Sparse Matrix Factorizations for Fast Linear Solvers with Application to Laplacian Systems

Michael T. Schaub †, Maguy Trefois ‡, Paul Van Dooren §, and Jean-Charles Delvenne ¶

Abstract. In solving a linear system with iterative methods, one is usually confronted with the dilemma of having to choose between cheap, inefficient iterates over sparse search directions (e.g., coordinate descent), or expensive iterates in well-chosen search directions (e.g., conjugate gradients). In this paper, we propose to interpolate between these two extremes, and show how to perform cheap iterations along non-sparse search directions, provided that these directions admit a new kind of sparse factorization. For example, if the search directions are the columns of a hierarchical matrix, then the cost of each iteration is typically logarithmic in the number of variables. Using some graph-theoretical results on low-stretch spanning trees, we deduce as a special case a nearly-linear time algorithm to approximate the minimal norm solution of a linear system $Bx = b$ where $B$ is the incidence matrix of a graph. We thereby can connect our results to recently proposed nearly-linear time solvers for Laplacian systems, which emerge here as a particular application of our sparse matrix factorization.

Key word. matrix factorization, linear system, Laplacian matrix, iterative algorithms, sparsity, hierarchical matrices

AMS subject classifications. 15A06, 15A23, 15A24

1. Introduction. A fundamental issue in almost all areas of mathematical sciences and quantitative research is finding the solutions of large linear systems of equations. For instance, partial differential equations, which arise in various areas of physics, mechanics and electro-magnetics, have often to be solved numerically. A spatial discretization of such a problem naturally leads to solving a large sparse or structured linear system [21].

In principle, two strategies to solve linear systems exist. First, one can apply direct methods [5] like Cholesky factorization or Gaussian elimination. Those methods provide an exact solution of the system after a finite number of computations, but might be computationally expensive. A second strategy is to use iterative methods [6,15,21], such as the Jacobi method or gradient descent. Unlike for direct methods, the result after every step of an iterative algorithm may be interpreted as an approximate solution to the problem, which keeps getting improved until a desired stopping criterion, e.g., a predefined precision is reached. While iterative methods provide only approximations of the system solution, they are generally less costly in terms of running time. For instance, the complexity of direct Gaussian elimination for a system of size $n$ is $O(n^3)$. In contrast, the iterative Jacobi method takes only $O(Nn^2)$ time, where $N$ is the number of iterations needed, which can usually be kept small.

* We thank Raf Vandebril and Francois Glineur for fruitful discussions and comments. This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, and the ARC (Action de Recherche Concertée) on Mining and Optimization of Big Data Models funded by the Wallonia-Brussels Federation.

†ICTEAM, Université catholique de Louvain & naXys, Université de Namur, Belgium (michael.schaub@uclouvain.be). Equally contributing first author

‡ICTEAM, Université catholique de Louvain, Belgium (maguy.trefois@uclouvain.be). Equally contributing first author

§ICTEAM, Université catholique de Louvain, Belgium (paul.vandooren@uclouvain.be).

¶ICTEAM and CORE, Université catholique de Louvain, Belgium (jean-charles.delvenne@uclouvain.be).
However, when the size $n$ of the system is large, effectively all classical direct and iterative methods become computationally prohibitive, unless the matrix is known to have a special structure (banded, Toeplitz, semiseparable, etc.). Methods which provide faster means for solving linear systems are thus highly demanded. For instance, a seminal result by Spielman and Teng [16], further improved and simplified in the last decade [4,10–14], shows that one can construct iterative algorithms to solve symmetric, diagonally dominant (SDD) systems in nearly-linear running time. Here, ‘nearly-linear’ refers to a complexity of the form $O(m \log^c m)$, where $m$ is the number of nonzero entries in the system matrix, and $c$ is an arbitrary positive constant. In this line of work, the problem is first reduced to solving a Laplacian system of the form $Lx = b$, where $L$ is the Laplacian matrix of an undirected graph. The Laplacian system is then solved efficiently using graph theoretic techniques.

In this paper, we propose a new class of iterative solvers with cheap iterations. From an abstract point of view, we consider the problem of finding the minimal norm vector $x$ within an affine space $X$. Note that this problem is naturally connected to iteratively solving a linear system: it amounts, e.g., to finding the minimal norm solution for an under-determined system $Ax = b$. Let $x_0 \in X$ be any point in our affine space. Then by updating $x$ within this search space along a set of chosen search directions $\{q_i\}$, one can find the minimal norm solution of $x$. More precisely, we iteratively solve:

\[
\begin{align*}
\min \|x\| \\
\text{s.t. } x - x_0 \in \text{span}(\{q_i\}).
\end{align*}
\]

The success of any iterative update scheme in solving such a problem depends on 2 intertwined factors. On the one hand, we would like to design our iterations such that each update brings us as close as possible towards the true solution. On the other hand we would like to make each iteration computationally as cheap as possible. Trivially, the best update in the first sense would be finding the correct solution directly, thus requiring only one iteration. However, this is clearly not satisfactory, or even feasible, if our initial problem evaded direct solution methods. A more realistic scheme, following this kind of update paradigm would be conjugate gradient descent, which tries to find good search directions at each step using gradient information. The downside of an approach like gradient descent is that each step can be computationally very costly, e.g., as in general all coordinates have to be updated at each step. On the other end of the spectrum in this respect are (canonical) coordinate descent approaches. Here the idea is to keep the updates very sparse and only update one (or a small number of $m$) coordinates at a time, thereby facilitating cheap iterations. However, as this imposes quite strong restrictions on the allowed search directions, this may in general result in a large number of iterations needed, possibly outweighing the gain in computational complexity for each iteration. A natural question is thus whether there is a way to perform cheap updates like in coordinate descent, while at the same time having more flexibility for the allowed search directions. As we show in the following the answer is indeed affirmative.

More specifically, in this work we show that if we choose a particular set of search directions for problem (1), given by the columns $q_i$ of a matrix $Q$, that we call $k$-sparsely factorizable, then all iterative updates of the form:

\[
x_{t+1} = x_t - x_t^T q_i \frac{q_i^T q_i}{q_i^T q_i}
\]
can be performed in $O(k)$ time, where $k$ is smaller than the dimension of the search space. Crucially, $Q$ can be a full matrix in general, i.e., our search directions are not limited to have only sparse support. For example, if $Q$ is hierarchical then it is $k$-sparsely factorizable, where $k$ is typically logarithmic in the size of the matrix. Using this decomposition we can provide a new iterative algorithm with a running time per iteration intermediate between coordinate descent and gradient descent. Finally, we show that this algorithm can be applied to solve Laplacian systems in nearly polynomial time, thereby establishing a connection to the previous literature. We remark that, rather than emphasizing one particular application and providing detailed implementations and simulations of our algorithms, the focus of the present paper is on the theoretical development of a new matrix factorization and its algebraic properties, which may then be used in a different context.

The outline of this paper is as follows. In Section 2, we first motivate and define a $k$-sparse matrix factorization, and then continue to show how one can compute an iteration of the form (2) in $O(k)$ time using this decomposition, which can be utilized to construct fast iterative solvers for linear systems. In Section 3, we review hierarchical matrices [8], as an example of $k$-sparse factorizable matrices where $k$ does not depend on the size of the matrix. In Section 4, we present an iterative algorithm that computes the minimal norm solution of an under-determined linear system and demonstrate how it can be applied if the system matrix is the incidence matrix of a graph. Section 5 showcases how we use our algorithm to solve in nearly-linear time a Laplacian system. Finally, Section 6 concludes the paper and discusses possible avenues for future work. To improve readability, some technical proofs are reported in the appendix.

2. A new sparse matrix factorization for fast iterative updates.

2.1. Background and Notation. We briefly recap the issue of finding the minimal norm solution of an under-determined system, which will be our guiding example in the following. For simplicity of notation we will consider only real vectors and matrices, although generalizations to the complex case are straightforward. Given a compatible linear system $Ax = b$, we are looking for the optimal solution of the following optimization problem:

$$
\min_{\|x\|} \quad \text{s.t. } Ax = b,
$$

where $\|x\| := \sqrt{x^T x}$. We denote this optimal solution by $x^*$:

$$
x^* := \arg \min_{s.t. Ax=b} \|x\|,
$$

(3)

This problem can be readily solved as follows. Suppose we are given a matrix $Q$ where the columns $q_i$ form a basis of the null space, null($A$), of $A$. If $x_0$ is a feasible solution to $Ax = b$, we can write (3) as

$$
x^* := \arg \min_{s.t. \ x=x_0+Qy} \|x\|,
$$

(4)

where $y$ is an arbitrary vector. Consequently, we may compute increasingly accurate approximations of $x^*$ by iteratively updating $x$ according to:

$$
x_{t+1} = x_t + \alpha^* q_i \quad \text{with} \quad \alpha^* = \arg \min_{\alpha \in \mathbb{R}} \|x_t + \alpha q_i\|.
$$
A short computation shows that \( \alpha^* = -\frac{x_t^T q_i}{q_i^T q_i} \) and therefore each iteration is of the form (2). Note that if we start with a feasible solution \( x_0 \), each iterate \( x_t \) is an exact solution of \( Ax = b \), since all updates added to \( x_0 \) are in the null space of \( A \). Therefore, the above iterative method converges to the optimal \( x^* \). Observe, that keeping track of the step sizes \( \alpha^* \) while following these update scheme effectively amounts to constructing the vector \( y^* \) such that \( Qy^* + x_0 = x^* \) in (4).

We remark that an iteration (2) also appears naturally when iteratively solving an over-determined system:

\[
\arg\min_y \|Ay - b\|. 
\]

By simply making the substitution \( x = Ay - b \), we can transform the above into the equivalent problem:

\[
\min \|x\| \\
\text{s.t. } x + b \in \text{span}(A),
\]
i.e., we are again trying to find the minimum norm solution of \( x \) within an affine space. Now an arbitrary \( y_0 \) will provide a starting point \( x_0 = Ay_0 - b \) for an iterative update procedure, and the search directions can be set to \( Q \), or more generally to \( Q = AS \) for any \( S \) with full-row-rank. While in the following we will concentrate on the under-determined case, most of our results can thus be simply recast, *mutatis mutandis*, to the over-determined setting.

### 2.2. \( k \)-sparse matrix factorization and efficient updates for iterative algorithms

We are now prepared to introduce the notion of \( k \)-sparse matrix factorization. Our motivation for this factorization is that it should enable fast iterative updates of the form (2), i.e., we would like that any iteration

\[
x_{t+1} = x_t - \frac{x_t^T q_i}{q_i^T q_i} q_i,
\]
can be computed in \( O(k) \) time, if \( q_i \) is a column of the \( k \)-sparsely factorizable matrix \( Q \). As we will show in the following, this will allow us to construct fast linear solvers.

The underlying idea is akin to the case where \( x_t \) and \( q_i \) are two vectors in \( \mathbb{R}^m \) and \( q_i \) is a sparse vector with only \( k \) non-zero entries. Then just \( k \) non-zero products need to be computed. Hence, the computational cost of the update is \( O(k) \). However, in order to solve a generic linear system efficiently, we need to ensure that we can find a set of vectors \( \{q_1, \ldots, q_n\} \) such that all necessary iterative updates can be performed with this complexity. This will be the key ingredient of our results on linear solvers presented in Section 4.

**Definition 1** (Support and sparsity of vectors and matrices). The support of a vector \( v = (v_1, \ldots, v_m)^T \in \mathbb{R}^m \) is the set of indices of the nonzero entries of \( v \):

\[
\text{supp}(v) = \{i \in \{1, \ldots, m\} : v_i \neq 0\}.
\]

A vector \( v \in \mathbb{R}^m \) is said to be \( k \)-sparse, if the size of its support, \( |\text{supp}(v)| \), is less than or equal to \( k \). Similarly, a matrix is said to be \( k \)-column (\( k \)-row) sparse if each of its columns (rows) is \( k \)-sparse.
Suppose that $x_i$ is not stored in the canonical basis, but in a different set of coordinates encoded by a matrix $C$. That is, instead of performing iterations (2) on $x_i$, we keep track of a vector $y_i$ such that $x_i = Cy_i$. To yield a sparse update, we may moreover choose $C$ such that $q$ is sparse in this representation, i.e., $q_i = Cd_i$, where $d_i$ is a $l$-sparse vector. This leads to an iteration of the form:

$$Cy_{t+1} = Cy_t - \frac{x_i^T q_i}{q_i^T q_i} Cd_i$$

Using this representation, every update would be sparse in that it would only effect $l$ components of $y$. However, this is not enough to perform each iteration (2) fast, as one also needs to compute the scalar product $x_i^T q_i$, which in the new basis becomes $y_i^T C^T Cd_i$, i.e., the iteration in terms of $y_t$ is of the form:

$$y_{t+1} = y_t - \frac{y_i^T C^T Cd_i}{d_i^T C^T Cd_i} d_i$$

To bound the complexity of this operation, one must understand the sparsity pattern of $C^T C$, which is dictated by how the supports of the columns of $C$ overlap. Observe that the $i$th column of $C^T C$ contains the scalar products between the $i$th column of $C$ and every column of the same matrix. Whence, if every column of $C$ overlaps in support with at most $c$ other columns, then every column of $C^T C$ contains at most $c$ non-zero entries. If we can find a matrix for which this is true, then $C^T Cd_i$ is a $k = cl$ sparse vector, since $d_i$ is $l$-sparse, and $y_i C^T Cd_i$ is computed in time $O(k)$. If we compile all such vectors $q$ into a matrix $Q$, then we say that $Q = CD$ is a $k$-sparse factorization.

While this reasoning provides us with some intuition, this definition must in fact be improved to reach tighter complexity bounds. First, we can exploit the symmetry of $C^T C$, by noting that it can be decomposed as $C^T C = U^T + U$, where $U$ is an upper-triangular matrix. Observe that the number of non-zero entries in the $i$th column of $U^T$ (or $i$th row of $U$) is bounded by the number of columns $c_j$ that overlap with $c_i$ for $j \geq i$. Second, two columns of $U^T$ may have their non-zero entries at the same positions. Therefore, the support of the sum of two columns does not necessarily increase. To bound the complexity we need to look at the size of the union of supports of all columns $u_j$ of $U^T$, for which $j$ belongs to the support of $d_i$. This number can indeed be much lower than the approximate estimate $cl$ above. This justifies the following definition.

**Definition 2.** Suppose a matrix $Q \in \mathbb{R}^{m \times n}$ has a factorization $Q = CD$. Let us denote the columns of $C \in \mathbb{R}^{m \times p}$ and $D \in \mathbb{R}^{p \times n}$ by $c_i$ and $d_j$, respectively. We define the forward-overlap $FO(c_i)$ of a column $c_i$ to be the list of columns $c_j$, with $j \geq i$, that have a support overlapping with the support of $c_i$. We call the factorization $Q = CD$ $k$-sparse if $|\bigcup_{j \in supp(d_j)} FO(c_i)| \leq k$ for all $j$ (see Figure 1 for an illustration). Without loss of generality each column of $C$ and each row of $D$ is supposed to be nonzero.

The example in Figure 1 shows an 8-sparse factorization of the given matrix $Q$. For instance, one can easily check that the forward overlap of column $c_{12}$ is $FO(c_{12}) = \{c_{12}, c_{13}, c_{16}, c_{20}\}$, and e.g.

$$|\bigcup_{i \in supp(d_i)} FO(c_i)| = |\{c_{11}, c_{12}, c_{13}, c_{15}, c_{16}, c_{19}, c_{20}\}| = 7 \leq k = 8.$$ 

To gain some further intuition, let us consider an alternative definition of a sparse factorization. Suppose we can define a partial order on the columns of $C$ with the
following properties. First, only columns with overlapping support are compared. Second, every set $T_i$ of columns of $C$ required to span a certain column $q_i$ has an upper set of at most $k$ elements (indeed the factorization $Q = CD$ expresses nothing but the fact that every column $q_i$ is a linear combination of a set $T_i$ of columns of $C$ with coefficients given by entries of $i$th column of $D$). Remember that the upper set consists of elements of $T_i$ and all columns of $C$ larger than an element of $T_i$ in the partial order.

The following properties of a $k$-sparse factorization are worth noting.

1. Any $m$-by-$n$ matrix $Q$ is min$(m,n)$-sparsely factorizable with either $Q = QI$ or $Q = IQ$.
2. If $Q = CD$ is a $k$-sparse factorization, then for every column $c_i$ of $C$, $|FO(c_i)| \leq k$, $C$ is $k$-row sparse and each column of $D$ is $k$-sparse.
3. Conversely, a matrix $C$ such that $|FO(c_i)| \leq k$ for all columns $c_i$ is trivially $k$-sparsely factorizable. A $k$-column sparse matrix $D$ is also trivially $k$-sparsely factorizable.
4. If $Q = CD$ is a $k$-sparse factorization and $F$ is $f$-column sparse, then $QF = C(DF)$ is a $kf$-sparse factorization of $QF$.
5. If $Q_1 = C_1D_1$ is a $k_1$-sparse factorization and $Q_2 = C_2D_2$ is a $k_2$-sparse factorization, then the matrix $(Q_1^T Q_2^T)^T$ is $(k_1 + k_2)$-sparsely factorizable. In order to see this, we write

$$
\begin{pmatrix}
Q_1 \\
Q_2
\end{pmatrix} =
\begin{pmatrix}
C_1 \\
C_2
\end{pmatrix}
\begin{pmatrix}
D_1 \\
D_2
\end{pmatrix}.
$$

In particular, if $Q_2$ is the identity, the compound matrix is $(k_1 + 1)$-sparsely factorizable.
The following theorem establishes the running time of \( N \) iterations of the form (2), when the vectors \( q \) are the columns of a \( k \)-sparsely factorizable matrix. The proof of the theorem is given in the appendix.

**Theorem 3.** Let \( Q \in \mathbb{R}^{m \times n}, C \in \mathbb{R}^{m \times p} \) and \( D \in \mathbb{R}^{p \times n} \) be matrices such that \( Q = CD \) is a \( k \)-sparse factorization of \( Q \), and consider iterations of the form (2) that start from an arbitrary vector \( x_0 \in \mathbb{R}^m \). If every \( q_i \) in (2) is a column of \( Q \), then the computational complexity of running \( N \) iterations of (2) is:

\[
O(Nk + pk + (m + n)k^2).
\]

With the same complexity, we can compute a \( y_N \) such that \( x_N = x_0 + Qy_N \), where \( x_N \) denotes the vector resulting from the \( N \) first iterations.

The remarkable point about Theorem 3 is that the running time of each iteration is merely \( O(k) \), even if some columns of \( Q \) are full. Hence, if \( k \ll m \), then the cost per iteration can be largely reduced through the use of a \( k \)-sparse factorization, and the overhead terms \( pk + (m + n)k^2 \) are more than compensated.

### 2.3. Ensuring fast convergence of efficient updates.

From our discussion above, we know that after sufficiently many iterations (2) over all columns of \( Q \), \( x_t \) converges to \( x^* \), which is the solution of

\[
(6) \quad x^* = \arg \min_{x \in x_0 + \text{Im}(Q)} \|x\|_2
\]

However, to ensure that we can construct an efficient algorithm based on such cheap updates, we need to ensure that the number of updates needed is not too large, as this would undermine the purpose of the fast updates. Stated differently, we need to ensure that the convergence rate of our iterations to the solution is not too slow.

Remarkably, using a random sampling of the columns of \( Q \) one can indeed ensure a sufficient convergence rate. To this end, at each iteration randomly select a column \( q_i \) with probability proportional to \( \|q_i\| \). This guarantees a convergence rate of the form

\[
\mathbb{E}\|x_t - x^*\|_2^2 = \left(1 - \frac{\sigma_{\text{min}}^2(Q)}{\|Q\|_{\text{Frob}}^2}\right)^t \|x_0 - x^*\|_2^2,
\]

where \( \|Q\|_{\text{Frob}} = \text{Tr}(Q^TQ) \) is the Frobenius norm and \( \sigma_{\text{min}}^2(Q) = \lambda_{\text{min}}(Q^TQ) \) is the smallest nonzero squared singular value \([7,17]\). The proof of this result is provided in the appendix.

The above results states effectively that the expected error in computing \( x^* \) is decreased by an order of magnitude (say by a factor 10) after \( N = O(\|Q\|_{\text{Frob}}^2/\sigma_{\text{min}}^2(Q)) \) iterations. The main challenge for the construction of a fast algorithm is thus to find a matrix \( Q \) spanning the desired search space, with efficient \( k \)-sparse factorization and low “condition number” \( \|Q\|_{\text{Frob}}/\sigma_{\text{min}}(Q) \).

### 3. A class of sparsely factorizable matrices: hierarchical matrices.

As our above result illustrates, \( k \)-sparsely factorizable matrices are extremely well-suited to reduce the complexity of an iteration of the form (2). A natural question is therefore what kind of matrices are \( k \)-sparsely factorizable. In the following, we will discuss hierarchical \( H_r \)-matrices, originally introduced by Hackbusch \([8]\), and show that they are \( k \)-sparsely factorizable. Importantly, in this case \( k \) depends only on the height or the degree of the hierarchical structure.
3.1. Definition of an $H_r$-matrix. As the name suggests, $H_r$-matrices are intimately related to hierarchical structures. As a hierarchy may be aptly represented as a tree we introduce these matrices here with the help of (tree-)graphs. As we will see this also enables us to establish a connection to graph-theoretic algorithms for solving Laplacian systems in subsequent sections.

**Definition 4 (Dendrogram).** A **dendrogram** is a hierarchical partitioning $\mathcal{P}$ of the set $\{1, \ldots, n\}$, namely a sequence of increasingly finer partitions $P_h, \ldots, P_0$ starting from the coarsest (global) partition $P_h$ given by the whole set, up to the finest (singleton) partition $P_0$ into $n$ sets. Every dendrogram is conveniently represented by a rooted directed tree. The nodes of this tree at height $i$ are the subsets of partition $P_i$. Thus the root ($i = h$) is the full set while the leaves ($i = 0$) are the $n$ single-element subsets. The children (out-neighbours) of a node at height $i$ correspond to the subsets of this node as specified by the next lower partition $P_{i-1}$. We call $h$ the height of the dendrogram, and the maximum number of children of a node in the tree is denoted as maximum degree $d$.

Figure 2a shows an example of a dendrogram with height 3 and maximum degree 2. For simplicity of notation and without loss of generality, we suppose throughout the paper that every node of a dendrogram has consecutive elements.

A dendrogram $\mathcal{P}$ induces a hierarchical block segmentation of a matrix $E \in \mathbb{R}^{n \times n}$ as follows. Let us denote the degree of the root node by $t \leq d$. The rows and columns of $E$ are first block-partitioned according to the partition $P_{h-1}$:

\[
E = \begin{pmatrix}
E_{I_1 \times I_1} & E_{I_1 \times I_2} & \cdots & E_{I_1 \times I_t} \\
E_{I_2 \times I_1} & E_{I_2 \times I_2} & \cdots & E_{I_2 \times I_t} \\
\vdots & \vdots & \ddots & \vdots \\
E_{I_t \times I_1} & E_{I_t \times I_2} & \cdots & E_{I_t \times I_t}
\end{pmatrix},
\]

where $I_1, \ldots, I_t$ are the elements of partition $P_{h-1}$. The diagonal blocks $E_{I_i \times I_i}$ are recursively sub-partitioned according to $P_{h-2}$, etc. This partitioning of $E$ is called $\mathcal{P}$-partitioning. See Figure 2(b) for an illustration.
**Definition 5. (Elementary block)** We use the term elementary block to refer to a sub-matrix of \( E \) generated by the \( \mathcal{P} \)-partitioning, that is not further subdivided. In other words it is a block of the form \( E_{I_i \times I_j} \) where \( I_i \) and \( I_j \) are either two different sets appearing in one same partition \( P_k \), or two single-element sets of the finest partition \( P_0 \).

**Definition 6. (Hierarchical Matrix)** An \( \mathcal{H}_r(\mathcal{P}) \)-matrix is a square matrix structured according to the dendrogram \( \mathcal{P} \), for which the elementary blocks have rank at most \( r \in \mathbb{N} \). We use the shorthand \( \mathcal{H}_r \) when the dendrogram is clear from the context. Note that a sub-matrix \( E_{I_i \times I_j} \) of an \( \mathcal{H}_r(\mathcal{P}) \)-matrix \( E \), where \( I_i \) is a set of some partition \( P_k \), is an \( \mathcal{H}_r \)-matrix as well.

### 3.2. Sparse factorization property.
In what follows, we show that an \( \mathcal{H}_r(\mathcal{P}) \)-matrix is \( k \)-sparsely factorizable and express \( k \) in terms of the rank \( r \), maximum degree \( d \) and height \( h \).

Recall that an \( \mathcal{H}_r(\mathcal{P}) \)-matrix \( E \) is of the form (7). Every non-elementary block \( E_{I_i \times I_j} \) on the diagonal is recursively of the same form until the diagonal block is just a scalar. Hence, every diagonal non-elementary block is a hierarchical matrix, too. Further, note that every column of the full matrix \( E \) is built by concatenating the corresponding columns of the \( E_{I_i \times I_j} \) blocks. For example, the first column of \( E \) can be built by stacking up the first columns of \( E_{I_1 \times I_1}, E_{I_2 \times I_2}, \ldots, E_{I_t \times I_t} \).

We can thus build a \( k \)-sparse factorization \( E = CD \) as follows. As every off-diagonal elementary block \( E_{I_i \times I_j} \) has a rank of at most \( r \), there is a matrix \( D_{ij} \) such that the elementary block can be decomposed as \( E_{I_i \times I_j} = C_{ij} D_{ij} \), where \( C_{ij} \) has at most \( r \) columns. Thus, we know how to express all the elements in the off-diagonal blocks using this factorization. Hence, if we knew a sparse decomposition of the diagonal blocks \( E_{I_i \times I_i} = C_{ii} D_{ii} \), we could assemble the whole matrix \( E \) by appropriate concatenation of the matrices \( C_{ij} \).

To factorize the diagonal blocks we apply this construction recursively. To make the recursion well defined, if the diagonal block \( E \) is a scalar (a \( 1 \times 1 \) matrix), we define \( E = CD \), where \( C \) is an arbitrary nonzero scalar, for instance we take \( C = E \) and take \( D = 1 \). Decomposing the columns of \( E \) in this recursive way, we obtain a sparse factorization \( E = CD \).

We illustrate this for the case \( t = 3 \), hereafter. For each \( i \in \{1, 2, 3\} \), let each diagonal block \( E_{I_i \times I_i} = C_{ii} D_{ii} \) be a \( k_i \)-sparse decomposition (recursively), and recall that each elementary block \( E_{I_i \times I_j} (i \neq j) \) can be factorized as \( E_{I_i \times I_j} = C_{ij} D_{ij} \). Then a \( k \)-sparse factorization of \( E \) is given by:

\[
E = \begin{pmatrix}
C_{11} & C_{12} & C_{13} \\
C_{21} & C_{22} & C_{23} \\
C_{31} & C_{32} & C_{33}
\end{pmatrix}
\quad \begin{pmatrix}
D_{11} & 0 & 0 \\
0 & D_{12} & 0 \\
0 & 0 & D_{13}
\end{pmatrix}
\quad \begin{pmatrix}
D_{22} & 0 & 0 \\
D_{21} & 0 & 0 \\
0 & 0 & D_{23}
\end{pmatrix}
\quad \begin{pmatrix}
D_{33} & 0 & 0 \\
D_{31} & 0 & 0 \\
0 & 0 & D_{32}
\end{pmatrix}
\quad \begin{pmatrix}
D
\end{pmatrix}
\]

where \( C_{11}, C_{22}, C_{33} \) are recursively defined according to the diagonal blocks of \( E \).
Having thus found a possible factorization, the question remains what sparsity, \( k \), it affords. To answer this questions, let us first consider, as an illustration, the union of the forward overlaps of those columns of \( C \), that are necessary to build the first columns of \( E \). There are two types of columns in \( C \):

1. the columns belonging to the \((C_{11} \ 0 \ 0)^T\) block. Their forward-overlap is \( k_1 + r(t - 1) \), where \( k_1 \) is the sparsity of the factorization of \( C_1 \), and the \( r(t - 1) \) term accounts for the overlap with the \((t - 1)\) \( r \)-column matrices \( C_{12} \) and \( C_{13} \).

2. The columns from the \( B \) blocks \((0 \ C_{21} \ 0)^T\) and \((0 \ 0 \ C_{31})^T\). Their forward overlap is \( r(t - 1) \) at most.

As this argument holds for any column of \( E \), the factorization \( E = CD \) is \( k \)-sparse for \( k = \max_i k_i + r(t - 1) \), where \( k_i \) is determined recursively from the decomposition of the diagonal block \( E_{I_i \times I_i} \). Unravelling the recursion over all \( h \) levels, we find that \( k = rd(d - 1)(h + 1) \), where \( d \) is the maximal degree of the dendrogram, as before.

Throughout the paper, in a \( k \)-sparse factorization \( E = CD \) of an \( H_\mathcal{P}(\mathcal{P}) \)-matrix, the matrix \( C \) is supposed to be of the generic form (8), for an accordingly determined degree \( d \). We will call this type of matrix a \( C \)-matrix. We now prove a property of \( C \)-matrices, which will be of interest in the next sections.

**Lemma 7.** The number of columns in \( C \) in the recursive construction in (8) is given by \( p \leq rd^2n \).

**Proof.** By induction on \( n \), we prove that \( p \leq rd(d - 1) \left( \frac{d}{d - 1} n - \frac{1}{d - 1} \right) \).

1. If \( n = 2 \), then \( d = 2 \) and \( p \leq 4 \leq rd(d - 1) \left( \frac{d}{d - 1} n - \frac{1}{d - 1} \right) \leq rd^2n \).

2. If \( n > 2 \), then \( E \) is of the form (7). Below, the size of a diagonal block \( E_{I_i \times I_i} \) is denoted by \( n_i \). Note that \( n = \sum_{i=1}^{d} n_i \).

   By construction of \( C \), we know that \( p \leq r(d - 1)d + \sum_{i=1}^{d} p_i \), where \( p_i \) is the maximum number of columns in the matrix \( C_i \) of \( E_{I_i \times I_i} \) \((1 \leq i \leq d)\). Consequently, by induction we have

\[
p \leq r(d - 1)d + r(d - 1)d \sum_{i=1}^{d} \left( \frac{d}{d - 1} n_i - \frac{1}{d - 1} \right) = r(d - 1)d \left( \frac{d}{d - 1} n - \frac{1}{d - 1} \right) \leq rd^2n.
\]

We formalize the above findings in the following theorem.

**Theorem 8.** Let \( E \in \mathbb{R}^{n \times n} \) be an \( H_\mathcal{P}(\mathcal{P}) \)-matrix with a dendrogram \( \mathcal{P} \) of height \( h \) and maximum degree \( d \). Then, there are matrices \( C \in \mathbb{R}^{n \times p} \) and \( D \in \mathbb{R}^{p \times n} \) such that \( p \leq rd^2n \) and the factorization \( E = CD \) is \( k \)-sparse for \( k = rd(d - 1)(h + 1) \).

### 3.3. Examples of hierarchical matrices.

#### 3.3.1. Example 1: semiseparable matrices.

**Definition 9.** [18] An \( n \times n \) matrix \( E \) is called \((p, q)\)-semiseparable if the following relations are satisfied:

\[
\text{rank}(E(1 : i + q - 1, i : n)) \leq q \quad \text{and} \quad \text{rank}(E(i : n, 1 : i + p - 1)) \leq p
\]

for all feasible \( 1 \leq i \leq n \).
THEOREM 10. An \( n \times n \) matrix that is \( (p,q) \)-semiseparable is an \( \mathcal{H}_{r}(\mathcal{P}) \)-matrix where \( r = \max\{p,q\} \) and \( \mathcal{P} \) is a binary dendrogram.

The proof is given in the appendix. We note that, due to their remarkable structural properties, algorithms solving semiseparable systems in linear time are well known in the literature [19, 20].

3.3.2. Example 2: reduced incidence matrices of trees and their inverse. In what follows, we define a reduced incidence matrix of a tree, and show that it is essentially an \( \mathcal{H}_{1}(\mathcal{P}) \)-matrix where \( \mathcal{P} \) is a binary dendrogram \( (d = 2) \).

We first give the definitions of an incidence matrix of a graph and of a reduced incidence matrix of a tree.

DEFINITION 11 (Incidence matrix, reduced incidence matrix). Let \( G \) be a positively weighted undirected graph on \( n \) nodes and \( m \) edges with an arbitrary direction chosen for each edge. An incidence matrix \( B \in \mathbb{R}^{n \times m} \) of \( G \) is a node-by-edge matrix such that given an edge \( e \) of \( G \) from node \( i \) to node \( j \) with weight \( w \), the \( i \)th column of \( B \) takes value \( -\sqrt{w} \) at the source node \( i \), value \( \sqrt{w} \) at the target node \( j \) and value 0 at any other node.

A reduced incidence matrix of a graph \( G \) is an incidence matrix of \( G \) from which one row has been removed.

To reveal the hierarchical structure in the reduced incidence matrix of a tree, one has to recursively split the nodes of the tree in a balanced way. A classic way to do so is provided by the tree-vertex-separator lemma.

LEMMA 12 (Tree Vertex Separator Lemma, [3, 9]). For any forest \( T \) with \( n \geq 2 \) nodes, one can divide \( T \) into two forests both of at most \( 2n/3 \) nodes, by removing at most one node \( d \), which can be computed in \( \mathcal{O}(n) \) time.

PROPOSITION 13. A reduced incidence matrix of an \( n \)-edge tree is, for some ordering of the nodes and edges, an upper-triangular \( \mathcal{H}_{1}(\mathcal{P}) \)-matrix for a binary dendrogram \( \mathcal{P} \) with height \( \mathcal{O}(\log n) \). The inverse of the reduced incidence matrix is, for the same ordering of nodes and edges, also an upper-triangular \( \mathcal{H}_{1}(\mathcal{P}) \)-matrix. The dendrogram \( \mathcal{P} \) and both hierarchical matrices can be computed in time \( \mathcal{O}(n \log n) \). Thus, a \( \mathcal{O}(\log n) \)-sparse factorization of (the inverse of) such a hierarchical matrix is computable in time \( \mathcal{O}(n \log^{2} n) \).

Proof. A tree \( T \) of \( n \) nodes has \( n - 1 \) edges, hence an \( n \)-by-(\( n - 1 \)) incidence matrix. Note that in this proof we consider \( T \) as an undirected tree with root \( v \) and the arbitrary direction on the edges used to assign a sign on the entries of the incidence matrix play no role here. Removing a row corresponding to, say, node \( v \), we obtain a square reduced incidence matrix of dimension \( n - 1 \).

We now order the nodes in two blocks as suggested by the Tree Vertex Separator Lemma. This splits the forests of non-root tree \( T \) into two forests \( T_{1} \) and \( T_{2} \), each with no more than \( 2n/3 \) nodes, where the separator node \( d \) (if any) is assigned to \( T_{2} \). The indices of the edges are assigned as follows. An edge connecting node \( i \) and \( j \), indexed by \( j \), if \( j \) is the son of \( i \) (i.e. one step further away from the root than \( i \)).

This splits the reduced incidence matrix \( E \) of \( T \) as

\[
E = \begin{pmatrix}
E_{I_{1} \times I_{1}} & E_{I_{1} \times I_{2}} \\
0 & E_{I_{2} \times I_{2}}
\end{pmatrix},
\]

where \( E_{I_{i} \times I_{i}} \) (for \( i = 1, 2 \)) is the reduced incidence matrix of \( T_{i} \) and \( E_{I_{1} \times I_{2}} \) is a rank-1 matrix with at most one non-zero entry corresponding to the edge linking \( d \) to its
We repeat this argument recursively which creates a dendrogram $P$ on the nodes of $T$ of height $O(\log n)$ and a corresponding upper triangular $H_1(P)$-matrix structure for $E$. From the ordering of edges, we see that the $i$th node is always incident to the $i$th edge, thus the diagonal entry of $E$ is $\pm \sqrt{w_i}$, making it invertible. The inverse of $E$ is computed recursively as

$$E^{-1} = \begin{pmatrix} E_{f_1}^{-1} & F \\ 0 & E_{f_2}^{-1} \end{pmatrix},$$

with $F = -E_{f_1}^{-1}E_{f_2}^{-1}$. Note that we may write $F = uv^T$ as it is clearly of rank one at most, thus leading to an upper-triangular $H_1(P)$-matrix for $E^{-1}$ as well. Both for $E$ and $E^{-1}$, every of the $O(\log n)$ steps of the recursion takes $O(n)$, required to finding the tree vertex separators and (in case of $E^{-1}$) computing $u$ and $v$, solutions of triangular systems. Therefore we get a total cost of $O(n \log n)$.

Finally, using the procedure outlined above we can decompose $E^{-1} = CD$. Using $E_{f_1} = C_{11}D_{11}$ and $E_{f_2} = C_{22}D_{22}$, we recursively construct:

$$E^{-1} = \begin{pmatrix} C_{11} & u \\ C_{22} & D_{22} \end{pmatrix}.$$ 

By unfolding this recursion we can see that this leads to a forward-overlap of size $O(\log(n))$ in $C$, and an $O(\log(n))$ column-sparse matrix $D$. Similarly, a $O(\log n)$-sparse factorization can be obtained for $E$.

4. Iterative linear solvers with fast iterations. To illustrate the utility of our results, in the following we showcase two concrete application scenarios in which the above developed theory can be employed.

4.1. A strategy for solving under-determined systems. In the following theorem, we focus again on the case of an under-determined system $Ax = b$. We devise a strategy that assumes a decomposition of the $n$-by-$m$ full-rank matrix $A$ (with $n < m$) of the form $A = (E \ F)$, where $E$ is an invertible $n \times n$ submatrix of $A$.

A matrix $Q$ whose columns span the null space of $A$ can then be constructed as:

$$Q = \begin{pmatrix} E^{-1}F \\ -I_{m-n} \end{pmatrix},$$

where $I_{m-n}$ is the identity matrix of dimension $m - n$. Indeed, we clearly have $AQ = 0$, and thus the columns of $Q$ belong to the null space of $A$. The rank of $Q$ is $m - n$, which is the dimension of null($A$).

If $E^{-1}$ is $k$-sparsely factorizable as $E^{-1} = CD$ and $F$ is $f$-column sparse, then we know from elementary properties of our sparse factorization that $E^{-1}F$ is $(kf)$-sparse factorizable. Further we know that $Q$ is $(kf + 1)$-sparsely factorizable, with factorization:

$$Q = \hat{C}\hat{D} = \begin{pmatrix} C & 0 \\ 0 & I_{m-n} \end{pmatrix} \begin{pmatrix} DF \\ -I_{m-n} \end{pmatrix}.$$ 

Moreover, we have that $\sigma_{\min}(Q) = \lambda_{\min}(F^T E^{-T} E^{-1} F + I_{m-n}) \geq 1$. Following section 2.3, the number of steps to decrease the error by one order of magnitude is
then at most of the order of \( \|Q\|_{\text{Frob}} = \|E^{-1}F\|_{\text{Frob}} + m - n \). Hence, we have to find \( E \) such that \( k, f \) and \( \|E^{-1}F\|_{\text{Frob}} \) are small, in particular. Let us consider for instance the case where \( E^{-1} \) is hierarchical. We can then combine Theorems 3 and 8 to obtain the following.

**Theorem 14.** Let \( A = (E \ F) \) be an \( n \times m \) matrix with \( n < m \), where \( E \in \mathbb{R}^{n \times n} \) is invertible and \( E^{-1} \) is an \( H_r(P) \)-matrix with an associated dendrogram \( P \) of maximum degree \( d \) and height \( h \). Further, let \( F \in \mathbb{R}^{n \times (m-n)} \) be \( f \)-column sparse. Then, we can compute an approximation of \( x^* := \arg \min_{x, t} Ax = b \|x\| \) by applying \( N \) iterations of the form (2), in time

\[
O(N frd^2 h + mf f^2 d^4 h^2 + \text{Cost}(CD)),
\]

where \( \text{Cost}(CD) \) is the cost of computing a \((rd^2(h+1))\)-sparse factorization of \( E^{-1} \). The number of iterations to gain one order of magnitude on the error is at most \( N = O(\|E^{-1}F\|) + m \).

**Proof.** According to Theorem 8, let \( E^{-1} = CD \) be a \( k \)-sparse factorization with \( k = rd(d-1)(h+1) \). By the second elementary property (see Property 2 on page 6) of sparse factorization, we know that \( C \) is \( k \)-row sparse and that each column of \( D \) is \( k \)-sparse. A feasible solution to \( Ax = b \) is then computed in \( O(kn) \) time. Indeed, such a solution is given by \( x_0 = (E^{-1} b) \) where \( E^{-1} b = CD b \) is computed in \( O(kn) \) time.

Now, consider the matrix \( Q \) given in (9). From our discussion above we know that the columns of \( Q \) are a basis of \( \text{null}(A) \) and that the matrix \( Q \) is \((kf+1)\)-sparsely factorizable. Let \( Q = \tilde{C} \tilde{D} \) be the \((kf+1)\)-sparse factorization given in (10). We start from the vector \( x_0 \) and iteratively pick a column \( q \) of \( Q \) and perform an iteration of the form (2). Theorem 3 with \( Q, \tilde{C} \) and \( \tilde{D} \) then shows that the running time is given by

\[
O(N k f + pk f + mk f^2 + \text{Cost}(CD)).
\]

We conclude the proof by noting that \( k = rd(d-1)(h+1) \) and that \( p \leq rd^2n \) (see Theorem 8).

**4.2. Square hierarchical systems.** The present technique can be also applied to solve square systems \( Ax = b \), where \( A \) is hierarchical and invertible.

**Theorem 15.** The system \( Ay = b \), where \( A \) is an invertible \( n \)-by-\( n \) \( H_r(P) \)-matrix with \( P \) a dendrogram of degree \( d \) and height \( h \), can be solved iteratively in time

\[
O(N rd^2 h + nr^2 d^4 h^2 + \text{Cost}(CD)),
\]

where \( N \) is the number of iterations and \( \text{Cost}(CD) \) is the running time needed to compute a \( k \)-sparse factorization of \( A \) with \( k = rd(d-1)(h+1) \).

**Proof.** In section 2.1, page 4, we show how to solve an over-determined system using iterations (2). Trivially, we can use the presented method for the square system \( Ay = b \). Following the notations of Theorem 3, here \( Q = A \), \( m = n \) and the running time is

\[
O(N k + pk + nk^2 + \text{Cost}(CD)).
\]

We moreover use Theorem 8 which states that \( p \leq rd^2n \) and \( k = rd(d-1)(h+1) \) to deduce that the running time is

\[
O(N rd^2 h + nr^2 d^4 h^2 + \text{Cost}(CD)).
\]
In particular, if $A$ is an $\mathcal{H}_1(\mathcal{P})$-matrix with $\mathcal{P}$ a binary dendrogram of height $h = O(\log n)$ (for instance, $A$ could be the reduced incidence matrix of a tree), then this running time becomes
\[
O(N \log n + n \log^2 n),
\]
where we have used Proposition 13 which states that a sparse factorization of $A$ is computed in time $O(n \log^2 n)$.

As far as we know, this is the best iterative method in terms of cost per iteration ($\log n$). Most standard methods would exhibit a cost of $O(n)$ per iteration, the cost of a matrix-vector product. However, a direct method exists that solves such problem in $O(n \log^2 n)$\textsuperscript{[2]}.  

5. Solving Laplacian systems in nearly linear time. In the following we demonstrate how the approach outlined above can be used to solve Laplacian systems. To this end we will make use of the following corollary.

Corollary 16. Let $A$ be a reduced incidence matrix of a connected undirected graph on $n$ nodes and $m$ edges. Then, the minimal norm solution of a compatible system of the form $Ax = b$ can be computed to any relative accuracy (defined as the ratio norm-of-the-error to norm-of-the-solution) $\epsilon$ in $O(m \log^2 n \log \log n \log(mn^{-1}))$ time.

Proof. Choosing an invertible (sub-)matrix $E$ such that $A = (E \ F)$ is equivalent to selecting a matrix $E$ that corresponds to the reduced incidence matrix of a spanning tree of $G$, with arbitrary orientation chosen for the edges. We now claim that we can choose $E$, i.e., choose an appropriate spanning tree, such that $\|E^{-1}F\|_{\text{Frob}} = O(m \log n \log \log n)$.

For any choice of spanning tree, we define the root as the node whose row has been removed from the incidence matrix to obtain a reduced incidence matrix. We choose the (arbitrary) orientation on the edges so as to go from root to leaves. We order the nodes from root to leaves (topological order) and edges so that any edge has the same index as its destination. With this choice, the unweighted, directed adjacency matrix $T$ of the tree is upper triangular. Then we can write $E = (I - T)\sqrt{W_T}$ where $W_T$ is the diagonal matrix weights on the edges.

Using a Neumann series expansion we can see that $E^{-1} = W_T^{-1/2}(I + T + T^2 + T^3 + \cdots + T^h)$ where $h$ is the height of the tree. The columns of $E^{-1}$ encode the paths between root and leaves, with positive inverse square root of the edge-weights as entries.

Therefore each column $i$ of $E^{-1}F$, which is the difference between two columns of $E^{-1}$, describes the (signed) path in the tree between the extremities of edge $i$, on which each edge $e$ has weight $\sqrt{w_i/w_e}$. Therefore the square Frobenius norm of $E^{-1}F$ is the so-called stretch of the tree in the graph with inverse weights, i.e. weight $w_e^{-1}$ on each edge $e$ of the graph, as already noticed in [10]. Using the algorithm in Ref. [1] we can therefore find a tree $E$ such that $\|E^{-1}F\|_{\text{Frob}}^2 = O(m \log n \log \log n)$ in time $O(m \log n \log \log n)$, where $m$ is the number of edges in the graph [1].

As shown by Proposition 13, $E^{-1}$ is an $\mathcal{H}_1(\mathcal{P})$-matrix with $\mathcal{P}$ a binary dendrogram of height $h = O(\log n)$, a sparse decomposition of which can be computed in time $O(n \log^2 n)$. From Theorem 14, we then derive a nearly-linear convergence time to compute the minimal norm solution of $Ax = b$. More precisely, the running time needed to decrease the square error by an order of magnitude is given by $O(m \log^2 n)$. One can start from the solution $x_0 = (E_0^{-1}b)$, computed in $O(n)$ time (solving a triangular system). Then we find
\[ \|x\|^2 \leq \|x_0\|^2 = \|E^{-1}b\|^2 = \|E^{-1}Ax\|^2 = \|(I - E^{-1}F)x\|^2 \leq O(m \log n \log \log n) \|x\|^2. \]

Therefore one needs to improve the initial relative accuracy \( \|x_0 - x\|/\|x\| \) by at most \( \log(m \epsilon^{-1}) \) orders of magnitude. The total computation time need for this is \( O(m \log^2 n \log \log n \log(m \epsilon^{-1})) \) thanks to Theorem 14 where \( f \) and \( r \) are constants and \( h = O(\log n) \).

The above corollary provides the critical step in solving a compatible Laplacian system \( Ly = c \), where \( L \) is the Laplacian of the same graph, as we show now. For a given incidence matrix \( B \) the Laplacian is defined as \( L = BB^T \), or equivalently as the node-by-node matrix with entries \( L_{ij} = -w_{ij} \) for every edge \( ij \) of weight \( w_{ij} \), \( L_{ij} = 0 \) if \( i \) is not adjacent to \( j \), and the weighted degree \( L_{ii} = \sum_k w_{ik} \) on diagonal entries. Such a system \( Ly = c \) can be solved in two steps:

1. solve \( Bx = c \) so that \( x \) is in the image of \( B^T \);
2. solve the over-determined system \( B^Ty = x \).

Note that this strategy of splitting the problem of solving a Laplacian system into 2 parts is in line with the approach followed by Kelner et al. [10], albeit their algorithm relies on graph-theoretic notions, rather than a matrix decomposition.

The first step is equivalent to finding the minimum-norm solution of \( Bx = c \). Indeed any solution of \( Bx = c \) is of the form \( x = B^Ty + v \), for any \( v \) such that \( Bv = 0 \). This implies that \( v \) is orthogonal to \( B^Ty \), thus \( B^Ty + v \) has a norm larger than \( B^Ty \), except for \( v = 0 \), where the minimum norm is reached. The goal is therefore to solve \( Bx = c \) in the minimum norm sense. Since the columns of \( B \) sum to zero, every single row is redundant to the system, thus we can remove an arbitrary row without affecting the solution. Calling \( A \) the so-obtained reduced incidence matrix of the graph, and \( b \) the vector obtained from \( c \) by removing one entry, and one has to solve \( Ax = b \), for \( A \) a reduced incidence matrix, which can be done efficiently as discussed above.

Note that the second step involves the solution of a compatible over-determined system. This can be solved by the square invertible triangular subsystem \( E^Ty = x_E \) where \( E \) is the reduced incidence matrix of the spanning tree used to solve \( Ax = b \) (see the proof of Corollary 16) and \( x_E \) is the corresponding part of vector \( x \). This triangular system can be solved in \( O(n) \) time, from leaves to root.

Solving a semi-definite positive system \( Ly = c \), the \( L \)-pseudo-norm \( \|v\|_L^2 = v^TLv \) is often used as the error norm. Note that all \( \|v\|_L^2 \) vanishes only if vector \( v \) has identical entries. The relative accuracy of iterate \( x_t \) is therefore defined as \( \|x_t - x\|_L/\|x\|_L \).

Putting these pieces together, we obtain the following theorem:

**Theorem 17 (Kelner et al. [10]).** Given a Laplacian matrix \( L \) of a connected graph with \( m \) edges, and any zero-sum vector \( c \), the (compatible) system \( Ly = c \) can be solved with relative accuracy \( \epsilon \), as measured in the \( L \)-pseudo-norm, within time \( O(m \log n \log \log n \log(m \epsilon^{-1})) \).
Proof. From Corollary 16 we find an approximate solution $x + \Delta x$ to the problem $Bx = c$, with $\|\Delta x\|/\|x\| \leq \delta$, in time $O(m \log n \log \log n \log(n\delta^{-1}))$.

We then find the approximate solution $y + \Delta y$ as $E^{-T}(x + \Delta x)$. The error $\Delta y$ checks, using $L = BB^T$ and $B = (E \ F)$:

\begin{align*}
    \|\Delta y\|^2_L &= \|E^{-T}\Delta x\|^2_L \\
    &= \|(I - E^{-1}F)^T \Delta x\|^2 \\
    &\leq O(m \log n \log \log n) \|\Delta x\|^2
\end{align*}

Moreover the exact solution verifies $\|y\|^2_L = \|x\|^2$ by definition $x = B^Ty$. Therefore we see that the relative accuracy on $x$ in terms of $\|\cdot\|_L$ is

$$\frac{\|\Delta x\|^2_L}{\|x\|^2_L} = O(m \log n \log \log n) \frac{\|\Delta y\|^2}{\|y\|^2}$$

Therefore we can choose $\delta = O(m \log n \log \log n) \epsilon$, for any required accuracy level $\epsilon$, and we conclude the proof by Corollary 16.

6. Conclusion. In this paper we have considered the problem of finding the minimum norm vector $x$ within an affine space, which arises naturally when solving an under- or over-determined linear system. We have shown that this problem can be solved very efficiently in an iterative manner by choosing the matrix of search directions $Q = (q_1, \ldots, q_m)$ for our updates of the form $x_{t+1} = x_t - \frac{x^T q_i}{q_i^T q_i} q_i$ in an appropriate way. Specifically, if there exists a $k$-sparse matrix factorization of $Q$, each iterative update can be computed in $O(k)$ time, enabling us to construct fast algorithm for solving linear systems. The notion of a $k$-sparse matrix factorization is indeed central to these findings, as it ensures the existence of a computationally efficient update scheme despite the fact that $Q$ might be full, i.e., the search directions are not formed by sparse vectors.

We have shown that the important class of hierarchical matrices are $k$-sparsely factorizable, and in particular that $k$ here does not depend on the size of the matrix, but merely on the depth of the hierarchy. From this, we have deduced an iterative method with fast iterations that approximates the minimal norm solution of under-determined linear systems. In particular, this approach can be applied when the coefficient matrix is the incidence matrix of a connected graph. From this we have deduced a method to solve Laplacian systems in nearly-linear time. In this context, our work provides a complementary algebraic perspective to the problem of solving Laplacian system, and connects combinatorial and graph-theoretic notions with the problem of finding a $k$-sparse matrix factorization.

REFERENCES

[1] I. Abraham and O. Neiman, Using petal-decompositions to build a low stretch spanning tree, in Proceedings of the forty-fourth annual ACM symposium on Theory of computing, ACM, 2012, pp. 395–406.
[2] S. Ambikasaran and E. Darve, An $O(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices, Journal of Scientific Computing, 57 (2013), pp. 477–501.
[3] F. R. K. Chung, Separator theorems and their applications, in Algorithms and Combinatorics 9, B. Korte, L. Lovász, and H. J. Prömel, eds., Springer, 1990.
[4] M. B. Cohen, R. Kyng, J. W. Pachocki, R. Peng, and A. Rao, Preconditioning in expectation, arXiv:1401.6236, (2014).
[5] T. A. Davis, Direct methods for sparse linear systems, vol. 2, Siam, 2006.
then the computational complexity of running (2) that start from an arbitrary vector \( x \) is of the form\( C \frac{m^k}{k!} \) such that \( C \) is a column of \( Q \).

Given \( x \) such that \( Qy \), With the same complexity, we can compute a \( y_N \) such that \( x_N = x_0 + Qy_N \), where \( x_N \) denotes the vector resulting from the \( N \) first iterations.

Proof. Let us first comment on the general strategy for computing fast iterations. Given \( x_t \in \mathbb{R}^m \) and a column \( q_j \) of \( Q \), recall that the next iteration we aim to compute is of the form

\[
x_{t+1} = x_t - \frac{x_t^T q_j}{q_j^T q_j} q_j.
\]

In order to get a running time for each iteration not depending on the system size \( m \), we make use of two generating sets of \( \mathbb{R}^m \). The sets are given by the columns of \( C \), as well as the columns of \( CU^{-T} \), where \( U \) is the \( p \times p \) upper triangular matrix such that \( C^T C = U^T + U \). Each column of \( Q \) has a decomposition in terms of these

---

**References**

[1] I. Koutis, G. L. Miller, and R. Peng, Approaching optimality for solving SDD linear systems, in Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on, IEEE, 2013, pp. 911–920.

[2] I. Koutis, G. L. Miller, and R. Peng, A nearly-m log n time solver for sdd linear systems, in Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on, IEEE, 2013, pp. 147–156.

[3] Y. Saad, Iterative methods for sparse linear systems, Siam, 2003.

[4] D. A. Spielman and S.-H. Teng, Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems, in Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, ACM, 2004, pp. 81–90.

[5] T. Strohmer and R. Vershynin, A randomized Kaczmarz algorithm with exponential convergence, Journal of Fourier Analysis and Applications, 15 (2009), pp. 262–278.

[6] R. Vandebril, M. Van Barel, and N. Mastronardi, Matrix computations and semiseparable matrices: linear systems, Johns Hopkins University Press, 2008.

[7] D. M. Young, Iterative solution of large linear systems, Elsevier, 2014.

---

**Appendix.**

**Proof of Theorem 3.**

**Theorem 18 (Theorem 3).** Let \( Q \in \mathbb{R}^{m \times n}, C \in \mathbb{R}^{m \times p} \) and \( D \in \mathbb{R}^{p \times n} \) be matrices such that \( Q = CD \) is a \( k \)-sparse factorization of \( Q \), and consider iterations of the form (2) that start from an arbitrary vector \( x_0 \in \mathbb{R}^m \), If every \( q_i \) in (2) is a column of \( Q \), then the computational complexity of running \( N \) iterations of (2) is:

\[
\mathcal{O}(Nk + pk + (m+n)k^2).
\]

With the same complexity, we can compute a \( y_N \) such that \( x_N = x_0 + Qy_N \), where \( x_N \) denotes the vector resulting from the \( N \) first iterations.

Proof. Let us first comment on the general strategy for computing fast iterations. Given \( x_t \in \mathbb{R}^m \) and a column \( q_j \) of \( Q \), recall that the next iteration we aim to compute is of the form (14)

\[
x_{t+1} = x_t - \frac{x_t^T q_j}{q_j^T q_j} q_j.
\]

In order to get a running time for each iteration not depending on the system size \( m \), we make use of two generating sets of \( \mathbb{R}^m \). The sets are given by the columns of \( C \), as well as the columns of \( CU^{-T} \), where \( U \) is the \( p \times p \) upper triangular matrix such that \( C^T C = U^T + U \). Each column of \( Q \) has a decomposition in terms of these
generating sets with a sparsity governed by \( k \). Using these sets we can thus express \( x_t \), with the coefficient vectors \( h_t, g_t \), defined via the relationships \( x_t = Ch_t \) and \( x_t = C^T e \). Note that such a vector \( g_t \) is given by \( g_t = U^T h_t \). Moreover we define the vector \( e_j = U^T d_j \). Now at each iteration, we only use the vectors \( h_t, g_t, d_j \) and \( e_j \), and do not need to store the full vector \( x_t \). In particular the inner-product can be computed as:

\[
x^T_t q_j = h^T_t (C^T C) d_j = h^T_t (U + U^T ) d_j = (U^T h_t)^T d_j + h^T_t (U^T d_j) = g_t^T d_j + h^T_t e_j.
\]

This can be done in \( O(k) \) time, as we will show in the following.

In order to establish this key result about the complexity of the inner product, which leads directly to an efficient algorithm for performing our iterative updates, we will proof the following facts.

**Fact 1** We can compute the matrix \( U \) in \( O(mk^2) \) (which is also the cost of computing \( C^T C \)).

**Fact 2** We can compute a vector \( h_0 \in \mathbb{R}^p \) such that \( x_0 = Ch_0 \) in time \( O(m) \).

**Fact 3** We can compute \( g_0 := U^T h_0 \) in time \( O(pk) \).

**Fact 4** The matrix \( U^T D \) can be computed in time \( O(nk^2) \).

**Fact 5** All the scalar products \( q_i^T q_i \), where \( q_i \) is a column of \( Q \) are computable in time \( O(nk) \).

**Proof of Fact 1** The cost of computing \( C^T C \) can be estimated by the number of scalar additions and multiplications involved. In fact the number of additions is the same as the number of multiplications, so we need only track the number of scalar multiplications. From the elementary properties of the \( k \)-sparse factorization, it follows that there are at most \( k \) entries per row. In the course of computing the entries of \( C^T C \) all the scalar products between the \( p \) columns of \( C \) will be computed. Thus we find that every entry of the first row of \( C \) will be multiplied with every of the \( k \) (or less) entries of first row, which gives \( O(k^2/2) \) scalar multiplications associated to the entries of the first row. Since every row can be treated similarly, the cost of computing \( C^T C \) is at most \( O(mk^2) \).

**Proof of Fact 2** We can assume without loss of generality that the columns of \( C \) contain the canonical basis of \( \mathbb{R}^m \). To see this, one can set \( \tilde{C} := (I_m \ C) \in \mathbb{R}^{m \times (p+m)} \) and \( \tilde{D} := (0 \ D^T)^T \in \mathbb{R}^{(p+m) \times n} \). It then follows that for each column \( \tilde{c}_i \) of \( \tilde{C} \), \( |F \tilde{O}(\tilde{c}_i)| \leq k + 1 \), that \( \tilde{D} \) is \((k + 1)\)-column sparse and that for each column \( \tilde{d}_j \), \( |\cup_{\ell \in \text{supp}(\tilde{d}_j)} F \tilde{O}(\tilde{c}_i)| \leq k + 1 \). Consequently, even though \( \tilde{D} \) has some zero rows, the factorization \( \tilde{C} \tilde{D} \) has all the properties of a \((k + 1)\)-sparse factorization and we say that \( \tilde{C} \tilde{D} \) is \((k + 1)\)-sparse. As a consequence, the running time does asymptotically not depend on the choice of the decomposition \( CD \) or \( \tilde{C} \tilde{D} \). Hence, we can assume without loss of generality that a vector \( h_0 \in \mathbb{R}^p \) such that \( x_0 = Ch_0 \) can be computed in \( O(m) \) time.

**Proof of Fact 3** Denote by \( U \) the \( p \times p \) upper triangular matrix such that \( C^T C = U^T + U \). Notice that the \( j \)th row of \( U \) is \( |F \tilde{O}(\tilde{c}_i)| \)-sparse. Since \( |F \tilde{O}(\tilde{c}_i)| \leq k \), this implies that the matrix \( U \) is \( k \)-row sparse. Moreover, as each column of \( C \) is assumed to be nonzero, we can deduce that \( U \) is invertible. Hence, given \( h_0 \), since \( U^T \) is \( k \)-column sparse, we compute the vector \( g_0 := U^T h_0 \) in time \( O(pk) \).

**Proof of Fact 4** Let \( d_j \) be a column of \( D \), which is \( k \)-sparse. Then, since \( Q = CD \) is a \( k \)-sparse factorization, the vector \( e_j := U^T d_j \) is \( k \)-sparse and is computed in time \( O(k^2) \). Consequently, we can compute the matrix product \( U^T D \), i.e., all vectors \( e_i \) in
Since $e_i$ and $d_i$ are $k$-sparse, it takes $O(k)$ time to compute $q_i^T q_i$, and thus $O(nk)$ to compute all the products.

**Fast iterative algorithm** Following the analogous reasoning as in the proof of fact 5, we see that

$$x_i^T q_j = g_t^T d_j + h_t^T e_j$$

is also computable in $O(k)$ time. Hence, we can compute a first iteration of (14) efficiently.

In order to make this computational gain available at every iteration, we have to find a way to update $h_t$ and $g_t$ in a fast manner, too. Given $h_t, g_t \in \mathbb{R}^p$ such that $x_t = Ch_t$ and $g_t = U^T h_t$ and given $e_j = U^T d_j$, the vectors

$$h_{t+1} := h_t - \frac{x_t^T q_j}{q_j^T q_j} d_j$$

$$g_{t+1} = g_t - \frac{x_t^T q_j}{q_j^T q_j} e_j$$

are such that $x_{t+1} = Ch_{t+1}$ and $g_{t+1} = U^T h_{t+1}$. Moreover, from fact 2 and 3, and the sparsity of $d_j$, it follows that the vectors $h_{t+1}$ and $g_{t+1}$ are computed in time $O(k)$.

Consequently, at each iteration, we only need the vectors $h_t, g_t, d_j$ and $e_j$ in order to compute $h_{t+1}$ and $g_{t+1}$. Note that both $h_{t+1}$ and $g_{t+1}$ are required to compute the scalar product $x_{t+1}^T q_j$ (needed in the next iteration) in time $O(k)$. Finally, the approximate solution after $N$ steps, $x_N$, is computed from the relation $x_N = Ch_N$. This can be done in $O(nk)$ time due to the sparsity of $C$.

Combining these results, it follows that $N$ iterative updates can be performed in time

$$O(Nk + pk + (m + n)k^2).$$

Finally, the computation of $y_N$ such that $x_N = x_0 + Q y_N$ can be performed while computing $x_N$ with the above described method without additional costs. Indeed, start with $y_0 = 0$. If the $(i+1)^{th}$ iteration is

$$x_{i+1} = x_i - \frac{x_i^T q_j}{q_j^T q_j} q_j,$$

then $y_{i+1}$ corresponds to updating the $j^{th}$ entry of $y_i$ by adding $\frac{-x_i^T q_j}{q_j^T q_j}$. As the required scalar products are computed for $x_{i+1}$, no additional cost is incurred.

**Semiseparable and hierarchical matrices.**

**Theorem 19** (Theorem 10). An $n \times n$ matrix that is $(p,q)$-semiseparable is an $\mathcal{H}_r(\mathcal{P})$-matrix where $r = \max\{p,q\}$ and $\mathcal{P}$ is a binary dendrogram.

**Proof.** Following the definition, we have $1 \leq p, q \leq n$ and we assume without loss of generality that $n \geq 2$. Now, let $E$ be an $n \times n$ matrix which is $(p,q)$-semiseparable
and let \( \{I_1, I_2\} \) be a partition of \( I = \{1, ..., n\} \) with \( I_1 = \{1, ..., \lfloor \frac{n}{2} \rfloor \} \) and \( I_2 = I \setminus I_1 \). Consider an integer \( i_1 \in I \) such that \( \lfloor \frac{n}{2} \rfloor - q + 1 \leq i_1 \leq \min(\lfloor \frac{n}{2} \rfloor, n - q + 1) \). Then, the submatrix \( E(1 : i_1 + q - 1, i_1 : n) \) is of rank \( \ell_1 \leq q \) and contains \( E_{I_1 \times I_2} \).

Similarly, if \( i_2 \in I \) such that \( \lfloor \frac{n}{2} \rfloor - p + 1 \leq i_2 \leq \min(\lfloor \frac{n}{2} \rfloor, n - p + 1) \), then the submatrix \( E(i_2 : n, 1 : i_2 + p - 1) \) is of rank \( \ell_2 \leq p \) and contains \( E_{I_2 \times I_1} \). Therefore, we have shown that the off-diagonal blocks \( E_{I_1 \times I_2} \) and \( E_{I_2 \times I_1} \) are of a rank smaller of equal to \( r \).

From the definition of semiseparable matrix, it follows that the diagonal blocks \( E_{I_1 \times I_1} \) and \( E_{I_2 \times I_2} \) of \( E \) are also \((p, q)\)-semiseparable matrices. Repeating the previous argument recursively on \( E_{I_1 \times I_1} \) and \( E_{I_2 \times I_2} \) shows that \( E \) is an \( H_r(P) \)-matrix with \( P \) being a binary dendrogram, i.e \( d = 2 \).

Convergence rate and number of needed iterations when the search directions are randomly selected

The proof is due to Strohmer and Vershynin [17] and has originally been given in the context of Kaczmarz’s method for solving linear systems. The version we give here is adapted to the context of this paper.

We want to establish the speed of convergence of iterations (2), when each column \( q_i \) of the matrix \( Q \) is chosen with probability proportional to \( \|q_i\|^2 \).

For any \( x \) we first consider the auxiliary quantity \( \sum_i (x, q_i)^2 = x^T QQ^T x \geq \sigma_{\min}^2(Q) \|x\|^2 \). Here \( (x, q_i) \) denotes the usual scalar product \( x^T q_i \). If each direction \( q_i \) is selected with probability \( p_i = \|q_i\|^2 / \sum_j \|q_j\|^2 = \langle q_i, q_i \rangle / \|Q\|^2_{\text{Frob}} \), we can rewrite this inequality as \( \sum_i p_i (x, q_i)^2 / \langle q_i, q_i \rangle \geq \sigma_{\min}^2(Q) \|x\|^2 / \|Q\|^2_{\text{Frob}} \). In fact as \( x^* \), the minimum norm point in the affine space \( x_0 + \text{span}\{q_i\} \), must be orthogonal to all directions \( q_i \) in the space, we know that \( (x, q_i) = \langle x - x^*, q_i \rangle \). Thus we can write:

\[
\sum_i p_i (x, q_i)^2 / \langle q_i, q_i \rangle \geq \frac{\sigma_{\min}^2(Q)}{\|Q\|^2_{\text{Frob}}} \|x - x^*\|^2.
\]

In addition, we have that \( \|x_t - x^*\|^2 = \|(x_{t+1} - x^*) - (x_{t+1} - x_t)\|^2 = \|x_{t+1} - x^*\|^2 + \|x_{t+1} - x_t\|^2 \). The second equality is due to orthogonality \( \langle x_{t+1} - x^*, x_{t+1} - x_t \rangle = 0 = \langle x_{t+1} - x^*, q_i \rangle \): the new error \( x_{t+1} - x^* \) is orthogonal to the search direction. This can be checked from the two following observations. First, \( x^* \), being the minimum norm point of the affine space \( x_0 + \text{span}\{q_i\} \), must be orthogonal to all directions \( q_i \) in the affine space. Second, \( x_{t+1} \) is also computed as the minimum norm point on the line \( x_t + \alpha q_i \), and is therefore also orthogonal to the current search direction \( q_i \). Thus the error \( x_{t+1} - x^* \) is also orthogonal to search direction \( q_i \).

Finally we combine the results and observe that the expected value of the error norm

\begin{align}
\sum_i p_i \|x_{t+1} - x^*\|^2 & = \sum_i p_i \|x_t - x^*\|^2 - \sum_i p_i \|x_{t+1} - x_t\|^2 \\
\quad & = \|x_t - x^*\|^2 - \sum_i p_i \frac{(x_t, q_i)^2}{\langle q_i, q_i \rangle} \|q_i\|^2 \leq (1 - \frac{\sigma_{\min}^2(Q)}{\|Q\|^2_{\text{Frob}}}) \|x_t - x^*\|^2,
\end{align}

which is the desired result.