Construction of data-sparse $\mathcal{H}^2$-matrices by hierarchical compression

Steffen Börm*

October 14, 2007

Discretizing an integral operator by a standard finite element or boundary element method typically leads to a dense matrix. Since its storage complexity grows quadratically with the number of degrees of freedom, the standard representation of the matrix as a two-dimensional array cannot be applied to large problem sizes.

$\mathcal{H}^2$-matrix techniques use a multilevel approach to represent the dense matrix in a more efficient data-sparse format. We consider the challenging task of finding a good multilevel representation of the matrix without relying on a priori information of its contents.

This paper presents a relatively simple algorithm that can use any of the popular low-rank approximation schemes (e.g., cross approximation) to find an "initial guess" and constructs a matching multilevel structure on the fly. Numerical experiments show that the resulting technique is as fast as competing methods and requires far less storage for large problem dimensions.

Keywords: Hierarchical matrices, data-sparse approximation, non-local operators

AMS Subject Classification: 65F30, 65N38

Acknowledgement. A significant part of this work was carried out during a stay at the Institut für Geometrie und Praktische Mathematik of the RWTH Aachen.

1 Introduction

We consider numerical methods for solving integral equations. A typical example is the boundary integral equation

$$\int_\Gamma g(x,y)u(y)\,dy = f(x) \qquad \text{for all } x \in \Gamma,$$

*Max-Planck-Institut für Mathematik in den Naturwissenschaften, Inselstraße 22–26, 04103 Leipzig, Germany

where $\Gamma \subseteq \mathbb{R}^3$ is the boundary of a three-dimensional domain $\Omega \subseteq \mathbb{R}^3$, g is the *kernel function*, and f is a suitable right-hand side function.

If we approximate the solution u by a Galerkin scheme using finite element basis functions $(\varphi_i)_{i\in\mathcal{I}}$, we have to handle the matrix $G \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$ given by

$$G_{ij} = \int_\Gamma \varphi_i(x) \int_\Gamma g(x,y)\varphi_j(y)\,dy\,dx \qquad \text{for all } i,j \in \mathcal{I}. \tag{1}$$

Unless very special basis functions have been chosen, the matrix G is densely populated, i.e., almost all of its entries will be non-zero.

Storing G in the standard way, i.e., as a two-dimensional array, requires $\mathcal{O}(n^2)$ units of storage, where $n := \#\mathcal{I}$ is the number of basis functions used in the discretization scheme. In order to ensure a sufficient accuracy, we have to use a large number n of basis functions, therefore the storage requirements for the matrix G will quickly exceed the capacity of most of today's computers.

Therefore several techniques have been developed to reduce the storage requirements by replacing G with more efficient approximations $\widetilde{G}$. Wavelet techniques [3, 12, 11, 30, 27] employ special basis functions in order to ensure that most entries of the resulting Galerkin matrix G are almost zero, thus leading to a sparse approximation $\widetilde{G}$ of the matrix G. This approach works well if good wavelet bases are available and if the corresponding entries of $\widetilde{G}$ can be computed efficiently, so its application to complicated geometries and complicated integral operators can be a challenging task.

An alternative approach is followed by panel-clustering techniques [21, 25] and the closely related multipole expansion schemes [24, 16, 17, 23]: the domain $\Gamma\times\Gamma$ is split into subdomains $\tau\times\sigma$ such that $g|_{\tau\times\sigma}$ is sufficiently smooth, and on each of the subdomains a separable approximation

$$g(x,y) \approx \tilde{g}(x,y) := \sum_{\nu=1}^{k} v_\nu(x) w_\nu(y) \qquad \text{for all } x \in \tau, y \in \sigma,$$

of g is constructed, e.g., by Taylor or multipole expansions or interpolation. Since the variables x and y are separated in $\tilde{g}$, the double integral in (1) is replaced by products of single integrals that can be computed very efficiently.

Hierarchical (or short $\mathcal{H}$-) matrices [18, 19, 13, 15] are the algebraic counterparts of panel-clustering techniques: approximating g by a degenerate $\tilde{g}$ on subdomains $\tau\times\sigma$ corresponds to approximating submatrices $G|_{t\times s}$ of G by low-rank matrices. An $\mathcal{H}$-matrix is given by a partition of the matrix into a hierarchy of submatrices and factorized low-rank approximations of these submatrices.

Since low-rank approximations can also be constructed using singular value decompositions [18, 13, 8] or cross approximation techniques [28, 29, 2, 1, 7], $\mathcal{H}$-matrices are far more flexible than panel-clustering techniques. It is even possible to carry out approximate arithmetic operations like matrix inversion or factorization [15, 22] in almost linear complexity, therefore new applications like the evaluation of matrix functions, the solution of matrix equations or just the construction of very efficient preconditioners for linear systems can be treated efficiently by this approach.

Compared to multipole expansions, $\mathcal{H}$-matrices offer the advantage that optimal low-rank approximations (e.g., for inverses and preconditioners) can be constructed adaptively, while standard multipole techniques require specially-designed expansion systems to reach optimal efficiency, and these expansion systems are only available for a small number of standard situations.

Compared to wavelet methods, $\mathcal{H}$-matrices offer the advantage that their implementation is far easier and that they, like most schemes based on the approximation of g in the entire space, can handle complex geometries without additional effort.

In an $\mathcal{H}$-matrix, each submatrix uses a low-rank factorization that is independent of the factorizations of the other submatrices. This leads to relatively simple algorithms, but also to a loss of efficiency: if a special factorization is used that takes advantage of connections between a submatrix and its "neighbors", the storage complexity can be reduced significantly. This is the idea of $\mathcal{H}^2$-matrices [20, 9]: each submatrix is expressed as an element of a low-dimensional matrix space, and the different matrix spaces are embedded in a multilevel structure. A typical $\mathcal{H}$-matrix requires $\mathcal{O}(nk \log n)$ units of storage, while an $\mathcal{H}^2$-matrix needs only $\mathcal{O}(nk)$, where k determines the accuracy of the approximation. The additional logarithmic factor means that $\mathcal{H}^2$-matrix approximations will become more efficient compared to $\mathcal{H}$-matrices as the problem size n increases.

We are therefore interested in an algorithm that combines the simplicity of $\mathcal{H}$-matrix constructions with the higher efficiency of $\mathcal{H}^2$-matrices. One solution is presented in [9]: we can convert arbitrary matrices into $\mathcal{H}^2$-matrices, and the conversion procedure can be adapted to handle $\mathcal{H}$-matrices efficiently. Unfortunately, this approach requires the entire $\mathcal{H}$-matrix to be available to the algorithm, therefore its total storage requirements will be at least as high as for a standard $\mathcal{H}$-matrix.

This paper presents a new algorithm that avoids the necessity of storing the entire $\mathcal{H}$-matrix. It starts with independent initial low-rank approximations of submatrices, as used in standard $\mathcal{H}$-matrix techniques, that are recursively converted into larger and larger $\mathcal{H}^2$-submatrices until the entire matrix has been approximated. Since each submatrix is converted as soon as it becomes available, the storage requirements of the resulting algorithm are close to those of the final $\mathcal{H}^2$-matrix. The basic building block is a *unification step* that takes several independent $\mathcal{H}^2$-matrices and turns them into a unified $\mathcal{H}^2$-matrix.

The new approach is not limited to the approximation of matrices resulting from the discretization of integral operators. Using the unification step, many algorithms developed for $\mathcal{H}$-matrices can be "refitted" easily to construct the more efficient $\mathcal{H}^2$-matrices. An important field of application for this paradigm are adaptive matrix arithmetic operations: algorithms like the approximative $\mathcal{H}$-matrix multiplication or inversion [15] can now work with $\mathcal{H}^2$-matrices that require far less storage and thus allow us to fit much larger problems into a given amount of storage.

The paper is organized in five sections: section 1 is the introduction you are currently reading, section 2 defines the basic structure of $\mathcal{H}^2$-matrices, section 3 recalls the fundamental theory of $\mathcal{H}^2$-matrix compression and error control and derives the unification step, section 4 describes the new conversion algorithm, and section 5 contains numerical experiments that demonstrate its efficiency regarding time and storage requirements.

2 $\mathcal{H}^2$-matrices

We will now briefly recall the structure of $\mathcal{H}^2$-matrices [20, 9].

2.1 Block structure

Hierarchical matrix techniques are based on detecting subblocks of the matrix which admit a data-sparse approximation. In order to find these *admissible* blocks efficiently, we introduce a hierarchy of subsets:

Definition 2.1 (Cluster tree) *Let $\mathcal{I}$ be an index set. Let $\mathcal{T}$ be a labeled tree. We denote its root by* $\operatorname{root}(\mathcal{T})$*, the label of $t \in \mathcal{T}$ by $\hat{t}$, and the set of sons by* $\operatorname{sons}(\mathcal{T}, t)$ *(or just* $\operatorname{sons}(t)$ *if this does not lead to ambiguity).*

$\mathcal{T}$ is a cluster tree *for $\mathcal{I}$ if it satisfies the following conditions:*

- $\widehat{\operatorname{root}(\mathcal{T})} = \mathcal{I}$.
- *If* $\operatorname{sons}(t) \neq \emptyset$ *holds for $t \in \mathcal{T}$, we have*

$$\hat{t} = \bigcup_{s \in \operatorname{sons}(t)} \hat{s} \quad \text{and}$$
$$\hat{s}_1 \cap \hat{s}_2 = \emptyset \text{ for all } s_1, s_2 \in \operatorname{sons}(t) \text{ with } s_1 \neq s_2.$$

If $\mathcal{T}$ is a cluster tree for $\mathcal{I}$, we will denote it by $\mathcal{T}_\mathcal{I}$ and call its nodes clusters*. The set of* leaves *of $\mathcal{T}_\mathcal{I}$ is denoted by*

$$\mathcal{L}_\mathcal{I} := \{t \in \mathcal{T}_\mathcal{I} \ : \ \operatorname{sons}(t) = \emptyset\}.$$

The definition implies $\hat{t} \subseteq \mathcal{I}$ for all clusters $t \in \mathcal{T}_\mathcal{I}$. We can use induction to prove that the set $\mathcal{L}_\mathcal{I}$ of leaves of $\mathcal{T}_\mathcal{I}$ is a disjoint partition of the index set $\mathcal{I}$, i.e.,

$$\mathcal{I} = \dot{\bigcup}_{t \in \mathcal{L}_\mathcal{I}} \hat{t}. \tag{2}$$

Given a cluster tree $\mathcal{T}_\mathcal{I}$ and a cluster $t \in \mathcal{T}_\mathcal{I}$, we denote the subtree with root t by $\mathcal{T}_\mathcal{I}^t$ and call its nodes, including t itself, the *descendants* of t. If $t \in \mathcal{T}_\mathcal{I}$ is a descendant of $t^+ \in \mathcal{T}_\mathcal{I}$, the cluster t^+ is called a *predecessor* of t. The *set of predecessors* of a cluster $t \in \mathcal{T}_\mathcal{I}$ is defined by

$$\operatorname{pred}(t) := \{t^+ \in \mathcal{T}_\mathcal{I} \ : \ t \in \mathcal{T}_\mathcal{I}^{t^+}\}.$$

Using cluster trees, we can now define a hierarchical partition of the matrix entries:

Definition 2.2 (Block cluster tree) *Let $\mathcal{I}$, $\mathcal{J}$ be index sets, and let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be corresponding cluster trees. Let $\mathcal{T}$ be a labeled tree. $\mathcal{T}$ is a* block cluster tree *for $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ if it satisfies the following conditions:*

- $\operatorname{root}(\mathcal{T}) = (\operatorname{root}(\mathcal{T}_\mathcal{I}), \operatorname{root}(\mathcal{T}_\mathcal{J}))$.

- *Each cluster $b \in \mathcal{T}$ has the form $b = (t, s)$ for $t \in \mathcal{T}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J}$ and its label satisfies $\hat{b} = \hat{t} \times \hat{s}$.*
- *Let $b = (t, s) \in \mathcal{T}$. If* $\operatorname{sons}(b) \neq \emptyset$*, we have*

$$\operatorname{sons}(b) = \begin{cases} \{t\} \times \operatorname{sons}(s) & \text{if } \operatorname{sons}(t) = \emptyset, \operatorname{sons}(s) \neq \emptyset, \\ \operatorname{sons}(t) \times \{s\} & \text{if } \operatorname{sons}(t) \neq \emptyset, \operatorname{sons}(s) = \emptyset \\ \operatorname{sons}(t) \times \operatorname{sons}(s) & \text{otherwise.} \end{cases}$$

If $\mathcal{T}$ is a block cluster tree for $\mathcal{I}$ and $\mathcal{J}$, we will denote it by $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and call its nodes blocks. *The leaves of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ are denoted by $\mathcal{L}_{\mathcal{I}\times\mathcal{J}}$.*

This definition implies that a block cluster tree for $\mathcal{I}$ and $\mathcal{J}$ is a cluster tree for the product index set $\mathcal{I} \times \mathcal{J}$, therefore the set of leaf labels

$$P := \{\hat{b} = \hat{t} \times \hat{s} \ : \ b = (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}\}$$

defines a disjoint partition of $\mathcal{I} \times \mathcal{J}$ into blocks of indices.

Definition 2.3 (Admissibility) *Let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be cluster trees for $\mathcal{I}$ and $\mathcal{J}$. Let $\alpha : \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{J} \to \mathbb{B} = \{true, false\}$ be a predicate, which we will call* admissibility condition. *A block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is called* admissible *if*

$$(\operatorname{sons}(\mathcal{T}_\mathcal{I}, t) \neq \emptyset \vee \operatorname{sons}(\mathcal{T}_\mathcal{J}, s) \neq \emptyset) \Longrightarrow \alpha(t, s) \qquad \text{holds for all } b = (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}},$$

i.e., if each leaf of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is either admissible or a pair of leaf clusters of the respective cluster trees. For an admissible block cluster tree, we split the set of leaves into

$$\mathcal{L}^+_{\mathcal{I}\times\mathcal{J}} := \{b = (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}} \ : \ \alpha(t, s) \text{ holds}\} \quad \text{and} \quad \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}} := \mathcal{L}_{\mathcal{I}\times\mathcal{J}} \setminus \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}},$$

i.e., into admissible and inadmissible leaves. Usually, we will not work with α, but only use $\mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ and $\mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$.

For matrices resulting from the discretization of elliptic problems, the admissibility condition

$$\alpha(t, s) := \begin{cases} \text{true} & \text{if } \max\{\operatorname{diam}(\Omega_t), \operatorname{diam}(\Omega_s)\} \leq \operatorname{dist}(\Omega_t, \Omega_s), \\ \text{false} & \text{otherwise} \end{cases} \tag{3}$$

is frequently used, where Ω_t and Ω_s are suitable domains containing the supports of the basis functions or functionals corresponding to t and s.

The condition (3) ensures that we are dealing with a region where we can expect Green's function to be smooth or at least separable. In the case $\mathcal{I} = \mathcal{J}$, this means that the block $\hat{t} \times \hat{s}$ lies "sufficiently far away" from the diagonal of the matrix.

If the indices in $\mathcal{I}$ and $\mathcal{J}$ correspond to locations in space, it is possible to construct good cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ by binary space partitioning and a good block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ by a simple recursion strategy [13, 15].

2.2 Matrix structure

Typical hierarchical matrices are defined based on the block partition P: given a *local rank* $k \in \mathbb{N}$, the submatrices $M|_{\hat{t}\times\hat{s}}$ corresponding to admissible leaf blocks $b = (t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ are required to be of low rank:

$$\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k) := \{M \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}} \ : \ \operatorname{rank}(M|_{\hat{t}\times\hat{s}}) \leq k \text{ for all } b = (t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}\}.$$

The parameter k is called the *local rank* of the $\mathcal{H}$-matrix set. For each $b = (t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$, the low-rank matrix $M|_{\hat{t}\times\hat{s}}$ is represented in factorized form, i.e., by matrices $A_b \in \mathbb{R}^{\hat{t}\times k}$ and $B_b \in \mathbb{R}^{\hat{s}\times k}$ with $M|_{\hat{t}\times\hat{s}} = A_b B_b^\top$.

The $\mathcal{H}^2$-matrix format is a specialization of this representation: we require not only that admissible blocks correspond to low-rank matrix blocks, but also that the ranges of these blocks and their adjoints are contained in predefined spaces.

Definition 2.4 (Cluster basis) *Let $\mathcal{T}_\mathcal{I}$ be a cluster tree, and let $K = (K_t)_{t\in\mathcal{T}_\mathcal{I}}$ be a family of index sets. A family $V = (V_t)_{t\in\mathcal{T}_\mathcal{I}}$ of matrices satisfying $V_t \in \mathbb{R}^{\hat{t}\times K_t}$ for all $t \in \mathcal{T}_\mathcal{I}$ is called* cluster basis *for $\mathcal{T}_\mathcal{I}$ and K. For each $t \in \mathcal{T}_\mathcal{I}$, the cardinality $\#K_t$ is called the* rank *of V in t.*

In the literature, the matrices V_t are sometimes embedded in $\mathbb{R}^{\mathcal{I}\times K_t}$ by extending them by zero. Obviously, both notations are equivalent.

The high efficiency of $\mathcal{H}^2$-matrix methods is owed to the fact that the cluster bases are designed to form a nested hierarchy matching the cluster tree:

Definition 2.5 (Nested cluster basis) *Let $V = (V_t)_{t\in\mathcal{T}_\mathcal{I}}$ be a cluster basis for a cluster tree $\mathcal{T}_\mathcal{I}$ and a family $(K_t)_{t\in\mathcal{T}_\mathcal{I}}$ of index sets. Let $E = (E_t)_{t\in\mathcal{T}_\mathcal{I}}$ be a family of matrices satisfying $E_{t'} \in \mathbb{R}^{K_{t'}\times K_t}$ and*

$$(V_t)|_{\hat{t}'\times K_t} = V_{t'} E_{t'} \tag{4}$$

for all $t \in \mathcal{T}_\mathcal{I}$ and all $t' \in \operatorname{sons}(t)$. Then the cluster basis V is called nested *with* transfer matrices *E.*

The case $t = \operatorname{root}(\mathcal{T}_\mathcal{I})$ is only included in order to avoid the necessity of treating a special case: we can see that the definition does not require the transfer matrix for the root of $\mathcal{T}_\mathcal{I}$ to satisfy any conditions. In practice, this matrix can be ignored completely.

If a cluster basis $V = (V_t)_{t\in\mathcal{T}_\mathcal{I}}$ is nested, we have

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ \vdots \\ V_{t_\tau} E_{t_\tau} \end{pmatrix} = \begin{pmatrix} V_{t_1} & & \\ & \ddots & \\ & & V_{t_\tau} \end{pmatrix} \begin{pmatrix} E_{t_1} \\ \vdots \\ E_{t_\tau} \end{pmatrix}$$

for all $t \in \mathcal{T}_\mathcal{I}$ with $\#\operatorname{sons}(t) = \tau > 0$ and $\operatorname{sons}(t) = \{t_1, \ldots, t_\tau\}$. This means that we have to store the matrices V_t only for leaf clusters $t \in \mathcal{L}_\mathcal{I}$ and can use the transfer matrices $E_{t'}$ to represent them *implicitly* for all other clusters. Since the transfer matrices $E_{t'}$

only require $(\#K_{t'})(\#K_t)$ units of storage, while the cluster basis matrices V_t require $(\#\hat{t})(\#K_t)$ units, this nested representation is very efficient for $\#\hat{t} \gg \#K_{t'}$.

The nested structure is the key difference between general hierarchical matrices and $\mathcal{H}^2$-matrices [20, 9], since it allows us to construct very efficient algorithms by re-using information across the entire cluster tree, similar to multigrid algorithms or other multilevel methods.

Definition 2.6 ($\mathcal{H}^2$-matrix) *Let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be cluster trees. Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be an admissible block cluster tree. Let V and W be nested cluster bases for $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ with families $K = (K_t)_{t\in\mathcal{T}_\mathcal{I}}$ and $L = (L_s)_{s\in\mathcal{T}_\mathcal{J}}$ of index sets. The set of $\mathcal{H}^2$-matrices for $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, V and W is given by*

$$\mathcal{H}^2(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, V, W) := \{X \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}} \ : \text{for all } b = (t,s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}} \text{ there is a matrix } S_b \in \mathbb{R}^{K_t\times L_s} \text{ satisfying } X|_{\hat{t}\times\hat{s}} = V_t S_b W_s^\top\}$$

In this context, V is called the row cluster basis*, W is called the* column cluster basis*, and the family $S = (S_b)_{b\in\mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}}$ is called the family of* coupling matrices*.*

Obviously, we have $\mathcal{H}^2(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, V, W) \subseteq \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ for the local rank

$$k := \max\{\min\{\#K_t, \#L_s\} \ : \ b = (t,s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}\},$$

i.e., the $\mathcal{H}$-matrix structure seems to be more flexible at first glance. A closer investigation [6] shows that in the most important application fields, $\mathcal{H}^2$-matrices can perform even better than $\mathcal{H}$-matrices if the cluster bases V and W are chosen correctly.

2.3 Complexity

Let us now consider the storage complexity of the $\mathcal{H}^2$-matrix representation.

Block cluster trees constructed for standard situations have an important property: for each $t \in \mathcal{T}_\mathcal{I}$, there is only a limited number of blocks of the form (t,s), i.e., the cardinalities of the sets

$$\operatorname{row}(t) := \{s \in \mathcal{T}_\mathcal{J} \ : \ (t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}\}, \qquad \operatorname{col}(s) := \{t \in \mathcal{T}_\mathcal{I} \ : \ (t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}\}$$

can be bounded by a constant. For cluster trees and block cluster trees constructed by geometric bisection, an explicit bound can be given, and this bound does not depend on the number of degrees of freedom [13, 15].

Definition 2.7 (Sparsity) *Let $C_{\mathrm{sp}} \in \mathbb{N}$. The block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is* C_{sp}-sparse *if we have*

$$\#\operatorname{row}(t) = \#\{s \in \mathcal{T}_\mathcal{J} \ : \ (t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}\} \leq C_{\mathrm{sp}} \qquad \text{for all } t \in \mathcal{T}_\mathcal{I}, \tag{5a}$$

$$\#\operatorname{col}(s) = \#\{t \in \mathcal{T}_\mathcal{I} \ : \ (t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}\} \leq C_{\mathrm{sp}} \qquad \text{for all } s \in \mathcal{T}_\mathcal{J}. \tag{5b}$$

The complexity of an $\mathcal{H}^2$-matrix representation can be bounded if the following conditions are fulfilled:

- the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is admissible and C_{sp}-sparse,
- each cluster has only a bounded number of sons, i.e., there is a constant $C_{\mathrm{sn}} \geq 1$ with

$$\#\operatorname{sons}(t) \leq C_{\mathrm{sn}}, \qquad \#\operatorname{sons}(s) \leq C_{\mathrm{sn}} \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}, s \in \mathcal{T}_{\mathcal{J}}, \tag{6}$$

- each leaf cluster is not too large, i.e., there is a constant $C_{\mathrm{lf}} \geq 1$ satisfying

$$\#\hat{t} \leq C_{\mathrm{lf}} \# K_t, \qquad \#\hat{s} \leq C_{\mathrm{lf}} \# L_s \qquad \text{for all leaves } t \in \mathcal{L}_{\mathcal{I}}, s \in \mathcal{L}_{\mathcal{J}}. \tag{7}$$

To keep the notations short, we introduce the abbreviations

$$k_t := \#K_t, \qquad l_s := \#L_s \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}, s \in \mathcal{T}_{\mathcal{J}}.$$

Let us first consider the storage complexity of the cluster bases V and W.

Lemma 2.8 (Storage of a cluster basis) *A cluster basis $V = (V_t)_{t\in\mathcal{T}_{\mathcal{I}}}$, represented by transfer matrices for all $t \in \mathcal{T}_{\mathcal{I}} \setminus \mathcal{L}_{\mathcal{I}}$, requires not more than*

$$(C_{\mathrm{lf}} + C_{\mathrm{sn}}^{1/2}) \sum_{t\in\mathcal{T}_{\mathcal{I}}} k_t^2 \quad \textit{units of storage.}$$

Proof. For each leaf cluster $t \in \mathcal{T}_{\mathcal{I}}$, we have to store the matrix V_t, which requires $(\#\hat{t})(\#K_t)$ units of storage. Due to (7), this is bounded by $C_{\mathrm{lf}}(\#K_t)^2 \leq C_{\mathrm{lf}} k_t^2$, and *all* of these matrices require not more than

$$C_{\mathrm{lf}} \sum_{t\in\mathcal{L}_{\mathcal{I}}} k_t^2 \leq C_{\mathrm{lf}} \sum_{t\in\mathcal{T}_{\mathcal{I}}} k_t^2 \quad \text{units of storage.}$$

For each non-leaf cluster $t \in \mathcal{T}_{\mathcal{I}}$, we have to store the transfer matrices $E_{t'}$ for all $t' \in \operatorname{sons}(t)$, which requires $(\#K_{t'})(\#K_t) = k_{t'}k_t$ units of storage and a total of

$$\begin{aligned}\sum_{t\in\mathcal{T}_{\mathcal{I}}} \sum_{t'\in\operatorname{sons}(t)} k_t k_{t'} &\leq \left(\sum_{t\in\mathcal{T}_{\mathcal{I}}} \sum_{t'\in\operatorname{sons}(t)} k_t^2\right)^{1/2} \left(\sum_{t\in\mathcal{T}_{\mathcal{I}}} \sum_{t'\in\operatorname{sons}(t)} k_{t'}^2\right)^{1/2} \\ &\overset{(6)}{\leq} \left(C_{\mathrm{sn}} \sum_{t\in\mathcal{T}_{\mathcal{I}}} k_t^2\right)^{1/2} \left(\sum_{t'\in\mathcal{T}_{\mathcal{I}}} k_{t'}^2\right)^{1/2} = C_{\mathrm{sn}}^{1/2} \sum_{t\in\mathcal{T}_{\mathcal{I}}} k_t^2,\end{aligned}$$

where we have used the Cauchy-Schwarz inequality and the fact that each cluster has at most one father. ∎

Let us now consider the storage requirements of the coupling matrices and the nearfield part of an $\mathcal{H}^2$-matrix.

Lemma 2.9 (Storage of matrices) *The coupling matrices* $(S_b)_{b\in\mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}}$ *and the near-field matrices* $(X|_{\hat{b}})_{b\in\mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}}$ *for an* $\mathcal{H}^2$*-matrix* X *require not more than*

$$\frac{C_{\mathrm{lf}}^2 C_{\mathrm{sp}}}{2}\left(\sum_{t\in\mathcal{T}_\mathcal{I}} k_t^2 + \sum_{s\in\mathcal{T}_\mathcal{J}} l_s^2\right) \quad \textit{units of storage.}$$

Proof. Let $b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J}}$. If b is admissible, the corresponding block is represented by the matrix S_b, which requires $(\#K_t)(\#L_s)$ units of storage. If b is not admissible, the admissibility of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ implies that t and s have to be leaves of $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$, respectively, and due to (7), the matrix $M|_{\hat{t}\times\hat{s}}$ requires only $(\#\hat{t})(\#\hat{s}) \leq C_{\mathrm{lf}}^2(\#K_t)(\#L_s)$ units of storage. In both cases not more than $C_{\mathrm{lf}}^2(\#K_t)(\#L_s) = C_{\mathrm{lf}}^2 k_t l_s$ units of storage are needed, and we get the bound

$$\begin{aligned}
C_{\mathrm{lf}}^2 \sum_{b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J}}} k_t l_s &\leq C_{\mathrm{lf}}^2 \left(\sum_{b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J}}} k_t^2\right)^{1/2} \left(\sum_{b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J}}} l_s^2\right)^{1/2} \\
&\leq C_{\mathrm{lf}}^2 \left(\sum_{t\in\mathcal{T}_\mathcal{I}} \sum_{s\in\mathrm{row}(t)} k_t^2\right)^{1/2} \left(\sum_{s\in\mathcal{T}_\mathcal{J}} \sum_{t\in\mathrm{col}(s)} l_s^2\right)^{1/2} \\
&\overset{(5)}{\leq} C_{\mathrm{lf}}^2 \left(C_{\mathrm{sp}} \sum_{t\in\mathcal{T}_\mathcal{I}} k_t^2\right)^{1/2} \left(C_{\mathrm{sp}} \sum_{s\in\mathcal{T}_\mathcal{J}} l_s^2\right)^{1/2} \\
&\leq \frac{C_{\mathrm{lf}}^2 C_{\mathrm{sp}}}{2}\left(\sum_{t\in\mathcal{T}_\mathcal{I}} k_t^2 + \sum_{s\in\mathcal{T}_\mathcal{J}} l_s^2\right)
\end{aligned}$$

for the total storage complexity. ∎

Adding the estimates of Lemma 2.8 and Lemma 2.9 yields an upper bound of

$$\left(\frac{C_{\mathrm{lf}}^2 C_{\mathrm{sp}}}{2} + 2C_{\mathrm{lf}} + 2C_{\mathrm{sn}}^{1/2}\right)\left(\sum_{t\in\mathcal{T}_\mathcal{I}} k_t^2 + \sum_{s\in\mathcal{T}_\mathcal{J}} l_s^2\right) \tag{8}$$

for the storage requirements of an $\mathcal{H}^2$-matrix.

In simple situations, we can assume that the ranks of the matrices V_t and W_s are bounded by a constant $k\in\mathbb{N}$, i.e., that

$$\#K_t \leq k, \qquad \#L_s \leq k, \qquad \text{holds for all } t\in\mathcal{T}_\mathcal{I}, s\in\mathcal{T}_\mathcal{J}.$$

This leads to an estimate of $\mathcal{O}((\#\mathcal{T}_\mathcal{I} + \#\mathcal{T}_\mathcal{J})k^2)$ for the storage requirements.

In practical applications, the cluster trees are constructed in such a way that $\#\mathcal{T}_\mathcal{I} \lesssim n/k$ and $\#\mathcal{T}_\mathcal{J} \lesssim n/k$ hold for

$$n := \max\{\#\mathcal{I}, \#\mathcal{J}\},$$

therefore an $\mathcal{H}^2$-matrix representation of X requires only $\mathcal{O}(nk)$ units of storage.

In the general case, we can take advantage of the fact that the estimate (8) depends only on the sums of k_t^2 and l_s^2 and not on the maximum k, therefore we can admit higher ranks for a small number of clusters and still get a low complexity. These variable-rank methods [25, 4] can reach the *optimal* complexity of $\mathcal{O}(n)$ while keeping the matrix error consistent with the discretization error.

3 Construction of cluster bases

Our goal is to approximate an arbitrary matrix $X \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ by an $\mathcal{H}^2$-matrix $\widetilde{X}$. We assume that the cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ and the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ are given.

3.1 Matrix approximation

If the row and column cluster bases V and W are also given, it is reasonable to wonder if we can compute the best approximation of X in the matrix space $\mathcal{H}^2(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, V, W)$.

Definition 3.1 (Orthogonal cluster basis) *Let $V = (V_t)_{t\in\mathcal{T}_\mathcal{I}}$ be a cluster basis. It is called* orthogonal *if*

$$V_t^\top V_t = I \qquad \text{holds for all } t \in \mathcal{T}_\mathcal{I}. \tag{9}$$

If V and W are orthogonal cluster bases, the computation of the least-squares approximation of X in $\mathcal{H}^2(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, V, W)$ is straightforward: for an admissible block $b = (t,s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$, we let $X_{t,s} := X|_{\hat{t}\times\hat{s}}$ and look for a good approximation of this block in the factorized form $V_t S_b W_s^\top$ used by $\mathcal{H}^2$-matrices. We let $S_b^* := V_t^\top X_{t,s} W_s$ and observe that the orthogonality of V_t and W_s implies

$$\begin{aligned}
\|X_{t,s} - V_t\widetilde{S}_b W_s^\top\|_F^2 &= \|X_{t,s} - V_t S_b^* W_s + V_t(S_b^* - \widetilde{S}_b)W_s^\top\|_F^2 \\
&= \|X_{t,s} - V_t S_b^* W_s\|_F^2 + \|V_t(S_b^* - \widetilde{S}_b)W_s^\top\|_F^2 \\
&= \|X_{t,s} - V_t S_b^* W_s\|_F^2 + \|S_b^* - \widetilde{S}_b\|_F^2 \quad \text{for all } \widetilde{S}_b \in \mathbb{R}^{K_t\times L_s},
\end{aligned}$$

i.e., S_b^* minimizes the blockwise approximation error. For the Frobenius norm, the sum of the squares of the blockwise approximation errors yields the square of the total approximation error, therefore the $\mathcal{H}^2$-matrix $\widetilde{X}$ defined by $\widetilde{X}|_{\hat{t}\times\hat{s}} = V_t S_b^* W_s^\top$ for all $b \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ is indeed the best approximation of X in the space $\mathcal{H}^2(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, V, W)$ with respect to the Frobenius norm. Using

$$\begin{aligned}
\|X_{t,s} - V_t S_b^* W_s^\top\|_F^2 &= \|X_{t,s} - V_t V_t^\top X_{t,s} + V_t V_t^\top X_{t,s} - V_t V_t^\top X_{t,s} W_s W_s^\top\|_F^2 \\
&= \|X_{t,s} - V_t V_t^\top X_{t,s}\|_F^2 + \|V_t V_t^\top (X_{t,s} - X_{t,s} W_s W_s^\top)\|_F^2 \\
&\leq \|X_{t,s} - V_t V_t^\top X_{t,s}\|_F^2 + \|X_{t,s} - X_{t,s} W_s W_s^\top\|_F^2,
\end{aligned} \tag{10}$$

the influence of V and W can be investigated independently.

We conclude that finding the matrix $\widetilde{X}$ is a relatively simple task once we have good orthogonal cluster bases at our disposal, and that orthogonal row and column cluster bases V and W can be considered "good" if

$$\|X_{t,s} - V_t V_t^\top X_{t,s}\|_F \leq \epsilon, \quad \|X_{t,s}^\top - W_s W_s^\top X_{t,s}^\top\|_F \leq \epsilon \qquad \text{for all } b = (t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}^+$$

hold for a given error tolerance $\epsilon \in \mathbb{R}_{>0}$.

Due to the symmetry of these inequalities, we can restrict our attention to the construction of a good row cluster basis V, since applying the same technique to the transposed matrix $X^\top$ will then yield a good column cluster basis W.

In practical applications, the Frobenius norm is usually only of little interest, more important are induced matrix norms like the spectral norm. The theory presented here carries over to this case [4].

3.2 Orthogonalization

Let us now consider the construction of "good" cluster bases. In a first step, we assume that a nested cluster basis $V = (V_t)_{t\in\mathcal{T}_\mathcal{I}}$ for the cluster tree $\mathcal{T}_\mathcal{I}$ and the index sets $K = (K_t)_{t\in\mathcal{T}_\mathcal{I}}$ is given that satisfies

$$\min_{Z_b \in \mathbb{R}^{K_t \times \hat{s}}} \|X_{t,s} - V_t Z_b\|_F \leq \epsilon \qquad \text{for all } b = (t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}^+. \tag{11}$$

We are looking for an *orthogonal* nested cluster basis $Q = (Q_t)_{t\in\mathcal{T}_\mathcal{I}}$ for the same cluster tree and the new index sets $(\widetilde{K}_t)_{t\in\mathcal{T}_\mathcal{I}}$ that satisfies

$$\|X_{t,s} - Q_t Q_t^\top X_{t,s}\|_F \leq \epsilon \qquad \text{for all } b = (t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}^+,$$

i.e., that allows us to compute the best approximation of $X_{t,s}$ explicitly by an orthogonal projection.

In general, the latter inequality will only hold if the range of V_t is contained in the range of Q_t, i.e., if there is a matrix $R_t \in \mathbb{R}^{\widetilde{K}_t \times K_t}$ such that $V_t = Q_t R_t$ holds.

This equation already suggests a good approach for an algorithm: we are looking for orthogonal factorizations of the matrices V_t, and we have the additional requirement that the new cluster basis $Q = (Q_t)_{t\in\mathcal{T}_\mathcal{I}}$ has to be nested. To illustrate the idea, let us consider a cluster $t \in \mathcal{T}_\mathcal{I}$ with $\operatorname{sons}(t) = \{t_1, t_2\}$ for $t_1 \neq t_2$. Since V is nested, there are transfer matrices E_{t_1} and E_{t_2} such that

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}$$

holds. We assume that the factorizations $V_{t_1} = Q_{t_1} R_{t_1}$ and $V_{t_2} = Q_{t_2} R_{t_2}$ and the corresponding index sets $\widetilde{K}_{t_1}$ and $\widetilde{K}_{t_2}$ have already been computed. We also assume that the $\widetilde{K}_{t_1} \subseteq \hat{t}_1$ and $\widetilde{K}_{t_2} \subseteq \hat{t}_2$ hold, since this implies $\widetilde{K}_{t_1} \cap \widetilde{K}_{t_2} = \emptyset$ and allows us to avoid formal problems in the construction of the matrix $\widehat{V}_t$ below. We get

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix} = \begin{pmatrix} Q_{t_1} R_{t_1} E_{t_1} \\ Q_{t_2} R_{t_2} E_{t_2} \end{pmatrix} = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \begin{pmatrix} R_{t_1} E_{t_1} \\ R_{t_2} E_{t_2} \end{pmatrix} = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{V}_t$$

for the auxiliary matrix

$$\widehat{V}_t := \begin{pmatrix} R_{t_1} E_{t_1} \\ R_{t_2} E_{t_2} \end{pmatrix} \in \mathbb{R}^{\widehat{K}_t \times K_t}, \qquad \widehat{K}_t := \widetilde{K}_{t_1} \dot{\cup} \widetilde{K}_{t_2}.$$

We compute the factorization

$$\widehat{V}_t = \widehat{Q}_t R_t$$

with an orthogonal matrix $\widehat{Q}_t \in \mathbb{R}^{\widehat{K}_t \times \widetilde{K}_t}$ and a matrix $R_t \in \mathbb{R}^{\widetilde{K}_t \times K_t}$ using Givens or Householder transformations. The index set $\widetilde{K}_t$ is chosen as a subset of $\widehat{K}_t \subseteq \hat{t}$ of cardinality $\min\{\#\widehat{K}_t, \#K_t\}$. The new cluster basis matrix Q_t is defined by

$$Q_t := \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t,$$

and it is obviously orthogonal. Due to $\widehat{K}_t = \widetilde{K}_{t_1} \dot{\cup} \widetilde{K}_{t_2}$, we can split $\widehat{Q}_t$ into its "upper half" $F_{t_1} := \widehat{Q}_t|_{\widetilde{K}_{t_1} \times \widetilde{K}_t}$ and its "lower half" $F_{t_2} := \widehat{Q}_t|_{\widetilde{K}_{t_2} \times \widetilde{K}_t}$ and observe

$$\widehat{Q}_t = \begin{pmatrix} F_{t_1} \\ F_{t_2} \end{pmatrix}, \qquad Q_t = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \begin{pmatrix} F_{t_1} \\ F_{t_2} \end{pmatrix} = \begin{pmatrix} Q_{t_1} F_{t_1} \\ Q_{t_2} F_{t_2} \end{pmatrix},$$

i.e., the cluster basis $Q = (Q_t)_{t \in \mathcal{T}_\mathcal{I}}$ is nested (cf. Definition 2.5), and the transfer matrices are given by F_t. Due to our construction, we have

$$V_t = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{V}_t = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t R_t = Q_t R_t,$$

i.e., the matrices Q_t and R_t indeed form an orthogonal decomposition of the matrix V_t.

If t is a leaf, the situation is simple: we let $\widehat{K}_t := \hat{t}$, $\widehat{V}_t := V_t$ and $Q_t := \widehat{Q}_t$, i.e., we just compute a standard orthogonal factorization of V_t by Householder or Givens transformations.

3.3 Truncation

The complexity of subsequent computations with the new cluster basis Q is determined by the cardinalities of the index sets $\widetilde{K} = (\widetilde{K}_t)_{t \in \mathcal{T}_\mathcal{I}}$, therefore we would like these sets to be as small as possible. The orthogonalization procedure guarantees $\#\widetilde{K}_t \leq \#K_t$ for all $t \in \mathcal{T}_\mathcal{I}$, i.e., the new cluster basis will at least not be less efficient than the original one, but this is not necessarily the optimal result.

In order to improve efficiency, we replace the exact factorization $V_t = Q_t R_t$ by an *approximate* factorization: we are looking for an orthogonal nested cluster basis $Q = (Q_t)_{t \in \mathcal{T}_\mathcal{I}}$ such that $V_t \approx Q_t R_t$ holds for the optimal coefficient matrices $R_t := Q_t^\top V_t$. We aim to choose the cardinalities $\#\widetilde{K}_t$ as small as possible given the desired accuracy of the approximation.

This goal is reached by the *truncation* algorithm: we replace the exact factorization $\widehat{V}_t = \widehat{Q}_t R_t$ used in the orthogonalization algorithm by an approximate factorization

Algorithm 1 Given $X \in \mathbb{R}^{M\times N}$, construct an index set $K \subseteq M$ of minimal cardinality and orthogonal $Q \in \mathbb{R}^{M\times K}$ such that $\|X - QQ^\top X\|_F \leq \epsilon$

procedure Lowrank(X, ϵ, **var** Q, K);
$m \leftarrow \#M$; $\{\mu_1, \ldots, \mu_m\} \leftarrow M$; $n \leftarrow \#N$; $\{\nu_1, \ldots, \nu_n\} \leftarrow N$;
$\widehat{X} \leftarrow 0 \in \mathbb{R}^{m\times n}$;
for $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ **do** $\widehat{X}_{ij} \leftarrow X_{\mu_i\nu_j}$;
Compute singular value decomposition $\widehat{X} = U\Sigma V^\top$;
$k \leftarrow \min\{m, n\}$; $\tilde{\epsilon} \leftarrow 0$;
while $k > 0$ and $\tilde{\epsilon} + \Sigma_{kk}^2 \leq \epsilon^2$ **do begin** $k \leftarrow k - 1$; $\tilde{\epsilon} \leftarrow \tilde{\epsilon} + \Sigma_{kk}^2$ **end**;
$K \leftarrow \{\mu_1, \ldots, \mu_k\}$; $Q \leftarrow 0 \in \mathbb{R}^{M\times K}$;
for $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, k\}$ **do** $Q_{\mu_i\mu_j} \leftarrow U_{ij}$

$\widehat{V}_t \approx \widehat{Q}_t R_t$, e.g., by computing the singular value decomposition of $\widehat{V}_t$ and dropping all singular values below a given error tolerance (cf. Algorithm 1).

In this setting we have

$$V_t = \begin{pmatrix} V_{t_1}E_{t_1} \\ V_{t_2}E_{t_2} \end{pmatrix} \approx \begin{pmatrix} Q_{t_1}R_{t_1}E_{t_1} \\ Q_{t_2}R_{t_2}E_{t_2} \end{pmatrix} = \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{V}_t \approx \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} \widehat{Q}_t R_t = Q_t R_t,$$

i.e., the approximation error $V_t - Q_t R_t$ is the sum of two errors: the first one is caused by the approximation of V_{t_1} and V_{t_2}, the second one is caused by the approximation of $\widehat{V}_t$. Using the orthogonality of Q_{t_1} and Q_{t_2} yields

$$\begin{aligned} \|(V_t - Q_t R_t)x\|_2^2 &= \left\| \begin{pmatrix} (V_{t_1} - Q_{t_1}R_{t_1})E_{t_1}x \\ (V_{t_2} - Q_{t_2}R_{t_2})E_{t_2}x \end{pmatrix} + \begin{pmatrix} Q_{t_1} & \\ & Q_{t_2} \end{pmatrix} (\widehat{V}_t - \widehat{Q}_t R_t)x \right\|_2^2 \\ &= \|(V_{t_1} - Q_{t_1}R_{t_1})E_{t_1}x\|_2^2 + \|(V_{t_2} - Q_{t_2}R_{t_2})E_{t_2}x\|_2^2 + \|(\widehat{V}_t - \widehat{Q}_t R_t)x\|_2^2 \end{aligned}$$

for all vectors $x \in \mathbb{R}^{K_t}$. The first two terms of this sum are of the same structure as the original term, only with the vectors $E_{t_1}x$ and $E_{t_2}x$ instead of x. Using the *long-range transfer matrices* defined by

$$E_{r,t} := \begin{cases} E_{r,t'}E_{t'} & \text{if } r \in \mathcal{T}_{\mathcal{I}}^{t'} \text{ for a } t' \in \operatorname{sons}(t), \\ I & \text{otherwise, i.e., if } r = t \end{cases} \qquad \text{for } t \in \mathcal{T}_{\mathcal{I}}, r \in \mathcal{T}_{\mathcal{I}}^t,$$

we can proceed by induction (cf. [5, Theorem 4] for a detailed general proof) to get

$$\|(V_t - Q_t R_t)x\|_2^2 = \sum_{r\in\mathcal{T}_{\mathcal{I}}^t} \|(\widehat{V}_r - \widehat{Q}_r R_r)E_{r,t}x\|_2^2. \tag{12}$$

This equation relates the total error to the local errors introduced in each step of the algorithm, therefore it can be used to implement sophisticated error-control schemes. Applying it to canonical unit vectors and summing up yields

$$\|V_t - Q_t R_t\|_F^2 = \sum_{r\in\mathcal{T}_{\mathcal{I}}^t} \|(\widehat{V}_r - \widehat{Q}_r R_r)E_{r,t}\|_F^2, \tag{13}$$

while taking the supremum of (12) yields a similar estimate for the spectral norm.

3.4 Matrix compression

Let us now return our attention to the problem of finding a "good" nested cluster basis for an arbitrary matrix $X \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$.

Let $b = (t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ be an admissible block. According to (10), we have to find an orthogonal nested cluster basis $Q = (Q_t)_{t\in\mathcal{T}_\mathcal{I}}$ such that

$$\|X_{t,s} - Q_t Q_t^\top X_{t,s}\|_F \leq \epsilon$$

holds for a given error tolerance $\epsilon \in \mathbb{R}_{>0}$.

We approach this problem by using the concept of the *total cluster basis* introduced in [6]: if we collect the relevant matrix blocks $X_{t,s}$ in a large matrix X_t with

$$X_{t,s} = X_t|_{\hat{t}\times\hat{s}} \tag{14}$$

and find an algorithm that guarantees

$$\|X_t - Q_t Q_t^\top X_t\|_F \leq \epsilon,$$

this would immediately imply

$$\|X_{t,s} - Q_t Q_t^\top X_{t,s}\|_F = \|(X_t - Q_t Q_t^\top X_t)|_{\hat{t}\times\hat{s}}\|_F \leq \|X_t - Q_t Q_t^\top X_t\|_F \leq \epsilon.$$

We choose the matrices X_t in such a way that they not only satisfy (14), but also form a nested cluster basis: we let

$$X_t := X|_{\hat{t}\times N_t}, \qquad N_t := \dot{\bigcup}\{\hat{s} \;:\; \text{with } (t^+, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}} \text{ for } t^+ \in \operatorname{pred}(t), s \in \mathcal{T}_\mathcal{J}\}$$

for all clusters $t \in \mathcal{T}_\mathcal{I}$. Obviously $(X_t)_{t\in\mathcal{T}_\mathcal{I}}$ is a cluster basis with the index sets $(N_t)_{t\in\mathcal{T}_\mathcal{I}}$, and since $N_{t'} \subseteq N_t$ holds for all $t \in \mathcal{T}_\mathcal{I}$, $t' \in \operatorname{sons}(t)$, this cluster basis is also nested with trivial transfer matrices. For all $b = (t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$, we have $\hat{s} \subseteq N_t$ and therefore $X_{t,s} = X_t|_{\hat{t}\times\hat{s}}$.

This means that the *total cluster basis* $(X_t)_{t\in\mathcal{T}_\mathcal{I}}$ can be used to represent each admissible block of X with zero error. The ranks of the total cluster basis are usually too large (on the order of $\#\mathcal{J}$) for useful algorithms, but we already know how to fix this: we apply the truncation algorithm to the total cluster basis. Due to the simplicity of its transfer matrices, the error estimate (13) takes the form

$$\|X_t - Q_t R_t\|_F^2 = \sum_{r\in\mathcal{T}_\mathcal{I}^t} \|(\widehat{X}_r - \widehat{Q}_r R_r)|_{\widehat{K}_r\times N_t}\|_F^2, \tag{15}$$

and due to $\hat{s} \subseteq N_t$ and $X_t|_{\hat{t}\times\hat{s}} = X_{t,s}$ for all admissible leaves $(t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$, we conclude

$$\|X_{t,s} - Q_t Q_t^\top X_{t,s}\|_F^2 = \sum_{r\in\mathcal{T}_\mathcal{I}^t} \|\widehat{X}_{r,s} - \widehat{Q}_r \widehat{Q}_r^\top \widehat{X}_{r,s}\|_F^2$$

for $\widehat{X}_{r,s} := \widehat{X}_r|_{\widehat{K}_r\times\hat{s}}$, therefore we can control the approximation error by making sure that the local errors on the right-hand side of this equation are under control.

3.5 Unification

Using the total cluster basis $(X_t)_{t\in\mathcal{T}_\mathcal{I}}$ directly to construct the adaptive cluster basis $(Q_t)_{t\in\mathcal{T}_\mathcal{I}}$ is only an option if nothing about the structure of X is known, since it means working with ranks on the order of $\#\mathcal{J}$ and leads to algorithms of at least quadratic complexity.

We consider the special matrix

$$X = \begin{pmatrix} X_1 & \dots & X_p \end{pmatrix}$$

for submatrices $X_i \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}_i}$ with disjoint index sets $\mathcal{J}_1, \dots, \mathcal{J}_p$. We assume that cluster trees $\mathcal{T}_{\mathcal{J}_i}$ are given for each $i \in \{1, \dots, p\}$ and that each of the matrices X_i is an $\mathcal{H}^2$-matrix with nested cluster bases $V_i = (V_{i,t})_{t\in\mathcal{T}_\mathcal{I}}$ and $W_i = (W_{i,s})_{s\in\mathcal{T}_{\mathcal{J}_i}}$ and coupling matrices $(S_{i,b})_{b\in\mathcal{L}^+_{\mathcal{I}\times\mathcal{J}_i}}$.

In order to make the computation of $Q = (Q_t)_{t\in\mathcal{T}_\mathcal{I}}$ as efficient as possible, we have to take the special structure of the $\mathcal{H}^2$-matrices X_i into account.

Let $t \in \mathcal{T}_\mathcal{I}$, $t^+ \in \operatorname{pred}(t)$ and $s \in \mathcal{T}_{\mathcal{J}_i}$ with $(t, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}_i}$. Since X_i is an $\mathcal{H}^2$-matrix, we have

$$X_{i,t^+,s} := (X_i)|_{\hat{t}^+\times\hat{s}} = V_{i,t^+} S_{i,b} W_{i,s}^\top,$$

and since V_i is a nested cluster basis, we can apply (4) inductively to get

$$V_{i,t^+}|_{\hat{t}\times K_{i,t^+}} = V_{i,t} E_{i,t,t^+}$$

and therefore

$$X_i|_{\hat{t}\times\hat{s}} = (V_{i,t^+} S_{i,b} W_{i,s}^\top)|_{\hat{t}\times\hat{s}} = V_{i,t^+}|_{\hat{t}\times K_{i,t^+}} S_{i,b} W_{i,s}^\top = V_{i,t} E_{i,t,t^+} S_{i,b} W_{i,s}^\top.$$

This means that the total cluster basis $(X_{i,t})_{t\in\mathcal{T}_\mathcal{I}}$ for the submatrix X_i satisfies

$$X_{i,t} = V_{i,t} Z_{i,t}^\top \qquad \text{for all } t \in \mathcal{T}_\mathcal{I},$$

where $Z_{i,t} \in \mathbb{R}^{N_{i,t}\times K_{i,t}}$ is given by

$$Z_{i,t}|_{\hat{s}\times K_{i,t}} = W_{i,s} S_{i,b}^\top E_{i,t,t^+}^\top \qquad \text{for all } t \in \mathcal{T}_\mathcal{I}, t^+ \in \operatorname{pred}(t), s \in \mathcal{T}_{\mathcal{J}_i} \text{ with } b = (t^+, s) \in \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}_i}.$$

The matrix $Z_{i,t}$ has only $\#K_{i,t}$ columns, therefore we can use an orthogonal transformation to turn it into a upper triangular matrix with not more than $\#K_{i,t}$ columns. More precisely, we can find an index set $\widetilde{N}_{i,t} \subseteq N_{i,t}$ with $\#\widetilde{N}_{i,t} \leq \#K_{i,t}$, an orthogonal matrix $P_{i,t} \in \mathbb{R}^{N_{i,t}\times\widetilde{N}_{i,t}}$ and a matrix $\widetilde{Z}_{i,t} \in \mathbb{R}^{\widetilde{N}_{i,t}\times K_{i,t}}$ with

$$Z_{i,t} = P_{i,t} \widetilde{Z}_{i,t} \qquad \text{for all } t \in \mathcal{T}_\mathcal{I}.$$

The *weight matrix* $\widetilde{Z}_{i,t}$ is relatively small, since both the number of rows and the number of columns are bounded by $\#K_{i,t}$, and using the factorization

$$X_{i,t} = V_{i,t} \widetilde{Z}_{i,t}^\top P_{i,t}^\top \tag{16}$$

allows us to handle the truncation algorithm far more efficiently: the orthogonality of the matrix $P_{i,t}$ implies

$$\|X_{i,t} - Q_t Q_t^\top X_{i,t}\|_F = \|V_{i,t}\widetilde{Z}_{i,t}^\top - Q_t Q_t^\top V_{i,t}\widetilde{Z}_{i,t}^\top\|_F, \tag{17a}$$

$$\|\widehat{X}_{i,t} - \widehat{Q}_t \widehat{Q}_t^\top \widehat{X}_{i,t}\|_F = \|\widehat{V}_{i,t}\widetilde{Z}_{i,t}^\top - \widehat{Q}_t \widehat{Q}_t^\top \widehat{V}_{i,t}\widetilde{Z}_{i,t}^\top\|_F \tag{17b}$$

for the matrices $R_{i,t} := Q_t^\top V_{i,t}$ and

$$\widehat{V}_{i,t} = \begin{cases} V_{i,t} & \text{if } \operatorname{sons}(t) = \emptyset, \\ \begin{pmatrix} R_{i,t_1} E_{i,t_1} \\ R_{i,t_2} E_{i,t_2} \end{pmatrix} & \text{otherwise} \end{cases}$$

already used in the orthogonalization algorithm.

Replacing all matrices X_t^i in the truncation algorithm by the more compact matrices $V_t^i \widetilde{Z}_t^i$ means that we can handle the matrix

$$\widetilde{X}_t := \begin{pmatrix} V_{1,t}\widetilde{Z}_{1,t}^\top & \dots & V_{p,t}\widetilde{Z}_{p,t}^\top \end{pmatrix} \in \mathbb{R}^{\hat{t}\times\widehat{N}_t}, \qquad \widehat{N}_t := \widetilde{N}_{1,t} \cup \ldots \cup \widetilde{N}_{p,t} \subseteq N_t \tag{18}$$

instead of the full total cluster basis matrix X_t without changing the result of the algorithm. We will call the matrix $\widetilde{X}_t$ the *condensed* counterpart of X_t: as far as our algorithm is concerned, both matrices contain essentially the same information, but $\widetilde{X}_t$ is far smaller than X_t. Using this approach, the matrices $\widehat{X}_t$ appearing in the orthogonalization and truncation procedures are given by

$$\widehat{X}_t := \begin{pmatrix} \widehat{V}_{1,t}\widetilde{Z}_{1,t}^\top & \dots & \widehat{V}_{p,t}\widetilde{Z}_{p,t}^\top \end{pmatrix} \in \mathbb{R}^{\widehat{K}_t\times\widehat{N}_t}.$$

We can proceed as in the truncation algorithm: the singular value decomposition of $\widehat{X}_t$ yields an orthogonal matrix $\widehat{Q}_t$ with minimal rank satisfying an error estimate of the form

$$\|\widehat{X}_t - \widehat{Q}_t \widehat{Q}_t^\top \widehat{X}_t\|_F \leq \epsilon_t$$

for an arbitrary $\epsilon_t \in \mathbb{R}_{>0}$. Due to (17) and (15), we get

$$\|X_{t,s} - Q_t Q_t^\top X_{t,s}\|_F^2 \leq \|X_t - Q_t Q_t^\top X_t\|_F^2 \leq \sum_{r\in\mathcal{T}_\mathcal{I}^t} \epsilon_t^2 \qquad \text{for all } b = (t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}^+$$

and conclude that by choosing the error tolerances $(\epsilon_t)_{t\in\mathcal{T}_\mathcal{I}}$ small enough an arbitrarily good approximation can be computed. A similar result [6, Theorem 4.2] can be derived for the total approximation error.

During the course of the algorithm, we also need the operators $R_{i,t} = Q_t^\top V_{i,t}$ describing the mapping from the old cluster bases to the new one. A direct computation would be too time-consuming, but since the orthogonality of Q_{t_1} and Q_{t_2} implies

$$R_{i,t} = Q_t^\top V_{i,t} = \widehat{Q}_t^\top \widehat{V}_{i,t},$$

Algorithm 2 Given $X \in \mathbb{R}^{M\times N}$, construct an index set $K \subseteq M$ and a factorization $X = QR$ such that $Q \in \mathbb{R}^{M\times K}$ is orthogonal, $R \in \mathbb{R}^{K\times N}$ and $\#K \leq \#N$

procedure Householder(X, **var** Q, R, K);
$m \leftarrow \#M$; $\{\mu_1, \ldots, \mu_m\} \leftarrow M$; $n \leftarrow \#N$; $\{\nu_1, \ldots, \nu_n\} \leftarrow N$; $k \leftarrow \min\{m, n\}$;
$\widehat{X} \leftarrow 0 \in \mathbb{R}^{m\times n}$; $\widehat{Q} \leftarrow I \in \mathbb{R}^{m\times m}$;
for $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ **do** $\widehat{X}_{ij} \leftarrow X_{\mu_i\nu_j}$;
Apply k Householder reflections to $\widehat{X}$ to make it upper triangular;
Apply the same reflections to $\widehat{Q}$;
$K \leftarrow \{\mu_1, \ldots, \mu_k\}$; $Q \leftarrow 0 \in \mathbb{R}^{M\times K}$; $R \leftarrow 0 \in \mathbb{R}^{K\times N}$
for $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, k\}$ **do** $Q_{\mu_i\mu_j} \leftarrow \widehat{Q}_{ji}$;
for $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, n\}$ **do** $R_{\mu_i\nu_j} \leftarrow \widehat{X}_{ij}$

we can perform this task more efficiently.

We intend to use this basis construction recursively, so we require the weight matrices $(\widetilde{Z}_t)_{t\in\mathcal{T}_\mathcal{I}}$ for the unified approximation with the new cluster basis Q. Fortunately, their construction is straightforward: the total cluster basis matrices for the unified approximation are given by $Q_t Q_t^\top X_t$, and due to (16) and (18), we have

$$\begin{aligned} Q_t Q_t^\top X_t &= Q_t Q_t^\top \begin{pmatrix} X_{1,t} & \ldots & X_{p,t} \end{pmatrix} = Q_t Q_t^\top \begin{pmatrix} V_{1,t}\widetilde{Z}_{1,t}^\top P_{1,t}^\top & \ldots & V_{p,t}\widetilde{Z}_{p,t}^\top P_{p,t}^\top \end{pmatrix} \\ &= Q_t Q_t^\top \widetilde{X}_t \begin{pmatrix} P_{1,t} & & \\ & \ddots & \\ & & P_{p,t} \end{pmatrix}^\top = Q_t \widetilde{Y}_t \begin{pmatrix} P_{1,t} & & \\ & \ddots & \\ & & P_{p,t} \end{pmatrix}^\top \end{aligned}$$

for the matrix $\widetilde{Y}_t := Q_t^\top \widetilde{X}_t \in \mathbb{R}^{\widetilde{K}_t\times\widehat{N}_t}$, and we can use a standard orthogonal factorization (cf. Algorithm 2) to find an index set $\widetilde{N}_t \subseteq \widehat{N}_t$, a matrix $\widetilde{Z}_t \in \mathbb{R}^{\widetilde{N}_t\times\widetilde{K}_t}$ and an orthogonal matrix $\widehat{P}_t \in \mathbb{R}^{\widehat{N}_t\times\widetilde{N}_t}$ with

$$\widetilde{Y}_t^\top = \widehat{P}_t \widetilde{Z}_t,$$

therefore we get

$$Q_t Q_t^\top X_t = Q_t \widetilde{Y}_t \begin{pmatrix} P_{1,t} & & \\ & \ddots & \\ & & P_{p,t} \end{pmatrix}^\top = Q_t (\widetilde{Z}_t \widehat{P}_t)^\top \begin{pmatrix} P_{1,t} & & \\ & \ddots & \\ & & P_{p,t} \end{pmatrix}^\top = Q_t \widetilde{Z}_t^\top P_t^\top$$

for the orthogonal matrix

$$P_t := \begin{pmatrix} P_{1,t} & & \\ & \ddots & \\ & & P_{p,t} \end{pmatrix} \widehat{P}_t,$$

and conclude that $\widetilde{Z}_t$ is a weight matrix for the new approximation defined by the projection into the new cluster basis Q. The resulting recursive procedure is given in Algorithm 3.

Algorithm 3 Construct unified cluster basis $V = (V_t)_{t\in\mathcal{T}_\mathcal{I}}$ with weight matrices $(\widetilde{Z}_t)_{t\in\mathcal{T}_\mathcal{I}}$ for cluster bases $V_1, \ldots, V_p$ with weight matrices $\widetilde{Z}_1, \ldots, \widetilde{Z}_p$

procedure Unify(t, V_1, …, V_p, $\widetilde{Z}_1$, …, $\widetilde{Z}_p$, **var** Q, $\widetilde{Z}$, R_1, …, R_p);
if $\operatorname{sons}(t) = \emptyset$ **then begin**
  $\widehat{K}_t \leftarrow \hat{t}$;
  for $i \in \{1, \ldots, p\}$ **do** $\widehat{V}_{i,t} \leftarrow V_{i,t}$
end else begin
  $\widehat{K}_t \leftarrow \emptyset$;
  for $t' \in \operatorname{sons}(t)$ **do begin**
    Unify(t', V_1, …, V_p, $\widetilde{Z}_1$, …, $\widetilde{Z}_p$, Q, $\widetilde{Z}$, R_1, …, R_p);
    $\widehat{K}_t \leftarrow \widehat{K}_t \dot\cup \widetilde{K}_{t'}$
  end;
  for $i \in \{1, \ldots, p\}$ **do begin**
    $\widehat{V}_{i,t} \leftarrow 0 \in \mathbb{R}^{\widehat{K}_t \times K_{i,t}}$;
    for $t' \in \operatorname{sons}(t)$ **do** $\widehat{V}_{i,t}|_{\widetilde{K}_{t'} \times K_{i,t}} \leftarrow R_{i,t'} E_{i,t'}$
  end
end;
$\widehat{N}_t \leftarrow \emptyset$;
for $i \in \{1, \ldots, p\}$ **do** $\widehat{N}_t \leftarrow \widehat{N}_t \dot\cup \widetilde{N}_{i,t}$;
$\widehat{X}_t \leftarrow 0 \in \mathbb{R}^{\widehat{K}_t \times \widehat{N}_t}$;
for $i \in \{1, \ldots, p\}$ **do** $\widehat{X}_t|_{\widehat{K}_t \times \widetilde{N}_{i,t}} \leftarrow \widehat{V}_{i,t} \widetilde{Z}_{i,t}$;
Lowrank($\widehat{X}_t$, ϵ_t, $\widehat{Q}_t$, $\widetilde{K}_t$);
for $i \in \{1, \ldots, p\}$ **do** $R_{i,t} \leftarrow \widehat{Q}_t^\top \widehat{V}_{i,t}$;
$\widetilde{Y}_t \leftarrow \widehat{Q}_t^\top \widehat{X}_t \in \mathbb{R}^{\widetilde{K}_t \times \widehat{N}_t}$;
Householder($\widetilde{Y}_t^\top$, $\widehat{P}_t$, $\widetilde{Z}_t$, $\widetilde{N}_t$);
if $\operatorname{sons}(t) = \emptyset$ **then**
  $Q_t \leftarrow \widehat{Q}_t$
else
  for $t' \in \operatorname{sons}(t)$ **do** $F_{t'} \leftarrow \widehat{Q}_t|_{\widetilde{K}_{t'} \times \widetilde{K}_t}$

3.6 Complexity of the unification

Let us now investigate the complexity of Algorithm 3. We assume that the computation of the Householder factorization of a $m \times n$ matrix requires $C_{\mathrm{qr}} mn^2$ operations and that its singular value decomposition (up to machine accuracy) can be found in $C_{\mathrm{svd}} mn^2$ operations.

We introduce the abbreviation

$$k_{i,t} := \#K_{i,t}, \qquad \text{for all } t \in \mathcal{T}_\mathcal{I}$$

and start by establishing a number of basic estimates: by construction, we have

$$\#\widetilde{N}_{i,t} \leq \#K_{i,t} = k_{i,t} \qquad \text{for all } i \in \{1,\ldots,p\}, t' \in \mathcal{T}_{\mathcal{I}},$$

and this implies

$$\#\widehat{N}_t = \sum_{i=1}^{p} \#\widetilde{N}_{i,t} \leq \sum_{i=1}^{p} k_{i,t} \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}.$$

Since $\widehat{X}_t$ has only $\#\widehat{N}_t$ columns, its rank cannot be higher, therefore also the rank of the new cluster basis Q_t has to be bounded by

$$\#\widetilde{K}_t \leq \#\widehat{N}_t \leq \sum_{i=1}^{p} k_{i,t} \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}. \tag{19}$$

Let us now consider the last remaining quantity $\#\widehat{K}_t$. If t is a leaf, we have $\widehat{K}_t = \hat{t}$ and get $\#\widehat{K}_t \leq C_{\mathrm{lf}} k_{i,t}$ for any $i \in \{1,\ldots,p\}$, which trivially implies

$$\#\widehat{K}_t \leq C_{\mathrm{lf}} \sum_{i=1}^{p} k_{i,t} \qquad \text{for all } t \in \mathcal{L}_{\mathcal{I}}.$$

Otherwise, we have

$$\#\widehat{K}_t = \sum_{t' \in \mathrm{sons}(t)} \#\widetilde{K}_{t'} \leq \sum_{i=1}^{p} \sum_{t' \in \mathrm{sons}(t)} k_{i,t'} \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}} \setminus \mathcal{L}_{\mathcal{I}}.$$

Using these preliminary estimates, we can now give a bound for the algorithmic complexity of Algorithm 3:

Lemma 3.2 (Complexity of unification) *There is a constant $C_{\mathrm{uni}} \in \mathbb{R}_{>0}$ depending only on C_{qr}, C_{svd}, C_{lf} and C_{sn} such that Algorithm 3 requires not more than*

$$C_{\mathrm{uni}} p^2 \sum_{i=1}^{p} \sum_{r \in \mathcal{T}_{\mathcal{I}}^t} k_{i,r}^3 \quad \textit{arithmetic operations.}$$

Proof. Let $t \in \mathcal{T}_{\mathcal{I}}$. Algorithm 3 starts by preparing the matrix $\widehat{V}_{i,t}$. If t is a leaf, the matrix $V_{i,t}$ is copied and no arithmetic operations are performed. If t is not a leaf, the matrices $R_{i,t'}$ and $E_{i,t'}$ are multiplied for all $i \in \{1,\ldots,p\}$ and $t' \in \mathrm{sons}(t)$, and this requires not more than

$$\sum_{i=1}^{p} \sum_{t' \in \mathrm{sons}(t)} 2(\#\widetilde{K}_{t'})(\#K_{i,t'})(\#K_{i,t}) \leq 2 \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{t' \in \mathrm{sons}(t)} k_{j,t'} k_{i,t'} k_{i,t} \quad \text{operations}$$

due to (19). We employ the elementary inequality

$$xyz \leq \frac{1}{3}(x^3 + y^3 + z^3) \qquad \text{for all } x, y, z \in \mathbb{R}_{\geq 0}$$

to bound this term by

$$\frac{2}{3}\sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{t'\in\operatorname{sons}(t)} k_{j,t'}^3 + k_{i,t'}^3 + k_{i,t}^3 \leq \frac{2}{3}p\sum_{i=1}^{p}\sum_{t'\in\operatorname{sons}(t)} 2k_{i,t'}^3 + k_{i,t}^3.$$

Since most of our estimates will be of a similar form with different constants, we introduce

$$\alpha_t := \sum_{i=1}^{p} k_{i,t}^3, \qquad \beta_t := \sum_{i=1}^{p}\sum_{t'\in\operatorname{sons}(t)} k_{i,t'}^3$$

and get

$$\sum_{i=1}^{p}\sum_{t'\in\operatorname{sons}(t)} 2(\#\widetilde{K}_{t'})(\#K_{i,t'})(\#K_{i,t}) \leq \left(\frac{2}{3}C_{\mathrm{sn}}p\right)\alpha_t + \left(\frac{4}{3}p\right)\beta_t.$$

We can use similar techniques to prove that the number of operations for the construction of $\widehat{X}_t$ by multiplying $\widehat{V}_{i,t}$ and $\widetilde{Z}_{i,t}$ is bounded by

$$\sum_{i=1}^{p} 2(\#\widehat{K}_t)(\#K_{i,t})(\#\widetilde{N}_{i,t}) \leq \left(2C_{\mathrm{lf}}p + \frac{4}{3}C_{\mathrm{sn}}p\right)\alpha_t + \left(\frac{2}{3}p\right)\beta_t,$$

that the computation of the singular value decomposition of $\widehat{X}_t$ takes not more than

$$C_{\mathrm{svd}}(\#\widehat{K}_t)(\#\widehat{N}_t)^2 \leq \left(C_{\mathrm{svd}}C_{\mathrm{lf}}p^2 + \frac{2}{3}C_{\mathrm{svd}}C_{\mathrm{sn}}p^2\right)\alpha_t + \left(\frac{1}{3}C_{\mathrm{svd}}p^2\right)\beta_t$$

operations, that the transformation matrices $R_{i,t}$ can be constructed in not more than

$$2\sum_{i=1}^{p}(\#\widetilde{K}_t)(\#\widehat{K}_t)(\#K_{i,t}) \leq \left(2C_{\mathrm{lf}}^2p^2 + \frac{2}{3}C_{\mathrm{sn}}p^2\right)\alpha_t + \left(\frac{4}{3}p^2\right)\beta_t$$

operations, that we can compute the "uncondensed" weight matrix $\widetilde{Y}_t$ in not more than

$$2(\#\widetilde{K}_t)(\#\widehat{K}_t)(\#\widehat{N}_t) \leq \left(2C_{\mathrm{lf}}^2p^2 + \frac{2}{3}C_{\mathrm{sn}}p^2\right)\alpha_t + \left(\frac{4}{3}p^2\right)\beta_t$$

operations and that the orthogonal factorization used to find the final weight matrix $\widetilde{Z}_t$ takes not more than

$$C_{\mathrm{qr}}(\#\widehat{N}_t)(\#\widetilde{K}_t)^2 \leq \left(C_{\mathrm{qr}}C_{\mathrm{lf}}^2p^2 + \frac{1}{3}C_{\mathrm{qr}}C_{\mathrm{sn}}p^2\right)\alpha_t + \left(\frac{2}{3}C_{\mathrm{qr}}p^2\right)\beta_t$$

arithmetic operations. Adding up these estimates yields a bound of

$$C_1 p^2 \alpha_t + C_2 p^2 \beta_t = C_1 p^2 \sum_{i=1}^{p} k_{i,t}^3 + C_2 p^2 \sum_{i=1}^{p} \sum_{t' \in \operatorname{sons}(t)} k_{i,t'}^3$$

for constants $C_1, C_2 \in \mathbb{R}_{\geq 0}$ depending only on C_{qr}, C_{svd}, C_{lf} and C_{sn}.

Since Algorithm 3 uses recursive calls to compute the matrices also for all descendants r of t, a bound for the total number of operations can be derived by summing this result over all descendants. Due to the fact that each of these descendants cannot have more than one father, we get the bound

$$\begin{aligned}
\sum_{r \in \mathcal{T}_{\mathcal{I}}^t} C_1 p^2 \alpha_t + C_2 p^2 \beta_t &= C_1 p^2 \sum_{i=1}^{p} \sum_{r \in \mathcal{T}_{\mathcal{I}}^t} k_{i,r}^3 + C_2 p^2 \sum_{i=1}^{p} \sum_{r \in \mathcal{T}_{\mathcal{I}}^t} \sum_{r' \in \operatorname{sons}(r)} k_{i,r'}^3 \\
&\leq C_1 p^2 \sum_{i=1}^{p} \sum_{r \in \mathcal{T}_{\mathcal{I}}^t} k_{i,r}^3 + C_2 p^2 \sum_{i=1}^{p} \sum_{r' \in \mathcal{T}_{\mathcal{I}}^t} k_{i,r'}^3 \\
&\leq (C_1 + C_2) p^2 \sum_{i=1}^{p} \sum_{r \in \mathcal{T}_{\mathcal{I}}^t} k_{i,r}^3,
\end{aligned}$$

and using $C_{\mathrm{uni}} := C_1 + C_2$, this is the estimate we need. ∎

We can again consider special cases: if $k_{i,t} \leq k$ holds for all $i \in \{1, \ldots, p\}$ and all $t \in \mathcal{T}_{\mathcal{I}}$, Algorithm 3 requires $\mathcal{O}((\#\mathcal{T}_{\mathcal{I}}) k^3 p^3)$ operations, and for $\#\mathcal{T}_{\mathcal{I}} \lesssim n/k$, we get $\mathcal{O}(nk^2p^3)$. Lemma 3.2 provides us with a worst-case bound: our estimates allow the case $\#\widetilde{K}_t = \#\widehat{N}_t$, corresponding to completely unrelated submatrices, while in practice we expect that the submatrices correspond to subblocks of a matrix that can be approximated globally by an $\mathcal{H}^2$-matrix, therefore the resulting rank $\#\widetilde{K}_t$ can be expected to be similar to $\#K_{i,t}$.

4 Hierarchical compression

Based on the unification technique presented in the previous section, we can now introduce the hierarchical compression algorithm.

A simple approach is to use the block cluster tree: for each admissible leaf, we construct a low-rank approximation of the corresponding submatrix using a standard technique (like interpolation or adaptive or hybrid cross approximation [2, 7]). Then we work towards the root of the block cluster tree and construct unified row and column cluster bases for each submatrix.

4.1 Leaf matrices

Let $b = (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}}^+$ be an admissible leaf of the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. We assume that we can find a low-rank approximation of $G|_{\hat{t}\times\hat{s}}$ in the form

$$G|_{\hat{t}\times\hat{s}} \approx A_b B_b^*, \qquad A_b \in \mathbb{R}^{\hat{t}\times k_b}, \qquad B_b \in \mathbb{R}^{\hat{s}\times k_b}$$

for $k_b \in \mathbb{N}$, e.g., by the simple and flexible cross approximation techniques [1, 2, 7]. In order to turn it into an $\mathcal{H}^2$-matrix representation of the submatrix, we have to construct row and column cluster bases and a coupling matrix.

For the purposes of error control and overall efficiency, we would like to ensure that the row and column cluster bases are orthogonal. We apply Householder factorizations in such a way that all intermediate results have minimal rank: let us first assume $\#\hat{t} \leq \#\hat{s}$. We compute the Householder factorization

$$V_{b,t}\widehat{A} = A_b$$

with an orthogonal matrix $V_{b,t}$ and observe that $\widehat{A}$ cannot have more than $\min\{\#\hat{t}, k_b\} \leq \tilde{k} := \min\{\#\hat{t}, \#\hat{s}, k\}$ rows. Now we compute the Householder factorization

$$W_{b,s}\widehat{B} = B_b\widehat{A}^\top$$

with an orthogonal matrix $W_{b,s}$. Due to our construction of $\widehat{A}$, the matrix $\widehat{B}$ cannot have more than $\tilde{k}$ columns, and due to the nature of the Householder factorization, it therefore cannot have more than $\tilde{k}$ rows. We let $S_{b,b} := \widehat{B}^\top$ and observe

$$V_{b,t}S_{b,b}W_{b,s}^\top = V_{b,t}(W_{b,s}\widehat{B})^\top = V_{b,t}(B_b\widehat{A}^\top)^\top = V_{b,t}\widehat{A}B_b^\top = A_bB_b^\top.$$

This is the three-term factorization used by $\mathcal{H}^2$-matrices. If $\#\hat{t} \geq \#\hat{s}$, we can use a similar approach, starting with B_b instead of A_b (cf. Algorithm 5). The matrices $V_{b,t}$ and $W_{b,s}$ can be subdivided in order to construct orthogonal cluster bases for the entire subtrees $\mathcal{T}_{\mathcal{I}}^t$ and $\mathcal{T}_{\mathcal{I}}^s$ without additional arithmetic operations.

We may encounter a situation where the rank of the initial approximation $A_bB_b^\top$ of the submatrix $G|_{\hat{t}\times\hat{s}}$ is too high. Under these circumstances, we can reduce the rank of the approximation by computing the singular value decomposition

$$Q_b\Sigma_bP_b = S_{b,b}$$

of the matrix $S_{b,b}$, dropping the sufficiently small singular values in the matrix Σ_b and multiplying $V_{b,t}$ and $W_{b,s}$ by Q_b and P_b, respectively.

4.2 Subdivided matrices

Let now $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}} \setminus \mathcal{L}_{\mathcal{I}\times\mathcal{J}}$. According to Definition 2.2, we have

$$\operatorname{sons}(b) = \begin{cases} \operatorname{sons}(t) \times \operatorname{sons}(s) & \text{if } \operatorname{sons}(t) \neq \emptyset, \operatorname{sons}(s) \neq \emptyset, \\ \{t\} \times \operatorname{sons}(s) & \text{if } \operatorname{sons}(t) = \emptyset, \operatorname{sons}(s) \neq \emptyset, \\ \operatorname{sons}(t) \times \{s\} & \text{if } \operatorname{sons}(t) \neq \emptyset, \operatorname{sons}(s) = \emptyset. \end{cases}$$

The handling of the second and third case is straightforward once the more general first case is treated, therefore we can assume $\operatorname{sons}(t) \neq \emptyset$ and $\operatorname{sons}(s) \neq \emptyset$.

Algorithm 4 Compute the coupling matrices $\widetilde{S}_b \leftarrow R_t S_b P_s^\top$ for all admissible leaves

procedure Convert(b, S, R, P, **var** $\widetilde{S}$);
if sons$(b) = \emptyset$ **then**
  if b admissible **then begin**
    $(t, s) \leftarrow b$;
    $X \leftarrow R_t S_b$; $\widetilde{S}_b \leftarrow X P_s^\top$
  end else Nothing to do for inadmissible leaves
else
  for $b' \in$ sons(b) **do** Convert(b', S, R, P, $\widetilde{S}$)

In order to keep the presentation simple, we again consider only the case $\#\operatorname{sons}(t) = 2 = \#\operatorname{sons}(s)$ and let $\{t_1, t_2\} := \operatorname{sons}(t)$, $\{s_1, s_2\} := \operatorname{sons}(s)$. We let

$$b_{11} := (t_1, s_1), \qquad b_{12} := (t_1, s_2), \qquad b_{21} := (t_2, s_1), \qquad b_{22} := (t_2, s_2).$$

and find

$$\operatorname{sons}(b) = \{b_{11}, b_{12}, b_{21}, b_{22}\}.$$

Due to Definition 2.1, we have

$$\hat{t} = \hat{t}_1 \dot{\cup} \hat{t}_2, \qquad\qquad \hat{s} = \hat{s}_1 \dot{\cup} \hat{s}_2$$

and find

$$G|_{\hat{b}} = \begin{pmatrix} G|_{\hat{b}_{11}} & G|_{\hat{b}_{12}} \\ G|_{\hat{b}_{21}} & G|_{\hat{b}_{22}} \end{pmatrix}.$$

We construct the $\mathcal{H}^2$-matrix construction of $G|_{\hat{t}\times\hat{s}}$ by recursion: first we find $\mathcal{H}^2$-matrix approximations of the submatrices

$$G|_{\hat{b}_{ij}} = G|_{\hat{t}_i \times \hat{s}_j} \approx \widetilde{G}_{ij} \in \mathcal{H}^2(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^{b_{ij}}, V_{ij}, W_{ij}) \qquad \text{for all } i \in \{1,2\}, j \in \{1,2\},$$

where V_{ij} is a nested row cluster basis for the subtree $\mathcal{T}_{\mathcal{I}}^{t_i}$ and W_{ij} is a nested column cluster basis for the subtree $\mathcal{T}_{\mathcal{J}}^{s_j}$. Now

$$\begin{pmatrix} \widetilde{G}_{11} & \widetilde{G}_{12} \\ \widetilde{G}_{21} & \widetilde{G}_{22} \end{pmatrix}$$

is an approximation of $G|_{\hat{t}\times\hat{s}}$ consisting of independent $\mathcal{H}^2$-submatrices, and we are in the situation of subsection 3.5: we have to find row cluster bases V_i for $i \in \{1,2\}$ that approximate V_{i1} and V_{i2} simultaneously, and we have to find column cluster bases W_j for $j \in \{1,2\}$ that approximate W_{1j} and W_{2j} simultaneously. This problem can be solved by Algorithm 3. Using the resulting bases we can then define the row cluster basis $V_b = (V_{b,r})_{r\in\mathcal{T}_{\mathcal{I}}^t}$ and its index sets $(K_{b,r})_{r\in\mathcal{T}_{\mathcal{I}}^t}$ by

$$(V_{b,r}, K_{b,r}) := \begin{cases} (V_{1,r}, K_{1,r}) & \text{if } r \in \mathcal{T}_{\mathcal{I}}^{t_1}, \\ (V_{2,r}, K_{2,r}) & \text{if } r \in \mathcal{T}_{\mathcal{I}}^{t_2}, \\ (0, \emptyset) & \text{otherwise} \end{cases} \qquad \text{for all } r \in \mathcal{T}_{\mathcal{I}}^t.$$

The column cluster basis $W_b = (W_{b,r})_{r\in\mathcal{T}^s_{\mathcal{J}}}$ and its index sets $(L_{b,r})_{r\in\mathcal{T}^s_{\mathcal{J}}}$ are defined similarly by

$$(W_{b,r}, L_{b,r}) := \begin{cases} (W_{1,r}, L_{1,r}) & \text{if } r \in \mathcal{T}^{s_1}_{\mathcal{I}}, \\ (W_{2,r}, L_{2,r}) & \text{if } r \in \mathcal{T}^{s_2}_{\mathcal{I}}, \\ (0, \emptyset) & \text{otherwise} \end{cases} \qquad \text{for all } r \in \mathcal{T}^s_{\mathcal{J}}.$$

Since V_1, V_2, W_1 and W_2 are nested and orthogonal, so are V_b and W_b.

Let $b^* = (t^*, s^*) \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$ be an admissible block. Since b itself is not admissible, there exist $i, j \in \{1, 2\}$ such that $b^* \in \mathcal{T}^{b_{ij}}_{\mathcal{I}\times\mathcal{J}}$, and due to our assumption, we have

$$G|_{\hat{b}^*} \approx \widetilde{G}_{ij}|_{\hat{b}^*} = V_{ij,t^*} S_{ij,b^*} W^\top_{ij,s^*}.$$

We switch to the new unified bases V_b and W_b by applying the orthogonal projections $V_{b,t^*}V^\top_{b,t^*}$ and $W_{b,s^*}W^\top_{b,s^*}$ to this approximation and get

$$G|_{\hat{b}^*} \approx V_{b,t^*}(V^\top_{b,t^*}V_{ij,t^*})S_{ij,b^*}(W^\top_{ij,s^*}W_{b,s^*})W_{b,s^*} = V_{b,t^*}S_{b,b^*}W_{b,s^*}$$

for the new coupling matrix

$$S_{b,b^*} := (V^\top_{b,t^*}V_{ij,t^*})S_{ij,b^*}(W^\top_{ij,s^*}W_{b,s^*}).$$

Using the matrices

$$R_{ij,t^*} := V^\top_{b,t^*}V_{ij,t^*}, \qquad P_{ij,s^*} := W^\top_{b,s^*}W_{ij,s^*}$$

provided by Algorithm 3, the computation of the new coupling matrix can be expressed in the form

$$S_{b,b^*} = R_{ij,t^*}S_{ij,b^*}P^\top_{ij,s^*},$$

and since all three factors of this product can be expected to be small, the change of basis is efficient. Applying this procedure to all submatrices of $G|_{\hat{b}}$ yields the desired $\mathcal{H}^2$-matrix approximation in the space $\mathcal{H}^2(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}, V_b, W_b)$ (cf. Algorithm 4).

In order to apply Algorithm 3, we require weight matrices Z_b and Y_b for V_b and W_b, respectively, but these are easily constructed for leaf blocks and provided by Algorithm 3 for all subdivided blocks.

Applying Algorithm 3 to find the new cluster bases and using Algorithm 4 to convert the submatrices into the new $\mathcal{H}^2$-matrix representation yields the recursive Algorithm 5.

4.3 Complexity of the hierarchical compression

At first glance, the analysis of the hierarchical compression Algorithm 5 seems to be a complicated task, since a large number of cluster bases (one row and one column cluster basis for each block in the block cluster tree) has to be taken into account. Fortunately we can simplify the analysis by an optimality argument: cluster bases constructed for a block $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ have to be able to approximate the entire matrix $G|_{\hat{b}}$. This means that they also

Algorithm 5 Find an approximation of $G|_{\hat{b}}$ in an $\mathcal{H}^2$-matrix space corresponding to the block cluster tree $\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$ and the adaptively constructed cluster bases V_b and W_b

procedure HierarchicalCompression(b, **var** V_b, W_b, S_b, Z_b, Y_b);
$(t,s) \leftarrow b$;
if sons$(b) = \emptyset$ **then**
  if b admissible **then begin**
    Find low-rank approximation $G|_{\hat{b}} \approx A_b B_b^\top$;
    if $\#\hat{t} \geq \#\hat{s}$ **then begin**
      Householder(B_b, $W_{b,s}$, $\widehat{B}$, $L_{b,s}$); Householder($A_b\widehat{B}^\top$, $V_{b,t}$, $\widehat{A}$, $K_{b,t}$); $S_{b,b} \leftarrow \widehat{A}$
    end else begin
      Householder(A_b, $V_{b,s}$, $\widehat{A}$, $K_{b,t}$); Householder($B_b\widehat{A}^\top$, $W_{b,s}$, $\widehat{B}$, $L_{b,s}$); $S_{b,b} \leftarrow \widehat{B}^\top$
    end;
    $Z_{b,t} \leftarrow S_{b,b}$; $Y_{b,s} \leftarrow S_{b,b}^\top$
  end else begin
    Store $G|_{\hat{b}}$ in standard representation;
    $Z_{b,t} \leftarrow 0$; $Y_{b,s} \leftarrow 0$
  end
else begin
  if sons$(t) = \emptyset$ **then begin** $\tau \leftarrow 1$; $t_1 \leftarrow t$ **end**
  else begin $\tau \leftarrow \#\,\text{sons}(t)$; $\{t_1, \ldots, t_\tau\} \leftarrow$ sons(t) **end**;
  if sons$(s) = \emptyset$ **then begin** $\sigma \leftarrow 1$; $s_1 \leftarrow s$ **end**
  else begin $\sigma \leftarrow \#\,\text{sons}(s)$; $\{s_1, \ldots, s_\sigma\} \leftarrow$ sons(s) **end**;
  for $i \in \{1, \ldots, \tau\}$, $j \in \{1, \ldots, \sigma\}$ **do**
    HierarchicalCompression((t_i, s_j), V_{ij}, W_{ij}, S_{ij}, Z_{ij}, Y_{ij});
  for $i \in \{1, \ldots, \tau\}$ **do** Unify(t_i, V_{i1}, …, $V_{i\sigma}$, Z_{i1}, …, $Z_{i\sigma}$, V_b, Z_b, R_{i1}, …, $R_{i\sigma}$);
  for $j \in \{1, \ldots, \sigma\}$ **do** Unify(s_j, W_{1j}, …, $W_{\tau j}$, Y_{1j}, …, $Y_{\tau j}$, W_b, Y_b, P_{1j}, …, $P_{\tau j}$);
  for $i \in \{1, \ldots, \tau\}$, $j \in \{1, \ldots, \sigma\}$ **do** Convert((t_i, s_j), S_{ij}, R_{ij}, P_{ij}, S_b);
end

have to be able to approximate each submatrix $G|_{\hat{b}^*}$ for $b^* \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$. Since Algorithm 1 uses the singular value decomposition to find *optimal* low-rank approximations, this implies that the ranks of the cluster bases for b^* are bounded by the ranks for b.

Therefore we can restrict our attention to the ranks used in the largest block when analyzing the complexity of Algorithm 5.

Let $b = (t,s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, and let $(K_{b,t^*})_{t^*\in\mathcal{T}^t_\mathcal{I}}$ and $(L_{b,s^*})_{s^*\in\mathcal{T}^s_\mathcal{J}}$ be the index families for the cluster bases $V_b = (V_{b,t^*})_{t^*\in\mathcal{T}^t_\mathcal{I}}$ and $W_b = (W_{b,s^*})_{s^*\in\mathcal{T}^s_\mathcal{J}}$. We introduce the abbreviations

$$k_{t^*} := \#K_{b,t^*}, \qquad l_{s^*} := \#L_{b,s^*} \qquad \text{for all } t^* \in \mathcal{T}^t_\mathcal{I}, s^* \in \mathcal{T}^s_\mathcal{J}$$

and again assume that

$$\#\hat{t}^* \leq C_{\text{lf}} k_{t^*}, \qquad \#\hat{s}^* \leq C_{\text{lf}} l_{s^*} \qquad \text{hold for all } t^* \in \mathcal{T}^t_\mathcal{I}, s^* \in \mathcal{T}^s_\mathcal{J}.$$

Lemma 4.1 (Leaf blocks) *Let $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. Let $k, C_{\mathrm{ff}}, C_{\mathrm{nf}} \in \mathbb{R}_{\geq 1}$ be such that*

- *for all admissible leaves $b^* = (t^*, s^*) \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}} \cap \mathcal{L}^+_{\mathcal{I}\times\mathcal{J}}$ an approximation $A_b B_b^\top$ with $A_b \in \mathbb{R}^{\#\hat{t}^*\times k_b}$ and $B_b \in \mathbb{R}^{\#\hat{s}^*\times k_b}$ can be computed in not more than*

$$C_{\mathrm{ff}}(\#\hat{t}^* + \#\hat{s}^*)k_b^2 \quad \text{operations with } k_b \leq k,$$

- *and for all inadmissible leaves $b^* = (t^*, s^*) \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}} \cap \mathcal{L}^-_{\mathcal{I}\times\mathcal{J}}$ the block $G|_{\hat{b}^*}$ can be computed in not more than*

$$C_{\mathrm{nf}}(\#\hat{t}^*)(\#\hat{s}^*) \quad \text{operations.}$$

Then Algorithm 5 treats all leaf blocks in $\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$ with not more than

$$C_{\mathrm{ff}}k^2 \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}})(\#\hat{t} + \#\hat{s}) + \frac{1}{2}C_{\mathrm{nf}}C_{\mathrm{lf}}k(\#\hat{t} + \#\hat{s}) \quad \text{arithmetic operations.}$$

Proof. If b is an admissible leaf, an initial rank k_b approximation $G|_{\hat{b}} \approx A_b B_b^\top$ is computed by a suitable algorithm (e.g., cross approximation). If $\#\hat{t} \geq \#\hat{s}$ holds, an orthogonal decomposition of B_b is computed, which takes not more than

$$C_{\mathrm{qr}}(\#\hat{s})k_b^2 \quad \text{operations.}$$

Then the product $A_b\widehat{B}^\top$ is computed in

$$2(\#\hat{t})k_b \min\{k_b, \#\hat{s}\} \leq 2(\#\hat{t})k_b^2 \quad \text{operations,}$$

and finally its orthogonal decomposition is constructed in not more than

$$C_{\mathrm{qr}}(\#\hat{t})(\min\{k_b, \#\hat{s}\})^2 \leq C_{\mathrm{qr}}(\#\hat{t})k_b^2 \quad \text{operations.}$$

We can apply the same reasoning to the case $\#\hat{t} \leq \#\hat{s}$ and conclude that an admissible leaf block is treated in not more than

$$(C_{\mathrm{ff}} + C_{\mathrm{qr}} + 2)k_b^2(\#\hat{t} + \#\hat{s}) \quad \text{operations}$$

by Algorithm 5.

If b is an inadmissible leaf, our assumption yields that not more than

$$C_{\mathrm{nf}}(\#\hat{t})(\#\hat{s}) \leq \frac{1}{2}C_{\mathrm{nf}}C_{\mathrm{lf}}((\#\hat{t})l_s + k_t(\#\hat{s})) \leq \frac{1}{2}C_{\mathrm{nf}}C_{\mathrm{lf}}k(\#\hat{t} + \#\hat{s}) \quad \text{operations}$$

are required for building $G|_{\hat{b}}$.

A simple recursion and standard arguments (cf. [15, Lemma 2.4]) yield a bound of

$$(C_{\mathrm{ff}} + C_{\mathrm{qr}} + 2)k_b^2 \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}})(\#\hat{t} + \#\hat{s}) + \frac{1}{2}C_{\mathrm{nf}}C_{\mathrm{lf}}k(\#\hat{t} + \#\hat{s})$$

for the total number of operations required by Algorithm 5 to treat all leaf blocks. ∎

In the context of matrices resulting from the discretization of integral operators, the conditions of Lemma 4.1 are satisfied by methods like adaptive [1, 2] and hybrid cross approximation [7], where the constants C_{ff} and C_{nf} may depend on the order of quadrature used to compute the singular and regular integrals.

Lemma 4.2 (Conversion) *The conversion Algorithm 4 requires not more than*

$$2 \sum_{b^*=(t^*,s^*)\in\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}} (k_{t^*}^3 + l_{s^*}^3) \quad \text{arithmetic operations.}$$

Proof. For an admissible block, the computation of the matrix X in Algorithm 4 requires not more than $2k_t^2 l_s$ operations, the subsequent computation of the matrix $\widetilde{S}_b$ requires $2k_t l_s^2$ operations, leading to a bound of

$$2(k_t^2 l_s + k_t l_s^2) \leq \frac{2}{3}(2k_t^3 + l_s^3 + k_t^3 + l_s^3) = 2(k_t^3 + l_s^3).$$

A simple induction yields the desired result for subdivided blocks. ∎

Lemma 4.3 (Non-leaf blocks) *Let $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. Algorithm 5 treats all non-leaf blocks in $\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$ with not more than*

$$C_{\mathrm{sp}}(C_{\mathrm{sn}}^3 C_{\mathrm{uni}} + 2)\operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}) \left(\sum_{t^*\in\mathcal{T}^t_{\mathcal{I}}} k_{t^*}^3 + \sum_{s^*\in\mathcal{T}^s_{\mathcal{J}}} l_{s^*}^3 \right) \quad \text{arithmetic operations,}$$

where C_{sp} is the sparsity constant introduced in Definition 2.7 and C_{uni} is the complexity constant of Lemma 3.2.

Proof. Except for the recursive call, Algorithm 5 handles the block b by applying the unification Algorithm 3 to the row and column cluster bases and then using the conversion Algorithm 4 to switch to the new bases.

Using $p \leq C_{\mathrm{sn}}$ and Lemma 3.2 yields a bound of

$$C_{\mathrm{uni}} C_{\mathrm{sn}}^3 \left(\sum_{t^*\in\mathcal{T}^t_{\mathcal{I}}} k_{t^*}^3 + \sum_{s^*\in\mathcal{T}^s_{\mathcal{J}}} l_{s^*}^3 \right)$$

for the number of operations used for the two unification steps, and due to Lemma 4.2, the conversion requires not more than

$$2 \sum_{b^*=(t^*,s^*)\in\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}} (k_{t^*}^3 + l_{s^*}^3)$$

operations. The recursive structure of Algorithm 5 ensures that these computations are carried out at most once for each subblock of b, therefore we can bound the total number of operations by

$$\sum_{b^+=(t^+,s^+)\in\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}} \left(2 \sum_{b^*=(t^*,s^*)\in\mathcal{T}^{b^+}_{\mathcal{I}\times\mathcal{J}}} (k_{t^*}^3 + l_{s^*}^3) + C_{\mathrm{uni}} C_{\mathrm{sn}}^3 \left(\sum_{t^*\in\mathcal{T}^{t^+}_{\mathcal{I}}} k_{t^*}^3 + \sum_{s^*\in\mathcal{T}^{s^+}_{\mathcal{J}}} l_{s^*}^3 \right) \right). \quad (20)$$

For the first term, we introduce the set

$$\mathcal{P}_{b^*} := \{b^+ \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}} \ : \ b^* \in \mathcal{T}^{b^+}_{\mathcal{I}\times\mathcal{J}}\} \qquad \text{for all } b^* \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$$

and observe that $b^+, b^* \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$ implies

$$\#\mathcal{P}_{b^*} \leq \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}) \qquad \text{for all } b^* \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}},$$

so we get

$$\begin{aligned}
\sum_{b^+=(t^+,s^+)\in\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}} 2 \sum_{b^*=(t^*,s^*)\in\mathcal{T}^{b^+}_{\mathcal{I}\times\mathcal{J}}} (k^3_{t^*} + l^3_{s^*}) &= 2 \sum_{b^*=(t^*,s^*)\in\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}} (k^3_{t^*} + l^3_{s^*}) \#\mathcal{P}_{b^*} \\
&\leq 2\operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}) \sum_{b^*=(t^*,s^*)\in\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}} (k^3_{t^*} + l^3_{s^*}) \\
&= 2\operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}) \left(\sum_{t^*\in\mathcal{T}^t_{\mathcal{I}}} k^3_{t^*} \# \operatorname{row}(t^*) + \sum_{s^*\in\mathcal{T}^s_{\mathcal{J}}} l^3_{s^*} \# \operatorname{col}(s^*) \right) \\
&\leq 2C_{\mathrm{sp}} \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}) \left(\sum_{t^*\in\mathcal{T}^t_{\mathcal{I}}} k^3_{t^*} + \sum_{s^*\in\mathcal{T}^s_{\mathcal{J}}} l^3_{s^*} \right).
\end{aligned}$$

For the second term in (20), we use

$$\begin{aligned}
\mathcal{P}_{t^*} &:= \{b^+ = (t^+, s^+) \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}} \ : \ t^* \in \mathcal{T}^{t^+}_{\mathcal{I}}\} && \text{for all } t^* \in \mathcal{T}^t_{\mathcal{I}}, \\
\mathcal{P}_{s^*} &:= \{b^+ = (t^+, s^+) \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}} \ : \ s^* \in \mathcal{T}^{s^+}_{\mathcal{J}}\} && \text{for all } s^* \in \mathcal{T}^s_{\mathcal{J}}.
\end{aligned}$$

We have

$$\begin{aligned}
\#\mathcal{P}_{t^*} &\leq C_{\mathrm{sp}} \#\{t^+ \in \mathcal{T}_{\mathcal{I}} \ : \ t^* \in \mathcal{T}^{t^+}_{\mathcal{I}} \text{ and there exists } s^+ \in \mathcal{T}_{\mathcal{J}} \text{ with } (t^+, s^+) \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}\} \\
&\leq C_{\mathrm{sp}} \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}})
\end{aligned}$$

due to $(t^+, s^+), b \in \mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}$. Applying the same argument to $\mathcal{P}_{s^*}$ yields

$$\#\mathcal{P}_{t^*} \leq C_{\mathrm{sp}} \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}), \quad \#\mathcal{P}_{s^*} \leq C_{\mathrm{sp}} \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}) \qquad \text{for all } t^* \in \mathcal{T}^t_{\mathcal{I}}, s^* \in \mathcal{T}^s_{\mathcal{J}}.$$

This means

$$\begin{aligned}
C_{\mathrm{uni}} C^3_{\mathrm{sn}} \sum_{b^+=(t^+,s^+)\in\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}} \left(\sum_{t^*\in\mathcal{T}^{t^+}_{\mathcal{I}}} k^3_{t^*} + \sum_{s^*\in\mathcal{T}^{s^+}_{\mathcal{J}}} l^3_{s^*} \right) \\
= C_{\mathrm{uni}} C^3_{\mathrm{sn}} \left(\sum_{t^*\in\mathcal{T}^t_{\mathcal{I}}} k^3_{t^*} \#\mathcal{P}_{t^*} + \sum_{s^*\in\mathcal{T}^s_{\mathcal{J}}} l^3_{s^*} \#\mathcal{P}_{s^*} \right)
\end{aligned}$$

	SLP matrix			DLP matrix			L^2 error	
n	Build	Mem	M/n	Build	Mem	M/n	ϵ_1	ϵ_2
2048	6.3	10.7	5.3	10.2	7.2	3.6	1.2_{-1}	2.3_{-2}
8192	33.1	37.1	7.1	54.3	43.0	5.4	6.2_{-2}	1.1_{-2}
32768	167.7	281.8	8.8	271.2	233.6	7.3	3.1_{-2}	6.7_{-3}
131072	1146.2	1331.9	10.4	1534.4	1032.3	8.1	1.5_{-2}	2.8_{-3}
524288	6081.9	6014.1	11.7	7867.0	4938.2	9.6	7.7_{-3}	1.4_{-3}

Table 1: Boundary integral operators on the unit sphere

$$\leq C_{\mathrm{sp}} C_{\mathrm{uni}} C_{\mathrm{sn}}^3 \operatorname{depth}(\mathcal{T}^b_{\mathcal{I}\times\mathcal{J}}) \left(\sum_{t^*\in\mathcal{T}^t_{\mathcal{I}}} k_{t^*}^3 + \sum_{s^*\in\mathcal{T}^s_{\mathcal{J}}} l_{s^*}^3 \right)$$

and adding our estimates yields the bound we have to prove. ∎

In standard situations, we have

$$k_t \leq k, \qquad l_s \leq k \qquad \text{for all } t \in \mathcal{T}_{\mathcal{I}}, s \in \mathcal{T}_{\mathcal{J}}$$

and typical cluster trees and block partitions (cf. [15]) satisfy

$$\#\mathcal{T}_{\mathcal{I}} \leq n/k, \qquad \#\mathcal{T}_{\mathcal{J}} \leq n/k, \qquad \operatorname{depth}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}) \sim \log(n),$$

so the estimates of Lemma 4.1 and Lemma 4.3 imply that Algorithm 5 constructs an $\mathcal{H}^2$-matrix approximation of G in $\mathcal{O}(nk^2 \log n)$ operations.

5 Numerical experiments

We apply the hierarchical compression scheme to a boundary integral problem: consider a Lipschitz domain $\Omega \subseteq \mathbb{R}^3$ and a harmonic function u in this domain. The Dirichlet values $u|_\Gamma$ on the boundary $\Gamma := \partial\Omega$ of Ω are connected to the Neumann values $\partial_n u|_\Gamma$ by Green's formula

$$\int_\Gamma g(x,y)\partial_n u(y)\,dy = \frac{1}{2}u(x) + \int_\Gamma \partial_{n(y)} g(x,y) u(y)\,dy \qquad \text{for all } x \in \Gamma, \tag{21}$$

which allows us to compute the Neumann values corresponding to given Dirichlet values, where the kernel function is given by

$$g(x,y) := \frac{1}{4\pi}\frac{1}{\|x-y\|_2} \qquad \text{for all } x, y \in \mathbb{R}^3, x \neq y.$$

The integral operator on the left-hand side of (21) is called the single layer potential operator, the integral operator on the right-hand side is called the double layer potential operator. We discretize (21) by a Galerkin scheme with piecewise constant basis

	$\mathcal{H}$-matrix			$\mathcal{H}^2$-matrix		
n	Build	Mem	$\|V - \widetilde{V}\|_2$	Build	Mem	$\|V - \widetilde{V}\|_2$
25744	141.2	184.0	1.8_{-8}	130.5	215.4	1.1_{-8}
102976	791.2	1327.9	5.7_{-10}	798.2	869.0	3.0_{-10}
411904	4520.9	7572.2	2.7_{-11}	4541.3	3695.0	1.3_{-11}
25088	160.8	219.7	1.7_{-8}	156.6	192.5	1.2_{-8}
100352	942.7	1312.5	6.1_{-10}	993.2	941.8	5.1_{-10}
401408	5860.8	7889.3	3.3_{-11}	5874.6	4304.6	1.1_{-11}
28952	313.1	347.1	3.2_{-8}	274.2	522.7	2.1_{-8}
115808	1470.9	2338.6	1.2_{-9}	1593.6	1981.9	3.7_{-10}
463232	8935.3	14171.0	7.9_{-11}	9710.6	9006.4	1.4_{-11}

Table 2: Comparison of $\mathcal{H}$- and $\mathcal{H}^2$-matrix compression

functions for the Neumann values $\partial_n u|_\Gamma$ and continuous piecewise linear basis functions for the Dirichlet values $u|_\Gamma$, thus approximating the single and double layer potential operators by matrices V and K.

These matrices are dense and can only be handled efficiently if a compression scheme and a cubature quadrature rule is applied. We use Algorithm 5 in combination with an initial low-rank approximation provided by the HCA method [7] and a grey-box quadrature rule [26]. Strang's lemma (e.g., [10, Theorem 4.1.1]) implies that error estimates of the form $\|V - \widetilde{V}\|_2 \lesssim h^4$ and $\|K - \widetilde{K}\|_2 \lesssim h^4$ would ensure that the optimal order of convergence of the overall scheme is preserved. We achieve this goal by using the advanced error control technique presented in [4].

Table 1 contains the results for a simple situation: we approximate the unit sphere by n plane triangles and apply our scheme to the harmonic functions

$$u_1(x) = x_1 + x_2 + x_3, \qquad u_2(x) = x_1^2 - x_3^2 \qquad \text{for all } x \in \mathbb{R}^3$$

to test the approximation properties of the approximation scheme. The columns "Build" contain the time for the construction of the matrices (including quadrature of the singular nearfield integrals) in seconds, measured on one processor of a SunFire X4600 computer, the columns "Mem" give the storage requirements for near- and farfield in megabytes, the columns "M/n" give the storage requirements per degree of freedom in kilobytes, while ϵ_1 and ϵ_2 are the L^2-norm errors of the Neumann values $\partial_n u|_\Gamma$.

We can see that the errors converge like $1/n$, which is the optimal rate for a piecewise constant approximation. We can also see that the storage requirements per degree of freedom increase roughly by a factor of 1.6 if the number of degrees of freedom is quadrupled, this hints at a storage complexity of $\mathcal{O}(n \log n)$.

In a second experiment, we compare the $\mathcal{H}^2$-matrix approximation provided by the hierarchical compression algorithm with an approximation in the $\mathcal{H}$-matrix format: the adaptive coarsening algorithm [14] not only uses near-optimal low-rank approximations in each admissible block, it also optimizes the block cluster tree in order to reduce the

storage requirements to the minimum. The resulting $\mathcal{H}$-matrix is very close to the best possible approximation in this representation.

We compare both techniques using three different geometries: the first two are examples from the NetGen package by Joachim Schöberl, namely an approximation of a crank shaft with 25744 plane triangles and an approximation of a pierced sphere with 25088 plane triangles, the third one is an approximation of a three-dimensional foam with 28952 plane triangles courtesy of Günther Of and Heiko Andrä. Each of the geometries is refined twice by splitting each triangle into four congruent subtriangles, thus providing us with a range of problem dimensions.

We approximate the single layer potential matrix V on each of the resulting nine surface meshes and compare the time for the construction and the storage requirements. The results are given in Table 2, and we can see that $\mathcal{H}$- and $\mathcal{H}^2$-matrices require approximately the same time to build, but that the storage requirements of the $\mathcal{H}^2$-matrices are clearly preferable for large problem dimensions: in the NetGen examples with roughly 400000 degrees of freedom, they require only half of the storage needed by their $\mathcal{H}$-matrix counterparts and reach a better accuracy. For higher resolutions of the foam example, the $\mathcal{H}^2$-matrix is also preferable: if offers an error of 1.4×10^{-11} using 9 gigabytes, while the $\mathcal{H}$-matrix only reaches an error of 7.9×10^{-11} using 14 gigabytes.

The experiments demonstrate that with the hierarchical compression Algorithm 5, the $\mathcal{H}^2$-matrix method is significantly better than the best known $\mathcal{H}$-matrix approach: it requires approximately the same amount of time (which is no surprise, since it is based on the same initial low-rank approximation), and its improved storage complexity is clearly visible for large problem dimensions n. The estimates of Lemma 2.8 and Lemma 2.9 indicate that the advantage of the $\mathcal{H}^2$-matrix format will become even more pronounced if the problem dimension n grows larger.

References

[1] M. Bebendorf. Approximation of boundary element matrices. *Numer. Math.*, 86(4):565–589, 2000.

[2] M. Bebendorf and S. Rjasanow. Adaptive Low-Rank Approximation of Collocation Matrices. *Computing*, 70(1):1–24, 2003.

[3] G. Beylkin, R. Coifman, and V. Rokhlin. The fast wavelet transform and numerical algorithms. *Comm. Pure and Appl. Math.*, 44:141–183, 1991.

[4] S. Börm. Adaptive variable-rank approximation of dense matrices. Preprint 114/2005, Max Planck Institute for Mathematics in the Sciences, 2005. To appear in SIAM J. of Sci. Comp.

[5] S. Börm. Approximation of integral operators by $\mathcal{H}^2$-matrices with adaptive bases. *Computing*, 74(3):249–271, 2005.

[6] S. Börm. Data-sparse approximation of non-local operators by $\mathcal{H}^2$-matrices. *Linear Algebra and its Applications*, 422:380–403, 2007.

[7] S. Börm and L. Grasedyck. Hybrid cross approximation of integral operators. *Numerische Mathematik*, 101:221–249, 2005.

[8] S. Börm, L. Grasedyck, and W. Hackbusch. Hierarchical Matrices. Lecture Note 21 of the Max Planck Institute for Mathematics in the Sciences, 2003.

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

[10] P. G. Ciarlet. *The finite element method for elliptic problems*. SIAM, 2002.

[11] W. Dahmen, H. Harbrecht, and R. Schneider. Compression techniques for boundary integral equations — Asymptotically optimal complexity estimates. *SIAM J. Num. Anal.*, 43(6):2251–2271, 2006.

[12] W. Dahmen and R. Schneider. Wavelets on manifolds I: Construction and domain decomposition. *SIAM Journal of Mathematical Analysis*, 31:184–230, 1999.

[13] L. Grasedyck. *Theorie und Anwendungen Hierarchischer Matrizen*. Doctoral thesis, Universität Kiel, 2001.

[14] L. Grasedyck. Adaptive recompression of $\mathcal{H}$-matrices for BEM. *Computing*, 74(3):205–223, 2004.

[15] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70:295–334, 2003.

[16] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[17] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace in three dimensions. In *Acta Numerica 1997*, pages 229–269. Cambridge University Press, 1997.

[18] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62:89–108, 1999.

[19] W. Hackbusch and B. Khoromskij. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part II: Application to multi-dimensional problems. *Computing*, 64:21–47, 2000.

[20] W. Hackbusch, B. Khoromskij, and S. A. Sauter. On $\mathcal{H}^2$-matrices. In H. Bungartz, R. Hoppe, and C. Zenger, editors, *Lectures on Applied Mathematics*, pages 9–29. Springer-Verlag, Berlin, 2000.

[21] W. Hackbusch and Z. P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik*, 54:463–491, 1989.

[22] M. Lintner. The eigenvalue problem for the 2d Laplacian in $\mathcal{H}$-matrix arithmetic and application to the heat and wave equation. *Computing*, 72:293–323, 2004.

[23] G. Of, O. Steinbach, and W. L. Wendland. The fast multipole method for the symmetric boundary integral formulation. *IMA J. Numer. Anal.*, 26:272–296, 2006.

[24] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60:187–207, 1985.

[25] S. A. Sauter. Variable order panel clustering. *Computing*, 64:223–261, 2000.

[26] S. A. Sauter and C. Schwab. *Randelementmethoden*. Teubner, 2004.

[27] J. Tausch and J. White. Multiscale bases for the sparse representation of boundary integral operators on complex geometries. *SIAM J. Sci. Comput.*, 24(5):1610–1629, 2003.

[28] E. E. Tyrtyshnikov. Mosaic-skeleton approximation. *Calcolo*, 33:47–57, 1996.

[29] E. E. Tyrtyshnikov. Incomplete cross approximation in the mosaic-skeleton method. *Computing*, 64:367–380, 2000.

[30] T. von Petersdorff and C. Schwab. Fully discretized multiscale Galerkin BEM. In W. Dahmen, A. Kurdila, and P. Oswald, editors, *Multiscale wavelet methods for PDEs*, pages 287–346. Academic Press, San Diego, 1997.