SIAM J. SCI. COMPUT.
Vol. 0, No. 0, pp. S24–S50

DATA-DRIVEN CONSTRUCTION OF HIERARCHICAL MATRICES WITH NESTED BASES*

DIFENG CAI†, HUA HUANG‡, EDMOND CHOW‡, AND YUANZHE XI†

Abstract. Hierarchical matrices provide a powerful representation for significantly reducing the computational complexity associated with dense kernel matrices. For example, the fast multipole method (FMM) and its variants are highly efficient when the kernel function is related to fundamental solutions of classical elliptic PDEs. For general kernel functions, interpolation-based methods are widely used for the efficient construction of hierarchical matrices. In this paper, we present a fast hierarchical data reduction (HiDR) procedure with $O(n)$ complexity for the memory-efficient construction of hierarchical matrices with nested bases where n is the number of data points. HiDR aims to reduce the given data in a hierarchical way so as to obtain $O(1)$ representations for all nearfield and farfield interactions. Based on HiDR, a linear complexity $\mathcal{H}^2$ matrix construction algorithm is proposed. The use of data-driven methods enables better efficiency than other general-purpose methods and flexible computation without accessing the kernel function. Experiments demonstrate significantly improved memory efficiency of the proposed data-driven method compared to interpolation-based methods over a wide range of kernels. For the Coulomb kernel, the proposed general-purpose algorithm offers competitive performance compared to FMM and its variants, such as PVFMM. The data-driven approach not only works for general kernels but also leads to much smaller precomputation costs compared to PVFMM.

Key words. hierarchical matrix, kernel matrix, data-driven construction, complexity analysis, data reduction, Gaussian

MSC codes. 15A23, 68W25, 65D40

DOI. 10.1137/22M1500848

1. Introduction.

In various applications, the pairwise interaction between two objects is characterized by a nonlocal kernel function. A system of n objects then gives rise to an n-by-n dense kernel matrix. Such matrices arise frequently in integral equations [23, 42, 43], astrophysics [4], statistics [27, 18], machine learning [7, 24], etc. A computational challenge is that the naive computational or storage cost associated with the dense kernel matrix is at least $O(n^2)$. For the Coulomb kernel, pioneering work such as the fast multipole method (FMM) [42, 43, 29] and the Barnes–Hut algorithm [4] use multilevel approximation to successfully reduce the cost to linear or quasi-linear complexity. Variants of FMM such as KIFMM [47] and PVFMM [38] were later developed to extend FMM to kernels related to fundamental solutions of classical constant-coefficient elliptic PDEs. Further computational savings can also be achieved by incorporating singular value decomposition in the multilevel compression [20]. These techniques can be generalized into the powerful

* Received by the editors June 6, 2022; accepted for publication (in revised form) November 1, 2022; published electronically July 13, 2023.
https://doi.org/10.1137/22M1500848
Funding: The work of the first and fourth authors is supported by NSF award OAC 2003720 and RTG grant DMS-2038118. The work of the second and third authors is supported by NSF award OAC 2003683.
† Department of Mathematics, Emory University, Atlanta, GA 30322 USA (dcai7@emory.edu, yxi26@emory.edu).
‡ School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (huangh223@gatech.edu, echow@cc.gatech.edu).

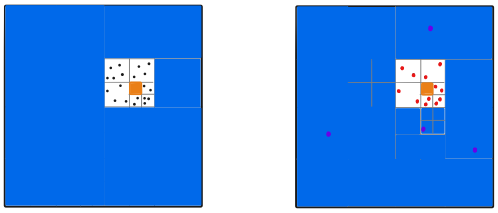
algebraic framework of hierarchically low-rank matrices [33, 34, 9, 31], or hierarchical matrices for short, for efficiently approximating dense kernel matrices associated with *general* kernel functions. Two widely used classes of hierarchical matrices are $\mathcal{H}$ matrices and $\mathcal{H}^2$ matrices. The $\mathcal{H}$ matrix yields an $O(n \log n)$ representation, and the $\mathcal{H}^2$ matrix yields an optimal $O(n)$ representation for approximating the kernel matrix and computing the matrix-vector multiplication. For dense kernel matrices, the efficient construction of these hierarchical representations associated with general kernels remains a challenging problem. In this paper, we address this issue for the $\mathcal{H}^2$ matrix representation. For general kernels, interpolation-based methods are often adopted as a black-box tool used in hierarchical matrix construction. However, as will be demonstrated in later sections, the use of interpolation nodes (or points outside the given dataset) is in general *not* robust with respect to the kernel function or the data distribution and may lead to loss of accuracy. We present a general-purpose data-driven framework for hierarchical matrix construction that resolves these issues and offers improved efficiency.

Given a set of points and a kernel function, hierarchical matrix construction starts with an adaptive partitioning of the data. The partitioning can be encoded by a tree structure in which each node represents a subset generated in the adaptive partitioning procedure. For example, the root node corresponds to the entire set of n points, and its children nodes correspond to the subsets generated from the first partitioning of the dataset. For hierarchical matrix construction, directly compressing the submatrix corresponding to a pair of nodes/subsets will be inefficient since one of the subsets may contain $O(n)$ points (see Figure 1.1(a)) and the total cost for all such pairs will be at least $O(n^2)$. We propose to first process the tree-structured data to obtain a reduced representation in which each node only corresponds to a small subset with $O(1)$ points while the farfield corresponds to $O(1)$ points as well. These $O(1)$ subsets are called *representor sets* in [6]. See Figure 1.1(b). Using representor sets, the multilevel compression of the kernel matrix can be rapidly computed. To achieve optimal efficiency, we design a hierarchical data reduction (HiDR) procedure with computational cost of $O(n)$. Different from many existing approaches that create out-of-data points to facilitate the low-rank compression (cf. [47, 9, 38]), the proposed procedure operates entirely on the given data without using artificial points outside the data or accessing the kernel function. Thus, the approach works for general kernel functions in addition to those related to fundamental solutions of elliptic PDEs. Furthermore, no *algebraic* compression is performed in the data reduction. Hence, the approach is termed *data-driven*. We show how to incorporate HiDR in hierarchical matrix construction to obtain a fast algorithm for general kernels. Numerical experiments demonstrate the competitive performance of the new method in terms of generality, memory use, speed, and accuracy.

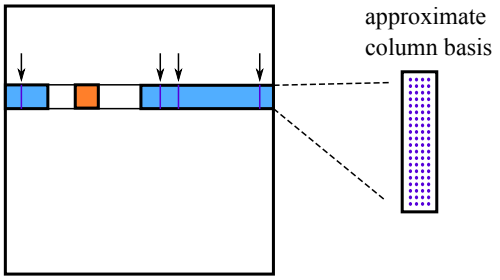
The rest of the manuscript is organized as follows. Section 2 reviews hierarchical matrix structures and existing methods for constructing hierarchical matrices. Section 3 introduces the HiDR algorithm. Based on this HiDR, the complete hierarchical matrix construction is presented in section section 4. Section 5 presents numerical experiments to investigate different data reduction techniques. Numerical results for the proposed data-driven hierarchical matrix construction are given in section 6. Concluding remarks are drawn in section 7.

The notation used in this paper is listed below:

- $|x - y|$ denotes the Euclidean distance between x and y in $\mathbb{R}^d$.
- $\mathrm{diam}(X)$ denotes the diameter of set X, i.e., $\max_{x,y \in X} |x - y|$.



(a) Left: the farfield (shown in blue) of the orange box contains $O(n)$ points. Right: the $O(n)$ points are reduced to a subset of $O(1)$ points. (For another orange box, the farfield is different and the reduced subset is generally a different set of points.)



(b) The kernel matrix is shown, highlighting the interaction between points in the orange region and points in its farfield. The arrows point to columns corresponding to the points selected in the $O(1)$ subset. These columns form an approximate basis for the farfield.

FIG. 1.1. *Farfield data reduction as a way to construct an approximate column basis for the farfield block row in the kernel matrix.*

- $\mathrm{dist}(X_1, X_2)$ denotes the distance between sets X_1 and X_2, i.e., $\min_{x\in X_1, y\in X_2} |x-y|$.
- $\mathrm{card}(X)$ denotes the cardinality of set X.
- X^* denotes a representor set of X.

2. Review of hierarchical matrix representations. Given a kernel function $\kappa(x,y)$ and a dataset $X = \{x_1, \ldots, x_n\}$, the associated kernel matrix is defined by

$$K = [\kappa(x_i, x_j)]_{i,j=1}^n.$$

Dense kernel matrices are ubiquitous and arise in various applications where the kernel function measures the interaction between objects. In Coulombic N-body simulations, $\kappa(x,y) = \frac{1}{|x-y|}$. In boundary integral equations, $\kappa(x,y)$ can take very different forms, including $\Phi(x,y)$, $\nabla_{v_y}\Phi(x,y)$, where $\Phi(x,y)$ denotes the fundamental solution of the underlying differential operator and v_y denotes the unit outer normal at y on the boundary. In certain structured matrix computations such as those involving Cauchy and Cauchy-like matrices, $\kappa(x,y)$ is taken as $\frac{1}{x-y}$ with $x, y \in \mathbb{C}$. In statistics and machine learning, $\kappa(x,y)$ is often taken as the Gaussian kernel $e^{-|x-y|^2}$ or the Laplace kernel $e^{-|x-y|}$. A major computational bottleneck in these applications lies in the $O(n^2)$ cost in storing the dense matrix K and performing operations such as matrix-vector multiplication.

To avoid the high cost in forming K explicitly, hierarchically low-rank matrix representations are used to approximate K. The hierarchical representation is based on the fact that K can be partitioned into blocks in a hierarchical fashion and many blocks are numerically low rank. Low-rank factors are computed for the blocks and are stored in the hierarchical representation to replace original dense blocks. Two widely used hierarchical representations are $\mathcal{H}$ and $\mathcal{H}^2$ [33, 9, 32, 31]. The $\mathcal{H}$ matrix representation in general has a complexity of $O(n \log n)$ in space, while $\mathcal{H}^2$ has the optimal complexity of $O(n)$. The matrix-vector multiplication can be computed in $O(n \log n)$ complexity for $\mathcal{H}$ matrices and in $O(n)$ complexity for $\mathcal{H}^2$ matrices, which is much more efficient than directly multiplying K by a vector.

We review the mathematical description of hierarchical matrices in subsection 2.1. Existing methods for constructing the hierarchical matrices are discussed in subsection 2.2.

2.1. Hierarchical matrix representations. In the following, we review the general algebraic framework of $\mathcal{H}$ and $\mathcal{H}^2$ matrices.

Given a dataset $X = \{x_i\}_{i=1}^n$ in $\mathbb{R}^d$, one builds a tree structure by recursively partitioning X spatially until no more than $m = O(1)$ points are contained in each partitioned subset. The tree encodes the subsets of X generated by the adaptive partitions. Namely, the root node is associated with X, and its children nodes are associated with subsets of X created by the first partition. Inductively, each node is associated with a nonempty subset of X. For node i, we denote by X_i the subset associated with that node. Hence, $X_{\text{root}} = X$. An illustration is given in Figure 2.1.

The tree structure automatically yields a blockwise partition of the matrix K. See Figure 2.2 for an example with data points lying inside an interval. We define $K_{i,j} = [\kappa(x,y)]_{\substack{x \in X_i \\ y \in X_j}}$. Block $K_{i,j}$ is approximated by a low-rank factorization if (i,j) satisfies an admissiblity condition that requires X_i and X_j to be separated from each other to some extent. For example, the pair (i,j) is considered admissible if

$$\operatorname{diam}(X_i) + \operatorname{diam}(X_j) \le 2\tau |a_i - a_j| \tag{2.1}$$

for some $\tau \in [0, 0.7]$, where a_i denotes the center of the box associated with X_i in the partition. The admissiblity condition in (2.1) is used in [46, 15]. The parameter τ, often called the *separation ratio* [46], controls how separated the two subsets are. Smaller τ implies better separation, and the value 0.7 can be replaced by other values less than 1 (but not close to 1). For any admissible (i,j), the corresponding submatrix $K_{i,j}$ is called an *admissible block*. An admissible block is also referred to as a *farfield* block. Nonadmissible blocks are referred as *nearfield* blocks. For a subset $X_i \subset X$, the union of all X_j that are separated from X_i in the sense of (2.1) is called the

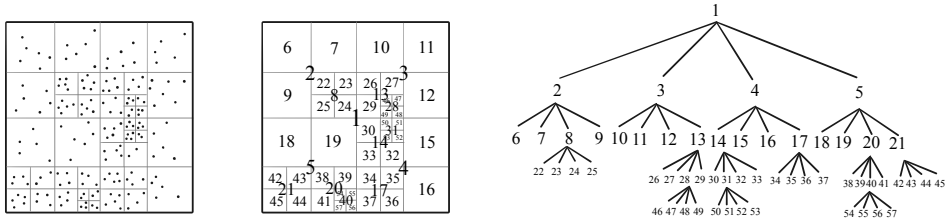


FIG. 2.1. *Adaptive partitioning of the dataset (left), label for each subset (middle), and the associated partition tree (right).*

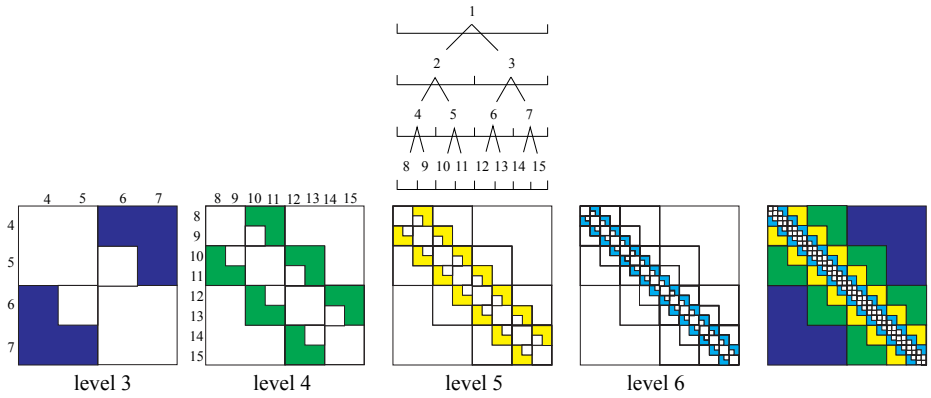


FIG. 2.2. *Hierarchical matrix structure for the one-dimensional problem (left to right): tree (only the top four levels are plotted); admissible blocks (colored) at levels* 3, 4, 5, *and* 6*; and all admissible blocks (colored) in the kernel matrix.*

farfield of X_i. Points that are not in the farfield of X_i constitute the *nearfield* of X_i. In hierarchical matrices, each admissible block is approximated by a low-rank factorization,

$$K_{i,j} \approx U_i B_{i,j} V_j^T,$$

where U_i, V_j are column and row basis matrices and $B_{i,j}$ is called a *coupling matrix*. The maximum column size of all basis matrices U_i and V_j is the approximation rank for K. A larger approximation rank yields a more accurate approximation. For each node i, the *interaction list* of i consists of nodes j such that X_j is well separated from X_i but, for the parent p of j, X_p is not well separated from X_i. The interaction list specifies the blockwise partition of the matrix in which each admissible block has a low-rank approximation. For example, in Figure 2.2, the interaction list of node 4 is $\{6,7\}$; the interaction list of node 10 is $\{8,12,13\}$. The number of elements in any interaction list is bounded by the so-called sparsity constant c_{sp}, which is independent of n. The blockwise low-rank representation yields the $\mathcal{H}$ matrix structure, which stores all nearfield blocks and low-rank factors $U_i, B_{i,j}, V_j$. $\mathcal{H}$ matrices generally admit $O(n \log n)$ storage complexity. The more refined $\mathcal{H}^2$ structure requires the basis matrices to be *nested*. The nested bases property states that if node p has children $c_1, \ldots, c_k$, then there exist *transfer matrices* R_{c_i}, W_{c_i} such that

$$U_p = \begin{bmatrix} U_{c_1} R_{c_1} \\ \vdots \\ U_{c_k} R_{c_k} \end{bmatrix}, \quad V_p = \begin{bmatrix} V_{c_1} W_{c_1} \\ \vdots \\ V_{c_k} W_{c_k} \end{bmatrix}.$$

See Figure 2.3 for an illustration. The sizes of the transfer matrices are equal to r if rank-r factorizations are used for approximations to admissible blocks. In practice, one uses $r = O(1)$ independent of n. Due to the nested bases property, only basis matrices U_i, V_i associated with leaf nodes need to be stored in an $\mathcal{H}^2$ representation, in addition to all transfer matrices (whose row and column sizes are $O(1)$) and nearfield blocks. This results in $O(n)$ storage cost.

Note that the above complexity estimates for storing hierarchical representations assume that the low-rank factors have already been computed. In practice, comput-

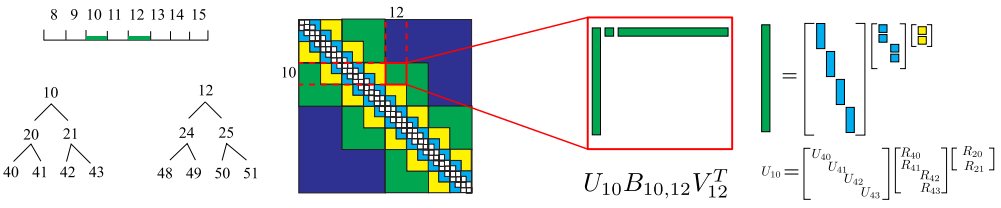


FIG. 2.3. *Admissible block* (10, 12) *and nested bases.*

ing these hierarchical low-rank factors is usually the most costly step, compared to applying the hierarchical representation to a vector. Extensive research has focused on the efficient computation of the hierarchical representation. We review several state-of-the-art methods in subsection 2.2.

2.2. General-purpose methods. For an n-by-n kernel matrix K associated with a general kernel function $\kappa(x, y)$, the hierarchical matrix representation can be computed in linear or quasi-linear time with a variety of techniques, including interpolation [33, 31], adaptive cross approximation [5, 6] and its high-performance extensions [37, 48], hybrid cross approximation [8], SMASH [15], etc. These methods work for general kernels (for example, nonsymmetric, nontranslationally invariant). For special kernel functions, such as fundamental solutions of certain elliptic PDEs, analytic methods such as the FMM [42, 43, 29, 46, 19] and its variants [1, 47, 38, 20] can be used to construct a hierarchical matrix representation efficiently. Two notable variants, KIFMM [47] and PVFMM [38], extend the original FMM [42, 43, 29] by solving integral equations on certain surfaces to obtain blockwise low-rank approximation instead of using an analytic expansion of the kernel function.

When constructing the hierarchical matrix with *nested bases*, one needs to compute basis matrices U_i, V_i for submatrices $K_{X_iY_i}$ that account for the interaction between X_i with $O(1)$ points and its entire farfield Y_i with $O(n)$ points (see Figure 1.1(a)). In order to achieve $O(n)$ optimal complexity for building the hierarchical representation, each basis matrix U_i (as well as V_i) must be computed with $O(1)$ complexity. This requires that the matrix $K_{X_iY_i}$, which is $O(1)$ by $O(n)$ in size, must not be formed. For general kernels, a widely used technique to construct a column basis matrix U_i in $O(1)$ complexity is based on interpolation. For $K_{X_iY_i}$, interpolating the kernel function $\kappa(x, y)$ at r nodes $Q = \{q_1, \ldots, q_r\}$,

$$\kappa(x, y) \approx \sum_{i=1}^{r} \kappa(q_i, y) L_i(x),$$

yields a rank-r approximation,

$$K_{X_iY_i} \approx U_i K_{QY_i}, \tag{2.2}$$

where L_i is the Lagrange polynomial corresponding to node q_i and $U_i = [L_k(x)]_{\substack{x\in X_i \\ k=1:r}}$. Due to its generality and efficiency in computing U_i, this interpolation method is used in a number of general-purpose hierarchical matrix algorithms, e.g., [33, 9, 8, 15].

Another way of finding a column basis matrix for $K_{X_iY_i}$ is through subset selection [6, 26]. The column basis matrix is chosen as the submatrix corresponding to a judiciously chosen $O(1)$ subset Y_i^* from Y_i. Similar to interpolation, subset selection can be applied to general kernel functions. The reference [6] presents an efficient

hierarchical scheme to select representor sets for all nodes using two steps: top-down and bottom-up. The cost of the algorithm in [6] is dominated by computing farfield representor sets in the top-down step. Computing representor sets for all nodes i leads to $O(n \log n)$ complexity for a balanced tree with $O(\log n)$ levels. The total complexity of the resulting hierarchical matrix construction is $O(n \log n)$ instead of the optimal $O(n)$ complexity achieved by interpolation-based methods. However, compared to using interpolation nodes (which are generally *outside* the given dataset), selecting subsets directly from the dataset is more memory efficient for low-rank approximation. See section 6 for a detailed discussion.

3. Fast HiDR. To facilitate the fast construction of hierarchically low-rank representations, we propose an efficient preprocessing scheme to reduce the tree-structured data so that each node in the partition tree induces $O(1)$ cost in the subsequent hierarchical matrix construction process. Specifically, let X_i be the set of points corresponding to node i and Y_i be the farfield of X_i. The data reduction aims to find representor sets $X_i^* \subset X_i$ with $O(1)$ points and $Y_i^* \subset Y_i$ with $O(1)$ points for each node i. Note that a naive data reduction for Y_i with $O(n)$ points into a subset of evenly spaced points as shown in Figure 1.1 will lead to $O(n)$ computational complexity. The cost can be reduced to $O(1)$ with a carefully designed hierarchical procedure presented in subsection 3.1.

We present the HiDR algorithm in subsection 3.1 and verify that it scales linearly with the size of the data in subsection 3.2. Several algorithms for performing data reduction are discussed in subsection 3.3.

3.1. Linear complexity HiDR. The fast HiDR consists of two traversals of the tree: bottom-up and top-down. The $O(1)$ representor set X_i^* for X_i is computed in the bottom-up pass, and the $O(1)$ farfield representor set Y_i^* for Y_i is computed in the top-down pass. An illustration of HiDR for a one-dimensional dataset of quasi-uniform points is shown in Figure 3.1, where X_i^* contains 2 points computed in the bottom-up pass and Y_i^* contains 3 points computed in the top-down pass (each set in blue is the farfield of X_i in orange).

A building block for the hierarchical scheme is a `DataReduct` subroutine that takes the form

$$\texttt{DataReduct}(X, k) \to X^*,$$

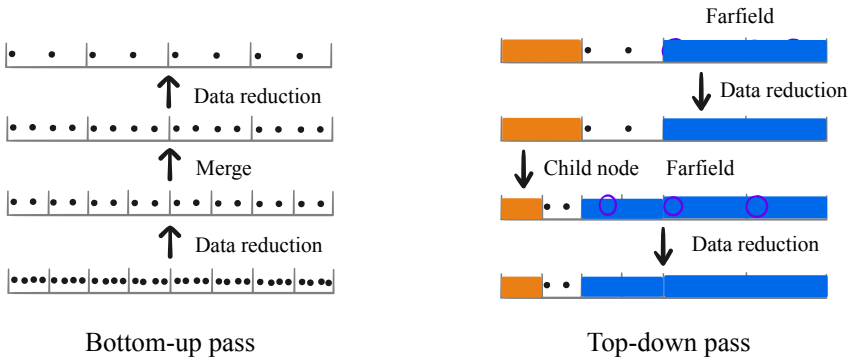


FIG. 3.1. *HiDR for a one-dimensional dataset (that yields a perfect binary partition tree).*

where X^* is a subset (representor set) of the input X and k is a parameter that specifies the size of X^*. There are several options for the subroutine `DataReduct` to obtain X^* from X. A detailed discussion is presented in subsection 3.3. Given a set of points, the subroutine selects a subset of evenly spaced points whose size is bounded by a prescribed constant and scales linearly with the size of the input data. The HiDR is designed such that the input dataset for `DataReduct` is always $O(1)$ in size.

In the bottom-up sweep, starting from leaf nodes i, each X_i contains $O(1)$ points, and thus the data reduction from X_i to X_i^* induces $O(1)$ cost only. After children nodes have been processed, we define for each parent p an intermediate set S_p as the union of reduced sets X_i^* for all its children i. The representor set X_p^* is obtained by applying data reduction to the intermediate set S_p. Since each parent has at most C children ($C = 2, 4, 8$ for binary tree, quadtree, and octree, respectively) and each X_i^* is $O(1)$ in size, the intermediate set S_p is always $O(1)$ in size. Thus, the cost of computing the representor set for p is always $O(1)$. Recursively, all nonroot nodes can be processed in $O(1)$ complexity in the bottom-up procedure (see Figure 3.2). Figure 3.3 shows the data reduction to generate X_i^* from the lowest level to upper levels of the tree constructed in Figure 2.1, where each X_i^* contains 2 points.

In the top-down sweep, starting from the top nodes i in the tree with the nonempty interaction list, we define the intermediate set T_i as the union of Y_p (p denotes the parent of i) and X_j^* for all nodes j in the interaction list of i. The representor set Y_i^* for the farfield Y_i of X_i is computed by applying data reduction to the intermediate set T_i. Since the cardinality of the interaction list of i is bounded by the sparsity constant c_{sp}, which is $O(1)$, and each X_j^* is already $O(1)$ in size, we see that T_i only contains $O(1)$ points, and consequently computing Y_i^* has $O(1)$ cost. Once parent nodes have been processed, we define the intermediate set T_i for each child i as the union of all X_j^* from its interaction list and Y_p^* from its parent p. From this definition, T_i is also $O(1)$ in size. Similar to the above, the representor set Y_i^* is then obtained by applying data reduction to T_i. Recursively, each reduced farfield representation Y_i^*

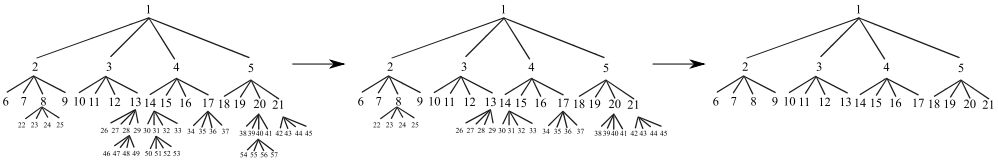


FIG. 3.2. *Bottom-up pass for a general tree. Representor sets X_i^* are first computed for nodes i at the deepest level and then at upper levels. Nodes with X_i^* computed are eliminated. The three trees from left to right correspond to the 1st, 3rd, and 5th configurations in Figure 3.3.*

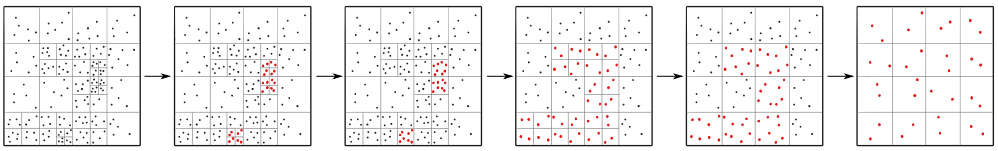


FIG. 3.3. *HiDR for two-dimensional nonuniform data: bottom-up pass. Each X_i is reduced to X_i^* such that X_i^* contains at most 2 points. The 1st, 3rd, and 5th arrows correspond to the subroutine `DataReduct` for the boxes at the bottom level of the tree, where points in red are output from `DataReduct`. The 2nd and 4th arrows correspond to merging children boxes and going up in the tree.*

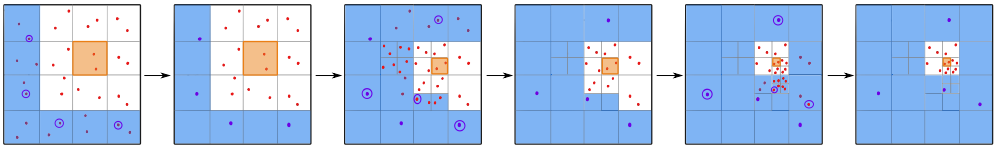


FIG. 3.4. *HiDR for two-dimensional nonuniform data: top-down pass. For each X_i^* in an orange box, the blue region contains the entire farfield, where each representor set Y_i^* (circled) contains at most 4 points.*

Algorithm 3.1 *HiDR*

Input: The adaptive partition tree $\mathcal{T}$ for dataset X, the collection of subsets X_i for all leaf nodes i, prescribed maximum size r_1 of X_i^* and maximum size r_2 of Y_i^*
Output: Reduced representations X_i^* and Y_i^* for all nodes i

for all $i \in \mathcal{T}$ **do**
  $Y_i^* = \emptyset$, $S_i = \emptyset$
  if i is a leaf node **then**
    $S_i = X_i$
  end if
end for
for each level (from bottom to top) **do**
  for all i at this level **do**
    if i is a parent **then**
      $S_i = \bigcup_{c \in \mathrm{ch}(i)} X_c^*$ with $\mathrm{ch}(i)$ the set of children of node i
    end if
    $X_i^* = \texttt{DataReduct}(S_i, r_1)$
  end for
end for
for each level (from top to bottom) **do**
  for all i at this level **do**
    if i has nonempty farfield **then**
      $T_i = Y_p^* \bigcup X_j^*$ over all j in the interaction list of i (p denotes the parent of i)
      $Y_i^* = \texttt{DataReduct}(T_i, r_2)$
    end if
  end for
end for
return X_i^*, Y_i^* for all nodes $i \in \mathcal{T}$

can be computed in $O(1)$ complexity via the top-down procedure. Figure 3.4 shows the top-down data reduction for farfield to generate Y_i^* (circled) as i goes from a node near the root node to a leaf node. In Figure 3.4, each Y_i^* contains at most 4 points only.

The HiDR algorithm is summarized in Algorithm 3.1. In practice, the cost of HiDR is lower than the hierarchical low-rank compression. HiDR reduces the computational cost of the subsequent hierarchical matrix construction.

3.2. Complexity analysis. In this section, we show that Algorithm 3.1 (HiDR) has linear complexity with respect to the number of points in X.

THEOREM 3.1. *For the given dataset X that contains n points, let $\mathcal{T}$ be a partition tree for X, in which each leaf node corresponds to a subset of X with $O(1)$ points and the sparsity constant $c_{sp} = O(1)$. Then the complexity of Algorithm* 3.1 *is $O(n)$.*

Proof. We first analyze the complexity in the bottom-up pass. Since for each leaf node i, X_i contains $O(1)$ points, the `DataReduct` for X_i has (1) cost. We next prove by induction that the cost for each nonleaf node is also $O(1)$. For a nonleaf node i, assume that for each child c, X_c^* contains $O(1)$ points. Then the intermediate set S_i in line 10 of Algorithm 3.1 contains $O(1)$ points because i has $O(1)$ children and X_c^* contains $O(1)$ points for each child c. Consequently, the cost of `DataReduct` applied to S_i is $O(1)$. This shows that the cost to obtain X_i^* is also $O(1)$. From the induction, we conclude that the cost for *each* node is $O(1)$ and that each reduced subset X_i^* has $O(1)$ points.

Now we analyze the complexity for the top-down pass. First note that each Y_i^* has at most $O(1)$ points according to the construction. Consequently, each T_i in line 18 of Algorithm 3.1 contains at most $O(1)$ points because the interaction list of i contains at most $c_{\text{sp}} = O(1)$ nodes and each X_j^* has $O(1)$ points. This implies that the `DataReduct` of T_i to generate Y_i^* has $O(1)$ complexity.

Overall, we see that for each node i, the associated total cost to compute X_i^* and Y_i^* is $O(1)$. Since there are $O(n)$ nodes in the tree, the total cost for all nodes is $O(n)$. This completes the proof of the theorem. ∎

3.3. Data reduction methods. In this section, we provide several algorithms for performing data reduction. The goal is to select a subset of X_i and of Y_i such that the selected subsets preserve the geometry of X_i and Y_i. For an input set X, the subroutine takes the simple form `DataReduct`$(X,k) \to X^*$ with $X^* \subset X$ the selected subset whose size is controlled by the parameter $k = O(1)$. Note that `DataReduct` only depends on the input data and is independent of any kernel function. It has been shown in [17] that the choice of the subset is essential for the accuracy and robustness of the low-rank approximation. According to the results in [17, 16], a subset evenly distributed over the containing set can offer an improved approximation robustness and accuracy over one that is not. The methods below can be used to generate such a subset efficiently. Many of them rely on a reference set with good uniformity, such as a uniform tensor grid. In addition to a tensor grid, it was shown recently that deep neural networks can be used to generate distributions with good uniformity [10].

We briefly review some of the existing data reduction methods below. An empirical comparison of these methods for the low-rank approximation of kernel matrices is presented in subsection 5.2.

Farthest point sampling. Given X and a target size k for the reduced subset S of X, farthest point sampling (FPS) constructs S in a sequential manner. S is initialized with 1 point only. Then FPS searches for a point in $X \backslash S$ that is farthest from S and adds the point to S. This procedure is repeated until S reaches size k. FPS generates evenly distributed subsets and has been widely used in computational geometry [25, 41, 44]. FPS was recently proposed for computing low-rank approximations [16].

Volume-based data reduction. Volume based data reductions choose a subset S of X via a reference grid Q with $O(1)$ points inside the computational domain. For example, Q can be chosen as a tensor grid (cf. [6]) inside the rectangular domain that encloses the data. S is chosen to be the collection of points in X that are closest to each point in Q.

Surface-based data reduction. Following the same idea as the volume-based method, we can also use a reference set Q based on surfaces constructed from the

given data X. In the surface-based method, we define Q as the union of points distributed on surfaces near the boundary of the computational domain. For example, we can construct ellipsoids centered at the center of a rectangular domain that encloses X. The principal semiaxes of the ellipsoids are chosen to be equal to γ times the width of the rectangular domain in each dimension, where $\gamma > 0$ is a hyperparameter. In subsection 5.2, we use three ellipsoids with $\gamma = 0.3, 0.6, 1.2$.

Anchor net method. The anchor net method [17] is a newly proposed subset selection method based on approximating the geometry of the given dataset with low-discrepancy subsets. It constructs a set of points with low discrepancy (termed anchor points) via a two-level scheme over the given data and then selects a subset of the given data close to the anchor points. For low-rank approximation, the construction helps avoid numerical instability encountered in random sampling or k-means clustering when the kernel matrix has rapidly decaying singular values (cf. [17]). The anchor net method is shown to achieve a good time-accuracy trade-off in practice and is particularly efficient for high-dimensional data (cf. [17, 16]).

4. Data-driven hierarchical matrix construction. In this section, we first show how to extract a low-rank factorization instantly for an admissible block after the prepossessing procedure HiDR in subsection 4.1 and then present an algorithm with $O(n)$ complexity (Algorithm 4.1) for constructing an $\mathcal{H}^2$ matrix representation in subsection 4.2. We analyze the computational complexity of Algorithm 4.1 in subsection 4.3. The proposed method enjoys the following features:

(a) black-box general-purpose (kernel-independent) construction of the hierarchical low-rank format;
(b) optimal $O(n)$ complexity, where n is the number of points in the given dataset;
(c) better efficiency for data from complex geometry compared to general-purpose approaches as well as specialized kernel-dependent techniques.

4.1. Approximating the entire farfield $K_{X_iY_i}$. In this section, we show how to derive a low-rank approximation for the entire farfield $K_{X_iY_i}$ based on the representor sets X_i^*, Y_i^* returned by Algorithm 3.1.

When computing an approximate column basis for $K_{X_iY_i}$, X_i contains $O(1)$ points, and its entire farfield Y_i contains $O(n)$ points for the leaf node i (see Figure 1.1). In order for the entire algorithm to have linear complexity, the column basis matrix of $K_{X_iY_i}$ must be computed in $O(1)$ complexity.

We first apply strong rank-revealing QR factorization [30] to the submatrix $K_{X_iY_i^*}$,

$$K_{X_iY_i^*} = P \begin{bmatrix} I \\ G \end{bmatrix} K_{\hat{X}_iY_i^*}, \tag{4.1}$$

where P is a permutation matrix, $||G||_{\max}$ is bounded by a prescribed constant, and $\hat{X}_i$ is a subset of X_i with $O(1)$ points. Then the column basis matrix is chosen as $U_i = P[I; G^T]^T$, and the low-rank approximation for the entire farfield reads

$$K_{X_iY_i} \approx P \begin{bmatrix} I \\ G \end{bmatrix} K_{\hat{X}_iY_i}. \tag{4.2}$$

For notational convenience, we denote the procedure in (4.1)–(4.2) for computing an approximate column basis for $K_{X_iY_i}$ by

$$\text{getBasis}(K_{X_iY_i}) = (U_i, \hat{X}_i), \tag{4.3}$$

Algorithm 4.1 *Data-driven HiDR-based $\mathcal{H}^2$ matrix construction*

Input: Dataset X, kernel function $\kappa(x,y)$, approximation tolerance ϵ, separation ratio τ, maximum number of points q for a leaf node

Output: $\mathcal{H}^2$ matrix representation

1: Apply adaptive partitioning to X to generate the partition tree $\mathcal{T}$ with at most q points for each leaf node and obtain subsets X_i for all *leaf* nodes i
2: Determine approximation parameters r_1, r_2 from ϵ
3: Apply HiDR in Algorithm 3.1 with approximation parameters r_1, r_2 to obtain $O(1)$ representor sets X_i^* and Y_i^* for all nodes $i \in \mathcal{T}$
4: **for all** $i \in \mathcal{T}$ **do**
5: Define $\bar{X}_i^{(\mathrm{row})} = \bar{X}_i^{(\mathrm{col})} = \emptyset$
6: **if** i is a leaf node **then**
7: define $\bar{X}_i^{(\mathrm{row})} = \bar{X}_i^{(\mathrm{col})} = X_i$
8: **end if**
9: **end for**
10: **for all** nonroot $i \in \mathcal{T}$ from bottom level to top level **do**
11: Apply getBasis($K_{\bar{X}_i^{(\mathrm{row})} Y_i^*}$) to obtain $U_i, \hat{X}_i^{(\mathrm{row})}$ and getBasis($K^T_{Y_i^* \bar{X}_i^{(\mathrm{col})}}$) to obtain $V_i, \hat{X}_i^{(\mathrm{col})}$
12: Update $\bar{X}_p^{(\mathrm{row})} = \bar{X}_p^{(\mathrm{row})} \cup \hat{X}_i^{(\mathrm{row})}$ and $\bar{X}_p^{(\mathrm{col})} = \bar{X}_p^{(\mathrm{col})} \cup \hat{X}_i^{(\mathrm{col})}$, where p is the parent of i
13: **end for**
14: Define the $\mathcal{H}^2$ column and row basis matrices: $U = \{U_i\}_{\text{leaf } i}$, $V = \{V_i\}_{\text{leaf } i}$, transfer matrices $R = \{R_i\}, W = \{W_i\}$:

$$\begin{bmatrix} R_{c_1} \\ \vdots \\ R_{c_k} \end{bmatrix} = U_i, \quad \begin{bmatrix} W_{c_1} \\ \vdots \\ W_{c_k} \end{bmatrix} = V_i \quad \text{if } i \text{ has children } c_1, \ldots, c_k,$$

coupling matrices $B = \{B_{i,j}\}$:

$$B_{i,j} = \begin{cases} K_{\hat{X}_i^{(\mathrm{row})} \hat{X}_j^{(\mathrm{col})}} & \text{if } (i,j) \text{ is admissible,} \\ K_{X_i X_j} & \text{otherwise.} \end{cases}$$

15: **return** $\mathcal{H}^2$ representation: U, V, R, W, B

where $U_i := P[I; G^T]^T$ is the computed column basis and $\hat{X}_i \subset X_i$. Note that the kernel matrix $K_{X_i Y_i}$ is never formed because the input of "getBasis" is the kernel function and the subsets X_i, Y_i^*.

The cost to obtain U_i and $\hat{X}_i$ from (4.3) is $O(1)$, as the matrix $K_{X_i Y_i^*}$ is $O(1)$ by $O(1)$. Also notice that

$$\mathrm{card}(\hat{X}_i) \le \mathrm{rank}(K_{X_i Y_i^*}) \le \mathrm{card}(Y_i^*) = O(1). \tag{4.4}$$

Thus, $\hat{X}_i$ always contains $O(1)$ points.

As we shall see in section 6, the column basis U_i derived from $K_{X_i Y_i^*}$ can yield better accuracy than analytic methods, such as interpolation.

4.2. Computing hierarchical matrices with nested bases using HiDR. Hierarchical matrices with nested bases, e.g., $\mathcal{H}^2$ matrices, can offer optimal $O(n)$ complexity in time and space when approximating an n-by-n kernel matrix. A black-box hierarchical matrix construction proposed in [15] works for general kernel functions and allows for arbitrary low-rank compression techniques. In this section, we show that the HiDR can be incorporated naturally into the construction of $\mathcal{H}^2$ matrix representations via the general framework proposed in SMASH [15]. SMASH employs a bottom-up procedure that recursively applies rank-revealing factorization to the initial basis matrix (with $O(1)$ entries) for each node in the tree. In [15], the initial basis matrices are constructed via either interpolation or analytic expansion of the kernel function. In this section, we leverage representor sets produced by HiDR in Algorithm 3.1 to construct the initial basis matrices.

The full data-driven construction is presented in Algorithm 4.1. The algorithm automatically determines the approximation parameters r_1, r_2 for HiDR in Algorithm 3.1 according to the approximation tolerance ϵ prescribed by the user. The idea here is to apply the low-rank approximation in (4.2) to the artificial kernel matrix $K_{Z_1 Z_2}$, where $Z_1, Z_2 \subset \mathbb{R}^d$ are well-separated subsets (in the sense of (2.1)) of $O(1)$ random points. The parameters r_1, r_2 are chosen adaptively by increasing from $r_1 = r_2 = 1$ to a point such that the approximation error to $K_{Z_1 Z_2}$ is smaller than $10^{-2}\epsilon$. More sophisticated techniques, like a posteriori error estimation (cf. [12, 13, 14, 11]), can also be studied to estimate the approximation error. After the parameters are determined, Algorithm 3.1 first applies HiDR to X associated with tree $\mathcal{T}$ to obtain representor sets. Then the hierarchical matrix representation can be computed rapidly by following the SMASH $\mathcal{H}^2$ construction and using $K_{X_i Y_i^*}$ as the initial basis matrix for each leaf node i.

We perform numerical experiments in section 6 to demonstrate that the new data-driven method improves the matrix approximation accuracy of the interpolation-based SMASH algorithm [15]. Moreover, the cost of HiDR is smaller than the subsequent hierarchical matrix compression.

4.3. Complexity analysis.

THEOREM 4.1. *For the given dataset X with n points, let $\mathcal{T}$ be a partition tree for X, in which each leaf node corresponds to a subset of X with $O(1)$ points and the sparsity constant $c_{sp} = O(1)$. Then the complexity of Algorithm* 4.1 *is $O(n)$.*

Proof. Algorithm 4.1 follows the $\mathcal{H}^2$ construction in SMASH [15] with an additional HiDR in line 3. According to Theorem 3.1, HiDR has $O(n)$ complexity. Thus, to prove the $O(n)$ complexity of Algorithm 4.1, it suffices to show that the cost per node is $O(1)$ in lines 11–12.

We first show that $\bar{X}_i^{(\mathrm{row})}$ and $\bar{X}_i^{(\mathrm{col})}$ contain at most $O(1)$ points. If i is a leaf node, then according to the definition in line 7, $\bar{X}_i^{(\mathrm{row})} = \bar{X}_i^{(\mathrm{col})} = X_i$ contain $O(1)$ points. If i is a parent node, then after all children of i have been updated, $\bar{X}_i^{(\mathrm{row})}$ and $\bar{X}_i^{(\mathrm{col})}$ can be written as

$$\bar{X}_i^{(\mathrm{row})} = \bigcup_{c \text{ is a child of } i} \hat{X}_c^{(\mathrm{row})}, \quad \bar{X}_i^{(\mathrm{col})} = \bigcup_{c \text{ is a child of } i} \hat{X}_c^{(\mathrm{col})}.$$

Since the number of children for every node is bounded from above by a constant and every subset $\hat{X}_c$ contains $O(1)$ points according to (4.4), we see that $\bar{X}_i^{(\mathrm{row})}$ and $\bar{X}_i^{(\mathrm{col})}$ contain $O(1)$ points. This implies that the complexity in line 12 is at most $O(1)$ for all i.

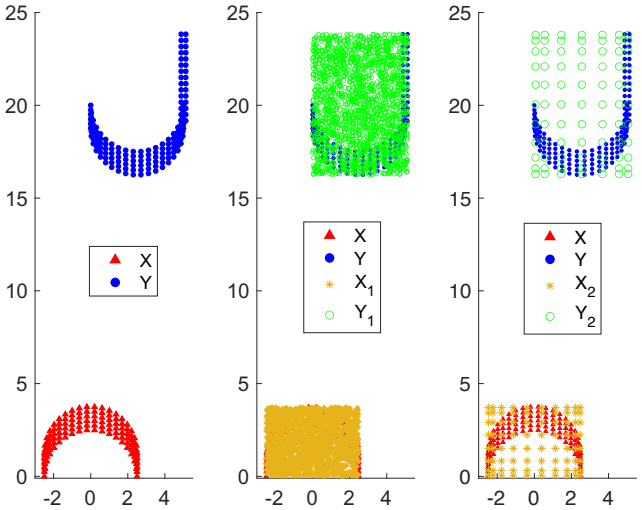
Next we analyze the complexity in line 11. Since Y_i^* contains $O(1)$ points only, it follows that in line 11, the input matrices $K_{\bar{X}_i^{(\mathrm{row})} Y_i^*}$ and $K^T_{Y_i^* \bar{X}_i^{(\mathrm{col})}}$ have $O(1)$ rows and columns. Consequently, performing "getBasis" in line 11 only takes $O(1)$ time.

Now we conclude that the total complexity in lines 11–12 is $O(1)$. Therefore, the total complexity for Algorithm 4.1 is $O(n)$. ∎

5. Data reduction and low-rank approximation. In this section, we investigate different data reduction techniques for low-rank approximation. Subsection 5.1 presents an example to reveal a drawback of methods that rely on points outside the given dataset (such as interpolation nodes or random points) for computing the low-rank approximation to kernel matrices. In subsection 5.2, we compare the performance of the data reduction methods of subsection 3.3 for low-rank approximation.

5.1. Drawback of using points outside the given data for irregular datasets. Interpolation nodes, random points, or, in general, points outside the given dataset are commonly used in low-rank approximation to obtain an approximate column basis efficiently without forming the original kernel matrix. Since these points are created artificially (not part of the given data), we call these points *virtual points*. For given data X and Y, virtual points are constructed in rectangular domains Ω_X and Ω_Y that cover X and Y, respectively, and the kernel matrix associated with these virtual points is used to compute the low-rank approximation.

One issue of of using virtual points is that it may lead to an incorrect approximation to the kernel matrix. This is because the virtual points lie outside the original data, and the kernel matrix involving these virtual points may have a very different spectrum from that of the kernel matrix to be approximated. Thus, methods using virtual points may not be robust for approximating general low-rank kernel matrices. To illustrate the issue, we use the "nJ" dataset as illustrated in Figure 5.1(a), where



(c) Dataset X, Y (d) 2000 random points in Ω_X, Ω_Y (e) 10×10 Chebyshev points in Ω_X, Ω_Y

FIG. 5.1. *Dataset X, Y and different types of "virtual points" within Ω_X and Ω_Y.*

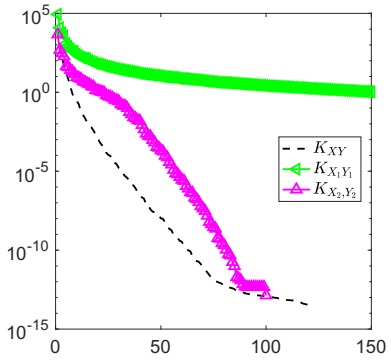


FIG. 5.2. *Distinct singular value patterns for kernel matrices with original data $X \times Y$ and virtual points $X_i \times Y_i$ $(i = 1, 2)$ of Figure* 5.1*: singular values of K_{XY} (*120 *by* 150*), $K_{X_1Y_1}$ (*2000 *by* 2000*), and $K_{X_2Y_2}$ (*100 *by* 100*). For $K_{X_1Y_1}$, the largest* 150 *singular values are plotted.*

X contains 120 points in $\Omega_X = [-2.5, 2.5] \times [0, 3.75]$ and Y contains 150 points in $\Omega_Y = [0, 5.13] \times [16.25, 23.84]$. We consider the smooth kernel function

$$\kappa(x, y) = \sqrt{1 + 100|x - y + a|^2}$$

with $a = [0, 20]^T$. The same issue also arises for other kernels, such as Gaussians. The corresponding kernel matrix K_{XY} (120 by 150) has rapidly decaying singular values as shown in Figure 5.2 (dashed line). Consequently, K_{XY} can be approximated very well by a low-rank matrix.

Now consider $K_{X_1Y_1}$, with X_1 being 2000 random points selected in Ω_X and Y_1 being 2000 random points selected in Ω_Y. The singular values of $K_{X_1Y_1}$ are an approximation to the continuous singular values of the problem in (5.1). The computed singular values of $K_{X_1Y_1}$ are also plotted in Figure 5.2. It can be seen that the singular values of $K_{X_1Y_1}$ do not decay rapidly, compared to K_{XY}.

Now consider $K_{X_2Y_2}$, where X_2 and Y_2 are 10×10 Chebyshev points in Ω_X and Ω_Y, respectively. Like $K_{X_1Y_1}$, this matrix does not have singular values that decay as rapidly as those of K_{XY}, and therefore an algebraic compression of $K_{X_2Y_2}$ will not be an effective approximation for K_{XY}.

Mathematical explanation. Employing a continuous treatment of the matrix approximation problem ignores the geometry of the discrete dataset. This can be problematic in general, as the continuous problem may have entirely different spectral properties compared to the matrix. It is even possible that the kernel function is *undefined* at virtual points. For the model problem in Figure 5.1, the matrix $K_{X_1Y_1}$ with virtual points X_1, Y_1 is related to the following integral operator:

$$T : L^2(\Omega_X) :\to L^2(\Omega_Y), \quad (Tf)(x) := \int_{\Omega_X} \kappa(x, y) f(y) dy. \tag{5.1}$$

The singular values of $K_{X_1Y_1}$ (with X_1 and Y_1 chosen as described above) approximate the singular values (up to a scaling constant) of the integral operator [3, 36, 45, 40, 2, 18]. These singular values do not decay rapidly like those of K_{XY}. In essence, we see that virtual point methods treat the matrix approximation as a *continuous* problem and thus ignore the geometry of the discrete data. When the continuous problem differs substantially from the original *discrete* problem (kernel matrix approximation), the performance of methods that utilize virtual points can be very unsatisfactory.

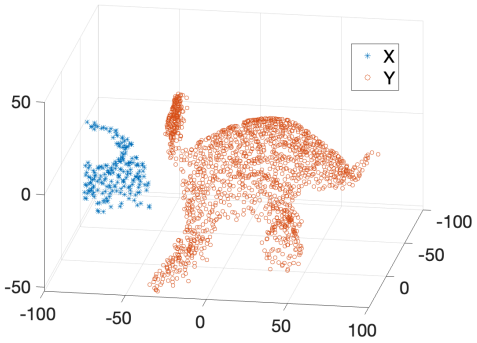


FIG. 5.3. *Subsection* 5.2 *dataset:* X *and* Y *well separated.*

5.2. Comparison of data reduction methods. In this section, we perform experiments to compare the performance of the four data reduction methods in subsection 3.3: farthest point sampling ('FPS'), volume-based reduction ('Volume'), surface-based reduction ('Surface'), and anchor net method ('AnchorNet'). These methods operate on the dataset and do not require any kernel function.

Experiment setup. We consider low-rank approximation to the kernel matrix K_{XY}, where X (198 points) and Y (1577 points) are well-separated subsets from a dinosaur manifold as shown in Figure 5.3. The diameter of X is 58.21, and the distance between X and Y is 21.275. We test three different kernel matrices K_{XY} corresponding to the kernel functions below:

$$\frac{1}{|x-y|}, \quad e^{-\frac{|x-y|^2}{900}}, \quad |x-y|^{11}.$$

To obtain the low-rank approximation, we first perform data reduction for Y and then build the factorization as described in subsection 4.1 using (4.1) and (4.2). The low-rank approximation error is measured by the relative matrix approximation error in the 2-norm.

The error plots for three different kernels are shown in Figure 5.4. For each plot, the horizontal axis denotes the number of points selected by the data reduction method, namely, the size of the subset $Y^* \subset Y$. Each curve shows how the low-rank approximation error for a specific data reduction method decays as we increase the size of Y^*. We see that 'Volume' and 'AnchorNet' offer the best performance and are almost indistinguishable from each other in performance across all three kernels. 'Surface' achieves similar performance for the first two kernels but is slightly worse than 'Volume' and 'AnchorNet' for the third kernel. The farthest point sampling 'FPS' performs well but is not as accurate as the other three methods for the same number of selected points for all kernels tested.

6. Numerical experiments. We present a series of numerical experiments in this section to illustrate the performance of the proposed data-driven construction in Algorithm 4.1. The code for the algorithm is available on GitHub.[1] The performance of the proposed data-driven hierarchical matrix construction is shown in subsection 6.1, inlcuding linear scaling, generality for various kinds of kernels, and the efficiency of HiDR for varying kernel parameters. Comparison to the state-of-the-art special-

[1] https://github.com/scalable-matrix/H2Pack/tree/sample_pt_opt.

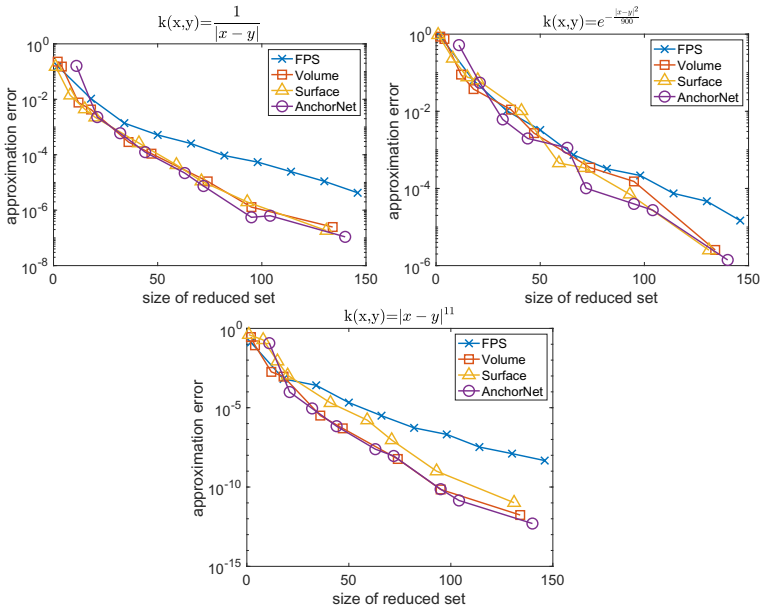


FIG. 5.4. *Comparison of data reduction methods for low-rank approximation to K_{XY} with dataset $X \times Y$ in Figure 5.3 and different kernels $k(x, y)$ on top of each plot.*

purpose methods for the Coulomb kernel is presented in subsection 6.2. Comparison to the widely used general-purpose method (interpolation) for various kernels is presented in subsection 6.3. For the data-driven hierarchical construction, volume-based data reduction is used. For experiments in subsections 6.1 and 6.2, we use one compute node on the Georgia Tech PACE-Hive cluster. This node has two sockets and 192 GB of DDR4 memory. Each socket has an Intel Xeon Gold 6226 12-core processor.

The approximation error is measured by the relative matrix-vector product error, $\frac{||Kz-\tilde{K}z||}{||z||}$, where $\tilde{K}$ denotes the hierarchical approximation to the kernel matrix K and z is a standard normal random vector. $||\cdot||$ denotes the 2-norm.

6.1. Data-driven construction: Scaling, generality, and once-for-all HiDR. This section has three objectives: (1) test the complexity of the proposed data-driven approach in subsection 6.1.1, including the HiDR and the resulting hierarchical matrix construction; (2) illustrate the generality of the data-driven approach by testing different kernels in subsection 6.1.2; and (3) apply HiDR once and use the representor sets to construct hierarchical matrices for various types of kernels, including Gaussian kernels with different bandwidths in subsection 6.1.3.

6.1.1. Scaling test for different datasets.

Datasets. Three datasets are used: cube, 3-sphere, and Dino. The cube dataset contains random samples from the uniform distribution in the unit cube $[0, 1]^3$. The Dino dataset is used in [15] and is illustrated in Figure 6.1. It consists of points distributed on a dinosaur-shaped surface in three dimensions. The 3-sphere dataset (see Figure 6.1) consists of random points distributed on the surface of three intersecting unit spheres whose centers form an equilateral triangle with side length close to 1. Roughly the same number of points is sampled from each sphere. Let n denote the number of points in the dataset. For the first three synthetic datasets, we test for n

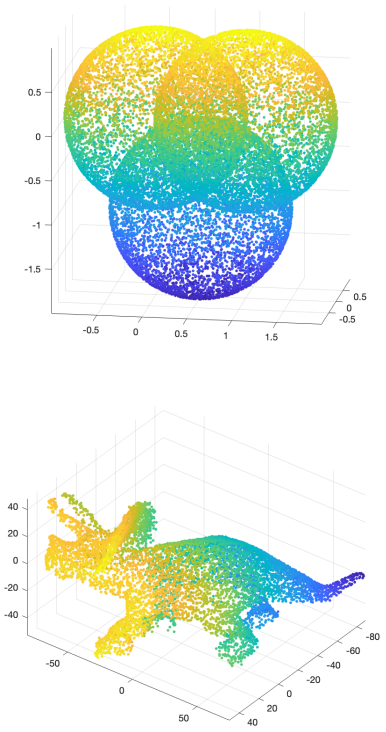


FIG. 6.1. 3-*sphere (left) and Dino (right) datasets.*

from 10^5 to 1.6×10^7. For the Dino dataset, since the size of the original data is fixed, we sample n points randomly and vary n from 10^4 to 1.5×10^5.

The kernel function is chosen to be the Coulomb kernel $\frac{1}{|x-y|}$, and the approximation error is 10^{-6} for each test. Figure 6.2 shows the timings for hierarchical data reduction (HiDR), $\mathcal{H}^2$ matrix construction (build), and the resulting matrix-vector multiplication (matvec) with respect to n, respectively. All timings scale linearly with n, and the hierarchical data reduction (HiDR) has a much lower cost than the subsequent hierarchical matrix construction (build). The low cost of data reduction and the kernel independence make the data-driven approach suitable for the case when the kernel matrix changes frequently due to changes in the data or kernel function.

6.1.2. Scaling test for different kernels. In this section, we test the proposed data-driven algorithm for the kernel functions in Table 6.1. We show that the general data-driven algorithm is scalable for different types of kernel functions for the same approximation accuracy. The 3-sphere dataset is used.

For each kernel function, we measure the time cost of the proposed algorithm as the size of data n increases. The three types of costs—HiDR, hierarchical matrix construction, and matrix-vector multiplication—correspond to the three plots in Figure 6.3.

In Figure 6.3, each plot shows the timing for all four kernels. The relative error for each case is 10^{-6}. It is easily seen from Figure 6.3 that each cost scales linearly with data size n. The algorithm is able to maintain accuracy for different types of kernel functions.

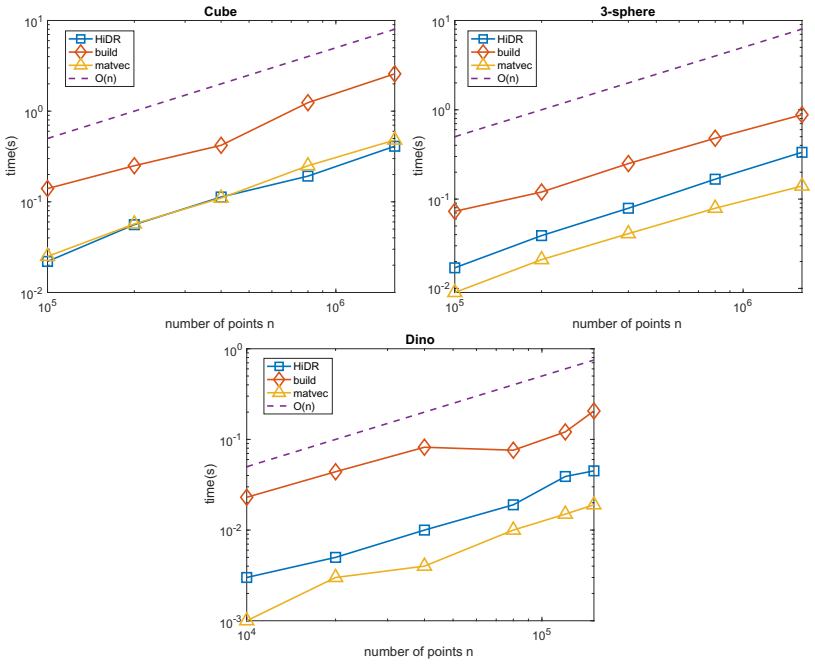


FIG. 6.2. *Subsection* 6.1.1 *experiment: timings of HiDR, $\mathcal{H}^2$ build, and matvec for n-by-n Coulomb kernel matrices with three datasets in $\mathbb{R}^3$.*

TABLE 6.1
Kernel functions used in the experiments in subsection 6.1. *Here $\kappa_1(x,x)=0$.*

$\kappa_1(x,y)$	$\kappa_2(x,y)$	$\kappa_3(x,y)$	$\kappa_4(x,y)$
$\frac{1}{\lvert x-y\rvert}$	$\exp(-\lvert x-y\rvert^2)$	$\cos(x\cdot y)$	$\exp\left(-\frac{1}{1-0.1\lvert x-y\rvert^2}\right)$

6.1.3. HiDR once for all. In this experiment—called "HiDR once for all"—we perform HiDR on the dataset to obtain representor sets and then use the representor sets to construct a hierarchical matrix representation for the different kernel functions in Table 6.1 and Gaussian kernels with different bandwidths. The key here is that, for a fixed compression level, HiDR is only performed *once* and that the same representor sets are used for *all* kernels.

Figure 6.4 shows the approximation error of the hierarchical matrix with respect to the average size of farfield representor sets Y_i^*. Different error curves correspond to approximations to different kernels. We see that by increasing the size of the farfield representor sets, the matrix approximation error is reduced effectively for all kinds of kernel functions. Since HiDR is only applied once, the precomputation cost is almost negligible when amortized over multiple kernels. The accuracy as seen from Figure 6.4 justifies the data-driven construction with the efficient kernel-independent HiDR. We see that the data-driven approach is particularly useful when hierarchical matrices for different kernel functions or kernel parameters need to be computed.

In applications like Gaussian processes, the bandwidth parameter for the Gaussian kernel is unknown and is determined by an iterative algorithm. Therefore, the bandwidth changes constantly, thus the kernel function. This makes existing hierarchical matrix constructions inefficient because the entire hierarchical algorithm needs

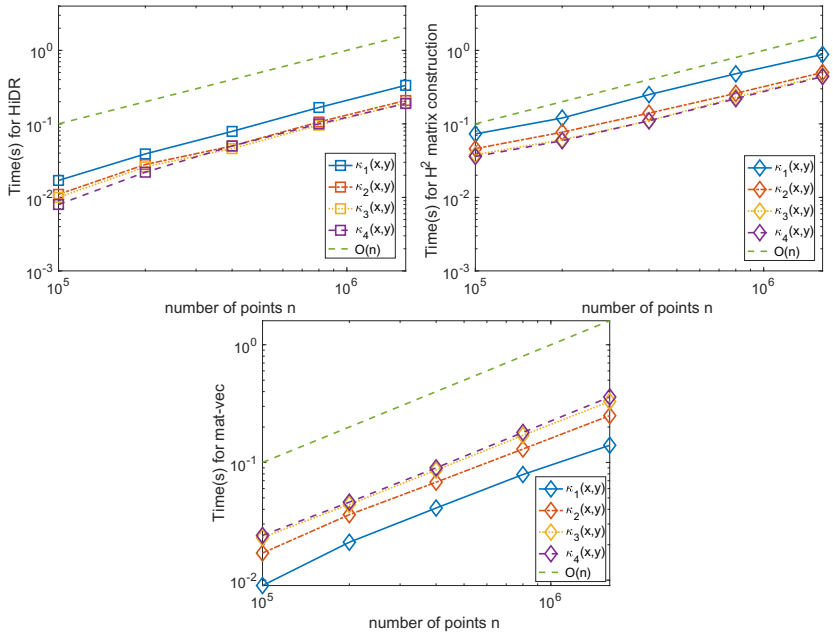


FIG. 6.3. *Subsection* 6.1.2 *scaling test for kernels in Table* 6.1*: CPU time for HiDR (left), hierarchical matrix construction (middle), and matrix-vector multiplication (right).*

to be run from scratch every time the bandwidth changes. The proposed data-driven approach, however, performs data reduction only *once,* and no matter what the bandwidth is, the hierarchical matrix representation can be constructed rapidly based on the computed representor sets. It can be seen from Figure 6.4 (right) that for a wide range of bandwidth values, the approximation error decays effectively as more points are used in the representor sets. The nearly zero approximation error for the bandwidth $L = 0.01$ is due to the fact that the admissible block is almost a zero matrix, as $\exp(-|x-y|^2/0.01^2) \approx 0$ when x and y are away from each other.

Overall, it can be seen from the experiments that the data-driven approach serves as a black-box tool for rapidly computing hierarchical matrices for general kernel functions. It is especially efficient in the situation when multiple kernel functions need to be approximated.

6.2. Comparison to special-purpose methods for the Coulomb kernel. In this section, we compare the new general-purpose data-driven (DD) construction to several optimized packages for the Coulomb kernel $\kappa(x,y) = \frac{1}{|x-y|}$, for example, FMM3D[2] [22], PVFMM [38], and proxy surface method (cf. [39, 21, 28]) implemented in H2Pack [35]. These methods are specialized for the Coulomb kernel to offer better efficiency in practice than the interpolation-based methods for constructing $\mathcal{H}^2$ matrices.

DD, FMM3D, PVFMM, and H2Pack are compiled using an Intel C/C++/Fortran compiler v19.0.5 with optimization flags "-xHost -O3." The Intel MKL 19.0.5 is used in all tested libraries to perform general matrix-vector and matrix-matrix multiplications. DD, H2Pack, and FMM3D use one thread per CPU core and 24 cores on one

[2]https://fmm3d.readthedocs.io/en/latest.

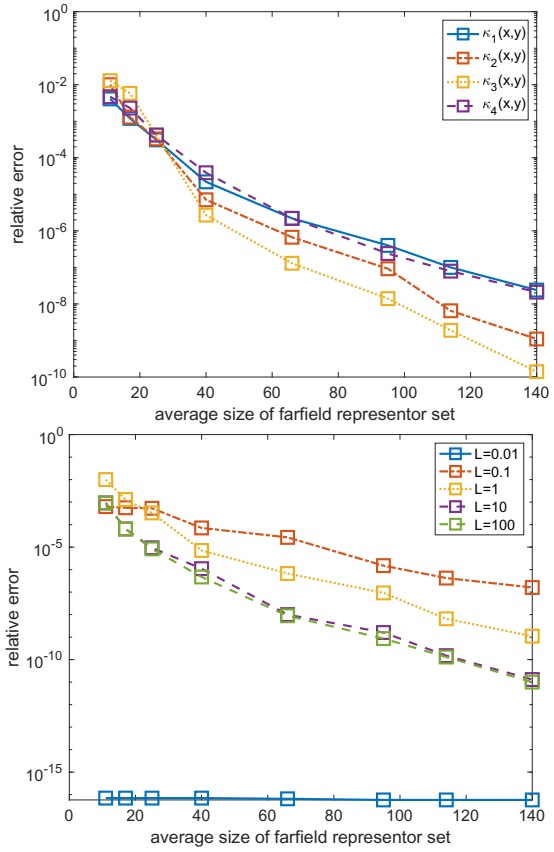


FIG. 6.4. *Subsection* 6.1.3*: perform HiDR only once to obtain representor sets and then use them to construct hierarchical matrices for multiple kernels. Matrix approximation error vs. average size of farfield representor sets* Y_i^**. Left: kernels in Table* 6.1*. Right: Gaussian kernel* $\exp(-|x-y|^2/L^2)$ *with different bandwidth* $L = 10^k$ *with* $k = -2, -1, 0, 1, 2$.

computing node. PVFMM uses MVAPICH2 2.3.2 as the MPI back end and uses one MPI process with 24 cores on one computing node.

We use the same datasets as in subsection 6.1. For every method, the total time is computed as

$$\text{total time} = \text{precomputation} + \mathcal{H}^2 \text{ construction} + \text{matrix-vector multiplication}.$$

FMM3D and the proxy surface method do not have precomputation, while PVFMM and DD require precomputation. For DD, the precomputation refers to HiDR.

Timings for precomputation, hierarchical build, and matvec are shown in Figure 6.5, Figure 6.6, and Figure 6.7, respectively. The total time is shown in Figure 6.8. The relative error (in 2-norm) for each test is 10^{-6}.

From Figure 6.5, we see that the proposed HiDR requires significantly lower precomputation cost compared to PVFMM. This is due to the fact that the kernel matrix is *never* accessed in HiDR and *no* algebraic compression is computed. Moreover, we see from Figure 6.5 that the advantage of the data-driven method becomes more obvious for irregular data from a manifold, such as 3-sphere and Dino. It can be seen that PVFMM has almost constant cost independent of the size of the dataset n. This

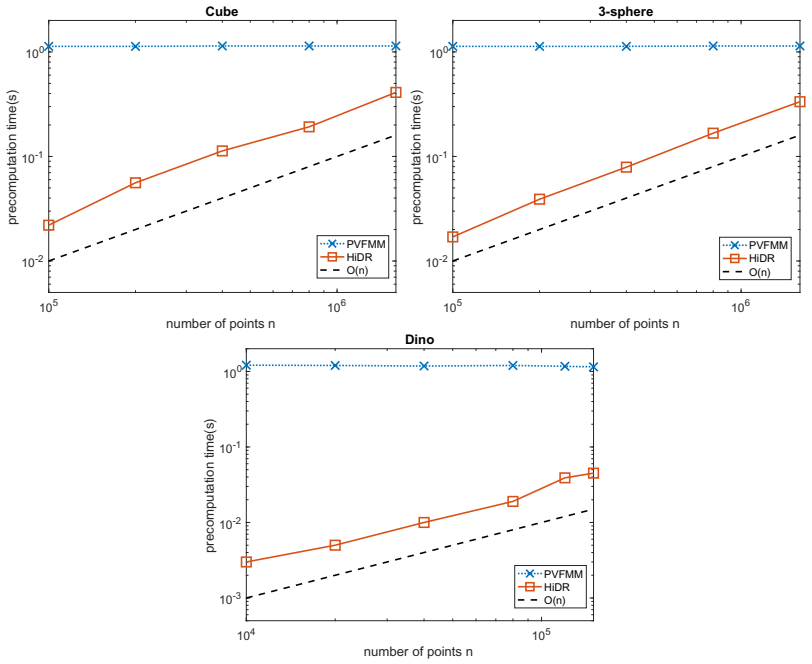


FIG. 6.5. *Subsection* 6.2 *experiment: precomputation time of PVFMM and DD for approximating n-by-n Coulomb kernel matrices with the cube, 3-sphere, and Dino datasets.*

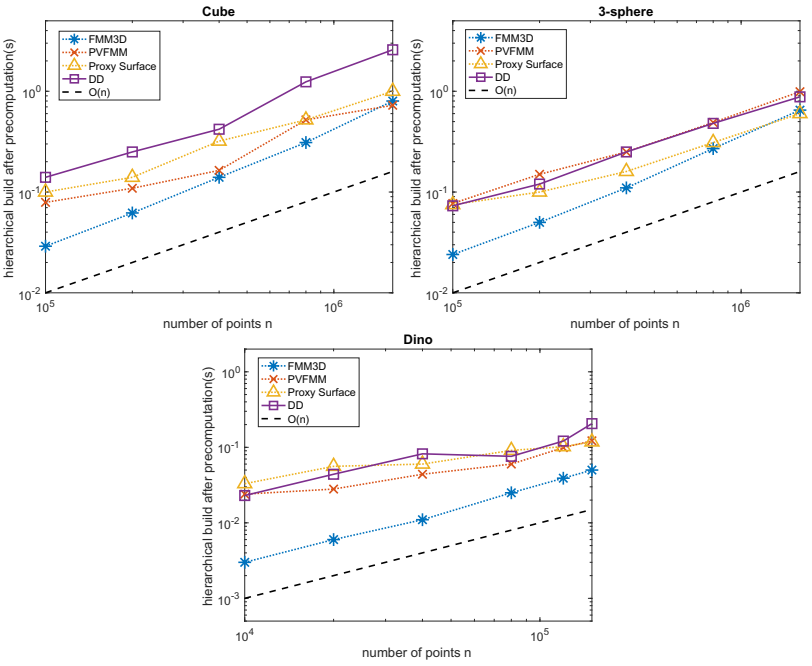


FIG. 6.6. *Subsection* 6.2*: Hierarchical construction time (after precomputation) of FMM3D, PVFMM, proxy surface, and DD for approximating n-by-n Coulomb kernel matrices with the cube, 3-sphere, and Dino datasets.*

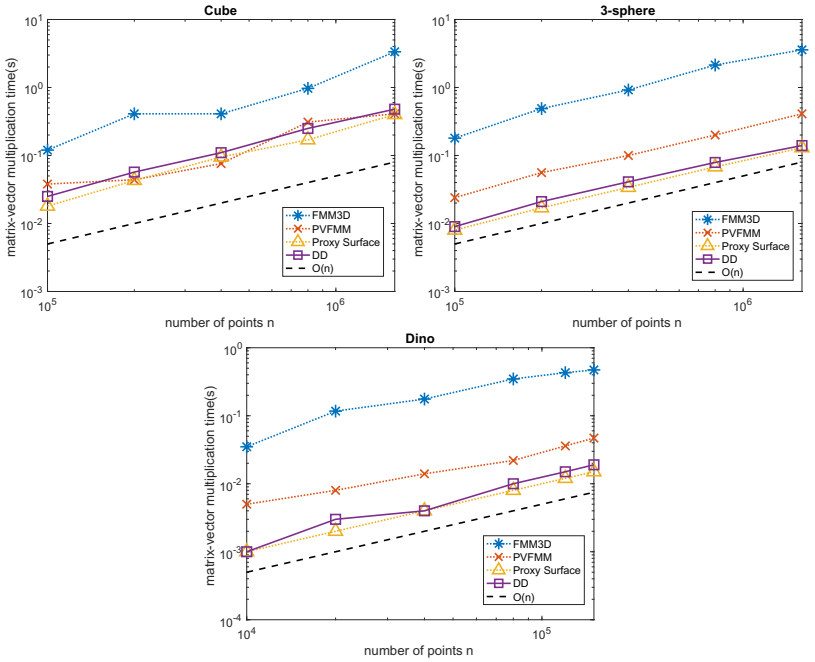


FIG. 6.7. *Subsection* 6.2*: matrix-vector multiplication time of FMM3D, PVFMM, proxy surface, and DD for approximating n-by-n Coulomb kernel matrices with the cube,* 3*-sphere, and Dino datasets.*

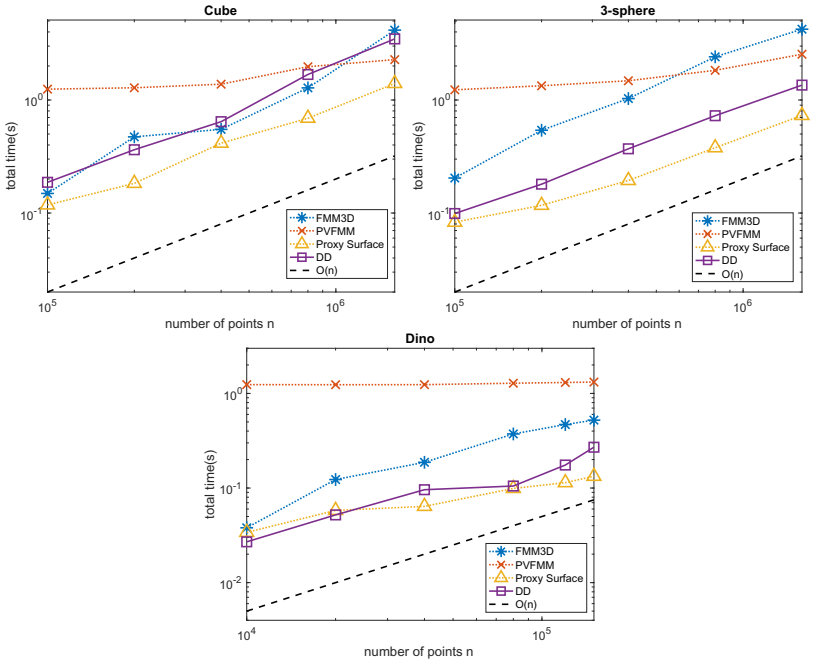


FIG. 6.8. *Subsection* 6.2*: total construction time of FMM3D, PVFMM, proxy surface, and DD for approximating n-by-n Coulomb kernel matrices with the cube,* 3*-sphere, and Dino datasets.*

is because PVFMM, based on KIFMM, treats matrix compression as a continuous problem and the precomputation involves solving ill-posed integral equations to compute "equivalent density points" in order to facilitate the farfield compression. The integral equations are imposed on the surface that surrounds the data, and thus the same integral equation will be used as long as the bounding surface remains the same regardless of the data distribution. For this reason, the resulting matrix compression may not reflect the potentially varying property of the kernel matrix for different data distributions, and as pointed out in [47], "in practice the cutoff number of equivalent density points in which the compression is effective, is very large." On the contrary, HiDR is entirely based on processing the data without solving any equation or performing any matrix compression, thus leading to a much smaller precomputation cost than PVFMM or related methods. Moreover, the "equivalent density" approach in KIFMM and PVFMM is not applicable to general kernel functions (such as sigmoid kernel, Gaussian kernel, exponential kernel, polynomial kernel, etc.) that are unrelated to fundamental solutions of classical constant-coefficient elliptic PDEs. As mentioned above, the HiDR-based approach is entirely data-driven and thus applicable to general kernel functions.

From Figure 6.6, we see that FMM3D outperforms other methods in hierarchical construction. This is because, unlike other methods, FMM3D does *not* compute and store a hierarchical matrix representation. Instead, it computes the hierarchical representation when performing matrix-vector multiplication. The other methods have similar performance, where no single method performs significantly better than others across all datasets. It should be noted that, for methods with precomputation, the hierarchical construction time can be further reduced at the expense of more precomputation time. In principle, more time spent in precomputation could yield faster hierarchical construction.

For matvec, Figure 6.7 shows that the data-driven method and proxy surface method achieve similar performance that is in general better than FMM3D and PVFMM. FMM3D is significantly slower than other methods due to the on-the-fly hierarchical construction. For data sampled from a manifold, PVFMM is outperformed by DD and proxy surface. The results in Figure 6.7 justify the efficiency of the hierarchical matrix representation built from the fast HiDR in Figure 6.5.

For the total computation time, as can be seen from Figure 6.8, we see that the proxy surface method provides the best performance overall, followed by DD. The advantage of DD is more evident for data from a manifold, e.g., 3-sphere and Dino. In general, we see that the data-driven method leads to a lot more computational savings when data are sampled from a low-dimensional manifold.

It should be emphasized that the proxy surface method is a specialized method optimized for the Coulomb kernel to offer superior efficiency, while DD is a general-purpose approach that can be applied to a variety of kernel functions (cf. Table 6.1). Unlike the special-purpose methods, no analytic property of the kernel function is used in the data-driven hierarchical construction. It is nonetheless possible to design specialized data-driven algorithms for the kernel function of interest to improve efficiency. Overall, we conclude from the results in Figure 6.8 that the data-driven method, as a black-box tool for hierarchical matrix computations, also offers excellent efficiency for special kernels without utilizing any specific property of the kernel.

6.3. Memory efficiency. In this section, we illustrate the memory efficiency of the proposed data-driven approach by comparing it to interpolation-based hierarchical matrix construction. We test the two general-purpose methods for the different

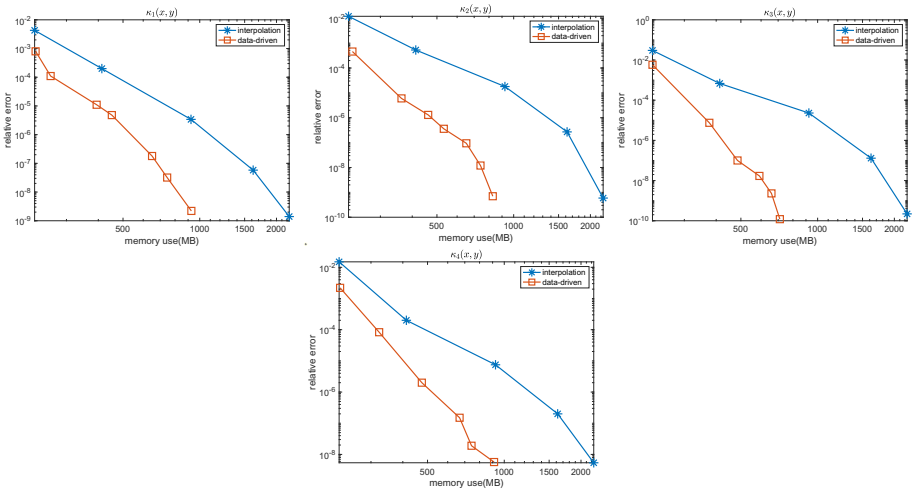


FIG. 6.9. *Subsection* 6.3 *experiment: error vs. memory use of interpolation-based and data-driven constructions for approximating the four kernel matrices (Table* 6.1*) with the* 3*-sphere dataset.*

kernel functions listed in Table 6.1. The 3-sphere dataset is used with $n = 20000$ points.

In Figure 6.9, we plot the approximation error vs. the memory use for each method and each kernel function. The memory use is measured by the cost for storing the hierarchical representation derived by the respective method. The high memory use of interpolation-based construction is clearly seen from the plots. In three dimensions, the number of interpolation nodes is k^3 if k interpolation nodes are used in each dimension. This number may exceed the size of the admissible block to be approximated. We found that, in practice, to achieve moderate to high approximation accuracy, a large number of interpolation nodes is needed. The proposed data-driven method, on the other hand, significantly reduces the memory needed to achieve a certain approximation accuracy. Equivalently, we can also conclude that, for the same approximation rank and memory requirement, the data-driven method is able to provide a much more accurate hierarchical representation than the one derived from interpolation. For large-scale data, the memory efficiency of data-driven construction would be even more prominent.

7. Conclusion. We proposed general-purpose data-driven hierarchical matrix construction accelerated by a novel HiDR. The algorithm first computes a reduced data representation following the tree structure and then performs the hierarchical low-rank compression. Different from all existing methods, HiDR entirely operates on the given dataset, *without* accessing the kernel function or kernel matrix. The complexity of the whole data-driven construction is linear with respect to the data size. Compared to general-purpose methods such as interpolation, the new data-driven framework requires less memory for the same matrix approximation accuracy. For special kernels like the Coulomb kernel, the general data-driven method, as a black-box approach, demonstrates competitive performance when compared to specialized methods optimized for the Coulomb kernel. The data-driven approach yields low computational cost and is particularly efficient for data sampled from low-dimensional manifolds. Future work includes extending the data-driven framework to the efficient construction of hierarchical matrices for high-dimensional data in machine learning

applications. One appealing feature is that, when constructing hierarchical matrices for different kernel functions associated with the same dataset, HiDR only needs to be performed once, which significantly reduces the total computational cost. Additionally, the data-driven procedure can be optimized toward the special kernel function under consideration. In the current presentation, we focus on providing a general approach that could be useful for a general-purpose library for accelerating kernel matrix computations with hierarchical matrix representations. Possible improvements for special kernel functions will be investigated at a future date.

REFERENCES

[1] C. R. Anderson, *An implementation of the fast multipole method without multipoles*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 923–947.

[2] K. Atkinson, *Convergence rates for approximate eigenvalues of compact integral operators*, SIAM J. Numer. Anal., 12 (1975), pp. 213–222.

[3] K. E. Atkinson, *The numerical solution of the eigenvalue problem for compact integral operators*, Trans. Amer. Math. Soc., 129 (1967), pp. 458–465.

[4] J. Barnes and P. Hut, *A hierarchical O(N log N) force-calculation algorithm*, Nature, 324 (1986), pp. 446–449.

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

[6] M. Bebendorf and R. Venn, *Constructing nested bases approximations from the entries of non-local operators*, Numer. Math., 121 (2012), pp. 609–635.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

[8] S. Börm and L. Grasedyck, *Hybrid cross approximation of integral operators*, Numer. Math., 101 (2005), pp. 221–249.

[9] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem., 27 (2003), pp. 405–422.

[10] D. Cai, *Physics-Informed Distribution Transformers via Molecular Dynamics and Deep Neural Networks*, preprint, https://ssrn.com/abstract=4028725, 2022.

[11] D. Cai and Z. Cai, *Hybrid A Posteriori Error Estimators for Conforming Finite Element Approximations to Stationary Convection-Diffusion-Reaction Equations*, preprint, arXiv:2107.06341, 2021.

[12] D. Cai and Z. Cai, *A hybrid a posteriori error estimator for conforming finite element approximations*, Comput. Methods Appl. Math. Engrg., 339 (2018), pp. 320–340.

[13] D. Cai, Z. Cai, and S. Zhang, *Robust equilibrated a posteriori error estimator for higher order finite element approximations to diffusion problems*, Numer. Math., 144 (2020), pp. 1–21.

[14] D. Cai, Z. Cai, and S. Zhang, *Robust equilibrated error estimator for diffusion problems: Mixed finite elements in two dimensions*, J. Sci. Comput., 83 (2020), pp. 1–22.

[15] D. Cai, E. Chow, L. Erlandson, Y. Saad, and Y. Xi, *SMASH: Structured matrix approximation by separation and hierarchy*, Numer. Linear Algebra Appl., 25 (2018), e2204.

[16] D. Cai, E. Chow, and Y. Xi, *Data-Driven Linear Complexity Low-Rank Approximation of General Kernel Matrices: A Geometric Approach*, preprint, 2022, https://arxiv.org/pdf/2212.12674.pdf.

[17] D. Cai, J. Nagy, and Y. Xi, *Fast deterministic approximation of symmetric indefinite kernel matrices with high dimensional datasets*, SIAM J. Matrix Anal. Appl., 43 (2022), pp. 1003–1028, https://doi.org/10.1137/21M1424627.

[18] D. Cai and P. S. Vassilevski, *Eigenvalue problems for exponential-type kernels*, Comput. Methods Appl. Math., 20 (2020), pp. 61–78.

[19] D. Cai and J. Xia, *A stable matrix version of the fast multipole method: Stabilization strategies and examples*, Electron. Trans. Numer. Anal., 54 (2021), pp. 581–609.

[20] Y. Cao, L. Wen, and J. Rong, *A SVD accelerated kernel-independent fast multipole method and its application to BEM*, WIT Trans. Model. Simul., 56 (2013), pp. 431–443.

[21] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404.

[22] H. Cheng, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm in three dimensions*, J. Comput. Phys., 155 (1999), pp. 468–498.

[23] D. Colton and R. Kress, *Integral Equation Methods in Scattering Theory*, Pure and Applied Mathematics, Wiley, New York, 1983.

[24] D. Decoste and B. Schölkopf, *Training invariant support vector machines*, Mach. Learn., 46 (2002), pp. 161–190.
[25] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi, *The farthest point strategy for progressive image sampling*, IEEE Trans. Image Process., 6 (1997), pp. 1305–1315.
[26] L. Erlandson, D. Cai, Y. Xi, and E. Chow. *Accelerating parallel hierarchical matrix-vector products via data-driven sampling*, in 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 749–758.
[27] B. Fornberg and G. Wright, *Stable computation of multiquadric interpolants for all values of the shape parameter*, Comput. Math. Appl., 48 (2004), pp. 853–867.
[28] A. Gillman, P. M. Young, and P.-G. Martinsson, *A direct solver with o(n) complexity for integral equations on one-dimensional domains*, Front. Math. China, 7 (2012), pp. 217–247.
[29] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[30] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.
[31] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis*, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2015.
[32] W. Hackbusch and S. Börm, *$\mathcal{H}^2$-matrix approximation of integral operators by interpolation*, Appl. Numer. Math., 43 (2002), pp. 129–143.
[33] W. Hackbusch, B. Khoromskij, and S. Sauter. *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, H.-J. Bungartz, R. H. W. Hoppe, and C. Zenger, eds., Springer-Verlag, Berlin, 2000, pp. 9–29.
[34] W. Hackbusch, B. N. Khoromskij, and R. Kriemann, *Hierarchical matrices based on a weak admissibility criterion*, Computing, 73 (2004), pp. 207–243.
[35] H. Huang, X. Xing, and E. Chow, *H2Pack: High-performance H-2 matrix package for kernel matrices using the proxy point method*, ACM Trans. Math. Softw., 47 (2020), pp. 1–29.
[36] H. B. Keller, *On the accuracy of finite difference approximations to the eigenvalues of differential and integral operators*, Numer. Math., 7 (1965), pp. 412–419.
[37] Y. Liu, W. Sid-Lakhdar, E. Rebrova, P. Ghysels, and X. S. Li, *A parallel hierarchical blocked adaptive cross approximation algorithm*, Int. J. High Perform. Comput. Appl., 34 (2020), pp. 394–408.
[38] D. Malhotra and G. Biros, *Pvfmm: A parallel kernel independent fmm for particle and volume potentials*, Commun. Comput. Phys., 18 (2015), pp. 808–830.
[39] P. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.
[40] J. Osborn, *Spectral approximation for compact operators*, Math. Comput., 29 (1975), pp. 712–725.
[41] G. Peyré and L. D. Cohen, *Geodesic remeshing using front propagation*, Int. J. Comput. Vis., 69 (2006), pp. 145–156.
[42] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187–207.
[43] V. Rokhlin, *Rapid solution of integral equations of scattering theory in two dimensions*, J. Comput. Phys., 86 (1990), pp. 414–439.
[44] T. Schlömer, D. Heck, and O. Deussen. *Farthest-point optimized point sets with maximized minimum distance*, in Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics, 2011, pp. 135–142.
[45] A. Spence, *Error bounds and estimates for eigenvalues of integral equations*, Numer. Math., 29 (1978), pp. 133–147.
[46] X. Sun and N. Pitsianis, *A matrix version of the fast multipole method*, SIAM Rev., 43 (2001), pp. 289–300.
[47] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, J. Comput. Phys., 196 (2004), pp. 591–626.
[48] Y. Zhao, D. Jiao, and J. Mao, *Fast nested cross approximation algorithm for solving large-scale electromagnetic problems*, IEEE Trans. Microw. Theory, 67 (2019), pp. 3271–3283.