 discusses existing algorithms for matrix-vector multiplication and how they are adapted to our class of matrices. Section 4 discusses an algorithm for transforming the hierarchical representation into a unitary-weight representation. Section 5 gives the numerical performance of the algorithm for certain $\mathcal{H}^2$-matrices. Section 6 states the conclusion.

2. Hierarchically rank structured matrices. In this section the class of matrices of interest in this paper is defined. In what follows, the matrix $H \in \mathbb{C}^{n\times n}$ is often partitioned into three parts: its block lower (L), block upper (U) and block diagonal part (D). It is assumed that the block lower triangular part of H is a union of contiguous submatrices of H that contain the bottom left corner element of H, and, similarly, that the block upper triangular part of H is a union of contiguous submatrices of H that contain the upper right corner element of H; see Figure 2.1.

The hierarchical structure is obtained by partitioning the block lower and upper triangular

ETNA
Kent State University
http://etna.math.kent.edu

parts of H into small, disjoint blocks of low rank, as in Figure 1.1(a). Additionally, we want certain relations to hold between these blocks in order to guarantee the existence of huge horizontal and vertical low-rank shafts, as in Figure 1.1(b). This can be achieved with the following definition.

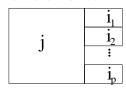
DEFINITION 2.1 (Hierarchically rank structured matrix). *Let $H \in \mathbb{C}^{n\times n}$, and let there be given a partition of H into its block lower, block upper and block diagonal part as described above. A* lower hierarchical structure *on the matrix H involves:*

(i) *A partition of the block lower triangular part of H into a set of disjoint blocks of low rank. If the jth low-rank block ($j = 1, \ldots, J$, where J is the total number of blocks) has size s_j by t_j and rank at most r_j, then we assume for this block a factorization of the form*

$$U_j B_j V_j, \tag{2.1}$$

with $U_j \in \mathbb{C}^{s_j\times r_j}$, $B_j \in \mathbb{C}^{r_j\times r_j}$, and $V_j \in \mathbb{C}^{r_j\times t_j}$. Here U_j is called the row shaft generator*, V_j the* column shaft generator*, and B_j the* intermediate matrix *of the jth low-rank block.*

(ii) *For all neighboring low-rank blocks which are distributed along the shape*



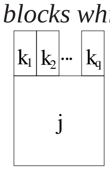
,

the row shaft generators in (2.1) satisfy the relation

$$U_j = \begin{bmatrix} U_{i_1}T_{i_1,j} \\ U_{i_2}T_{i_2,j} \\ \vdots \\ U_{i_p}T_{i_p,j} \end{bmatrix}, \tag{2.2}$$

for certain $T_{i,j} \in \mathbb{C}^{r_i\times r_j}$. The matrices $T_{i,j}$ are called row transition matrices.

(iii) *For all neighboring low-rank blocks which are distributed along the shape*



,

the column shaft generators in (2.1) satisfy the relation

$$V_j = \begin{bmatrix} S_{j,k_1}V_{k_1} & S_{j,k_2}V_{k_2} & \ldots & S_{j,k_q}V_{k_q} \end{bmatrix}, \tag{2.3}$$

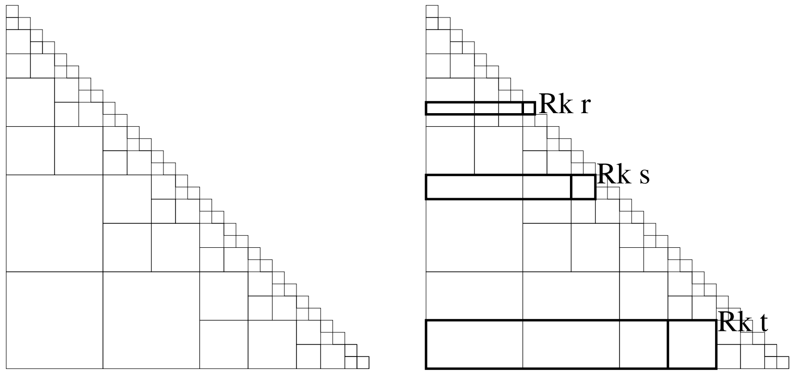
for certain $S_{j,k} \in \mathbb{C}^{r_j\times r_k}$. The matrices $S_{j,k}$ are called column transition matrices.

(iv) *Neighboring low-rank blocks that are not distributed along the shape of the two previous items, are not allowed.*

Finally, one can define an upper *hierarchical structure in a similar way to how the lower hierarchical structure is defined above. A matrix H is said to be* hierarchically rank structured *if it has hierarchical rank structure in both its lower and its upper triangular parts, possibly combined with some unstructured matrix part around the main diagonal of the matrix.*

We note that Definition 2.1 implies that the different low-rank blocks are compatible in the sense that they form large horizontal and vertical *shafts*. This means that for each low-rank block, the submatrix obtained by extending this low-rank block completely to the left-hand side or the bottom of the matrix must have the same rank upper bound r_j as the low-rank block j itself, forming what we call a horizontal or vertical shaft, respectively. Figure 2.2(a) shows an example of a lower hierarchical structure underlying a typical class of $\mathcal{H}^2$-matrices [16], and Figure 2.2(b) shows some horizontal shafts. The vertical shafts are analogous.

ETNA
Kent State University
http://etna.math.kent.edu



(a) (b)

FIGURE 2.2. *(a) A typical example of a lower hierarchical rank structure of an $\mathcal{H}^2$-matrix. (b) Some examples of horizontal shafts induced by this partitioning. In each case, the horizontal shaft is obtained by extending a low-rank block j completely to the left border of the matrix. By (2.2), the shaft has the same rank (Rk) upper bound r_j as its rightmost block j; compare with Figure 1.1.*

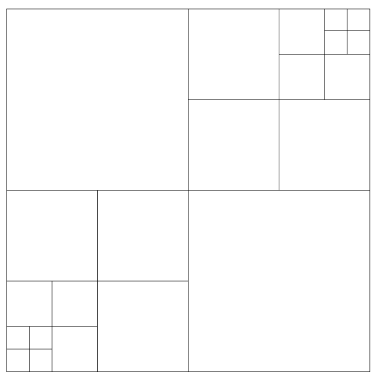


FIGURE 2.3. *Example $\mathcal{H}^2$-matrix structure which is not in the class defined in Definition 2.1*

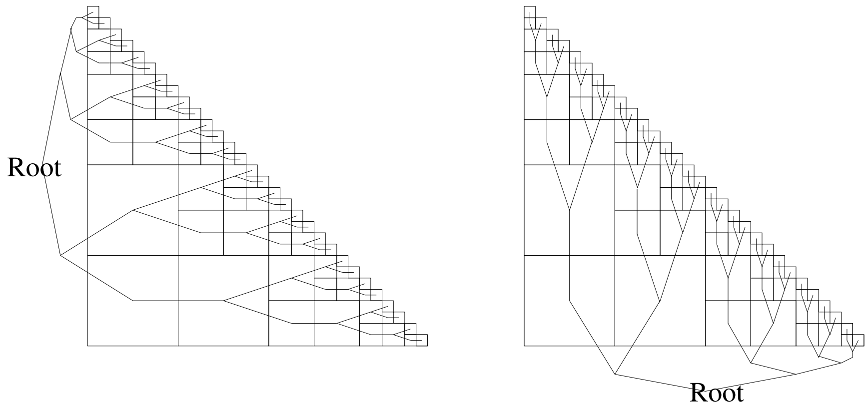
Notice that not all $\mathcal{H}^2$-matrices belong to class defined in Definition 2.1. There are $\mathcal{H}^2$-matrices for which the elements in the left bottom and the right upper corner of the matrix are difficult to approximate with low-rank blocks. Such an example is shown in Figure 2.3.

The main feature that distinguishes Definition 2.1 from the hierarchically rank structured matrices in the literature is the decoupling between the block lower and upper triangular parts of the matrix; compare Figure 1.1 with Figure 2.2. The reason why this decoupling has been done is because we believe that Definition 2.1 yields the natural class of matrices for which the algorithm of Section 4 works.

In the next paragraphs, some auxiliary attributes are defined.

DEFINITION 2.2 (2D graph, row and column tree). *For any hierarchically rank structured matrix as in Definition 2.1, there is a naturally associated planar graph whose nodes correspond to the low-rank blocks j into which the matrix is partitioned, $j = 1, \ldots, J$. This graph is referred to as the* two-dimensional graph, *or the* 2D graph *for short. Its nodes are connected in two ways: by means of the* row *and* column tree *(sometimes referred to as the* 2D row *and* 2D column tree*). These trees are a model for the horizontal and the vertical connections between neighboring low-rank blocks, respectively.*

ETNA
Kent State University
http://etna.math.kent.edu



(a) (b)

FIGURE 2.4. *$\mathcal{H}^2$-matrix hierarchical structure: (a) underlying 2D row tree, (b) underlying 2D column tree. Note that to each block of the matrix there corresponds a node in the tree, and in addition there are some virtual nodes near the roots of the trees.*

Let us provide some examples. First, for the example of the lower hierarchical rank structure in Figure 2.2, the underlying 2D row and column tree are shown in Figure 2.4. Note that in addition to the 'real' nodes these trees also have some 'virtual' nodes, at the left-hand side in the 2D row tree and at the bottom in the 2D column tree. *Virtual nodes* are nodes to which no physical block of the matrix corresponds. These virtual nodes are used only for organizational purposes (and most of them could, in fact, be removed if desired); they serve to remind us how the hierarchical structure is obtained by recursively subdividing a given matrix until all of its blocks are of sufficiently low rank [13–16].

Another example of a hierarchical rank structure is the class of HSS matrices introduced in [2, 4]. The underlying 2D row and column trees are shown in Figure 2.5. Note that these trees also have virtual nodes, not only near the root, but also throughout the rest of the tree; there are even virtual *leaves*! Once again these virtual nodes could, in fact, be removed; but note that the resulting tree would then not be binary anymore. Moreover, the (virtual) root and the virtual leaves play a special role in our algorithm, and hence cannot simply be removed.

Yet another example of a hierarchical rank structure is the class of lower block quasiseparable (also called sequentially semiseparable) representations [8]. In this case the underlying row tree specializes to sequential shape; we omit the details.

In addition to the 2D row and column trees, we can also define the following one-dimensional versions.

DEFINITION 2.3 (1D row and column tree). *Any hierarchically rank structured matrix as in Definition 2.1 has an associated* one-dimensional row tree, *or* 1D row tree *for short. The nodes of this tree are defined as the subsets of* $\{1, \dots, n\}$ *that occur as the row index set of one of the low-rank blocks, and the edges are defined by the natural inclusion relations between these subsets. Definition 2.1 guarantees that this graph is indeed a tree. The* 1D column tree *is defined in an analogous way.*

The 1D row and column tree are often closely related to the process that produces the hierarchically rank structured matrix [14, 16]. They are usually uniform binary trees, corresponding to an interval $I \subseteq \mathbb{R}$ on which a certain integral equation is defined. This interval is gradually cut into finer and finer pieces, leading to the nodes of the 1D row tree. Blocks of the matrix that can be well-approximated by a low rank matrix are kept fixed, while the

ETNA
Kent State University
http://etna.math.kent.edu

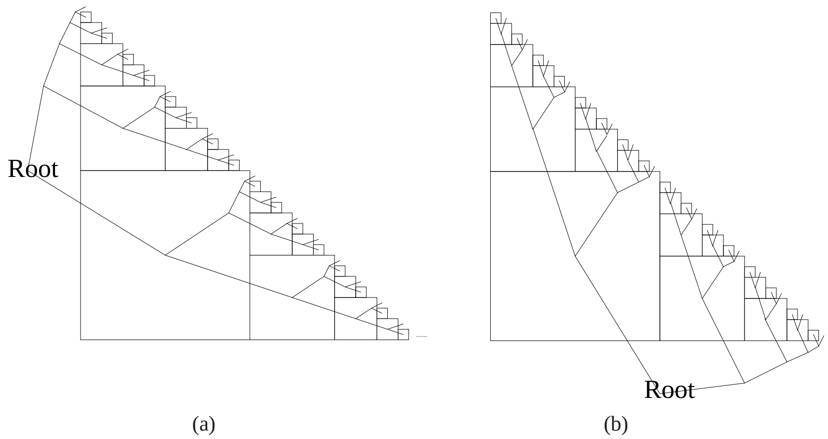


FIGURE 2.5. *Hierarchically semiseparable (HSS) structure: (a) underlying 2D row tree, (b) underlying 2D column tree.*

other blocks are again recursively subdivided, and so on. The virtual nodes in the 2D row tree that we discussed above could then be interpreted as 'placeholders' for those levels of the 1D row tree to which no physical low-rank block of the matrix corresponds. In fact, to each node of the 1D row tree there can correspond zero, one, or more than one nodes of the 2D graph. Some examples where multiple nodes of the 2D graph lie on the same 1D row level can be found in Figure 2.4(a). Examples of virtual nodes can be found in Figures 2.4(a) and 2.5(a).

3. Matrix-vector multiplication.

In this section, the multiplication $\mathbf{y} = H\mathbf{x}$ between a hierarchically rank structured matrix $H \in \mathbb{C}^{n\times n}$ and a vector $\mathbf{x} \in \mathbb{C}^{n\times 1}$ is discussed. The algorithm allows a description in terms of the 2D graph, and thus in terms of the 2D row and column trees. It is reminiscent of the matrix-vector multiplication algorithm for the *Fast Multipole Method* in the literature [1, 12], although the class of matrices for which it applies is slightly different, since we allow the structure in the block lower and upper triangular parts of our matrices to be decoupled. The description of the algorithm is included here only for completeness of the paper.

By the additivity of the matrix-vector multiplication, it clearly suffices to describe the matrix-vector multiplication with the block lower triangular part of the hierarchically rank structured matrix H. Indeed, the matrix-vector multiplication with the block upper triangular part then can be treated similarly, while the matrix-vector multiplication with the unstructured part of the matrix around the diagonal can be performed using standard matrix techniques.

It will be convenient to denote by $\mathbf{x}_k$ the part of the given vector $\mathbf{x}$ that corresponds to the indices of the *vertical* shaft induced by node k. Similarly, we denote by $\mathbf{y}_k$ the part of the matrix-vector product $\mathbf{y}$ that corresponds to the indices of the *horizontal* shaft induced by node k.

In the first phase of the computation, we want to compute for each node k the matrix-vector product $\mathbf{w}_k := B_k V_k \mathbf{x}_k$. (Recall the notations of Definition 2.1). To do this in an efficient way, the column vector $\mathbf{w}_k \in \mathbb{C}^{r_k}$ will be initialized for each k to be zero. The recursive relation (2.3) suggests that we can run through the 2D *column* tree (e.g., in depth-first order). The root of the column tree is used as starting node and the column children of each node are recursively considered. Each edge of the column tree is visited twice, once in the parent-child and once in the child-parent direction. The algorithm is as follows:

- When arriving at a leaf: update $\mathbf{w}_{\text{leaf}} = V_{\text{leaf}}\mathbf{x}_{\text{leaf}}$.

- For each transition child $\rightarrow$ parent:
  update $\mathbf{w}_{\text{parent}} = \mathbf{w}_{\text{parent}} + S_{\text{parent,child}}\mathbf{w}_{\text{child}}$, and
  update $\mathbf{w}_{\text{child}} = B_{\text{child}}\mathbf{w}_{\text{child}}$.

If the root is a real node (not virtual), update $\mathbf{w}_{\text{root}} = B_{\text{root}}\mathbf{w}_{\text{root}}$. At the end of this phase, the auxiliary vector $\mathbf{w}_k := B_k V_k \mathbf{x}_k$ for each node k will have been computed.

In the second phase of the algorithm, we want to compute the different pieces $\mathbf{y}_{\text{leaf}}$ of the required matrix-vector product $\mathbf{y} = H\mathbf{x}$. To do this in an efficient way, an auxiliary column vector $\mathbf{z}_k \in \mathbb{C}^{r_k}$, initialized to be $\mathbf{w}_k$, is defined for each node k. The recursive relation (2.2) suggests then that we can run through the 2D *row* tree (e.g., in depth-first order) and

- For each transition parent $\rightarrow$ child:
  update $\mathbf{z}_{\text{child}} = \mathbf{z}_{\text{child}} + T_{\text{child,parent}}\mathbf{z}_{\text{parent}}$.
- When arriving at a leaf: update $\mathbf{y}_{\text{leaf}} = U_{\text{leaf}}\mathbf{z}_{\text{leaf}}$.

At the end of this phase, we will have computed the different pieces $\mathbf{y}_{\text{leaf}}$ of the required matrix-vector product $\mathbf{y} = H\mathbf{x}$.

4. Transition to a unitary-weight representation.

In this section, we discuss how one can compute a unitary-weight representation as defined in [5] for the hierarchically rank structured matrices of Section 2. As we explained in Section 1.3, this can be considered as *embedding* the hierarchically rank structured matrices into the larger class of rank structured matrices.

REMARK 4.1. It is possible to devise a *sequential* method for computing the unitary-weight representation. Such a method was implemented for the rank-one case and presented by the authors at the International Conference on Matrix Methods and Operator Equations, Moscow, Russia, June 2005. The algorithm was also reported in the master thesis of Yvette Vanberghen, Faculty of Science and Applied Science, K. U. Leuven, Leuven, Belgium (written in the Dutch language). However, this method involves taking certain Schur complements of the data, and we found that it unfortunately becomes numerically unstable for the higher rank case. For this reason, in the present section, we describe an alternative, *hierarchical* method for achieving this goal. This method does not always lead to the technically correct ranks of the structure blocks, but this is compensated by a better efficiency and numerical stability.

In what follows we will describe a hierarchical method for computing the unitary-weight representation. We start with some preliminaries.

4.1. Basics of the unitary-weight representation.

A unitary-weight representation is a compact representation of a rank structured matrix [5]. It consists of only a small number of parameters written as a pair $(\{Q_l\}_{l=1}^{L}, W)$, where Q_l are *elementary unitary operations* and W is the *weight matrix*; L is the total number of structure blocks. An example is shown in Figure 1.2(b); the upward pointing arrows on the left-hand side denote the unitary operations, and the elements in the grey area denote the weight matrix.

The basic idea behind the unitary-weight representation is to 'compress' a given rank structure by means of elementary row operations, proceeding from the bottom to the top of the matrix and storing in each step the non-zero elements just before they reach the top border of the rank structure. In other words, we want to create as many zeros as possible in the rank structure and thereby bring some 'condensed' information ('weights') to the top of the rank structure.

An elementary row operation is a unitary operation $Q = I \oplus \tilde{Q} \oplus I$, where the I denote identity matrices of appropriate sizes and $\tilde{Q}$ is a unitary operation. If Q is applied to a matrix

ETNA
Kent State University
http://etna.math.kent.edu

$H \in \mathbb{C}^{n\times n}$, then only the rows that correspond to $\tilde{Q}$ are changed:

$$QH = \begin{bmatrix} I & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \tilde{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I \end{bmatrix} \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} = \begin{bmatrix} H_1 \\ \tilde{H}_2 \\ H_3 \end{bmatrix}.$$

The following technique is used in the construction of the unitary-weight representation in order to create zeros in all but the top rows. Consider a matrix $M \in \mathbb{C}^{\tilde{m}\times\tilde{n}}$ of low rank r. This matrix can be factored as $M = QR$, where $Q \in \mathbb{C}^{\tilde{m}\times\tilde{m}}$ is unitary and $R \in \mathbb{C}^{\tilde{m}\times\tilde{n}}$ is upper triangular. Because M is of low rank, R has the form

$$Q^H M = R = \begin{bmatrix} \tilde{M} \\ \mathbf{0} \end{bmatrix},$$

where $\tilde{M} \in \mathbb{C}^{r\times\tilde{n}}$. This means that if the conjugate transpose of the unitary operation Q is applied to M, all but the top r rows are converted to zeros.

The construction of the unitary-weight representation always starts at the bottom of the rank structure, so the aforementioned technique is applied to the bottommost structure block of the rank structure, for instance block 5 in Figure 1.2(a). This results in zeros except in the top rows (these non-zero elements are called weights). The weights that do not lie in the next structure block (block 4 in Figure 1.2(a)) are saved in the weight matrix. (In Figure 1.2, these are the elements in the structure block 5 with column index from $j_4 + 1, \ldots, j_5$.) The other weights are combined with the original elements of the next structure block (In Figure 1.2, elements of structure block 4 with row indices $i_4, \ldots, i_5 - 1$). On this combined matrix, the technique is applied again. Then the same procedure of saving the weights outside the next structure block and combining the weights with the original elements of the next structure block is followed until the top of the rank structure is reached. At the end, one obtains a weight matrix and a set of unitary operations, one for each structure block. In Figure 1.2(b), the unitary weight-representation of Figure 1.2(a) is shown (with $r_5 = r_4 = 2, r_3 = r_2 = r_1 = 1$).

The previous paragraphs gave a short introduction to the concept of unitary-weight representation; for more information, the reader is referred to [5].

4.2. Basic idea of the embedding algorithm. In this subsection, we discuss the basic idea of the algorithm to embed the hierarchically rank structured matrix into the class of rank structured matrices. The idea of the algorithm is to compress the given matrix H by means of elementary unitary row operations. Because we start from a structure according to Definition 2.1, this is done for the subsequent levels of the 2D row tree, going from finer to coarser levels and from the bottom to the top of the structure, and in between transmitting information from a child to its parent. Since this process can be considered as computing the first part of a QR factorization of H, we should then also store the resulting 'weights' at the top border of the structured lower triangular part during this process. This storage is performed at the nodes of the 2D graph. The final weights at the end of the algorithm arrive in the *leaves of the column tree* (the reader should try to see this!). At the end of the algorithm, we also obtain the elementary unitary operations $\{Q_l\}_{l=1}^{L}$ of the unitary-weight representation. These are stored in the *leaves of the row tree*. This is shown in Figure 4.1: Figure 4.1(a) shows the schematic begin configuration of the algorithm (in fact, the algorithm starts with the hierarchically rank structured formulation as defined in Definition 2.1); and Figure 4.1(b) shows the final result of the algorithm, with the elementary unitary operations indicated at the left and with weight blocks (depicted on a grey background) indicated at each column leaf.

Notice the difference between Figure 1.2(b) and Figure 4.1(b). In Figure 1.2(b), each structure block has a weight in which the number of nonzero rows remains the same. In

ETNA
Kent State University
http://etna.math.kent.edu

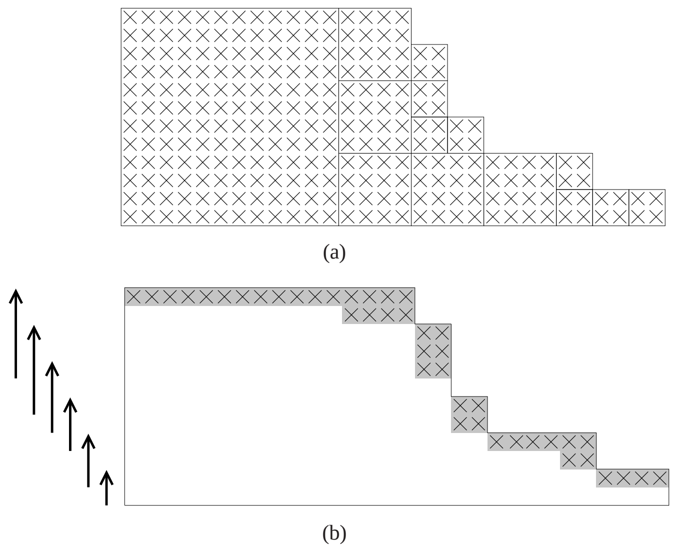


FIGURE 4.1. *(a) Starting situation for the algorithm. (b) Final situation for the algorithm. Here all the hierarchical blocks are assumed to be of rank 1. Note that the ranks of the induced structure blocks are all 1, 2 or 3.*

Figure 4.1(b), this is not the case; in the top structure block, there is a weight consisting of a part with one row and another with two rows. This difference is because in the second case the rank structure is generated from hierarchical blocks according to Definition 2.1.

4.3. Organization of the algorithm. In this subsection, an overall description of the organization of the embedding algorithm is given. We discuss the variables that are used during the algorithm and give pseudocode for the two most important components of the algorithm. The detailed explanation of these algorithms and their subroutines is given in Section 4.4. An illustration with a worked-out example is given in Section 4.6.

We start by listing the variables that are used by the algorithm. The input of the algorithm is the 2D graph of the given hierarchically rank structured matrix. According to Definitions 2.1 and 2.2, this is practically organized as follows:

- Each node i of the 2D graph has references to its parent and children in the (2D) column and row tree.
- Each node i of the 2D graph that corresponds to a real block in the matrix (real node) contains its intermediate matrix B_i. Nodes that do not correspond to a real block in the matrix, are called virtual nodes. For instance, nodes 1, 2, 3, 5, 6, and 9, are virtual in Figure 4.4.
- Each node i of the 2D graph contains the row transition matrix $T_{i,j}$ to its row parent (node j) and the column transition matrix $S_{k,i}$ to its column parent (node k).
- The row and column shaft generators U and V are defined in the row and column leaves, respectively.

The following data is also used during the algorithm:

- For each node i of the 2D graph, a memory element containing a 'weight' W_i and a 'temporary weight '$W\text{temp}_i$ is created.
- For the row leaves a memory element Q_{leaf} is created which contains at the end of the algorithm the product of the applied unitary compression operations.

Note that the above variables are expressed only in terms of the 2D graph and the cor-

ETNA
Kent State University
http://etna.math.kent.edu

responding (2D) row and column trees as in Definition 2.2. Indeed, the corresponding one-dimensional (1D) row and column trees are *not* used in our implementation, although they are implicitly considered by the use of virtual nodes in the 2D graph. We stress that this is a purely organizational point, and is by no means essential.

During the first part of the algorithm, only the row shaft generators U, the intermediate matrices B, and the transition matrices T and S are considered. The column shaft generators V are used only at the very end of the algorithm. In fact, the main purpose of the algorithm is to create zero weight matrices in the nodes which are not column leaves by applying elementary unitary row operations. At the end of the algorithm, each column leaf contains a non-zero weight matrix and each row leaf contains a unitary operation; these are the main components for a unitary-weight representation.

In the remainder of this subsection, pseudocode for the two main programs of the algorithm is given. The pseudocode may be difficult to follow now, but it will be gradually explained in the following subsections. At the beginning of the algorithm, the root of the row tree is used as starting node. For the example in Figure 2.4(a), this is the virtual node labelled 'Root 'on the left-hand side of the figure. The first program is the main program, which recursively calls itself for each of the row children of the input node node in order from the bottom child to the top child. The second program, which is called by the first program, executes the most important transition operations of the algorithm.

Program transform_to_unitary_weight_representation(node):

1. FOR $i = p$ (number of row children of node) down to 1
   transform_to_unitary_weight_representation(node.rowchild(i))
   END FOR
2. IF the number of row children is zero (node is a row leaf)
   Apply QR factorization on row shaft generator U of node.
   END IF
3. IF the number of row children is greater than zero (node is not a row leaf)
   Expand information from bottommost row child to parent (node).
   Update the weight of bottommost row child.
   END IF
4. Transmit information upwards:
   FOR $i = p$ (number of row children) down to 2 (node has more than one row child)
   - Define
     child := node.rowchild(i),
     toplevel := topdescendant(nextchild),
     nextchild := node.rowchild(i-1),
     currentlevel := topdescendant(child),
     The function topdescendant is returns the topmost row descendant of a given 2D node, which is always a row leaf.
   - Set auxnode := currentlevel.
   - WHILE auxnode $\neq$ node
     - IF auxnode is non-virtual
       (currentlevel)= transmit_upwards(auxnode,currentlevel,toplevel)
       END IF
     - Set auxnode := row parent of auxnode.
     END WHILE
   - Expand information of nextchild to node.
   - compress(node,toplevel)
   END FOR

ETNA
Kent State University
http://etna.math.kent.edu

5. IF node = row root and non-virtual
   Update the weight of node.
   END IF

End program transform_to_unitary_weight_representation.

Program (currentlevel)=transmit_upwards(auxnode, currentlevel, toplevel)

1. Compress:
   IF auxnode has its topmost row index less than or equal to the one of currentlevel, and if currentlevel is different from toplevel, and if auxnode has non-virtual column children, then
   - compress(auxnode,currentlevel). This compression is redundant when it is invoked for the first time, i.e., when currentlevel still equals the value to which it was initialized in the main program, since then a compression has been done already there.
   - Update currentlevel := currentlevel.nextleaf. Here, nextleaf is a function that returns the next leaf in the 2D row tree, i.e., the row leaf whose bottommost row index is adjacent to the topmost row index of currentlevel. Note that this leaf could be virtual.

   END IF
2. Recursively transmit information upwards:
   IF auxnode is not at toplevel and there is a non-virtual column descendant
   FOR $i = q$ (number of column children) down to 1
   - Set columnchild := auxnode.columnchildren(i).
   - IF columnchild is non-virtual
     – Bring weight of auxnode upwards by postmultiplying the weight with the transition matrix $S_{\text{auxnode,columnchild}}$ to obtain a new weight.
     – IF weight of columnchild is empty
       Save the new weight in the variable $W_{\text{columnchild}}$
       ELSE
       Save the new weight in the variable $W\text{temp}_{\text{columnchild}}$
       END IF
     – (currentlevel)=transmit_upwards(columnchild,currentlevel,toplevel)

     END IF

   END FOR
   Set weight of auxnode to empty
   END IF

End program transmit_upwards.

4.4. Detailed description of the algorithm. In this section, we give a detailed description of the programs mentioned in the previous section.

4.4.1. Main program. As described in the previous section, the main program transform_to_unitary_weight_representation recursively visits the nodes in the row tree in depth-first order, always processing the bottom children first. Nodes are processed in different ways depending on how many row children they have. We explain below the different actions that can occur when processing node j (we use the same labeling scheme that we used in the pseudocode of Section 4.3).

1. The program first recursively processes the row children of node j.
2. If node j is a row leaf, the row shaft generator $U_j \in \mathbb{R}^{s_j \times r_j}$ is decomposed as $U_j = Z_j R_j$, where $Z_j \in \mathbb{R}^{s_j \times s_j}$ is unitary and $R_j \in \mathbb{R}^{s_j \times r_j}$ is upper triangular.

ETNA
Kent State University
http://etna.math.kent.edu

The unitary operation $Q_{\text{leaf}} = Z_j^H$ and the weight $W_j = R_j(1:\min(s_j, r_j), 1:r_j)$ are stored.

3. If node j is not a row leaf, then we will denote by p the number of row children of node j; the bottom-most row child of node j is i_p. Information about node i_p is brought to node j by using the transtion matrix $T_{ip,j}$. That is, the weight of node j is set to $W_j := W_{i_p} T_{i_p,j}$; and the weight of the child i_p is updated by $W_{i_p} := W_{i_p} B_{i_p}$, because all information has now been transmitted to the left. Because every node has a different intermediate matrix B, it is not possible to do the update earlier.
4. When node j has more than one row child, information has to be transmitted upwards, and at the end a compression has to take place.

   FOR $i = p$ (number of row children) down to 2:
   - We define several variables that indicate which nodes transit information upwards and how far the information from those nodes is transmitted:
     child := node.rowchild(i),
     toplevel := topdescendant(nextchild),
     nextchild := node.rowchild(i-1),
     currentlevel := topdescendant(child),

     The function topdescendant returns the topmost row descendant of a given 2D node, which is always a row leaf. The variable currentlevel denotes the highest row level that has already been accessed, and the variable toplevel is the 'ceiling' through which information cannot pass.

     To derive the nodes which have to transmit information upwards, the horizontal line between child and nextchild has to be followed to the right until the end of the rank structure. This line is called the *level line*. The line above the topmost column child is called the *top line*. See Figure 4.5(e) for an illustration. All the nodes that are to the level line from below are considered sequentially (from the finest to the coarsest node) in the transmit_upwards phase. Information from these nodes is recursively transmitted upward to any column descendants until toplevel is reached.
   - The following loop now is executed while auxnode $\neq$ node.
     - The transmit_upwards phase consists of two parts. In the first part, an extra compression is applied (if necessary); and in the second part, the information is transmitted upwards until the toplevel is reached. This is explained in detail in Section 4.4.2.
     - Next, auxnode becomes the row parent of the previous auxnode. If auxnode $\neq$ node, transmit_upwards is called on the new auxnode.
   - When auxnode is node, all information has been brought upwards from all the nodes below the level line. Now nextchild has to be expanded to node, its row parent. This means that
     - The weight is stored in a temporary weight:
$$W\text{temp}_{\text{node}} = W_{\text{node}}.$$
     - The new weight is computed as follows:
$$W_{\text{node}} = W_{\text{nextchild}} T_{\text{nextchild,node}}.$$
     - The weight of nextchild is updated,
$$W_{\text{nextchild}} = W_{\text{nextchild}} B_{\text{nextchild}}.$$

ETNA
Kent State University
http://etna.math.kent.edu

- When all the information has been transmitted upwards, compression can take place for node j and the nodes that are attached to the top line from below. All these nodes have two weights, a weight W and a temporary weight Wtemp. The information in the two weights has to be merged into a single, hopefully smaller weight by applying a unitary row operation. The compression phase is explained in detail in Section 4.4.3.

END FOR

5. At the very end of the algorithm, in case when the row root is non-virtual, the weight of the row root has to be updated with its intermediate matrix B.

Then, from the information stored in the weight W and the column shaft generator V in the column leaves, the weight matrix of the unitary-weight representation is extracted. Together with the unitary operations in the row leaves, the unitary-weight representation is obtained.

4.4.2. Transmit upwards phase. The transmit upwards phase is called with three arguments: auxnode, currentlevel, and toplevel. The variable auxnode is the node which is going to transmit information upwards to its column children and descendants. The variable currentlevel is the highest level already accessed and the variable toplevel is the ‘ceiling’ above which information may not pass. This means that auxnode is going to transmit information upwards to its column children and further descendants, until the toplevel is reached.

The transmit upwards phase consists of two parts. In the first part, an extra compression is applied (if necessary); and in the second part, the information is transmitted upwards. The description is as follows (we use the same labelling as in the pseudo-code in Section 4.3).

1. In the first part, a compression occurs (if not the first time) when the level of auxnode is at least as high as that of currentlevel, currentlevel is different from toplevel, and auxnode has non-virtual column children. After this the variable currentlevel becomes the next row leaf encountered in the direction of the top of the row tree.
2. In the second part, information is transmitted upwards when auxnode has non-virtual column children and auxnode does not lie on the same level as the toplevel (transition parent $\rightarrow$ column children):

   FOR $i = q$ (number of column children) down to 2:

   Set columnchild = auxnode.columnchild(i).

   IF columnchild is non-virtual.

   - The information which has to be transmitted upwards is constructed using the column transition matrix S between these nodes to obtain the new weight

   $$W_{\text{auxnode}} S_{\text{auxnode,columnchild}}.$$

   - If the weight of the columnchild is not empty, store the new weight as the temporary weight $W\text{temp}_{\text{columnchild}}$; otherwise, store it as the weight $W_{\text{columnchild}}$.
   - Now the transmit upwards program is called again, but with columnchild as its first argument.

   END IF

   END FOR

   When all column children of auxnode have been considered, all information has been transmitted upwards. This means that the weight of auxnode is of no further use, therefore it is set empty.

4.4.3. Compression phase. Finally, we can now describe the actual compression routine compress. This routine is invoked at two different places in the algorithm: (i) in the main program when all the information has been transmitted upwards and (ii) during the transmit upwards phase itself. The compression will take place on a horizontal chain of nodes lying

ETNA
Kent State University
http://etna.math.kent.edu

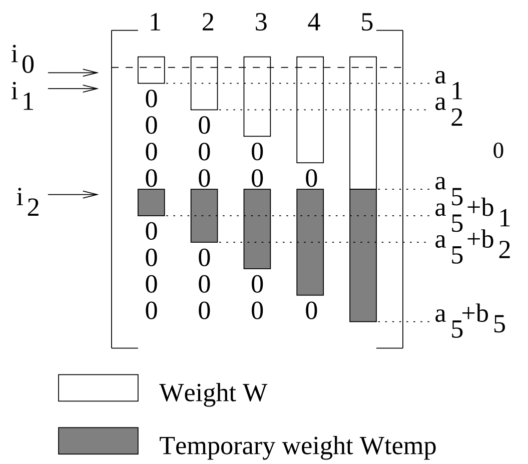


FIGURE 4.2. *Matrix A which contains the weights W and temporary weights Wtemp.*

between a starting node j and a row leaf leaf. All the nodes in the chain have two weights, a weight W and a temporary weight Wtemp. The information of the two weights has to be merged into a single, hopefully smaller weight by applying a unitary row operation. Before the actual compression can take place some parameters have to be introduced:

- Number the nodes that have to be compressed as $k = 1, \ldots, K$ and store the original node numbers in a vector: $\mathbf{s} = [j, \ldots, \text{leaf}]$ (the last node is always a row leaf).
- Define two vectors $\mathbf{a}$ and $\mathbf{b}$ of length K that contain the number of rows of the weights W and temporary weights Wtemp, i.e., a_k and b_k are the number of rows of the weight W_{s_k} and the temporary weight $W\text{temp}_{s_k}$, respectively, for $k = 1, \ldots, K$.
- Define a vector $\mathbf{c}$ of length K that contains the number of columns of the weights W, or, what is the same, the number of columns of the temporary weights Wtemp.
- The different weights W and Wtemp of all the nodes have to be placed in one matrix A. The number of rows of the matrix A is the sum: $a_K + b_K$ because these values are the highest possible number of rows of the weights (the highest rank is obtained at the leaves). The number of columns of the matrix A is the sum: $\sum_{k=1}^{K} c_k$.
- Place the weights in the matrix A. The values in the vectors $\mathbf{a}$ and $\mathbf{b}$ are (mostly) not the same. The weight W_{s_k} starts from the first row of the matrix A to row a_k (with $k = 1, \ldots, K$). If $a_k < a_K$, then zeros are added to fill the matrix. The temporary weight starts from row $a_K + 1$ to row $a_K + b_k$ (with $k = 1, \ldots, K$). If $b_k < b_K$, then zeros are added to fill the matrix. Figure 4.2 shows matrix A when five nodes are involved ($K = 5$).
- To make matrix A correspond to the actual matrix, the unitary operation corresponding to the leaf, Q_{leaf}, is extended with an identity matrix of the size b_K and on this matrix a preliminary permutation P is applied such that Q_{leaf} becomes (sq is the size of the old matrix Q_{leaf}):

$$Q_{\text{leaf}} = \begin{bmatrix} I_{a_K} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_{b_K} \\ \mathbf{0} & I_{sq-a_K} & \mathbf{0} \end{bmatrix} \begin{bmatrix} Q_{\text{leaf}} & \mathbf{0} \\ \mathbf{0} & I_{b_K} \end{bmatrix}.$$

- Define three row indices i_0, i_1 and i_2. Index $i_0 - 1$ denotes the number of fixed rows that cannot be touched because they were already considered in a previous step. This can be seen in Figure 4.5(f), where the weights of node 12 and 17 cannot be touched

ETNA
Kent State University
http://etna.math.kent.edu

because they are a result of a compression in a previous phase of the algorithm. Index i_1 denotes the first zero row in the top part of the matrix A, index i_2 denotes the first non-zero row in the bottom part of the matrix A. These indices are shown in Figure 4.2.

The intention of the compression is to make the matrix A as sparse as possible by running through the nodes for $k = 1, \ldots, K$ and applying in each step a permutation P_k and a compression C_k. At the end, the unitary operation Q_{leaf} is decomposed as follows:

$$Q_{\text{leaf}} = C_K P_K \ldots C_2 P_2 C_1 P_1 Q_{\text{leaf}}.$$

Now run through the nodes $k = 1, \ldots, K$. Consider the columns of matrix A which correspond to k.

- A permutation is applied to A to bring the weight of the bottom part to the top part of block k, such that all the zeros, which are in between the two weights, appear in the bottom rows of the block k. The permutation is as follows (set $b_0 = 0$, $c_0 = 1$):

$$\bar{P}_k = \begin{bmatrix} I_{i_1-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_{b_k-b_{k-1}} & \mathbf{0} \\ \mathbf{0} & I_{i_2-i_1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I_{a_K+b_K-(i_2+b_k-b_{k-1}-1)} \end{bmatrix}.$$

  The result of applying $\bar{P}_k$ to A is shown for $k = 1$ in Figure 4.3(a). The permutation has to be applied to the same rows of the unitary operation Q_{leaf}; therefore $\bar{P}_k$ has to be extended (because A and Q_{leaf} are of different size) to

$$P_k = \begin{bmatrix} \bar{P}_k & \mathbf{0} \\ \mathbf{0} & I_{sq-a_K} \end{bmatrix}.$$

  Set $Q_{\text{leaf}} = P_k Q_{\text{leaf}}$.
- When the permutation has been applied to both matrices, the actual compression can take place. If the number of non-zero rows starting at row index i_0 is greater than the number of columns c_k of the corresponding block k ($i_1 + b_k - b_{k-1} - i_0 > c_k$), a QR factorization of that part of the matrix A is computed: $Q_A R = A(i_0 : i_1 + b_k - b_{k-1} - 1, 1 + \sum_{K=1}^{k-1} c_k : \sum_{K=1}^{k} c_k)$. Figure 4.3(a) shows the part of A which is compressed for $k = 1$ in a bold box. The unitary operation to be applied to matrix A is as follows ($A = \bar{C}_k A$):

$$\bar{C}_k = \begin{bmatrix} I_{i_0-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & Q_A^H & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_{a_K+b_K-(i_1+b_k-b_{k-1}-1)} \end{bmatrix}.$$

  In the other case, if there are more columns than non-zero rows, it is disadvantageous to compute a QR factorization because no zero rows will be created. Then $\bar{C}_k$ is the identity matrix of size $a_K + b_K$. The compression $\bar{C}_k$ has to be applied to the same rows of the unitary operation Q_{leaf}, therefore $\bar{C}_k$ has to be extended for the same reason as before:

$$C_k = \begin{bmatrix} \bar{C}_k & \mathbf{0} \\ \mathbf{0} & I_{sq-a_K} \end{bmatrix}.$$

  Set $Q_{\text{leaf}} = C_k Q_{\text{leaf}}$.

ETNA
Kent State University
http://etna.math.kent.edu

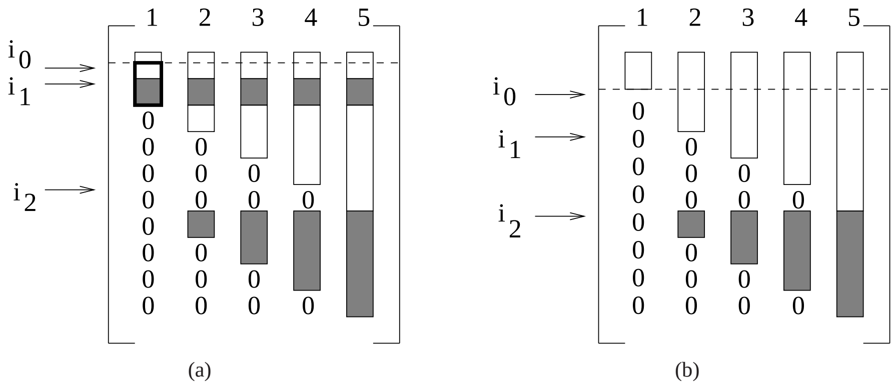


FIGURE 4.3. *(a) Matrix A after permutation $\bar{P}_1$. (b) Matrix A after compression $\bar{C}_1$.*

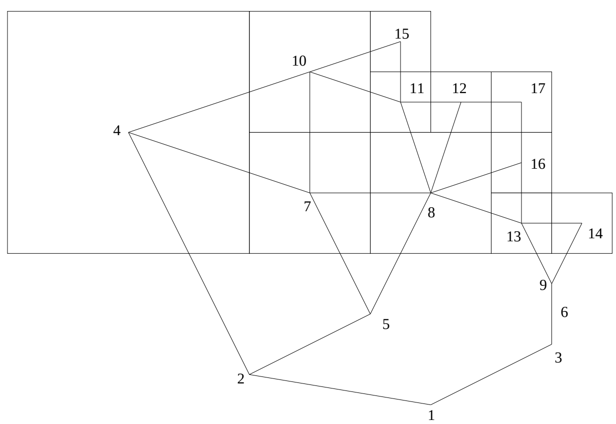


FIGURE 4.4. *$2D$ graph.*

- The weight W_{s_k} of the node s_k is now updated using all the non-zero rows of the matrix A in block k, and the temporary weight $W\text{temp}_{s_k}$ is emptied. Also the three row indices are updated (see Figure 4.3(b)):

$$\begin{aligned} i_0 &= i_0 + \min(c_k, i_1 + b_k - b_{k-1} - i_0), \\ i_1 &= a_{k+1} + b_k + 1, \\ i_2 &= i_2 + b_k - b_{k-1}. \end{aligned}$$

At the end, every node contains a compressed weight W_{s_K}, an empty temporary weight $W\text{temp}_{s_K} = 0$ and the memory element of the leaf contains an adjusted unitary operation Q_{leaf}.

4.5. Computational complexity. The main computational cost of the algorithm is during the transmit upwards phase. Information has to be transmitted upwards $O(n \log(n))$ times, and each transmission involves a matrix multiplication between two matrices of size $r \times r$, which costs $O(r^3)$ operations. Thus, the total cost of this phase gives $O(r^3 n \log(n))$, which is the computational complexity of the algorithm.

4.6. Example. To give a better idea of the algorithm, an example with 17 nodes is elaborated. The corresponding 2D graph is shown in Figure 4.4. First the notation used in

ETNA
Kent State University
http://etna.math.kent.edu

Figure 4.5 is explained. The lines at the top of a node denote the weights W. The lines in the middle or at the bottom of a node denote the temporary weights Wtemp coming from information of the bottom row child or the column parent, respectively. The horizontal lengths of the weights are not correctly represented because the weights are small blocks (recall that the column space generators V are only considered at the very end of the algorithm). Grey areas in a node denote that the weight of this node is set to zero. The unitary operations are denoted by upward pointing arrows at the left of the structure.

The software of this example is available from the authors on request. The algorithms were implemented in MATLAB[2].

The algorithm starts at the row root, node 4. Then the row children are recursively considered (bottom first). Thus, node 14 is the first node where a computation will be done. This is a row leaf; therefore, the computations for a row leaf have been executed. Then the nodes 13, 16, and 8 are considered, in this order, and the corresponding computations of the algorithm is executed. Because node 8 has two row children, information has to be transmitted upwards from node 13 to 16 with transition matrix $S_{13,16}$. This phase of the algorithm is shown in Figure 4.5(a). At the left, the two unitary operations computed in node 14 and 16 are shown by means of the small arrows.

Figure 4.5(b) shows the algorithm when the information has been transmitted upwards and node 16 has been expanded to node 8. When this is done, node 8 and 16 have to be compressed. This results in a bigger weight for node 16, a combined weight for node 8, and an updated unitary operation. This is shown in Figure 4.5(c).

After the compression, we continue to run through the row tree. The nodes 7, 17, 12, 11, 15, and 10 are considered, in this order, and the corresponding computations are executed. Node 10 has more than one row child, so information has to be transmitted upwards between its row children (see Figure 4.5(c)); and node 15 has to be expanded to node 10 (see Figure 4.5(d)). After this, a compression has to be applied to node 10 and 15. The result is shown in Figure 4.5(e). Note that, again, the weight matrices of nodes 10 and 15 have been updated by the compression.

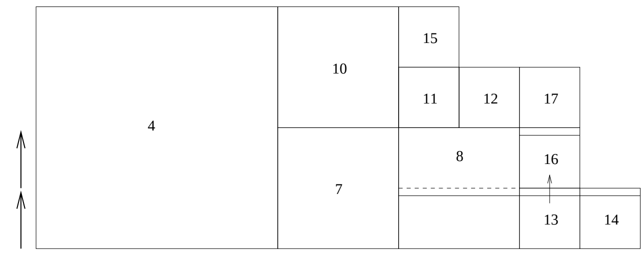
As we continue to run through the row tree, the only node which has not been considered yet is node 4. It gets a temporary weight from node 7. Node 4 has more than one row child. Therefore information has to be transmitted upwards. The nodes which have to transmit information upwards are the nodes that are attached to the level line from below; these are nodes 16, 8, and 7, shown in Figure 4.5(e). These nodes have to transmit the information upwards until the top line is reached. The following variables are set, child := node 7, nextchild := node 10, currentlevel := node 16, toplevel := node 15, and auxnode := node 16.

The transmit upwards phase starts at auxnode = node 16. This node lies on the same level as the currentlevel but not on toplevel and it has column children. Therefore, part 1 of the transmit upwards phase occurs; but this is the first time, so there is no compression. Only currentlevel has to become the next leaf in the row going to the top. This means that currentlevel becomes node 17. Now part 2 of the transmit upwards phase has to be executed, node 16 has column children so information has to be transmitted upwards to node 17. When this has been done, the transmit upwards routine is called with node 17 in its first argument. Node 17 does not fulfill the conditions for part 1 and 2 (it has no column children), so nothing happens.

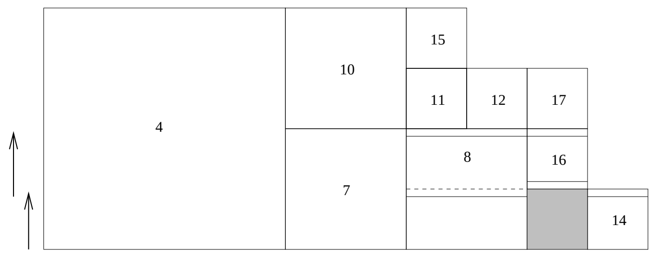
Now the row parent of node 16 is considered, auxnode = node 8. This node does not fulfill the condition for part 1 (it does not lie on the same level as currentlevel), but part 2 will be executed. Hence, information has to be transmitted upwards to node 12, and when this has been done the transmit upwards routine will be called with node 12 as the first argument.

[2]MATLAB is a registered framework of The MathWorks, Inc.

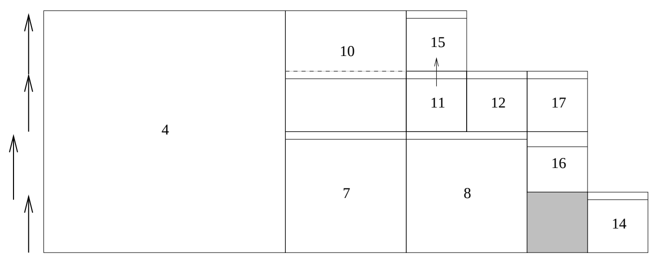
ETNA
Kent State University
http://etna.math.kent.edu



(a) Node 14, 13, 16 and 8 are compressed. Transmit information upwards from node 13 to node 16.



(b) Weight matrix of node 13 is set to zero and a compression is applied on node 8 and 16.



(c) Nodes 7, 17, 12, 11, 15, and 10 are compressed. Transmit information upwards from node 11 to node 15.

FIGURE 4.5. *Constructing unitary-weight representation.*

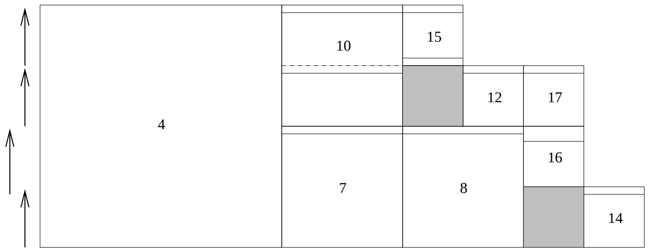
For node 12, the same happens as in node 17: the conditions for part 1 and 2 are not fulfilled. So, the next column child (node 11) of node 8 considered is considered next.

Information has to be transmitted upwards to node 11; and because node 11 has no weight, this information is stored as the weight and not in the temporary weight. The top rows of node 11 are already compressed; therefore these rows will not be touched. The weight will be placed below these rows, as shown in Figure 4.5(f). Now the transmit upwards routine will be called with node 11 as its first argument.

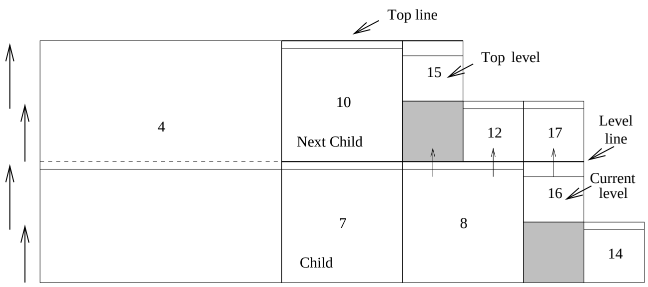
For node 11, the conditions of part 1 are now fulfilled. This means that there has to be a compression from node 11 to node 17; see Figure 4.5(f). In fact, node 11 has already been compressed in a previous phase, so only node 12 and 17 have to be compressed; see Figure 4.5(g). After the compression, the variable currentlevel is set to node 15 and information from node 11 is transferred upwards to node 15. Then node 15 is considered; but this node lies just below the top line, so nothing happens.

Node 8 has transmitted all its information upwards to its column descendants. Now we can go to the row parent of node 8 (auxnode becomes node 7), so that this node can transmit its information upwards to its column children; see Figure 4.5(h). When this has

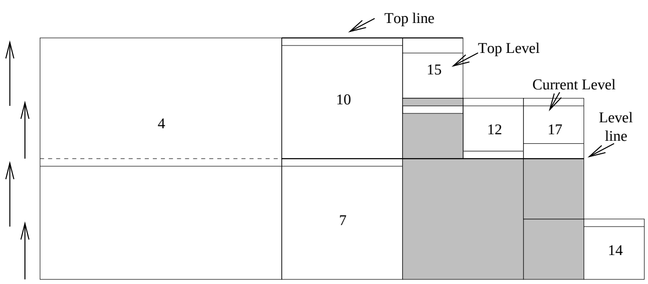
ETNA
Kent State University
http://etna.math.kent.edu



(d) Weight matrix of node 11 is set to zero and a compression is applied tp node 10 and 15.



(e) Transmit information upwards from node 16 to column child 17, and from node 8 to nodes 12 and 15.



(f) Special compression step: Compression of node 11, 12, and 17.

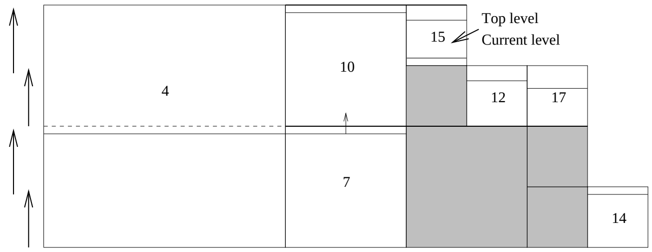
FIGURE 4.5. *Constructing unitary-weight representation (continued).*

been done, auxnode becomes node 4. This is the node where the transmit upwards phase started; therefore, the transmit upwards phase ends here.

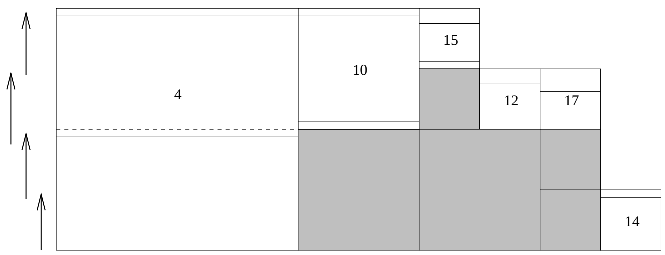
The last phase of the algorithm is to compress nodes 4, 10, and 15; see Figure 4.5(i). After this, the weight of node 4 has to be updated with its intermediate matrix, because it is the row root and non-virtual. At the end, every column leaf contains a weight and every row leaf a unitary operation. The weights in the column leaves have to be multiplied with the corresponding column shaft generators V, to obtain the weight matrix. The weight matrix and the unitary operations in the row leaves are the main components of the unitary-weight representation.

5. Numerical experiments. In this section, the results of numerical experiments on the stability of the transition to a unitary-weight representation are reported. Consider a hierarchically rank structured matrix underlying a typical class of $\mathcal{H}^2$-matrices of size $n = 2^k$ with $k = 9, 10$, as shown in Figure 2.2. Every example is tested for different levels of rank structure. Level 0 is the full matrix, level 1 is the matrix divided into four blocks, level 2 denotes that the inadmissible blocks of level 1 are further divided into four parts, and so

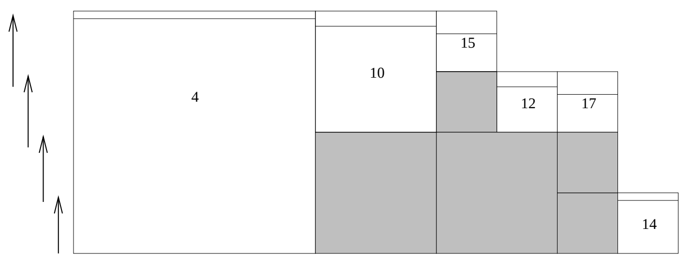
ETNA
Kent State University
http://etna.math.kent.edu



(g) Transmit information from node 7 upwards to node 10.



(h) Compression of node 4, 10, and 15.



(i) Final result: Unitary-weight representation consisting of weights in column leaves and unitary operations in the row leaves.

FIGURE 4.5. *Constructing unitary-weight representation (continued).*

on. For instance, Figure 2.2 is of level 5. In the numerical tests only level 2 through level 5 are considered. Also, three different possibilities corresponding to the rank of the blocks are considered. The first possibility is that all the blocks of low rank have the same rank ($r = 1, \ldots, 5$); the second and third possibilities are that the rank decreases and increases, respectively, from the leaves to the left bottom matrix corner (blocks of the same size have the same rank). The construction of the generators U, V, the transition matrices S, T, and the intermediate matrices B is done with a random number generator which generates numbers uniformly between 0 and 1.

The results of the experiments are shown in Figure 5.1. Based on ten samples, the average error between the original $\mathcal{H}^2$-matrix M and the reconstructed matrix $\tilde{M}$,

$$\|M - \tilde{M}\|_2 / \|M\|_2,$$

is shown. Figure 5.1(a)-5.1(b) shows results for blocks with constant rank, Figure 5.1(c)-5.1(d) shows results for blocks with decreasing rank, and Figure 5.1(e)-5.1(f) shows results for blocks with increasing rank (figures at the left are for $k = 9$ and at the right for $k = 10$). The rank values on the x-axis denote the rank which is defined in the leaves, and when the rank decreases or increases it means that the rank decreases or increases by one when going to a coarser block.

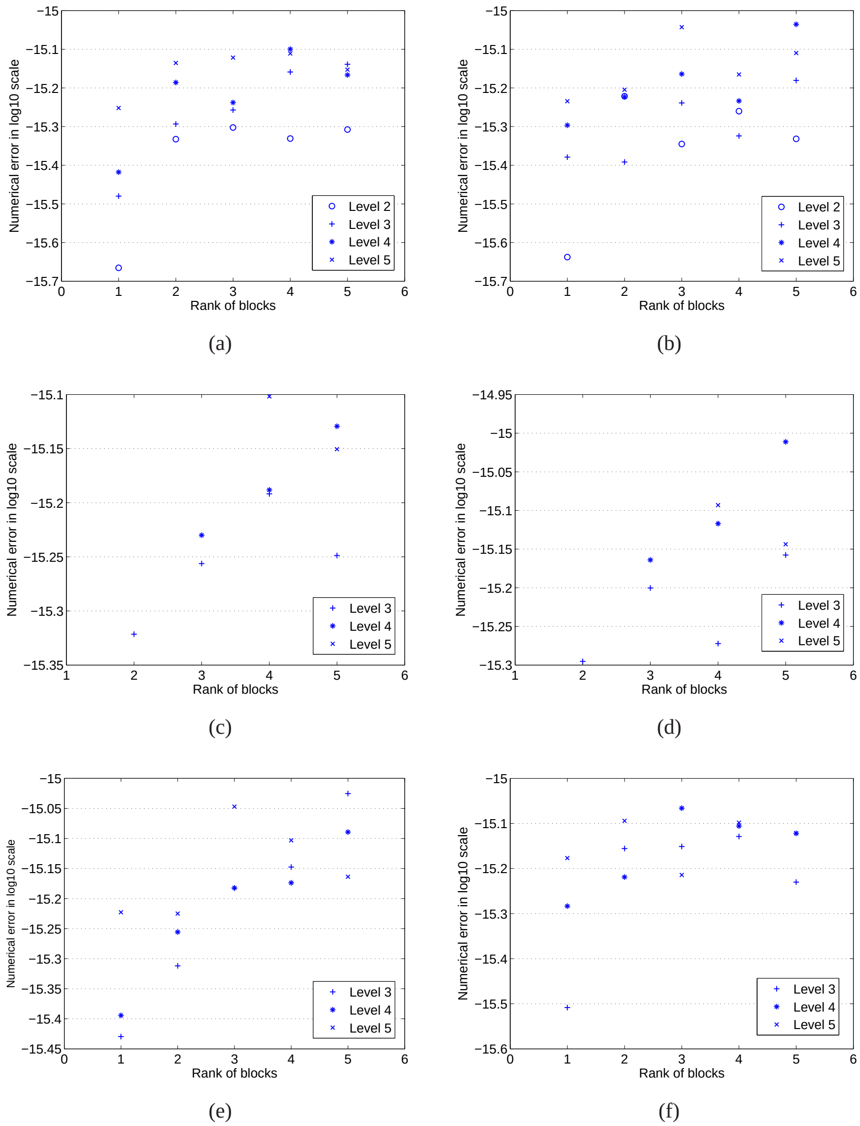


FIGURE 5.1. *Numerical results for* $n = 2^k$, $k = 9$ *(left),* 10 *(right). (a)-(b) For constant rank, (c)-(d) for decreasing rank, (e)-(f) for increasing rank.*

All six figures show that the relative error is of the order 10^{-16}. When the rank of the blocks increases, the relative error is still of the order 10^{-16}. This is also the case when the level increases. When blocks of different size have different rank, the relative error is still of the order 10^{-16}.

In Section 1.3, we gave a description of the expected rank upper bounds of the structure blocks for a typical class of $\mathcal{H}^2$-matrices. For these matrices, the rank increases by a factor of $\log(n)$. Table 5.1 shows the maximal obtained numerical rank of the structure blocks for a test matrix of size 2^9, for different levels and with hierarchical blocks of rank 1 and 2. It shows that the numerical computed ranks of the structure blocks are slightly larger than the

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 5.1
Maximal obtained rank of structure blocks, for $n = 2^9$ and for the different levels. The rank of the hierarchical blocks is considered constant, rank =1, 2.

level	2	3	4	5	6
rank $= 1$	1	2	3	5	7
rank $= 2$	2	4	6	10	14

expected rank upper bounds of the rank structure. For level 5 (rank $= 1$), a rank upper bound of 4 is expected (see Figure 1.3(b)), but numerically several structure blocks of rank 5 were found.

Also, numerical experiments were performed with the unitary-weight representation obtained for the test matrices. The unitary-weight representation was used as input for solving linear systems and computing the eigenvalues of the given hierarchically rank structured matrix; see [6, 7]. The conclusions of these numerical experiments is similar to the results for the test matrices reported in [6, 7].

6. Conclusion. In this paper we described an algorithm to transform a hierarchical representation into a unitary-weight representation in $O(r^3 n \log(n))$ operations. The algorithm is based on compression of the blocks and the transmission of information between blocks. The numerical experiments showed that in all cases the relative error is of order 10^{-16}.

REFERENCES

[1] J. CARRIER, L. GREENGARD, AND V. ROKHLIN, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669–686.
[2] S. CHANDRASEKARAN, P. DEWILDE, M. GU, W. LYONS, AND T. PALS, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.
[3] S. CHANDRASEKARAN, P. DEWILDE, M. GU, T. PALS, AND A.-J. VAN DER VEEN, *Fast stable solver for sequentially semi-separable linear systems of equations*, in Proceedings of High Performance Computing 2002, S. Sahni, V. K. Prasanna, and U. Shukla, eds., Lecture Notes in Computer Science, 2552 (2002), Springer, Berlin, 2002, pp. 545–554.
[4] S. CHANDRASEKARAN, M. GU, AND W. LYONS, *A fast adaptive solver for hierarchically semiseparable representations*, Calcolo, 42 (2005), pp. 171–185.
[5] S. DELVAUX AND M. VAN BAREL, *A Givens-weight representation for rank structured matrices*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1147–1170.
[6] ———, *A Hessenberg reduction algorithm for rank structured matrices*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 895–926.
[7] ———, *A QR-based solver for rank structured matrices*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 464–490.
[8] P. DEWILDE AND A.-J. VAN DER VEEN, *Time-Varying Systems and Computations*, Kluwer Academic Publishers, Boston, Massachusetts, 1998.
[9] ———, *Inner-outer factorization and the inversion of locally finite systems of equations*, Linear Algebra Appl., 313 (2000), pp. 53–100.
[10] Y. EIDELMAN AND I. C. GOHBERG, *On a new class of structured matrices*, Integral Equations Operator Theory, 34 (1999), pp. 293–324.
[11] ———, *A modification of the Dewilde-van der Veen method for inversion of finite structured matrices*, Linear Algebra Appl., 343–344 (2002), pp. 419–450.
[12] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[13] W. HACKBUSCH, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[14] W. HACKBUSCH AND S. BÖRM, *Data-sparse approximation by adaptive H^2-matrices*, Computing, 69 (2002), pp. 1–35.
[15] W. HACKBUSCH AND B. N. KHOROMSKIJ, *A sparse $\mathcal{H}$-matrix arithmetic, part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

ETNA
Kent State University
http://etna.math.kent.edu

[16] W. HACKBUSCH, B. N. KHOROMSKIJ, AND S. A. SAUTER, *On H^2-matrices*, in Lectures on Applied Mathematics, H. Bungartz and L. Horsten, eds., Springer-Verlag, Berlin, 2000, pp. 9–29.

[17] P. G. MARTINSSON AND V. ROKHLIN, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.

[18] P. G. MARTINSSON, V. ROKHLIN, AND M. TYGERT, *A fast algorithm for the inversion of general Toeplitz matrices*, Comput. Math. Appl., 50 (2005), pp. 741–752.

[19] X. SUN AND N. P. PITSIANIS, *A matrix version of the fast multipole method*, SIAM Rev., 43 (2001), pp. 289–300.

[20] E. E. TYRTYSHNIKOV, *Mosaic-skeleton approximations*, Calcolo, 33 (1996), pp. 47–58.

[21] ———, *Mosaic ranks and skeletons*, in Numerical Analysis and Its Applications, L. Vulkov, J. Wasniewski, and P. Y. Yalamov, eds., Lecture Notes in Computer Science, 1196, Springer-Verlag, 1997, pp. 505–516.