Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

Adaptive Recompression of $\mathcal{H}$-Matrices for BEM

(revised version: July 2004)

by

Lars Grasedyck

Preprint no.: 17 2004

Adaptive Recompression of $\mathcal{H}$-Matrices for BEM

L. Grasedyck, Leipzig

Abstract

The efficient treatment of dense matrices arising, e.g., from the finite element discretisation of integral operators requires special compression techniques. In this article we use a hierarchical low-rank approximation, the so-called $\mathcal{H}$-matrix, that approximates the dense stiffness matrix in admissible blocks (corresponding to subdomains where the underlying kernel function is smooth) by low rank matrices. The low rank matrices are assembled by the ACA+ algorithm, an improved variant of the well-known ACA method. We present an algorithm that can determine a coarser block structure that minimises the storage requirements (enhanced compression) and speeds up the arithmetic (e.g., inversion) in the $\mathcal{H}$-matrix format. This coarse approximation is done adaptively and on-the-fly to a given accuracy such that the matrix is assembled with minimal storage requirements while keeping the desired approximation quality. The benefits of this new recompression technique are demonstrated by numerical examples.

AMS Subject Classification: 65F05, 65F30, 65F50

Key words: hierarchical matrices, data-sparse approximations, formatted matrix operations, fast solvers, preconditioning, boundary elements.

1 Introduction

The efficient treatment of dense matrices arising, e.g., from the finite element discretisation of integral operators requires special compression techniques to avoid the quadratic cost for the assembly and storage.

The standard techniques in this field of research include but are not limited to panel clustering [15, 19], multipole expansions [17, 11], interpolation [5] or (adaptive) cross approximation [20, 1, 3]. All of these methods yield a matrix that consists blockwise of low-rank matrices.

A convenient format to store these matrices is the $\mathcal{H}$-matrix format [12, 13, 9, 10] which is at the same time useful to construct an efficient preconditioner or even an accurate inverse. However, the (hierarchical) partition of the matrix into blocks that allow for a low-rank approximation is not unique. Even the question whether or not a single block is suitable for a low-rank approximation is non-trivial [14]. Typically, one demands that the singular values of the matrix

block decay exponentially and ensures this by sufficient conditions (standard admissibility, weak admissibility).

For a rather coarse approximation of a matrix block an exponential decay of the singular values is not necessary. Therefore, standard (or weak) admissibility conditions for the blocks are too restrictive. The partitions generated by these sufficient conditions serve as input for our algorithm that coarsens the structure. The coarsening process is compatible with the assembly of the low-rank blocks by the standard methods and can thus be done on-the-fly, i.e., the input matrix is not generated as a whole but only for single blocks. The output of the algorithm is a recompressed matrix of reduced storage complexity with a coarser block structure.

The coarse block structure is especially useful to construct coarse preconditioners, e.g., by LU or Cholesky decomposition or the approximate inversion based on the formatted $\mathcal{H}$-matrix arithmetic.

The rest of this article is organised as follows: in Section 2 we introduce a simple model problem, describe in short the $\mathcal{H}$-matrix format and introduce a new variant of ACA for the assembly for the low rank blocks. In Section 3 the recompression algorithm is presented along with numerical experiments for realistic surface geometries in three space dimensions. In the last Section 4 we comment on the solution of the discrete system.

2 The $\mathcal{H}$-Matrix Format

2.1 Model Problem: Integral Equation

We consider a Fredholm integral operator of the form

$$\mathcal{G}[u](x) = \int_\Omega g(x,y)u(y)\,\mathrm{d}y \tag{2.1}$$

on a submanifold or subdomain Ω of $\mathbb{R}^3$ with a kernel function

$$g : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}.$$

The kernel function might be (but is not limited to) the single layer or double layer kernel for the Laplacian on a manifold Ω:

$$g^{\mathrm{SLP}}(x,y) := \frac{1}{4\pi\|x-y\|}, \quad g^{\mathrm{DLP}}(x,y) := \frac{\partial}{\partial n_y}\frac{1}{4\pi\|x-y\|}.$$

$\mathcal{H}$-matrices are based on the fact that, at least for typical kernel functions $g(\cdot,\cdot)$, singularities only occur at the diagonal and the function is smooth everywhere else. This is described by the *asymptotic smoothness*: the kernel function $g(\cdot,\cdot)$ is called *asymptotically smooth*, if there exist constants C_{as1} and C_{as2} and a singularity degree $s \geq 0$ such that for all $z \in \{x_j, y_j\}$ the inequality

$$|\partial_z^\nu g(x,y)| \leq C_{\mathrm{as1}}(C_{\mathrm{as2}}\|x-y\|)^{-\nu-s}\nu! \tag{2.2}$$

holds. This kind of operator occurs, e.g., in the integral equation formulation of the Poisson problem in $\mathbb{R}^3$, where g is the single layer or double layer kernel $g^{\mathrm{SLP}}, g^{\mathrm{DLP}}$.

A standard Galerkin discretisation of $\mathcal{G}$ for a basis $(\varphi_i)_{i\in I}$, $I = \{1, \dots, n\}$, $V_n := \mathrm{span}\{\varphi_1, \dots, \varphi_n\}$, yields a matrix G with entries

$$G_{ij} := \int_\Omega \int_\Omega \varphi_i(x) g(x,y) \varphi_j(y) \, \mathrm{d}x \, \mathrm{d}y. \tag{2.3}$$

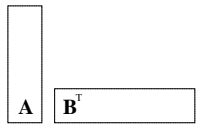
Since the support of the kernel g is in general not local, one expects a dense matrix G. The algorithmic complexity for computing and storing a dense matrix is quadratic in the number of degrees of freedom, therefore different approaches have been introduced to avoid dense matrices: for translation-invariant kernel functions and simple geometries, the matrix G has Toeplitz structure, which can be exploited by algorithms based on the fast Fourier transformation. If the underlying geometry can be described by a small number of smooth maps, wavelet techniques can be used in order to compress the resulting dense matrix [7].

2.2 Low Rank Approximation by ACA+

Hierarchical matrices [12, 13, 6, 10, 5] are based on the idea of replacing the kernel function locally by degenerate approximations. On the discrete level this means that blocks of the matrix are approximated by low rank matrices.

Definition 2.1 ($R(k)$-Matrix Format) *Let $M \in \mathbb{R}^{n\times m}$, $k \in \mathbb{N}$ and M of rank at most k. An $R(k)$-matrix representation of M is a factorisation of the form*

$$M = AB^T$$



with matrices $A \in \mathbb{R}^{n\times k}, B \in \mathbb{R}^{m\times k}$. Sometimes it is convenient to write $M = \sum_{\nu=1}^{k} a_\nu b_\nu^T$ for the column vectors a_i, b_i of A, B.

The storage requirements for an $R(k)$-matrix are $k(n+m)$ instead of the quadratic cost nm for standard full matrices. Since the rank k is expected to be $k \approx \log(n)$ this is a data-sparse representation of the matrix M. Moreover, the matrix-vector multiplication $v \mapsto Mv$ can be split into two multiplications $v \mapsto B^T v \mapsto A(B^T v)$ so that only $2k(n+m)-k-n$ additions and multiplications of real numbers are necessary to perform the exact matrix-vector multiplication.

Here, we compute each of those low rank blocks by the *adaptive cross approximation* (ACA) [20, 1, 3]. However, our method is not restricted to ACA but works equally well for Taylor expansion, multipole expansion or interpolation. Any method that yields a blockwise low rank approximation can serve as an input for our coarsening algorithm.

Let $t \times s \subset I \times I$, $\Omega_t := \cup_{i\in t} \mathrm{supp}\varphi_i, \Omega_s := \cup_{i\in s}\varphi_i$ and B_t, B_s axially parallel boxes that contain Ω_t, Ω_s (cf. Figure 1) and fulfil the η-admissibility condition

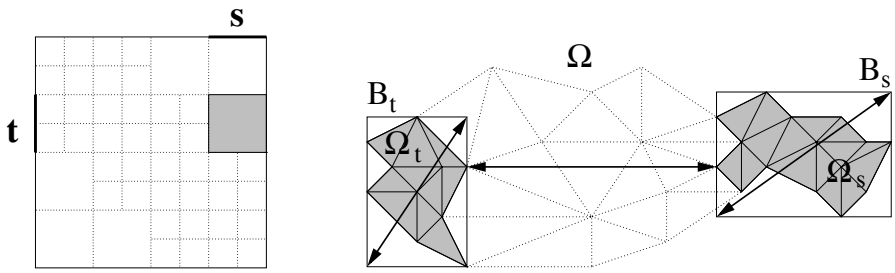


Figure 1: The block $t \times s \subset I \times I$ corresponds to a subset $\Omega_t \times \Omega_s$ of $\Omega \times \Omega$.

$$\min\{\mathrm{diam}(B_t), \mathrm{diam}(B_s)\} \leq 2\,\eta\,\mathrm{dist}(B_t, B_s). \tag{2.4}$$

Then the sub-matrix $G|_{t\times s} := (G_{ij})_{i\in t, j\in s}$ of G from (2.3) allows for a low rank approximation up to a relative error of ε where the rank k depends logarithmically on $1/\varepsilon$ [5]. A good heuristic (proofs exist for Nyström methods and asymptotically smooth kernels) to generate a low rank approximation follows.

Construction 2.2 (ACA with full pivoting) *The construction of an approximation of the form $\sum_{\nu=1}^{k} a_\nu b_\nu^T$ to a matrix $M \in \mathbb{R}^{n\times m}$ up to a relative error $\|M - \sum_{\nu=1}^{k} a_\nu b_\nu^T\|_2 \approx \varepsilon \|M\|_2$ is done in k steps $\mu = 1, \ldots, k$:*

Input: A function that returns the matrix entry M_{ij} for an index pair (i, j).

Step $\mu = 1, \ldots, k$:

1. Determine a pivot index (i^*, j^*) that maximises $\delta := |M_{ij} - \sum_{\nu=1}^{\mu-1} (a_\nu)_i (b_\nu)_j|$.
2. Stop if $\delta = 0$.
3. Compute the entries of the two vectors $a_\mu \in \mathbb{R}^n, b_\mu \in \mathbb{R}^m$ by

$$(a_\mu)_i := M_{ij^*} - \sum_{\nu=1}^{\mu-1} (a_\nu)_i (b_\nu)_{j^*}, \quad (b_\mu)_j := \frac{1}{\delta}\left(M_{i^*j} - \sum_{\nu=1}^{\mu-1} (a_\nu)_{i^*} (b_\nu)_j\right).$$

Stop if $\|a_\mu\|_2 \|b_\mu\|_2 \leq \varepsilon \|a_1\|_2 \|b_1\|_2$.

Output: The factorisation $AB^T \approx M$.

In each step $\mu = 1, \ldots, k$ of Construction 2.2 we have to determine the pivot index $(i^*, j^*) = (i^*_\mu, j^*_\mu)$. In order to avoid the quadratic cost for assessing all entries M_{ij}, one has to come up with a decent strategy. The state of the art **partial pivoting** approach used in [3] is to choose the first row index $i^*_1 := 1$ (or at random) and determine the column index j^*_1 by

$$j^*_1 := \mathrm{argmax}_{j=1,\ldots,m} |M_{i^*_1 j}|.$$

In the later steps $\mu = 2, \dots, k$, the row index i^*_μ is the maximiser of the column corresponding to the index $j^*_{\mu-1}$ from the previous step $\mu - 1$,

$$i^*_\mu := \operatorname{argmax}_{i=1,\dots,n} |M_{ij^*_{\mu-1}} - \sum_{\nu=1}^{\mu-1} (a_\nu)_i (b_\nu)_{j^*_{\mu-1}}|,$$

and j^*_μ is the corresponding row maximiser

$$j^*_\mu := \operatorname{argmax}_{j=1,\dots,m} |M_{i^*_\mu j} - \sum_{\nu=1}^{\mu-1} (a_\nu)_{i^*_\mu} (b_\nu)_j|.$$

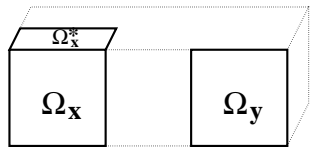
For matrix entries M_{ij} that stem from the evaluation of an asymptotically smooth function, as it is the case for the Nyström method, Theorem 4 from [1] states

$$|M_{ij} - (AB^T)_{ij}| \leq c\eta^{\sqrt[d]{k}},$$

i.e., exponential convergence. The above mentioned partial pivoting strategy from [3] fails, if the kernel function is not asymptotically smooth with respect to both variables or if we use a Galerkin discretisation. In order to illustrate this, we give a simple example.

Example 2.3 (Counterexample for Partial Pivoting) *Let M be a matrix with entries*

$$\begin{aligned} M_{ij} &:= \frac{\langle n(x_i), x_i - y_j \rangle}{\|x_i - y_j\|} \\ x_i &\in \Omega_x \subset \mathbb{R}^3, \quad i = 1, \dots, n_1 < n, \\ x_i &\in \Omega^*_x \subset \mathbb{R}^3, \quad i = n_1 + 1, \dots, n, \\ y_j &\in \Omega_y \subset \mathbb{R}^3, \quad j = 1, \dots, n, \end{aligned}$$



where $n(x_i)$ is the outer unit normal vector to $\Omega_x \cup \Omega^*_x$ in x_i as it typically arises in the discretisation of the double layer potential. The domains Ω_x and Ω_y are parallel. Thus, all entries M_{ij} with $x_i \in \Omega_x$ and $y_j \in \Omega_y$ are zero. In Algorithm 4.2 from [3] the first k row indices are $i^*_\nu = \nu$ for $\nu = 1, \dots, k$ and since $M_{\nu,j} = 0$ for all j, the generated approximation is zero, while $M_{i,j} \neq 0$ for $x_i \in \Omega^*_x$.

Obviously, one can circumvent the above counterexample by taking an initial row and column of the matrix and determine a row or column pivot index from these. In the further steps, the initial row and column can be reused. Thereby, we replace the above mentioned "look-back" strategy by the following "look-ahead" strategy in order to obtain the $ACA+$ algorithm:

Construction 2.4 (Partial Pivoting in ACA+) *1. In a setup phase we choose two reference indices (r, c) at random and compute the row r and column c of the matrix M.*

	$\eta = \sqrt{2}/2$			$\eta = \sqrt{2}$			$\eta = 2\sqrt{2}$		
ε	SVD	ACA+	ACA	SVD	ACA+	ACA	SVD	ACA+	ACA
10^{-1}	1	2	3	2	**4**	**4**	3	7	9
10^{-2}	3	5	6	5	6	14	7	13	14
10^{-3}	5	**9**	**7**	8	14	16	14	19	27
10^{-4}	8	11	12	14	17	20	23	30	37
10^{-5}	12	15	16	18	24	25	31	41	44

Table 1: A comparison of ACA,ACA+ and SVD. The table contains the rank necessary in order to get a relative approximation error less than ε.

2. *In each step $\mu = 1, \dots, k$ of Construction 2.2 we determine the row index i^* as the column maximiser*

$$i^* := \text{argmax}_{i=1,\dots,n} |M_{ic} - \sum_{\nu=1}^{\mu-1} (a_\nu)_i (b_\nu)_c|$$

and the column index j^* as the row maximiser

$$j^* := \text{argmax}_{j=1,\dots,m} |M_{rj} - \sum_{\nu=1}^{\mu-1} (a_\nu)_r (b_\nu)_j|.$$

We choose the index pair (i^*, c) or (r, j^*) that yields the larger matrix entry. The missing pivot index j^* or i^* is chosen by

$$\begin{aligned} j^* &:= \text{argmax}_{j=1,\dots,m} |M_{i^*j} - \sum_{\nu=1}^{\mu-1} (a_\nu)_i^* (b_\nu)_j| \quad \textit{or} \\ i^* &:= \text{argmax}_{i=1,\dots,n} |M_{ij^*} - \sum_{\nu=1}^{\mu-1} (a_\nu)_i (b_\nu)_{j^*}|. \end{aligned}$$

3. *If one of the pivot indices i^* or j^* coincides with r or c, then we choose a new reference index (at random) and compute the corresponding row or column of the matrix.*

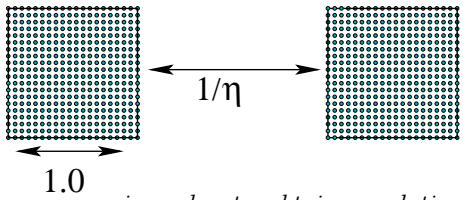
The behaviour of $ACA+$ compared to ACA is investigated for a model problem in Section 2.4. Even for matrices generated by an asymptotically smooth function the new pivoting pays, as is demonstrated by the next example.

Example 2.5 *We consider the matrix $M \in \mathbb{R}^{n \times n}$, $n = 81$, with entries*

$$M_{ij} := \frac{1}{\|x_i - y_j\|},$$

where the vertices x_i, y_j are chosen as in the following figure, i.e., a uniform 20×20 grid for a square. For the two squares we consider three model situations:

1. *Admissibility with* $\eta = \sqrt{2}/2$.
2. *Admissibility with* $\eta = \sqrt{2}$.
3. *Admissibility with* $\eta = 2\sqrt{2}$.



In Table 1 we have denoted the rank necessary in order to obtain a relative accuracy of $\varepsilon = 10^{-j}$, $j = 1, \ldots, 5$. *Except for two cases, where ACA requires rank* 4 *(*7*) and ACA+ requires rank* 4 *(*9*), the new pivoting needs to compute at least one matrix cross less than ACA, which compensates for the extra reference row and column. For large* η *the savings are the most.*

2.3 Clustering and Standard $\mathcal{H}$-Matrix Compression

The low rank approximation of Construction 2.2 applies for matrix blocks $t \times s \subset I \times I$ that are admissible with respect to (2.4). Therefore, we have to subdivide the matrix into blocks that are admissible, preferably large blocks in order to compress the matrix by a maximal factor. The standard way to choose the blocks (testing all blocks would be too expensive) is to *cluster* the index set I hierarchically in a cluster tree T_I and use a canonical construction for the partition of the matrix.

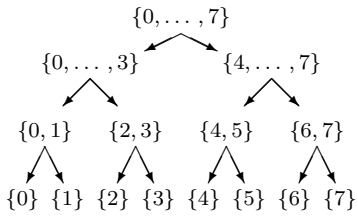
Notation 2.6 (Tree) *Let* $T = (V, E)$ *be a tree with vertex set* V *and edge set* E. *The unique vertex* $v \in V$ *with* $(w, v) \notin V$ *for all* $w \neq v$ *is called the root of* T *and denoted by* $\operatorname{root}(T)$. *The levels of the tree are defined by*

$$T^0 := \{\operatorname{root}(T)\}, \qquad T^{i+1} := \{w \in V \mid \exists v \in T^i : (v, w) \in E\}.$$

The set of successors of a node $v \in T^i$ *is defined as* $\operatorname{Sons}(v) := \{w \in T^{i+1} \mid (v, w) \in E\}$. *The set of leaves (*$\operatorname{Sons}(v) = \emptyset$*) is denoted by* $\mathcal{L}(T)$. *The depth of the tree is* $p(T) := 1 + \max_{T^i \neq \emptyset} i$ *or for a fixed tree* T *simply* p. *We use the short notation "*$v \in T$*" instead of "*$v \in V$*".*

Definition 2.7 (Cluster Tree T_I) *A cluster tree* T_I *for an index set* I *is a tree with root* $\operatorname{root}(T_I) = I$ *that fulfils*

$$\forall\, t \in T_I : \qquad t = \dot{\bigcup_{s \in S(t)}} s \qquad \text{and } t \neq \emptyset.$$



For all practical cases the cluster tree T_I is a binary tree (each node has exactly two successors or it is a leaf), the depth p is $\mathcal{O}(\log \#I)$ and the cardinality of T_I is $\mathcal{O}(\#I)$. Next, we want to construct the cluster tree T_I with underlying index set $I = \{1, \ldots, n\}$. Each index $i \in I$ corresponds to one of the basis functions $\varphi_i \in V_n$ (cf. (2.3)). The geometric location of the index i is given by the Chebyshev centre x_i of the support of φ_i (the Chebyshev centre is the centre of the smallest ball containing the support of φ_i). Instead of the Chebyshev centre one could choose any point x_i from the support of φ_i.

Construction 2.8 (Geometrically Balanced Clustering) *Let*

$$B := \bigotimes_{j=1}^{3} [\alpha_j, \beta_j]$$

be an axially parallel box containing the domain Ω. The cluster tree T_I is built from root to the leaves by the procedure `split` *defined next. We call this procedure with the parameters* `split`*(I,$(x_i)_{i\in I}$,B,$n_{\min}$), where n_{min} is the size of clusters that will not be split further, e.g. $n_{min} = 1$.*

`split`(t: NODE OF THE CLUSTER TREE,
$(x_i)_{i\in I}$: VECTOR OF CHEBYSHEV CENTRES,
B: BOX CONTAINING THE CENTRES,
$n_{\min}$: MINIMAL LEAF SIZE)

1. IF $\#t < n_{\min}$ THEN $\mathrm{Sons}(t) := \{\}$ AND RETURN, OTHERWISE
2. DETERMINE THE COORDINATE j SUCH THAT $\beta_j - \alpha_j$ IS THE LARGEST;
   DEFINE $s_1 := \{i \in t \mid (x_i)_j < (\alpha_j + \beta_j)/2\}, s_2 := t \setminus s_1$;
   SET $\mathrm{Sons}(t) := \{s_1, s_2\} \setminus \{\emptyset\}$;
   DEFINE B_{s_1} AS B BUT REPLACE β_j BY $(\alpha_j + \beta_j)/2$;
   DEFINE B_{s_2} AS B BUT REPLACE α_j BY $(\alpha_j + \beta_j)/2$;
   FOR EACH $s \in \mathrm{Sons}(t)$ CALL `split`(s,$(x_i)_{i\in s}$,B_s,$n_{\min}$);
   RETURN.

Construction 2.8 terminates if and only if the number of points x_i with the same geometric position is less than $n_{\min}$, but this can be determined a priori with complexity $\mathcal{O}(\#I)$.

Pairs of clusters $(t, s) \in T_I \times T_I$ are candidates for blocks of the matrix that we want to approximate in the $R(k)$-matrix format (Definition 2.1) by Construction 2.2. The number of possible pairs is too large, therefore we test only pairs of clusters on the same level of the tree. This motivates the definition of a so-called block cluster tree $T_{I\times I}$ that stores the admissible pairs of clusters in a hierarchical form.

Definition 2.9 (Block Cluster Tree $T_{I\times I}$) *A block cluster tree $T_{I\times I}$ based on the cluster tree T_I is a cluster tree for $I \times I$ such that (cf. Figure 2)*

$$\forall\, v \in T^i_{I\times I} \;\; \exists\, t, s \in T^i_I : \qquad v = t \times s.$$

Remark 2.10 (Cluster Tree yields Partition) *The leaves of a cluster tree T_I form a partition of the index set I. As a consequence, the leaves of a block cluster tree $T_{I\times I}$ yield a partition of the product index set $I \times I$. On each level ℓ of a cluster tree T_I there holds*

$$I = \left(\dot{\bigcup_{v\in T_I^\ell}} v \right) \dot\cup \left(\dot{\bigcup_{v\in \mathcal{L}(T_I)\cap T_I^{\ell-1}}} v \right) \dot\cup \cdots \dot\cup \left(\dot{\bigcup_{v\in \mathcal{L}(T_I)\cap T_I^{0}}} v \right).$$

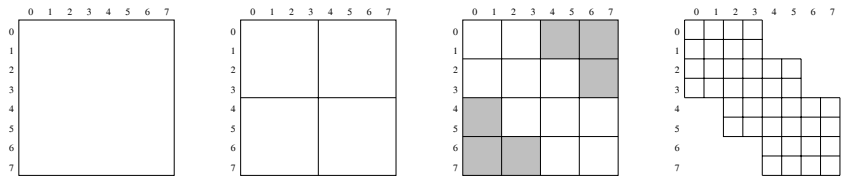


Figure 2: Depicted are four levels of a block cluster tree based on the cluster tree from Definition 2.7. On level 2 of the tree there are six leaves (shaded), e.g., the block $\{0,1\} \times \{6,7\}$.

If the underlying tree T_I is a binary tree, then the block cluster tree $T_{I\times I}$ is a quad-tree. Based on the cluster tree T_I (Construction 2.8) and the η-admissibility condition (2.4), we construct the canonical block cluster tree $T_{I\times I}$ as follows.

Construction 2.11 (Canonical Block Cluster Tree $T_{I\times I}$) *Let the cluster tree T_I be given. We define the block cluster tree $T_{I\times I}$ by* $\operatorname{root}(T) := I \times I$ *and for each vertex $t \times s \in T$ the set of successors by*

$$S(t\times s) := \begin{cases} \emptyset & \text{if } \min\{\#t, \#s\} \le n_{\min} \text{ or } t\times s \text{ is } \eta\text{-admissible (2.4)} \\ S(t)\times S(s) & \text{otherwise.} \end{cases}$$

The block cluster tree $T_{I\times I}$ is the basis for hierarchical matrix format. It defines the partition of the matrix into sub-matrices that are represented in the $R(k)$-matrix format.

Definition 2.12 ($\mathcal{H}$-Matrix) *Let $T := T_{I\times I}$ be a block cluster tree and $k : \mathcal{L}(T) \to \mathbb{N}_0$ a rank distribution. We define the set $\mathcal{H}(T,k)$ of hierarchical matrices ($\mathcal{H}$-matrices) by*

$$\mathcal{H}(T,k) := \left\{ M \in \mathbb{R}^{I\times I} \;\middle|\; \forall t\times s \in \mathcal{L}(T) : \operatorname{rank}(M|_{t\times s}) \le k(t\times s) \right\}.$$

A matrix $M \in \mathcal{H}(T,k)$ is said to be given in $\mathcal{H}$-matrix representation, if for all leaves $t \times s$ with $\#t \le n_{\min}$ or $\#s \le n_{\min}$ the corresponding matrix block $M|_{t\times s}$ is given in full matrix representation and in $R(k)$-matrix representation for the other leaves.

The result of Construction 2.8 followed by Construction 2.11 is a block cluster tree $T_{I\times I}$ for which we can estimate the depth and the sparsity C_{sp} defined next. The sparsity is needed to estimate the storage requirements and complexity of the matrix-vector multiplication of an $\mathcal{H}$-matrix.

Definition 2.13 (Sparsity) *Let $T_{I\times I}$ be a block cluster tree. We define the* sparsity (constant) C_{sp} *of $T_{I\times I}$ by*

$$C_{\text{sp}} := \max_{t\in T_I} \#\{s \in T_I \mid t\times s \in T_{I\times I}\}.$$

So far, we have not posed any condition on the locality of the supports of the basis functions φ_i. If all the supports cover the whole domain Ω, then the only admissible block $t \times s \subset I \times I$ is $t \times s = \emptyset$. Therefore, we demand the locality of the supports.

Assumption 2.14 (Locality) *We assume that the supports are locally separated in the sense that there exist two constants C_{sep} and $n_{\min}$ such that*

$$\max_{i \in I} \#\{j \in I \mid \text{dist}(\text{supp}\varphi_i, \text{supp}\varphi_j) \leq C_{\text{sep}}^{-1}\text{diam}(\text{supp}\varphi_i)\} \leq n_{\min}. \tag{2.5}$$

The left-hand side is the maximal number of basis functions with 'relatively close' supports (see Figure 3). Note that we will apply Construction 2.8 with a parameter $n_{\min}$ that satisfies (2.5).

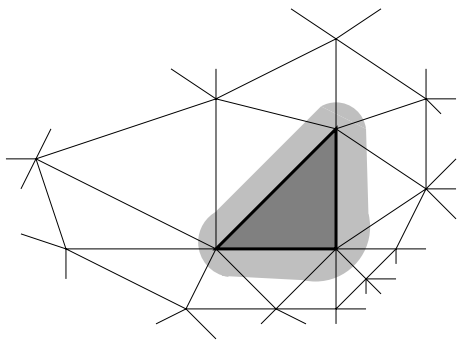


Figure 3: The triangle $\Omega_i := \text{supp}\varphi_i$ under consideration is dark grey. The area with a distance of $C_{\text{sep}}^{-1}\text{diam}(\Omega_i)$ is light grey ($C_{\text{sep}} := 4\sqrt{2}$). Here, 15 triangles (including Ω_i) are 'rather close' to Ω_i.

Based upon Assumption 2.14 the sparsity constant C_{sp} as well as the depth p of the block cluster tree $T_{I \times I}$ is estimated in [10, Lemma 4.5]. It should be noted, that $C_{\text{sp}} = \mathcal{O}(\eta^{-3})$ and thus the choice of η in the admissibility condition (2.4) enters the estimate in a critical way.

The estimates for the storage requirements and the complexity of the matrix-vector multiplication of an $\mathcal{H}$-matrix depend only on the cardinality of I and the depth and sparsity of the block cluster tree T:

Lemma 2.15 (Storage, Lemma 2.4 in [10]) *Let T be a block cluster tree based on the cluster tree T_I with sparsity constant C_{sp} (Definition 2.13) and minimal block size $n_{\min}$. Then the storage requirements $N_{\mathcal{H},St}(T,k)$ for an $\mathcal{H}$-matrix $M \in \mathcal{H}(T,k)$ are bounded by*

$$N_{\mathcal{H},St}(T,k) \leq 2(1 + \text{depth}(T))C_{\text{sp}} \max\{k, n_{\min}\}\#I.$$

Lemma 2.16 (Matrix-Vector Product) *Let T be a block cluster tree. The complexity $N_{\mathcal{H}\cdot v}(T,k)$ of the matrix-vector product in the set of $\mathcal{H}$-matrices can be bounded from above and below by*

$$N_{\mathcal{H},St}(T,k) \leq N_{\mathcal{H}\cdot v}(T,k) \leq 2N_{\mathcal{H},St}(T,k).$$

2.4 Numerical Test: ACA and ACA+

In order to illustrate the compression by using the $\mathcal{H}$-matrix format and also to underline the advantages of the new ACA+ algorithm to assemble the $R(k)$-matrix approximation we consider the model problem from Section 2.1 with the kernel function g^{DLP} of the double layer potential. The geometry is the surface of a (three-dimensional) crank shaft (Figure 4) and stems from the NETGEN package of Joachim Schöberl. We use $n = 28288$ panels for the

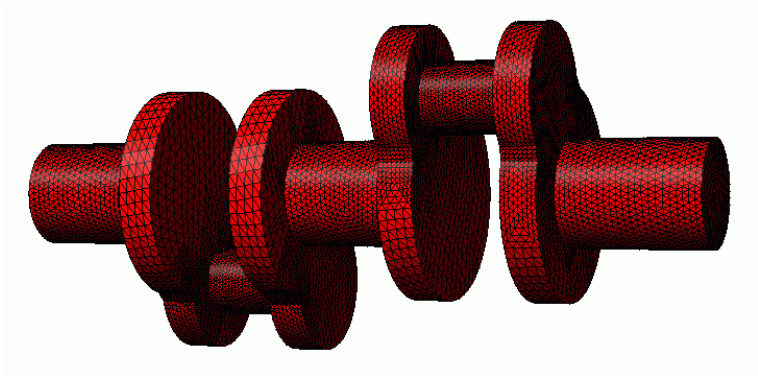


Figure 4: The surface of the crank shaft from NETGEN with $n = 28288$ panels.

discretisation and constant basis functions φ_i on each panel. The stiffness matrix G from (2.3) is compressed in the $\mathcal{H}$-matrix format by use of the ACA algorithm (Construction 2.2) with the standard partial pivot search and the new ACA+ pivot search. Both use $\varepsilon := 10^{-3}$ in the stopping criterion, and the ACA+ algorithm switches to ACA as soon as the stopping criterion is fulfilled with 3ε, so that both algorithms terminate with almost the same target accuracy. The nearfield integration uses the automatic quadrature of Erichsen and Sauter [8]. For the admissibility parameter η we choose the values $\eta \in \{0.5, 1.0, 2.0, 4.0\}$. The minimal blocksize is $n_{min} = 20$ for the geometrically balanced clustering (Construction 2.8). The results are contained in Table 2 and were obtained by the HLIB package [4] on a SUN UltraSPARC machine with a 900 MHz CPU. We conclude that η should be taken large enough. For the geometrically balanced clustering the maximal ratio diam/dist that can occur is ≈ 6 so that $\eta = 4$ is effectively the maximal parameter for the admissibility condition. Smaller values of η result in higher computational complexity and increased storage requirements.

		Time [Sec.]	Storage [KB/DoF]	Accuracy $\frac{\|G-\tilde{G}\|}{\|G\|}$	$\frac{\|x-(\tilde{G})^{-1}Gx\|_{L^2}}{\|x\|_{L^2}}$
$\eta = 0.5$	ACA	248.7	24.3	2.1×10^{-3}	5.3×10^{-4}
	ACA+	259.8	23.9	2.4×10^{-4}	2.7×10^{-5}
$\eta = 1$	ACA	174.2	16.7	6.3×10^{-3}	6.7×10^{-4}
	ACA+	174.3	16.2	1.1×10^{-3}	6.8×10^{-5}
$\eta = 2$	ACA	143.8	13.6	6.1×10^{-3}	9.6×10^{-5}
	ACA+	140.5	13.1	9.8×10^{-4}	1.2×10^{-4}
$\eta = 4$	ACA	140.2	13.2	1.6×10^{-3}	1.4×10^{-4}
	ACA+	136.8	12.7	5.5×10^{-4}	1.3×10^{-4}

Table 2: Comparison of ACA and ACA+ for the crank shaft example and different values of η.

3 Recompression of $\mathcal{H}$-Matrices

The standard $\mathcal{H}$-matrix compression by blockwise low rank approximation (via interpolation, ACA, etc.) yields a matrix $\tilde{G}$ where we still have to store a large amount of data per degree of freedom. For the crank shaft example from Section 2.4 we need approximately 12.7 KB per degree of freedom. This can be reduced by applying a recompression scheme. The first recompression is done for each $R(k)$-matrix block of the matrix separately, the second recompression changes the entire block structure. The benefit is twofold: the amount of data to be stored is reduced (hence the name) but also the $\mathcal{H}$-matrix arithmetic (addition, multiplication, inversion) is sped up.

3.1 Blockwise Recompression

The first recompression method is commonly used when applying sub-optimal rank revealing algorithms. Each assembled block in the $R(k)$-matrix format is immediately decomposed by the SVD. Since the block is already given in the factorised form AB^T, $A \in \mathbb{R}^{n\times k}, B \in \mathbb{R}^{m\times k}$, the SVD of such a matrix can be computed in $\mathcal{O}(k^2(n+m))$ [10]. All singular values σ_i with

$$\sigma_i \leq \varepsilon\, \sigma_1$$

are discarded and the rank thereby reduced. Here, we choose a parameter $\varepsilon := 2 \times 10^{-3}$ different from the one in Construction 2.2 (the stopping criterion in Construction 2.2 is just a heuristic).

We demonstrate the effect of this first recompression by the crank shaft example from Section 2.4. Here, we consider only the case $\eta = 4$ and the new ACA+ approximation. The results are contained in Table 3.

The first recompression reduces the storage requirements per degree of freedom from 12.7KB down to 8.9KB and retains the same approximation quality.

	Time [Sec.]	Storage [KB/DoF]	Accuracy $\frac{\|G-\tilde{G}\|}{\|G\|}$	$\frac{\|x-(\tilde{G})^{-1}Gx\|_{L^2}}{\|x\|_{L^2}}$
ACA+	136.8	12.7	5.5×10^{-4}	1.3×10^{-4}
Recompression	13.5	8.9	5.8×10^{-4}	1.1×10^{-4}

Table 3: The first recompression applied the the crank shaft example with $\eta = 4$.

Since this is blockwise a best approximation, the result is (almost) a best approximation in the given block structure. After the first recompression, it doesn't matter where the initial approximation stems from (e.g., ACA or interpolation).

The recompression can be applied "on-the-fly", i.e., after the assembly of each $R(k)$-matrix block by ACA+ we immediately apply the SVD and store only the recompressed block. Therefore, only the time needed to generate the initial approximation is crucial.

3.2 Coarsening of the Block Structure

While the first recompression reduces the blockwise rank of the $\mathcal{H}$-matrix approximation, the second recompression aims at a coarsening of the entire block structure of the $\mathcal{H}$-matrix. The reason why the block structure from Construction 2.8 and 2.11 does not necessarily yield the optimal partition is threefold:

First, the parameter η from the admissibility condition (2.4) might be too small (and it enters the complexity estimates in the power 6). For the discretisation this is not crucial but for the $\mathcal{H}$-matrix arithmetic it is. Our recompression scheme will automatically choose the right blocks, so that only the extra time for the assembly of the stiffness matrix is increased, but the storage requirements stay the same independently of η.

Second, blocks that are not admissible might be regarded as admissible, because the admissibility condition (2.4) is sufficient but not necessary. This was first observed in [14] under the name *weak admissibility*.

Third, the block cluster tree $T_{I\times I}$ based on the given cluster tree T_I does not take the accuracy of the discretisation and ACA+ compression into account. Asymptotically (k large enough or ε small enough) the optimal block structure might be that from the standard admissibility (2.4), but for all practical purposes the optimal structure is coarser.

The block cluster tree $T'_{I\times I}$ that we construct corresponds to an *optimal admissibility* with respect to the storage requirements. The tree $T_{I\times I}$ from Construction 2.11 serves as a fine initial guess from which we construct the optimised tree.

Remark 3.1 (Weak Admissibility and Optimal Admissibility) *The weak admissibility condition is an a priori condition, this means one can construct the block cluster tree $T_{I\times I}$ based on this condition and use this tree for the $\mathcal{H}$-matrix arithmetic. This is important for FEM applications, because there the*

initial stiffness matrix is sparse and only the inverse, which has to be computed, is a dense matrix. Therefore the matrix entries of the inverse to be stored in the $\mathcal{H}$-matrix format are not known.

The situation is different for BEM applications. Here, the matrix entries are known and one can construct an (almost) optimal block cluster tree $T'_{I\times I}$ a posteriori. For the purpose of discretisation this reduces the amount of storage, but for the purpose of the $\mathcal{H}$-matrix arithmetic (addition, multiplication, inversion) this also reduces the computational time. This is a pure algebraic construction and no knowledge of the continuous problem is needed.

Construction 3.2 ($\mathcal{H}$-Matrix Recompression) *Let $T_{I\times I}$ be a block cluster tree with admissible leaves in the sense that either (2.4) holds for the corresponding clusters or the block is small enough (c.f. $n_{\min}$ in Definition 2.12). Let $\varepsilon > 0$ be a prescribed accuracy (later we will comment on the choice of ε).*

We call the procedure `coarsen` *with parameters* `coarsen`*$(I \times I)$. Implicitly, we also use the son-relation* $\mathrm{Sons}(\cdot)$ *for the tree $T_{I\times I}$ and* $\mathrm{Son}'(\cdot) := \mathrm{Sons}(\cdot)$ *for the tree $T'_{I\times I}$ to be constructed, as well as the matrix $\tilde{G}$ and the desired accuracy ε.*

`coarsen`($t \times s$: NODE IN THE BLOCK CLUSTER TREE)

1. CALL `coarsen`$(t' \times s')$ FOR ALL SONS $t' \times s'$ OF $t \times s$.
2. IF AT LEAST ONE OF THE SONS $\mathrm{Sons}'(t \times s)$ IS NOT A LEAF THEN RETURN, OTHERWISE
3. COMPUTE A SVD OF THE SUBMATRIX $\tilde{G}|_{t\times s}$;
   DISCARD ALL SINGULAR VALUES $\sigma_i \le \varepsilon\sigma_1$ AND STORE THE RESULT R IN THE $R(k)$-MATRIX FORMAT;
4. IF THE STORAGE REQUIREMENTS OF $\tilde{G}|_{r\times s}$ ARE LARGER THAN THOSE OF R THEN
   SET $\tilde{G}|_{t\times s} := R$;
   SET $\mathrm{Sons}'(t \times s) := \{\}$;
   RETURN.

The computation of the singular value decomposition in step 3 of Construction 3.2 is simple, because we have verified in step 2 that the successors of $t \times s$ are all leaves. Therefore we have to decompose an $R(k)$-matrix $\tilde{G}|_{t\times s}$ of rank at most $k = \sum_{t'\times s' \in \mathrm{Sons}(t\times s)} \mathrm{rank}(\tilde{G}|_{t'\times s'})$, where each of the leaves $\tilde{G}|_{t'\times s'}$ is given in the $R(k)$-matrix format.

The effect of the recompression with regard to the storage requirements can be seen in Table 4, where we were able to reduce the storage requirements per degree of freedom from the crank shaft example of Section 2.4 from 8.9KB after the first recompression down to 6.4KB by Construction 3.2. Even if the initial compression is done by tensor interpolation of the underlying kernel function [5] with a small parameter $\eta = 0.5$, the matrix after the second recompression is almost the same and requires only 6.4KB storage per degree of freedom, cf. Table 4.

	Time [Sec.]	Storage [KB/DoF]	$\frac{\|G-\tilde{G}\|}{\|G\|}$	$\frac{\|x-(\tilde{G})^{-1}Gx\|_{L^2}}{\|x\|_{L^2}}$
ACA+	136.8	12.7	5.5×10^{-4}	1.3×10^{-4}
1st Rec.	13.5	8.9	5.8×10^{-4}	1.1×10^{-4}
2nd Rec.	85.6	6.4	6.0×10^{-4}	2.2×10^{-4}
Interpol.	694.9	189.5	4.6×10^{-4}	1.3×10^{-4}
1st Rec.	433.5	18.0	4.6×10^{-4}	1.2×10^{-4}
2nd Rec.	188.0	6.4	5.3×10^{-4}	3.3×10^{-4}

Table 4: The first recompression (with $\varepsilon = 2 \times 10^{-3}$) and second recompression (with $\varepsilon = 2 \times 10^{-3}$) applied to the crank shaft example. On top the ACA+ approximation with $\eta = 4$, below a tensor interpolation with $\eta = 0.5$.

The block structure (with parameter $\eta = 1.0$) before and after the coarsening is depicted in Figure 5. The effect of the coarsening can best be seen when

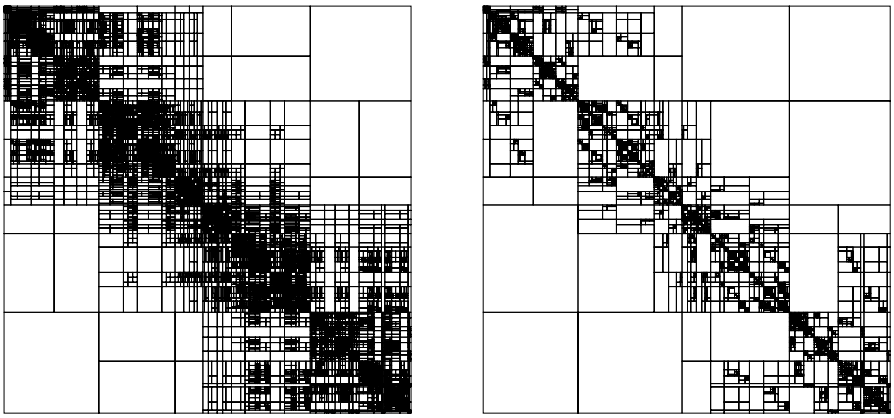


Figure 5: The block structure of the $\mathcal{H}$-matrix $\tilde{G}$ before and after the second recompression for the crank shaft example with $\eta = 1.0$.

applying the formatted $\mathcal{H}$-matrix arithmetic [12, 13, 10]. Here, we just add, multiply, invert or decompose the BEM stiffness matrix $\tilde{G}$ in the $\mathcal{H}$-matrix format for the purpose of illustrating the speed-up-factor achieved by the second recompression. The example is again the crank shaft from Section 2.4 ($\eta = 4$) and we use the formatted arithmetic defined in [10] with fixed rank as it is given by the recompressed matrix $\tilde{G}$. The results are contained in Table 5. The multiplication and inversion are sped up by a factor of 3 for an accurate recompression. Even the matrix-vector multiplication is two times faster for the coarsened matrix. The last column contains the times for the computation of an LU decomposition of $\tilde{G}$ within the $\mathcal{H}$-matrix format (this was first proposed

	$M \cdot v$	$M + M$	$M \cdot M$	M^{-1}	$M = LU$
1st Rec.	0.53	15	1233	1258	391
2nd Rec., $\varepsilon = 2 \times 10^{-3}$	0.27	17	421	407	155
2nd Rec., $\varepsilon = 2 \times 10^{-2}$	0.15	6.1	143	137	57
2nd Rec., $\varepsilon = 6 \times 10^{-1}$	0.02	0.5	4.9	5.3	2.2

Table 5: Time (in seconds) for the formatted $\mathcal{H}$-matrix arithmetic before and after the second recompression from Construction 3.2 for the crank shaft example.

in [16] and later used in [4, 2]). In the next section we consider the efficient solution of the discrete system in more detail.

4 Solution of the Discrete System

In the previous section we were concerned with the discretisation of the integral operator in the $\mathcal{H}$-matrix format and the fast matrix-vector multiplication. In order to solve the discrete system efficiently, we need the formatted $\mathcal{H}$-matrix arithmetic. The algorithms that perform this arithmetic as well as their complexity estimates for general cluster trees are already published in [10], especially the formatted addition and multiplication of $\mathcal{H}$-matrices. The word "formatted" in this context means that the result of the arithmetic operation $A + B$ or $A \cdot B$ has to be projected onto the set $\mathcal{H}(T_{I \times I}, k)$. For a fixed target tree T and rank k we denote the formatted addition by $\oplus$ and the formatted multiplication by $\odot$. Both are of complexity $\mathcal{O}(k^2 n \log^2 n)$, where k is the (maximal) rank for the $R(k)$-matrix blocks.

4.1 Direct Solution

For a direct solver (up to the initial compression error), we do not need extra memory. Instead, we overwrite the memory allocated for the stiffness matrix $\tilde{G}$ in $\mathcal{H}$-matrix format by the two factors L, U of an LU decomposition. The matrix $\tilde{G}$ is not needed anymore since the solution x for a right-hand side f is given by $x := (LU)^{-1} f$ without any iterative process. The disadvantage of this approach is that the time for the setup of the two factors L, U is of complexity $\mathcal{O}(k^2 n \log^2 n)$, where k is the average rank in the $R(k)$-matrix blocks. Since no evaluations of the kernel and thus no integration is involved this might still be faster than the assembly of the stiffness matrix. Once the two factors are computed, we can solve the system in $\mathcal{O}(kn \log n)$, such that for several right-hand sides the overall complexity of this approach might be less than for an iterative solution of the system. The results for the crank shaft example are contained in Table 6.

As a second example we consider the single layer potential operator on the same crank shaft surface as before (discretised with constant basis functions).

	Assembly & Rec.	$\tilde{G} = LU$	$\frac{\|x-(LU)^{-1}Gx\|_{L^2}}{\|x\|_{L^2}}$	$(LU)^{-1} \cdot f$
DLP	236	220	4.6×10^{-3}	0.32
SLP	125	79	4.0×10^{-3}	0.27

Table 6: Time (in seconds) for the setup and solution of the crank shaft example for the double layer and single layer potential.

Here, the system matrix is ill-conditioned and a simple iterative solution (e.g., by the conjugate gradient method) requires many evaluations of the stiffness matrix $\tilde{G}$. The results for an - up to the initial compression error - exact Cholesky decomposition of $\tilde{G}$ in the $\mathcal{H}$-matrix format are contained in Table 6.

4.2 Iterative Solution and Coarse Preconditioning

If the solution of the discrete system is only needed for few right-hand sides, then it is advantageous to compute only a coarse preconditioner. In order to reduce the arithmetic complexity, we first generate a coarse approximation $\tilde{G}'$ of the stiffness matrix $\tilde{G}$ by use of Construction 3.2 with parameter ε larger than for the assembly of the stiffness matrix, e.g., $\varepsilon = 0.3$. In Figure 6 we have depicted the structure of the coarsened $\mathcal{H}$-matrix for different parameters ε.

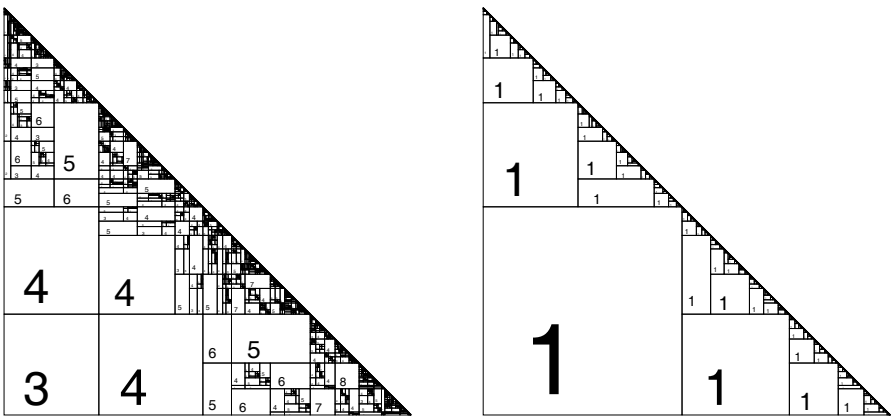


Figure 6: The block structure of the lower triangular part of $\tilde{G}$ recompressed with $\varepsilon = 0.01$ and $\varepsilon = 0.3$ for the crank shaft example. In each block the rank of the approximation is denoted.

For this coarse approximation we compute a Cholesky / LU decomposition in the $\mathcal{H}$-matrix format with a complexity of $\mathcal{O}(n \log^2 n)$. The results are contained in Table 7. We observe that the setup of a coarse preconditioner ($\varepsilon = 0.1$) takes less than 5% of the time for the assembly and initial compression of the stiffness

SLP $n = 28288$	Assembly & Recompression	Cholesky Decomposition	GMRES Time [Sec.]	Steps
$\varepsilon = 3 \times 10^{-3}$	125	47.0	4.1	5
$\varepsilon = 1 \times 10^{-2}$	125	29.8	5.1	7
$\varepsilon = 3 \times 10^{-2}$	125	16.0	7.8	12
$\varepsilon = 1 \times 10^{-1}$	125	5.0	11.7	20
$\varepsilon = 3 \times 10^{-1}$	125	**1.0**	**15.3**	28
no precond.	125	0	103.7	180

Table 7: Time (in seconds) for the setup and iterative solution (Cholesky-preconditioned GMRES until relative residual $< 10^{-6}$) of the crank shaft example.

matrix, so that the system can be solved in 20 iterative steps, again less than 10% of the time for the assembly. For this small problem an unpreconditioned GMRES [18] is able to compute the solution in 180 steps (104 seconds).

Since the discretisation in the compressed $\mathcal{H}$-matrix format is of complexity $\mathcal{O}(n \log n)$ while the setup of the preconditioner has a complexity of $\mathcal{O}(n \log^2 n)$, one could expect that for large enough problems the setup will exceed the time for the assembly. However, the compression has to be accurate up to the discretisation error while the preconditioner can be much coarser. We illustrate this by refining the surface triangulation of the crank shaft from Section 2.4 regularly to obtain a grid with $n = 113152$ panels. The time for the setup is reported in Table 8, and we observe that for the larger problem the setup and solution for one right-hand side can be done in 10% of the time for the assembly of the (compressed) stiffness matrix. An unpreconditioned solve by GMRES for a single right-hand side takes 242 steps (649 seconds), i.e., longer than the setup of the stiffness matrix.

SLP $n = 113152$	Assembly & Recompression	Cholesky Decomposition	GMRES Time [Sec.]	Steps
$\varepsilon = 3 \times 10^{-3}$	542	290.7	15.9	6
$\varepsilon = 1 \times 10^{-2}$	542	193.2	18.8	8
$\varepsilon = 3 \times 10^{-2}$	542	127.0	26.4	13
$\varepsilon = 1 \times 10^{-1}$	542	40.3	41.7	24
$\varepsilon = 3 \times 10^{-1}$	542	**6.3**	**50.2**	32
no precond.	542	0	648.8	242

Table 8: Time (in seconds) for the setup and iterative solution (Cholesky-preconditioned GMRES until relative residual $< 10^{-6}$) of the crank shaft example.

References

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

[2] M. Bebendorf. Hierarchical LU decomposition based preconditioners for BEM. Technical Report 28, Max Planck Institute for Mathematics in the Sciences, 2004.

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

[4] S. Börm and L. Grasedyck. HLib - a library for $\mathcal{H}$- and $\mathcal{H}^2$-matrices, 1999. Available at: `http://www.hlib.org`.

[5] S. Börm and L. Grasedyck. Low-rank approximation of integral operators by interpolation. *Computing*, 72:325–332, 2004.

[6] S. Börm, L. Grasedyck, and W. Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27:405–422, 2003.

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

[8] S. A. Sauter, S. Erichsen. Efficient automatic quadrature in 3-d Galerkin BEM. *Comput. Meth. Appl. Mech. Eng.*, 157:215–224, 1998.

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

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

[11] 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.

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

[13] 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.

[14] W. Hackbusch, B. Khoromskij, and R. Kriemann. Hierarchical matrices based on a weak admissibility criterion. Technical Report 2, Max Planck Institute for Mathematics in the Sciences, 2003. submitted to Computing.

[15] 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.

[16] 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.

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

[18] Y. Saad, M. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[19] S. A. Sauter. Variable order panel clustering (extended version). Technical Report 52, Max-Planck-Institut für Mathematik, Leipzig, Germany, 1999.

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

Lars Grasedyck
Max Planck Institut für Mathematik in den Naturwissenschaften
Inselstrasse 22-26
D-04109 Leipzig
Germany
`lgr@mis.mpg.de`