

Identifying Kronecker product factorizations

Yannis Voet ^{*1} and Leonardo De Novellis ^{†1}

¹MNS, Institute of Mathematics, École polytechnique fédérale de Lausanne, Station 8, CH-1015 Lausanne, Switzerland

October 30, 2025

Abstract

The Kronecker product is an invaluable tool for data-sparse representations of large networks and matrices with countless applications in machine learning, graph theory and numerical linear algebra. In some instances, the sparsity pattern of large matrices may already hide a Kronecker product. Similarly, a large network, represented by its adjacency matrix, may sometimes be factorized as a Kronecker product of smaller adjacency matrices. In this article, we determine all possible Kronecker factorizations of a binary matrix and visualize them through its decomposition graph. Such sparsity-informed factorizations may later enable good (approximate) Kronecker factorizations of real matrices or reveal the latent structure of a network. The latter also suggests a natural visualization of Kronecker graphs.

Keywords: Binary matrix, Kronecker product, Kronecker graph.

2020 MSC: 15A23, 15B34, 65F50.

1 Introduction

This work focuses on finding all possible Kronecker product factorizations of a binary (or Boolean) matrix A . That is, given $A \in \{0, 1\}^{n \times n}$, we seek all possible ways of writing

$$A = \bigotimes_{i=1}^{\ell} A_i \tag{1.1}$$

for $\ell > 1$ factor matrices $A_i \in \{0, 1\}^{n_i \times n_i}$, $n_i > 1$, whose sizes satisfy $\prod_{i=1}^{\ell} n_i = n$.

More than 25 years ago, Van Loan [1] rightly predicted that the Kronecker product would emerge as an increasingly useful tool in scientific computing. The problem stated in (1.1) is one of many instances of Kronecker products and is at the forefront of several important applications. For instance, if A is the adjacency matrix of an unweighted graph G representing a network, decomposing it as a Kronecker product effectively represents the graph as a Kronecker product of smaller graphs G_i . Factorizing a large graph into smaller graphs allows compressing the network and often helps both analyzing and visualizing it [2, 3, 4]. For those reasons, graph products, and in particular the Kronecker product (sometimes also called direct, cardinal, categorical or tensor product) have been well studied by the graph theory community [5, 6]. A graph G is a Kronecker product if there exists a graph G' isomorphic to G such that its adjacency matrix is a Kronecker product [7]. In other words, there exists a permutation matrix P such that $P^T A P$ is a Kronecker product [8]. This definition implies that identifying Kronecker product graphs is tightly intertwined with graph isomorphism problems, which are notoriously difficult. Additionally, computational tools for identifying Kronecker products graphs are still rather primitive. Imrich et al. [5] mostly studied theoretical questions of existence and uniqueness of graph factorizations (also for other products) and proposed a polynomial time algorithm for factorizing connected non-bipartite graphs. Recognizing the limitations of Imrich's algorithm, Calderoni et al. [9] recently proposed a heuristic for factorizing general graphs.

Apart from the theoretical and computational challenges of graph factorizations, Kronecker product graphs play an important role in modeling large networks. In the machine learning community, Leskovec et al. [2, 3] realized that Kronecker product graphs successfully mimic several key properties of real networks and proposed a Kronecker graph model by taking successive products with an initiator graph. In this context, the problem consists in fitting a real network to a Kronecker graph instead of exactly factorizing it.

The problem addressed in this contribution is related albeit different from the graph factorization problem. Here we assume that the binary matrix A is given and directly attempt to factorize it. On the one hand, this eliminates

^{*}yannis.voet@epfl.ch

[†]leonardo.denovellis@epfl.ch

the hurdle of finding a permutation of A such that it becomes factorizable. On the other hand, we must still cope with potential problems of non-uniqueness. A prominent application of our research lies in (approximately) factorizing large sparse matrices. Given *fixed* sizes n_1 and n_2 such that $n = n_1 n_2$, Van Loan and Pitsianis [10] presented in the early 1990s an algorithm for computing the best Kronecker product factorization of a general matrix $A \in \mathbb{R}^{n \times n}$ in the Frobenius norm. In other words, they find the best factor matrices $B_i \in \mathbb{R}^{n_i \times n_i}$ such that

$$\|B - B_1 \otimes B_2\|_F \quad (1.2)$$

is minimized. Their algorithm has since then been generalized to Kronecker products of an arbitrary number of matrices B_i for $i = 1, \dots, \ell$ [11], although the resulting factorization may no longer be optimal. Unfortunately, neither Van Loan’s original algorithm nor subsequent generalizations are easy to apply in a “black-box” fashion. Indeed, some of the key parameters are the sizes n_i of the factor matrices A_i and their number ℓ . Even for a fixed factorization length ℓ , there generally exist several tuples of sizes $\mathbf{n} = (n_1, n_2, \dots, n_\ell)$ satisfying $n_1 n_2 \dots n_\ell = n$ and without any background information on the problem’s origin, choosing the “right” length ℓ and the “right” sizes $\mathbf{n}$ is not obvious at all. By “right” we mean the factorization that minimizes some objective function or error measure such as the Frobenius norm in (1.2). Clearly, Van Loan’s algorithm (and subsequent generalizations) will deliver an approximate factorization for arbitrary choices of length ℓ and compatible sizes $\mathbf{n}$. However, it may produce a very poor “approximation” unless ℓ and $\mathbf{n}$ are suitably chosen. Since such approximations are oftentimes used as preconditioners within iterative solvers [10, 11, 12, 13], it could have rippling effects down the computational pipeline. Instead, depending on the origin of the matrix, there often exist natural choices for ℓ and $\mathbf{n}$ that will deliver remarkably good factorizations. In some instances, those choices are naturally encoded in the sparsity pattern of the matrix. For example, matrices arising from tensorized finite element discretizations of partial differential equations (PDEs) often hide a Kronecker product in their sparsity pattern, although the matrix itself is rarely exactly factorizable [14, 15]. Thus, our work will not attempt to factorize the matrices themselves but only their sparsity pattern encoded by a binary matrix. Clearly, if a real matrix B already happens to be a Kronecker product, then so is its sparsity pattern A . Although the converse does not hold, the structure encoded in the sparsity pattern may still suggest suitable sizes for approximately factorizing B . We will therefore assume that the matrices one attempts to (approximately) factorize are sparse. We note that the specific problem of factorizing permutations has already been addressed in [16], where an algorithm based on group theory arguments is also presented. Conversely, for completely dense matrices, the sparsity pattern does not help and one may need to compute all factorizations and retain the best one, as was suggested in [17] for applications in image analysis.

In this article, we present a theory and algorithm for computing all possible factorizations of the form (1.1). In particular, the algorithm finds suitable lengths ℓ and sizes $\mathbf{n}$ that may be directly supplied as input parameters to Van Loan’s algorithm. In general, one is interested in computing the factorization of greatest length since it yields the greatest compression.

The outline for the rest of the article is as follows: after setting up the problem in Section 2 and recalling some basic properties of the Kronecker product, we propose in Section 3 a fast algorithm for determining whether a sparse binary matrix A is factorizable for *fixed* sizes (n_1, n_2) . Remarkably, as shown later in the same section, factorizations of length $\ell = 2$ completely describe the structure of A and enable finding factorizations of length $\ell > 2$. Subsequently, we present in Section 4 the decomposition graph, a simple and elegant way of visualizing all possible Kronecker factorizations of A . Section 5 is then devoted to applications. The first one pertains to space-time discretizations of PDEs, whose coefficient matrix is approximated by a length 4 Kronecker product. The second one exploits the Kronecker structure of the adjacency matrix of a network for visualizing Kronecker graphs. Lastly, the third one identifies a Kronecker product structure within a subsequence of unitary operations encoding a quantum gate. Finally, conclusions are stated in Section 6.

2 Problem description

In this section, we introduce some first definitions and preliminary results to set up the problem. Throughout this article, we assume that $A \in \mathbb{B}^{n \times n}$ is a large sparse binary matrix, where $\mathbb{B} = \{0, 1\}$. This matrix may for instance define the adjacency matrix of a large network or encode the sparsity pattern of a matrix with real or complex entries. The sparsity pattern of a matrix $X \in \mathbb{R}^{n \times n}$, denoted $\text{sp}(X)$, simply keeps track of the position of nonzero entries:

$$\text{sp}(X) = \{(i, j) : x_{ij} \neq 0, 1 \leq i, j \leq n\}.$$

The binary matrix A is then defined as

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \text{sp}(X), \\ 0 & \text{if } (i, j) \notin \text{sp}(X). \end{cases}$$

When handling binary variables, the addition “+” and multiplication “.” refer to the standard Boolean addition and multiplication, whose truth table is recalled in Tables 2.1a and 2.1b, respectively.

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

(a) Boolean addition

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

(b) Boolean multiplication

Table 2.1: Boolean operations

Since $\text{sp}(X+Y) \subseteq \text{sp}(X) \cup \text{sp}(Y)$, for two matrices $X, Y \in \mathbb{R}^{n \times n}$, the Boolean addition is a natural choice in this context and simply describes the worst case fill-in when summing two sparse matrices. Also note that the Boolean multiplication reduces to the standard multiplication of real or complex numbers. However, the operation that is really at the center of our attention is the *Kronecker product*. Given two matrices $X \in \mathbb{R}^{n_1 \times n_1}$ and $Y \in \mathbb{R}^{n_2 \times n_2}$, their Kronecker product $X \otimes Y \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$ is defined as the block matrix

$$X \otimes Y = \begin{pmatrix} x_{11}Y & x_{12}Y & \cdots & x_{1n_1}Y \\ x_{21}Y & x_{22}Y & \cdots & x_{2n_1}Y \\ \vdots & \vdots & \ddots & \vdots \\ x_{n_1 1}Y & x_{n_1 2}Y & \cdots & x_{n_1 n_1}Y \end{pmatrix}.$$

The Kronecker product satisfies some basic properties of associativity and distributivity, whose proof is commonly found in standard textbooks (see e.g. [18]). Given matrices X, Y, Z of conforming size,

$$\begin{aligned} (X \otimes Y) \otimes Z &= X \otimes (Y \otimes Z) && \text{(associativity),} \\ X \otimes (Y + Z) &= X \otimes Y + X \otimes Z && \text{(distributivity I),} \\ (Y + Z) \otimes X &= Y \otimes X + Z \otimes X && \text{(distributivity II).} \end{aligned}$$

Obviously, the same definition and properties hold for binary matrices, provided one substitutes the standard addition and multiplication with their Boolean counterpart. Among the binary matrices encountered in applications, decomposable ones are of great interest.

Definition 2.1 (Decomposable matrix). A matrix $A \in \mathbb{B}^{n \times n}$ is called *decomposable* (with respect to the Kronecker product) if there exists an integer $\ell > 1$ and factor matrices $A_i \in \mathbb{B}^{n_i \times n_i}$ with $n_i > 1$ such that

$$A = \bigotimes_{i=1}^{\ell} A_i. \quad (2.1)$$

The integer ℓ is called the *length* of the factorization.

Definition 2.2 (Prime matrix). A matrix $A \in \mathbb{B}^{n \times n}$ is called *prime* if it cannot be decomposed.

In particular, note that A is necessarily prime if n is prime. Among all possible decompositions of A , so-called *prime decompositions* are particularly important in the forthcoming discussion.

Definition 2.3 (Prime decomposition). A decomposition (2.1) is called *prime* if all its factor matrices are prime.

In other words, prime decompositions cannot be further expanded into a factorization of greater length. A decomposition (or factorization) of A is commonly identified with the tuple $\mathbf{n} = (n_1, n_2, \dots, n_\ell)$ specifying the sizes of the factor matrices in the decomposition. The size of the matrix A is then read from the product $n = \Pi(\mathbf{n}) := \prod_{i=1}^{\ell} n_i$. The next lemma shows that this identification does not cause any ambiguity in all practically relevant cases.

Lemma 2.4. For fixed sizes $(n_1, n_2, \dots, n_\ell)$, the factorization of $A \in \mathbb{B}^{n \times n} \setminus \{0\}$, if it exists, is unique.

Proof. Assume that A admits two $(n_1, n_2, \dots, n_\ell)$ factorizations with factor matrices $\{A_i\}_{i=1}^{\ell}$ and $\{\hat{A}_i\}_{i=1}^{\ell}$. Then,

$$A = A_1 \otimes A_2 \otimes \cdots \otimes A_{\ell-1} \otimes A_\ell = \hat{A}_1 \otimes \hat{A}_2 \otimes \cdots \otimes \hat{A}_{\ell-1} \otimes \hat{A}_\ell.$$

From the associativity of the Kronecker product, the above relation may be rewritten

$$X \otimes A_\ell = \hat{X} \otimes \hat{A}_\ell$$

where $X = A_1 \otimes A_2 \otimes \cdots \otimes A_{\ell-1}$ and $\hat{X} = \hat{A}_1 \otimes \hat{A}_2 \otimes \cdots \otimes \hat{A}_{\ell-1}$. Thus, A is an $n_1 n_2 \dots n_{\ell-1} \times n_1 n_2 \dots n_{\ell-1}$ block matrix with blocks of size $n_\ell \times n_\ell$. Its (i, j) th block $A_{i,j}$ is given by

$$A_{i,j} = x_{ij} A_\ell = \hat{x}_{ij} \hat{A}_\ell \quad i, j = 1, \dots, n_1 n_2 \dots n_{\ell-1}.$$

Recalling that X, A_ℓ and $\hat{X}, \hat{A}_\ell$ are binary matrices and none of them can be zero (since $A \neq 0$), two cases must be distinguished:

- If $A_{i,j} \neq 0$, then $x_{ij} = \hat{x}_{ij} = 1$ and $A_\ell = \hat{A}_\ell$.
- If $A_{i,j} = 0$, then $x_{ij} = \hat{x}_{ij} = 0$.

In either case $x_{ij} = \hat{x}_{ij}$ and consequently, $X = \hat{X}$. Moreover, since $A \neq 0$, the first case is necessarily encountered at least once and yields $A_\ell = \hat{A}_\ell$. Recursively applying the same arguments on $X = \hat{X}$ proves that $A_i = \hat{A}_i$ for all $i = 1, \dots, \ell$. $\square$

Note that Definition 2.4 does not hold for real valued matrices due to the scaling indeterminacy. Indeed, for $X \in \mathbb{R}^{n_1 \times n_1}$ and $Y \in \mathbb{R}^{n_2 \times n_2}$, $X \otimes Y = (\alpha X) \otimes (\alpha^{-1} Y)$ for any $\alpha \neq 0$. For binary matrices, Definition 2.4 eliminates this problem and allows identifying a factorization with the size of its factors. Nevertheless, as shown in the next couple of examples, a given matrix $A \in \mathbb{B}^{n \times n}$ may still have multiple distinct factorizations with factor matrices of different size.

Example 2.5. From the following string of identities

$$\begin{aligned} A &= \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \end{aligned}$$

it appears that A admits $(2, 2, 3)$, $(2, 3, 2)$ and $(3, 2, 2)$ factorizations. Moreover, those factorizations are all prime since both 2 and 3 are prime.

Interestingly, decomposable matrices may have prime factorizations of different length.

Example 2.6. From the equalities

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \end{aligned}$$

we conclude that A admits $(3, 4)$ and $(2, 2, 3)$ factorizations. Note that the first factorization is prime since the 4×4 matrix cannot be further decomposed into a Kronecker product of 2×2 matrices and the second factorization is evidently prime since 2 and 3 are both prime.

Among all possible factorizations of A , we are particularly interested in those of greatest length. Such factorizations are relevant in many applications since they generally provide the greatest compression. Indeed, storing the factor matrices $\{A_i\}_{i=1}^\ell$ only requires a fraction of the memory for A . The same holds true when subsequently (approximately) factorizing a real matrix B having the same sparsity pattern as A . In the next section, we explain how to find all length 2 factorizations of a binary matrix A and how to eventually combine them to produce factorizations of greater length.

3 Decomposable matrices

As explained later in the article, factorizations of length $\ell = 2$ allow finding factorizations of length $\ell > 2$. Thanks to this remarkable property, we may exclusively focus on the former, which drastically simplifies the problem. In this case, if A admits an (n_1, n_2) factorization, then n_1 and n_2 are divisors of n . Denoting $[n] \subset \mathbb{N}$ the set of positive divisors of n , a pair of integers (n_1, n_2) with $n_1, n_2 \in [n] \setminus \{1, n\}$ and $n_1 n_2 = n$ is called a *compatible pair* and we denote $\mathcal{C}$ the set of all compatible pairs. Clearly, there are as many compatible pairs as there are non-trivial divisors of n and this number is related to the prime decomposition of n . Recall that for any natural integer $n \in \mathbb{N}$, there exist prime numbers $p_1, \dots, p_m$ and integer exponents $k_1, \dots, k_m$ such that

$$n = \prod_{j=1}^m p_j^{k_j}.$$

The number of divisors of n is therefore $\prod_{j=1}^m (k_j + 1)$. Excluding the trivial divisors 1 and n , the cardinality of $\mathcal{C}$ is

$$|\mathcal{C}| = \prod_{j=1}^m (k_j + 1) - 2.$$

The set $\mathcal{C}$ contains all pairs of candidate sizes for attempting a factorization. Decomposable matrices are factorizable for at least one compatible pair and clearly form a very special subset of binary matrices. As a matter of fact, the next lemma shows that most binary matrices one could form are in fact prime (even if n is not prime).

Lemma 3.1. Let $A \in \mathbb{B}^{n \times n}$. Then,

$$\mathbb{P}[A \text{ is factorizable}] \leq \sum_{(n_1, n_2) \in \mathcal{C}} \frac{2^{n_1 + n_2}}{2^{n^2}}.$$

Proof. Let $A \in \mathbb{B}^{n \times n}$ have i.i.d. $\text{Be}(1/2)$ random entries (i.e. independent and identically distributed random variables that take value 1 or 0, each with probability $1/2$). The number of binary matrices A is 2^{n^2} while the number of (n_1, n_2) factorizable matrices for a given compatible pair (n_1, n_2) is $2^{n_1 + n_2}$. Thus, the probability that A admits an (n_1, n_2) factorization is $2^{n_1 + n_2} / 2^{n^2}$. The probability that A is factorizable for any compatible pair is then obtained by bounding the probability of the union by the sum of probabilities. $\square$

This article would not make much sense if we were to consider random binary matrices. However, for specific applications, the sparsity pattern of binary matrices often hides a Kronecker product. Our algorithm for detecting it relies on a very special subset of matrices that are factorizable for any compatible pair.

Definition 3.2 (Maximal matrix). A decomposable matrix $A \in \mathbb{B}^{n \times n}$ is called *maximal* if it admits an (n_1, n_2) factorization for any compatible pair $(n_1, n_2) \in \mathcal{C}$.

So far, the only means of checking whether a matrix is maximal amounts to verifying that it is factorizable for any compatible pair $(n_1, n_2) \in \mathcal{C}$. Some simple examples of maximal matrices include the identity matrix and the matrix of all ones. One can also easily verify that the matrix A in Definition 2.5 is maximal. Other less obvious examples are the so-called *basis matrices*.

Definition 3.3 (Basis matrix). The *basis matrices* $\{E_{ij}\}_{i,j=1}^n \subset \mathbb{B}^{n \times n}$ are defined as

$$(E_{ij})_{kl} = \delta_{ik} \delta_{jl}$$

where the Kronecker delta symbol is defined as $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$.

In other words, the basis matrix E_{ij} has a single component equal to 1 in position (i, j) and 0 elsewhere. Over the real (or complex) field, the set of all basis matrices $\{E_{ij}\}_{i,j=1}^n$ forms the so-called canonical basis for the vector space of real (or complex) matrices of size n . However, they are brought up here for another reason: decomposable basis matrices are maximal. This result is the object of the next lemma whose constructive proof is central to our forthcoming analysis.

Lemma 3.4. All decomposable basis matrices E_{ij} are maximal for $i, j = 1, \dots, n$.

Proof. Firstly, since E_{ij} is decomposable, n cannot be prime and it has at least one non-trivial divisor. Let $n_2 \in [n] \setminus \{1, n\}$. Then, the Euclidean division of $i - 1$ and $j - 1$ by n_2 yields

$$\begin{aligned} i - 1 &= \tilde{i}_1 n_2 + \tilde{i}_2, \\ j - 1 &= \tilde{j}_1 n_2 + \tilde{j}_2, \end{aligned}$$

for integers $0 \leq \tilde{i}_1, \tilde{j}_1 < n_1 = d/n_2$ and $0 \leq \tilde{i}_2, \tilde{j}_2 < n_2$. Finally, setting $i_k = \tilde{i}_k + 1$ and $j_k = \tilde{j}_k + 1$ for $k = 1, 2$, we obtain

$$E_{ij} = E_{i_1 j_1} \otimes E_{i_2 j_2}$$

for basis matrices $E_{i_1 j_1} \in \mathbb{B}^{n_1 \times n_1}$ and $E_{i_2 j_2} \in \mathbb{B}^{n_2 \times n_2}$. Thus, E_{ij} admits an (n_1, n_2) factorization. Since this construction holds for any divisor $n_2 \in [n] \setminus \{1, n\}$, E_{ij} is maximal. $\square$

Definition 3.4 lays the foundation of our method and allows us to easily check whether *any* binary matrix $A \in \mathbb{B}^{n \times n}$ is factorizable. Some interesting ideas in that direction were drafted in [19] but the authors reached erroneous conclusions. The next few paragraphs present a complete and correct algorithm. Firstly, the matrix is expanded in terms of basis matrices as

$$A = \sum_{(i,j) \in \text{sp}(A)} E_{ij}. \quad (3.1)$$

Secondly, given a compatible pair (n_1, n_2) , each E_{ij} is factorized as $E_{ij} = E_{i_1 j_1} \otimes E_{i_2 j_2}$ with $1 \leq i_1, j_1 \leq n_1$ and $1 \leq i_2, j_2 \leq n_2$ as done in the proof of Definition 3.4. Note that the only information that matters in (3.1) are the integer pairs (i_1, j_1) and (i_2, j_2) , not the actual matrices. Moreover, for convenience, we map each index pair (i_1, j_1) and (i_2, j_2) to a linear index. For a matrix of size n , $l_n(i, j) = (j - 1)n + i$ is the linear index corresponding to (i, j) . For instance, the linear indices corresponding to the basis matrices $\{E_{11}, E_{21}, E_{12}, E_{22}\} \subset \mathbb{B}^{2 \times 2}$ are $\{1, 2, 3, 4\}$ (in the same order). Reverting back to index pairs and basis matrices is also straightforward from the Euclidean division. To lighten the notation, we drop the dependency on the sizes and simply denote

$$\begin{aligned} l_1 &= l_{n_1}(i_1, j_1) \\ l_2 &= l_{n_2}(i_2, j_2) \end{aligned}$$

the linear indices for the index pairs (i_1, j_1) and (i_2, j_2) , respectively. A superscript k is then appended to linear indices and index pairs for identifying the k th term in the sum (3.1). Finally, we define the set of pairs of linear indices

$$S = \{(l_1^k, l_2^k) : k = 1, \dots, \text{nnz}(A)\},$$

where $\text{nnz}(A)$ denotes the number of non-zero entries of A . Our algorithm then uses the following equivalence for determining whether A is factorizable.

Lemma 3.5. A matrix $A \in \mathbb{B}^{n \times n}$ admits an (n_1, n_2) factorization if and only if there exist subsets $S_1, S_2 \subset \mathbb{N}$ such that

$$S = S_1 \times S_2.$$

Proof. The matrix A is (n_1, n_2) factorizable if and only if

$$A = \left(\sum_{(i_1, j_1) \in I_1} E_{i_1 j_1} \right) \otimes \left(\sum_{(i_2, j_2) \in I_2} E_{i_2 j_2} \right) \quad (3.2)$$

for subsets

$$I_1 \subseteq \{(i_1, j_1) : 1 \leq i_1, j_1 \leq n_1\}, \quad I_2 \subseteq \{(i_2, j_2) : 1 \leq i_2, j_2 \leq n_2\}.$$

After mapping the index pairs within I_1 and I_2 to linear index sets S_1 and S_2 , respectively, the matrix relation (3.2) is then equivalent to the set relation $S = \{(l_1, l_2) : l_1 \in S_1, l_2 \in S_2\} = S_1 \times S_2$ on linear indices. $\square$

Thanks to Definition 3.5, our algorithm checks whether A is factorizable by attempting to write S as a Cartesian product. The latter merely requires ordering the elements of S and is efficiently coded up. Thus, we have substituted the matrix problem with a scalar one that only uses the position of nonzero entries rather than the matrix itself. Once S_1 and S_2 have been found, reverting back to index pairs yields I_1 and I_2 , the sparsity patterns of the factor matrices constituting the product. The next couple of examples illustrate our argument.

Example 3.6. We would like to determine whether the binary matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

is factorizable for the (only) compatible pair $(2, 2)$. By inspection, the answer is trivially negative but we will show it algebraically from the procedure described above. Firstly, the sparsity pattern of A is

$$\text{sp}(A) = \{(1, 1), (2, 2), (3, 3)\}.$$

Secondly, for the $(2, 2)$ compatible pair, we obtain the sets

$$\begin{aligned} \{(i_1^k, j_1^k)\}_{k=1}^3 &= \{(1, 1), (1, 1), (2, 2)\}, \\ \{(i_2^k, j_2^k)\}_{k=1}^3 &= \{(1, 1), (2, 2), (1, 1)\}, \end{aligned}$$

which are mapped to linear indices

$$S = \{(l_1^k, l_2^k)\}_{k=1}^3 = \{(1, 1), (1, 4), (4, 1)\}.$$

This set cannot be expressed as a Cartesian product and consequently A is not $(2, 2)$ factorizable. Consider now the slightly modified matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Its left and right index pairs for the $(2, 2)$ compatible pair are

$$\begin{aligned} \{(i_1^k, j_1^k)\}_{k=1}^3 &= \{(1, 1), (1, 1), (1, 1)\}, \\ \{(i_2^k, j_2^k)\}_{k=1}^3 &= \{(1, 1), (2, 1), (2, 2)\}, \end{aligned}$$

to which correspond the linear indices

$$S = \{(l_1^k, l_2^k)\}_{k=1}^3 = \{(1, 1), (1, 2), (1, 4)\}.$$

This set is evidently expressed as the Cartesian product $S = S_1 \times S_2$, where

$$S_1 = \{1\} \quad \text{and} \quad S_2 = \{1, 2, 4\}.$$

Therefore, A admits a $(2, 2)$ factorization $A = A_1 \otimes A_2$ and by reverting back to index pairs, we obtain

$$I_1 = \text{sp}(A_1) = \{(1, 1)\} \quad \text{and} \quad I_2 = \text{sp}(A_2) = \{(1, 1), (2, 1), (2, 2)\}.$$

These examples again highlight one of the key properties of our method: it is completely matrix-free and only requires the position of the non-zero entries of A . Apart from the Euclidean divisions needed for computing the index pairs, the factorizability of S as a Cartesian product is easily verified by sorting its elements.

Remark 3.7. The procedure described above is closely related to Van Loan's algorithm [10]. As a matter of fact, the index set S contains the position of the nonzero entries of $\mathcal{R}(A)$, the so-called rearrangement of A [10]. Hence, Definition 3.5 is equivalent to verifying that $\mathcal{R}(A)$ is rank 1. Whereas Van Loan's algorithm explicitly forms $\mathcal{R}(A)$, our method only maps the position of the nonzero entries from A to $\mathcal{R}(A)$, which is all that is needed for verifying its factorizability. In [9], the authors have instead directly used Van Loan's algorithm for checking the factorizability of A .

Our method easily allows checking whether a matrix is factorizable for a given compatible pair. However, in order to completely describe the Kronecker structure of A , *all* possible length 2 factorizations are needed, which we ensure by iterating over all compatible pairs. Although this may seem prohibitive, the number of compatible pairs is bounded by $\sqrt{n}$, yielding complexity estimates. Running the algorithm over all compatible pairs eventually yields the set of all length 2 factorizations. For future reference, we denote this set $\mathcal{F}$ such that

$$\mathcal{F} = \{(n_1, n_2) : A \text{ is } (n_1, n_2) \text{ factorizable}\} \subseteq \mathcal{C}.$$

Note that the order within a pair is important: an (n_1, n_2) factorization may exist independently of an (n_2, n_1) factorization. The next lemma is a cornerstone of our method and explains how to combine length 2 factorizations to produce factorizations of greater length.

Lemma 3.8. For $A \in \mathbb{B}^{n \times n}$ and integers $l, p, r > 1$, the existence of the following factorizations is equivalent

$$\exists \begin{cases} (l, pr) \\ (pl, r) \end{cases} \iff \exists (l, p, r).$$

Proof. For the forward implication, assuming there exists two factorizations (l, pr) and (pl, r) , then

$$A = A_1 \otimes A_2, \tag{3.3}$$

$$A = \hat{A}_1 \otimes \hat{A}_2. \tag{3.4}$$

If $A = 0$, the statement trivially holds. If $A \neq 0$, we deduce from (3.3) that A is a $l \times l$ block matrix with blocks of size $pr \times pr$ that are either zero or copies of A_2 . But since pr is a multiple of r , the (blockwise) equality of (3.3) and (3.4) ensures that there exists a matrix $P \in \mathbb{B}^{p \times p}$ such that $A_2 = P \otimes \hat{A}_2$ and consequently, from (3.3),

$$A = A_1 \otimes A_2 = A_1 \otimes P \otimes \hat{A}_2$$

and there exists an (l, p, r) factorization.

The backward implication immediately follows from the associativity of the Kronecker product: assuming there exists an (l, p, r) factorization, then

$$A = A_1 \otimes A_2 \otimes A_3 = A_1 \otimes (A_2 \otimes A_3) = (A_1 \otimes A_2) \otimes A_3$$

and there exist (l, pr) and (pl, r) factorizations. $\square$

Remark 3.9. The statement of Definition 3.8 also holds for real or complex matrices with essentially the same proof arguments.

In short, Definition 3.8 shows how to combine two length 2 factorizations to produce a single length 3 factorization. More generally, we may combine two factorizations of length $q+1$ and $s+1$ to produce a single factorization of length $q+s+1$. The result is stated in the next corollary, where for a vector $\mathbf{n} = (n_1, \dots, n_d) \in \mathbb{N}^d$, $\Pi(\mathbf{n}) = \prod_{i=1}^d n_i$.

Corollary 3.10. For an integer $p > 1$ and integer vectors $\mathbf{l} = (l_1, \dots, l_q) \in \mathbb{N}^q$, $\mathbf{r} = (r_1, \dots, r_s) \in \mathbb{N}^s$ all greater than 1 componentwise, the existence of the following factorizations is equivalent

$$\exists \begin{cases} (l, p\Pi(\mathbf{r})) \\ (p\Pi(\mathbf{l}), \mathbf{r}) \end{cases} \iff \exists (l, p, \mathbf{r}).$$

Proof. If A admits two factorizations $(l, p\Pi(\mathbf{r}))$ and $(p\Pi(\mathbf{l}), \mathbf{r})$, then

$$A = \left(\bigotimes_{i=1}^q L_i \right) \otimes R = L \otimes \left(\bigotimes_{i=1}^s R_i \right)$$

where the factor matrices L_i and R_i have size l_i and r_i , respectively. The proof then follows exactly the same lines as in Definition 3.8 with $A_1 = \bigotimes_{i=1}^q L_i$, $A_2 = R$, $\hat{A}_1 = L$ and $\hat{A}_2 = \bigotimes_{i=1}^s R_i$. $\square$

The next corollary is the workhorse of our method and allows to combine m factorizations of length 2 to produce a single factorization of length $m+1$.

Corollary 3.11. For integers $l, p_i, r > 1$ for $i = 1, \dots, q$, the existence of the following factorizations is equivalent

$$\exists \begin{cases} (l, p_1 p_2 \dots p_q r) \\ (p_1 l, p_2 \dots p_q r) \\ \vdots \\ (p_1 p_2 \dots p_q l, r) \end{cases} \iff \exists (l, p_1, p_2, \dots, p_q, r).$$

Proof. The proof consists in combining factorizations to gradually increase the length of the product. Considering the first two factorizations of the list and invoking Definition 3.10 (with $\mathbf{l} = l$, $p = p_1$ and $\mathbf{r} = p_2 \dots p_q r$), we obtain

$$\begin{cases} (l, p_1 p_2 \dots p_q r) \\ (p_1 l, p_2 \dots p_q r) \end{cases} \implies \exists (l, p_1, p_2 \dots p_q r).$$

Now combining the result with the third factorization and invoking Definition 3.10 (with $\mathbf{l} = (l, p_1)$, $p = p_2$ and $r = p_3 \dots p_q r$) yields

$$\begin{cases} (l, p_1, p_2 \dots p_q r) \\ (p_1 p_2 l, p_3 \dots p_q r) \end{cases} \implies \exists (l, p_1, p_2, p_3 \dots p_q r).$$

We continue the process by repeatedly combining the result with the next factorization in the list until we end up with $(l, p_1, p_2, \dots, p_q, r)$. The proof for the backward implication again trivially follows from the associativity of the Kronecker product. $\square$

Based on Definition 3.11, we may restrict our attention to length 2 factorizations to produce factorizations of greater length. Definition 3.11 also immediately leads to an algorithm where length 2 factorizations that are related (through the integers p_i) are grouped in a single *branch*. In practice, we might end up with multiple branches and a given length 2 factorization might belong to different branches. The algorithm for constructing all different branches is fairly simple. Starting from the set of all possible length 2 factorizations

$$\mathcal{F} = \{(l_1, r_1), \dots, (l_s, r_s)\}$$

for some integer s , let $\mathcal{L} = \{l_1, \dots, l_s\}$ and $\mathcal{R} = \{r_1, \dots, r_s\}$ denote the set of left and right indices, respectively. Without loss of generality, we may assume that $\mathcal{L}$ is sorted such that $l_1 < l_2 < \dots < l_s$ and the right indices are paired accordingly. Hereafter, we present an algorithm for computing the various branches. Given a set $\mathcal{X} \subset \mathbb{N}$, we denote $\overline{\mathcal{X}}$ the reduced set obtained by eliminating all elements that are multiples of other elements in the set. For instance, if $\mathcal{X} = \{2, 3, 4, 6\}$, then $\overline{\mathcal{X}} = \{2, 3\}$.

For building the various branches, let $\mathcal{M}_0 = \mathcal{L}$ and choose an integer $\ell_0 \in \overline{\mathcal{M}_0}$. Then, form the set $\mathcal{M}_1 \subseteq \mathcal{L}$ of multiples of ℓ_0 and choose an integer $\ell_1 \in \overline{\mathcal{M}_1}$. Repeat the process until finding an integer ℓ_q that does not have any multiples in $\mathcal{L}$. The integers $\{\ell_0, \ell_1, \dots, \ell_q\} \subseteq \mathcal{L}$ are then paired to $\{\tau_0, \tau_1, \dots, \tau_q\} \subseteq \mathcal{R}$ such that $\ell_k \tau_k = n$ for all k . Together, they form the left and right indices, respectively, of *one* possible branch. By construction, those indices are all consecutive multiples and there exist integers $\{p_1, \dots, p_q\}$ such that $\ell_k = p_k \ell_{k-1}$ for $k = 1, \dots, q$ (and similarly for the right indices). By keeping track of all possible choices at each stage of the process, we may construct all possible branches. Assuming that m branches have been constructed and indexing the k th branch with the superscript k , we eventually obtain

$$\begin{cases} (\ell_0^k, \tau_0^k) = (\ell_0^k, p_1^k p_2^k \dots p_{q_k}^k \tau_{q_k}^k) \\ (\ell_1^k, \tau_1^k) = (p_1^k \ell_0^k, p_2^k \dots p_{q_k}^k \tau_{q_k}^k) \\ \vdots \\ (\ell_{q_k}^k, \tau_{q_k}^k) = (p_1^k p_2^k \dots p_{q_k}^k \ell_0^k, \tau_{q_k}^k) \end{cases} \quad k = 1, \dots, m.$$

By convention, we assume that $q_k = 0$ corresponds to a branch with the single factorization (ℓ_0^k, τ_0^k) . Obviously, an analogous construction holds for the right indices and choosing one or the other is just a matter of taste. We will later present a convenient way of visualizing the factorizations resulting from different branches but at this stage an example is opportune.

Example 3.12. Let us return to Definition 2.6. Forming the matrix A and computing all length 2 factorizations, we find

$$\mathcal{L} = \{2, 3, 4\}.$$

Following the procedure described earlier we set $\mathcal{M}_0 = \mathcal{L}$ and find $\overline{\mathcal{M}_0} = \{2, 3\}$. Thus, there are two possibilities for choosing ℓ_0 :

- If $\ell_0 = 2$, $\mathcal{M}_1 = \overline{\mathcal{M}_1} = \{4\}$ and we must necessarily choose $\ell_1 = 4$.
- If $\ell_0 = 3$ the algorithm ends here since $\mathcal{L}$ does not contain any multiples of 3.

Consequently, we only have two branches from which we deduce two factorizations

$$\begin{cases} (2, 6) \\ (4, 3) \end{cases} \implies (2, 2, 3), \quad \left\{ (3, 4). \right.$$

Those are precisely the two factorizations listed in Definition 2.6.

The only remaining theoretical question is whether the factorizations resulting from different branches are prime. The next lemma provides a first answer.

Lemma 3.13. For a matrix $A \in \mathbb{B}^{n \times n}$, the right (resp. left) factor of an (n_1, n_2) factorization is decomposable if and only if there exists an $(\hat{n}_1, \hat{n}_2)$ factorization where $\hat{n}_2$ divides n_2 (resp. $\hat{n}_1$ divides n_1).

Proof. If the right factor A_2 of the (n_1, n_2) factorization is decomposable, then there exist matrices P and $\hat{A}_2$ such that $A_2 = P \otimes \hat{A}_2$. Thus,

$$A_1 \otimes A_2 = (A_1 \otimes P) \otimes \hat{A}_2$$

and there exists an $(\hat{n}_1, \hat{n}_2)$ factorization where $\hat{n}_2$ divides n_2 .

Now assume there exists an $(\hat{n}_1, \hat{n}_2)$ factorization where $\hat{n}_2$ divides n_2 . Then, there exists an integer p such that $n_2 = p\hat{n}_2$ and therefore $\hat{n}_1 = pn_1$. But then, by Definition 3.8,

$$\begin{cases} (n_1, p\hat{n}_2) \\ (pn_1, \hat{n}_2) \end{cases} \implies \exists (n_1, p, \hat{n}_2)$$

such that $A = A_1 \otimes P \otimes \hat{A}_2$ and from the uniqueness of the factorization (Definition 2.4) we deduce that $A_2 = P \otimes \hat{A}_2$ and A_2 is decomposable. The proof for the left factor follows similar arguments. $\square$

Assume we have constructed a branch

$$\begin{cases} (l, p_1 p_2 \dots p_q r) \\ (p_1 l, p_2 \dots p_q r) \\ \vdots \\ (p_1 p_2 \dots p_q l, r) \end{cases}$$

according to the procedure described earlier. We will now show that the $(l, p_1, p_2, \dots, p_q, r)$ factorization obtained by combining all length 2 factorizations within the branch is prime provided $\mathcal{F}$ lists *all* possible length 2 factorizations.

Corollary 3.14. If $\mathcal{F}$ contains all possible length 2 factorizations, then the factorizations constructed from each separate branch are prime.

Proof. We will first prove that the leftmost and rightmost factors of the $(l, p_1, p_2, \dots, p_q, r)$ factorization are prime. Indeed, by construction, l does not have any divisor in $\mathcal{L}$ so by Definition 3.13, the leftmost factor is prime. Moreover, assume by contradiction that the rightmost factor is decomposable such that $r = \tilde{p}\tilde{r}$. But then $\mathcal{F}$ must contain the $(\tilde{p}p_1 p_2 \dots p_q l, \tilde{r})$ factorization and consequently there would exist a multiple of $p_1 p_2 \dots p_q l$ in $\mathcal{L}$. This contradicts the termination criterion for constructing a branch and consequently the rightmost factor must also be prime. To prove that all the inner factors of the decomposition are also prime, assume by contradiction that one of the inner factors, say P_i , is decomposable such that $p_i = \tilde{l}_i \tilde{r}_i$. Therefore, $\mathcal{F}$ must contain the $(p_1 p_2 \dots p_{i-1} \tilde{l}_i l, \tilde{r}_i p_{i+1} \dots p_q r)$ factorization. But then there would exist a multiple of $\ell_{i-1} = p_1 p_2 \dots p_{i-1} l$ in $\mathcal{L}$ that divides $\ell_i = p_1 p_2 \dots p_i l$. Once again, this clashes with the construction of a branch since any left index ℓ that is a multiple of ℓ_{i-1} cannot be a divisor of ℓ_i . Thus, we conclude that all internal factors are also prime. $\square$

Definition 3.14 proves that distinct branches lead to distinct prime factorizations and remarkably, we may re-construct those factorizations from the enumeration of all length 2 factorizations. However, the primality guarantee is lost if $\mathcal{F}$ only *partially* lists length 2 factorizations. This does not prevent us from applying the algorithm but the factor matrices in the resulting decompositions may still be decomposable. Also bear in mind that even if a factorization is prime, the sizes of the factor matrices in the decomposition are generally not prime numbers (see Definition 2.6). Nevertheless, maximal matrices are again special in this regard and their properties are summarized in the next theorem.

Theorem 3.15. Let $A \in \mathbb{B}^{n \times n}$ be a decomposable maximal matrix and let $n = \prod_{j=1}^m p_j^{k_j}$ denote the prime decomposition of n . Then, for an $(n_1, \dots, n_\ell)$ factorization resulting from a given branch,

1. $n_i \in \{p_1, \dots, p_m\}$ for all $i = 1, \dots, \ell$.
2. $\ell = \sum_{j=1}^m k_j$.
3. The number of branches is $\frac{\ell!}{\prod_{j=1}^m k_j!}$.

Proof. Since A is maximal, $\mathcal{F} = \mathcal{C}$ and $\mathcal{L}$ contains all non-trivial divisors of n . Thus,

$$\mathcal{L} = \left\{ \prod_{j=1}^m p_j^{q_j} : 0 < \sum_{j=1}^m q_j < \sum_{j=1}^m k_j, 0 \leq q_j \leq k_j, j = 1, \dots, m \right\}.$$

From our algorithmic construction, the sizes $n_1, \dots, n_\ell$ resulting from any branch form a sequence of prime numbers, where each p_j appears k_j times among $n_1, \dots, n_\ell$. Consequently, the length of any such factorization is

$$\ell = \sum_{j=1}^m k_j.$$

Moreover, the number of distinct sequences (or branches) is the number of permutations of ℓ objects, among which k_1 are indistinguishable, k_2 are indistinguishable and so on. From elementary statistics, this number is

$$\frac{\ell!}{\prod_{j=1}^m k_j!}.$$

□

Definition 3.15 also implies that for a general $A \in \mathbb{B}^{n \times n}$, the length of any factorization cannot exceed $\sum_{j=1}^m k_j$, where the k_j are the exponents in the prime decomposition of n . This result merely confirms what could be straightforwardly inferred without setting up any heavy machinery. The results of Definition 3.15 are easily verified on Definition 2.5. For completeness, we present another more descriptive example by applying the algorithm outlined earlier.

Example 3.16. Let

$$\mathcal{L} = \{2, 3, 4, 6, 8, 12\}$$

be the (sorted) left indices of all possible length 2 factorizations of a maximal matrix of size $24 = 2^3 \cdot 3$. Thus, $m = 2$ with prime numbers $p_1 = 2$, $p_2 = 3$ and integers $k_1 = 3$, $k_2 = 1$. Let us first apply our algorithm while ignoring the fact that A is maximal. As explained earlier we initialize $\mathcal{M}_0 = \mathcal{L}$ and choose an integer in $\overline{\mathcal{M}}_0 = \{2, 3\}$. The integers in $\overline{\mathcal{M}}_0$ are the roots of any factorization we may form. Keeping track of the various choices at each stage of the process exhibits the various branches, leading to four distinct prime factorizations.

- If $\ell_0 = 2$, $\mathcal{M}_1 = \{4, 6, 8, 12\}$ and $\overline{\mathcal{M}}_1 = \{4, 6\}$
 - If $\ell_1 = 4$, $\mathcal{M}_2 = \{8, 12\}$ and $\overline{\mathcal{M}}_2 = \{8, 12\}$.
 - * If $\ell_2 = 8$ the branch ends since $\mathcal{L}$ does not contain any multiple of 8.
 - * If $\ell_2 = 12$ the branch ends since $\mathcal{L}$ does not contain any multiple of 12.
 - If $\ell_1 = 6$, $\mathcal{M}_2 = \{12\}$ and $\overline{\mathcal{M}}_2 = \{12\}$. Consequently, $\ell_2 = 12$ and the branch ends.
- If $\ell_0 = 3$, $\mathcal{M}_1 = \{6, 12\}$, $\overline{\mathcal{M}}_1 = \{6\}$ and the only one possible path is $\ell_1 = 6$ and $\ell_2 = 12$.

We have therefore found four different branches

$$\begin{array}{ll} \left\{ \begin{array}{l} (2, 12) \\ (4, 6) \\ (8, 3) \end{array} \right\} & \implies (2, 2, 2, 3), & \left\{ \begin{array}{l} (2, 12) \\ (4, 6) \\ (12, 2) \end{array} \right\} & \implies (2, 2, 3, 2), \\ \left\{ \begin{array}{l} (2, 12) \\ (6, 4) \\ (12, 2) \end{array} \right\} & \implies (2, 3, 2, 2), & \left\{ \begin{array}{l} (3, 8) \\ (6, 4) \\ (12, 2) \end{array} \right\} & \implies (3, 2, 2, 2). \end{array}$$

In agreement with Definition 3.15, the sizes of the factor matrices in each factorization are prime numbers, the length of each factorization is $\ell = k_1 + k_2 = 4$ and the number of factorizations is $\ell!/(k_1!k_2!) = 4$.

However, it is worthwhile noting that the Kronecker product of two maximal matrices is generally not maximal, as shown in the next example.

Example 3.17. Consider the maximal matrix

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}. \end{aligned}$$

Taking the Kronecker product of A with itself, we obtain a matrix of size $36 = 2^2 \cdot 3^2$. However, it only admits 5 prime factorizations, which is smaller than the 6 factorizations it could potentially have. Indeed, the $(2, 2, 3, 3)$ factorization is missing because 4 does not belong to the set of left indices.

In addition to the sizes of the factor matrices, one may also be interested in their sparsity pattern. Fortunately, once the sizes of the factor matrices in a decomposition are known, one may also easily retrieve their sparsity pattern as a simple post-processing step. This operation only requires labeling the length 2 factorizations constituting a branch and storing the sparsity pattern of their factors. Note that the sparsity pattern of length 2 factorizations is an immediate byproduct of the factorizability check and does not entail any extra cost. The sparsity pattern of factorizations of length $\ell > 2$ is then deduced one factor at a time by combining the sparsity pattern of consecutive length 2 factorizations. Indeed, the length 2 factorizations entering a branch are obtained by simply regrouping an increasingly large number of left factors together. More specifically, the length 2 factorizations constituting the branch of an arbitrary $(l, p_1, p_2, \dots, p_q, r)$ factorization $L \otimes P_1 \otimes P_2 \otimes \dots \otimes P_q \otimes R$ correspond to the following products:

$$\begin{cases} (l, p_1 p_2 \dots p_q r) & L \otimes (P_1 \otimes P_2 \otimes \dots \otimes P_q \otimes R), \\ (p_1 l, p_2 \dots p_q r) & (L \otimes P_1) \otimes (P_2 \otimes \dots \otimes P_q \otimes R), \\ \vdots & \\ (p_1 p_2 \dots p_q l, r) & (L \otimes P_1 \otimes P_2 \otimes \dots \otimes P_q) \otimes R. \end{cases}$$

Clearly, the sparsity pattern of the leftmost factor L is simply the sparsity pattern of the left factor of the first length 2 decomposition constituting the branch. The sparsity pattern of P_1 is then deduced from the sparsity patterns of L and $L \otimes P_1$, the latter simply being the sparsity pattern of the left factor in the second length 2 factorization. More generally, one deduces the sparsity pattern of P_i from the sparsity patterns of $L \otimes P_1 \otimes \dots \otimes P_{i-1}$ and $L \otimes P_1 \otimes \dots \otimes P_{i-1} \otimes P_i$, which are nothing but the left factors of the i th and $(i+1)$ th factorizations entering the branch. Finally, the sparsity pattern of the rightmost factor R is simply the sparsity pattern of the right factor of the last factorization. In summary, this process consists in repeatedly finding the sparsity pattern of the right factor of a Kronecker product knowing the sparsity pattern of the left factor and the one of the product. The solution to this problem is straightforward: assuming we are given an (n_1, n_2) factorization of $A = A_1 \otimes A_2$, where the sparsity patterns of A and A_1 are known, then the sparsity pattern of A_2 is simply read from the (i, j) th block of A of size n_2 corresponding to any nonzero entry (i, j) of A_1 . Similarly, if instead the sparsity patterns of A and A_2 were known, the sparsity pattern of A_1 would easily be deduced by tracking down the position of the nonzero blocks of A of size n_2 . Hence, one could equivalently apply the aforementioned process from right to left instead of from left to right. In either case, all that is needed is the sparsity pattern of the factor matrices for the length 2 factorizations entering a branch.

Now that we have a convenient way of constructing prime factorizations and finding the sparsity pattern of its factors, we present in the next section an elegant way of visualizing them.

4 The decomposition graph

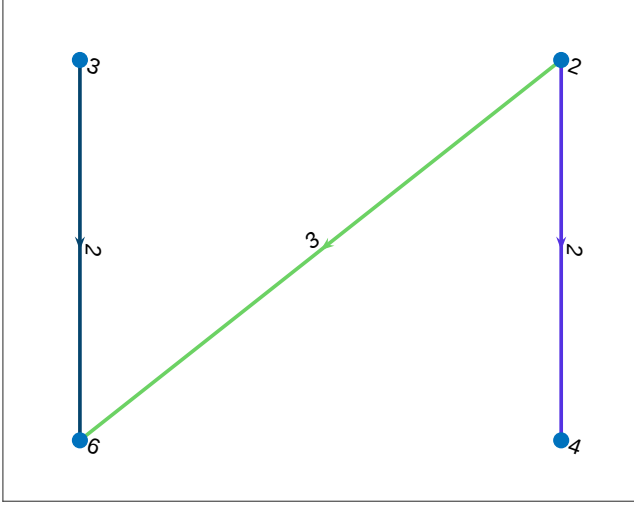
In this section, we build a directed graph G that encodes the construction of factorization branches. A graph is characterized by a set of vertices $V(G)$ and a set of edges $E(G)$, that are pairs of vertices. In our setting, this graph is more specifically a multigraph: it may have multiple edges connecting the same pair of vertices. Its construction obeys the following rules:

- $V(G) = \mathcal{L}$.
- For each branch k we connect successive left indices by an edge such that $\mathcal{L}^k = \{\ell_0^k, \ell_1^k, \dots, \ell_{q_k}^k\} \subseteq \mathcal{L}$ forms a path. Each path is identified by a distinct color or label.
- For the k th path $\mathcal{L}^k$, the “weights” on each edge are the integers p_j^k .

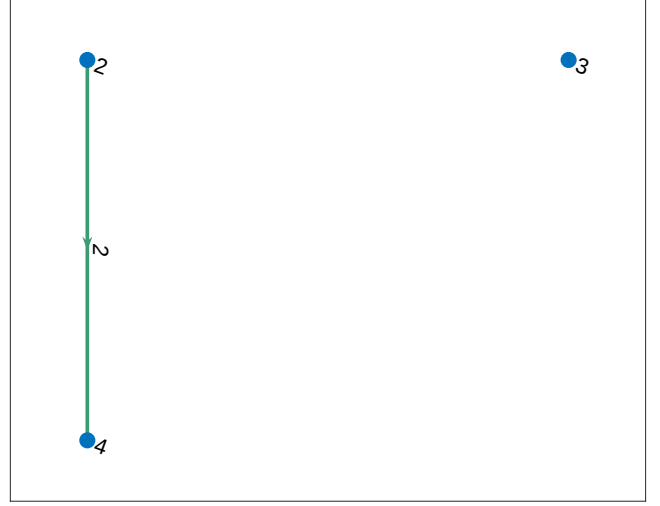
From this construction, we immediately deduce the following properties:

- The length of the k th factorization $\ell_k = |\mathcal{L}^k| + 1$ is related to the length of the k th path.
- Isolated vertices correspond to prime length 2 factorizations (but are still identified as “paths”).
- The number of paths is equal to the number of branches, which is itself equal to the number of prime factorizations.

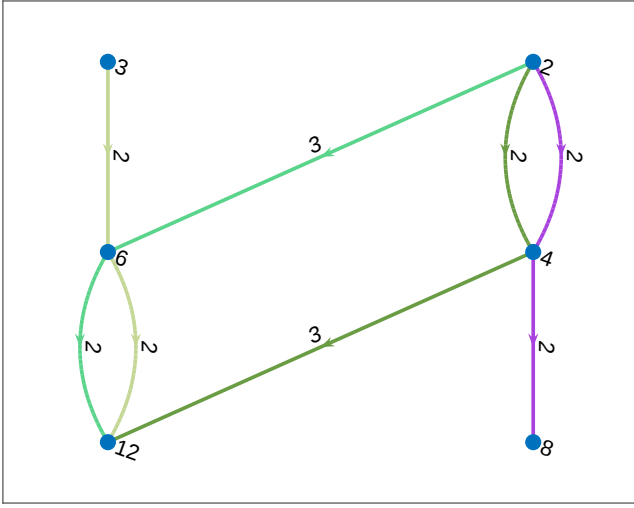
For reading the sizes of the factor matrices in the k th factorization, we start from the root ℓ_0^k and read the successive weights on the edges until arriving at the end of the path. The size of the last factor matrix is then $n/\ell_{q_k}^k$. The decomposition graphs for the various examples we have encountered so far are shown in Figure 4.1. It unveils in a single figure the Kronecker structure of a matrix and becomes especially useful for matrices admitting multiple distinct factorizations. Furthermore, since it visualizes the structure rather than the matrix, different matrices may have the same decomposition graph.



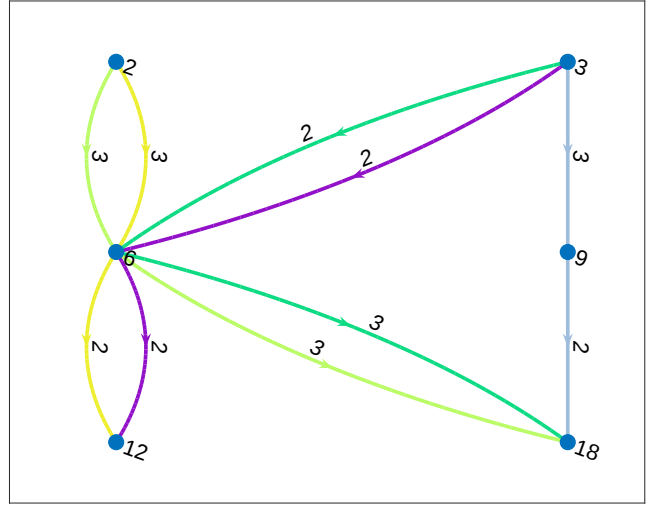
(a) Definition 2.5



(b) Definition 2.6



(c) Definition 3.16



(d) Definition 3.17

Figure 4.1: Decomposition graphs

5 Applications

5.1 Space-time isogeometric discretizations

One of the most significant applications of this research consists in identifying potentially good (approximate) Kronecker factorizations of a real or complex matrix B just by analyzing its sparsity pattern. As an instructive example, we consider a space-time isogeometric discretization of the heat equation, a well studied parabolic PDE. For time-dependent PDEs, such as the heat or the wave equation, space-time methods discretize both spatial and temporal domains with finite elements [20, 21]. In isogeometric analysis, finite element spaces are tensor products of smooth spline spaces [22, 23] and lead to large structured system matrices where the spatial and temporal degrees of freedom are solved all at once [24]. We consider in this section the discretization of the heat equation

$$\begin{aligned} \partial_t u(\mathbf{x}, t) - \Delta u(\mathbf{x}, t) &= f(\mathbf{x}, t) && \text{in } \Omega \times (0, T], \\ u(\mathbf{x}, t) &= 0 && \text{on } \partial\Omega \times (0, T], \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) && \text{in } \Omega, \end{aligned}$$

over the space-time cylinder $Q = \Omega \times (0, T)$, where $\Omega \subset \mathbb{R}^3$ is the magnet-shaped domain shown in Figure 5.1 and $T > 0$ is the final time. The unknown function $u: \bar{Q} \rightarrow \mathbb{R}$ represents a temperature field, f is a known source term and u_0 is an initial condition.

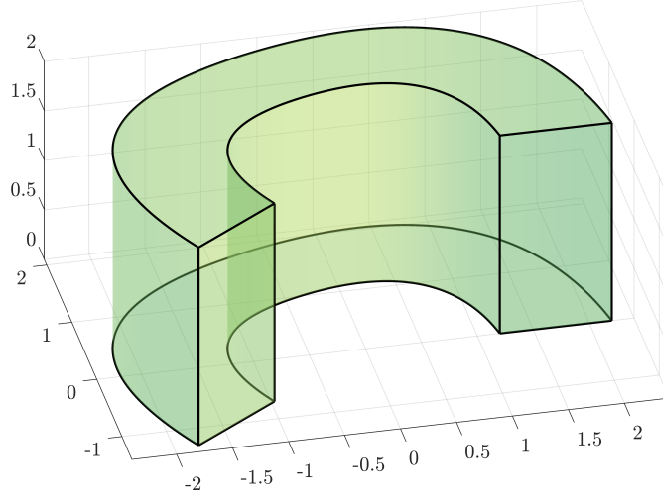


Figure 5.1: Magnet shaped domain taken from [15, Figure 5.7]

Discretizing the problem with space-time isogeometric methods requires solving a large linear system whose coefficient matrix is (see e.g. [20])

$$B = W_t \otimes M_s + M_t \otimes K_s \quad (5.1)$$

where

$$(W_t)_{ij} = \int_0^T b'_j(t)b_i(t)dt \quad (M_t)_{ij} = \int_0^T b_j(t)b_i(t)dt \quad i, j = 1, \dots, n_t$$

are the “temporal” finite element matrices and $\{b_i(t)\}_{i=1}^{n_t}$ are the B-spline basis functions for the temporal domain $(0, T)$. Similarly,

$$(K_s)_{ij} = \int_{\Omega} \nabla B_j(\mathbf{x}) \cdot \nabla B_i(\mathbf{x}) d\Omega \quad (M_s)_{ij} = \int_{\Omega} B_j(\mathbf{x}) B_i(\mathbf{x}) d\Omega \quad i, j = 1, \dots, n_s$$

are the “spatial” stiffness and mass matrices and $\{B_i(t)\}_{i=1}^{n_s}$ are the B-spline basis functions for the spatial domain Ω . The interested reader may consult [22, 23] for the construction of the B-spline basis and a gentle introduction to isogeometric analysis. Although the spatial stiffness and mass matrices K_s and M_s are generally not a Kronecker product, for certain spline parameterizations they are often exceedingly well approximated by a sum of Kronecker products [14, 25, 26], a property that is also reflected in their sparsity pattern. Indeed, these matrices exhibit a hierarchical block structure, where the block sizes and their bandwidths are directly related to the spline discretization parameters [14, 15]. Their Kronecker product with the banded temporal stiffness and mass matrices W_t and M_t in (5.1) just adds another hierarchical level, as shown in Figure 5.2. Thus, the sparsity pattern of B still hides a Kronecker product although the matrix itself may not be exactly factorized. In this example, we apply our algorithm to identify the sizes of the constituting factor matrices before computing an approximate Kronecker factorization. This example is obviously contrived since the sizes are commonly deduced from the discretization parameters and boundary conditions. However, that information may not always be available if the discretization is carried out independently of the linear system solver. In contrast, our algorithm operates in a “black-box” manner and allows reading the discretization parameters just by analyzing the sparsity pattern of the matrix. As a matter of fact, the discretization parameters for this example lead to a system matrix B of size $n = 53568$ and our algorithm perfectly recovers the (unique) $(31, 12, 12, 12)$ prime factorization of its sparsity pattern, in agreement with Figure 5.2¹. Once the sizes are known, one may easily deduce the sparsity pattern of the factors constituting the decomposition, as well as their bandwidth. In this example, each factor is banded of bandwidth 2. We may relate all that information back to the mesh sizes and spline orders of the discretization. However, in this example, we are primarily interested in computing an approximate Kronecker factorization of B , which may later serve as a preconditioner. The Kronecker structure already exhibited in (5.1) suggests approximating B by a sum of two length 4 Kronecker products

$$B \approx \tilde{B} = A_1 \otimes B_1 \otimes C_1 \otimes D_1 + A_2 \otimes B_2 \otimes C_2 \otimes D_2.$$

¹ The code for reproducing the results is freely available at the following address: <https://github.com/YannisVoet/Kronecker>

Computing this approximation requires tensor decomposition techniques, as described in [11]. For this purpose, we have used Matlab’s Tensor Toolbox [27]. The sizes of the factor matrices entering the decomposition are directly supplied by our algorithm. Those sizes directly reflect the structure of the matrix and are a rather natural choice for attempting an approximate factorization. Of course, there exist multiple other candidate sizes for computing approximate factorizations but the large error incurred may later impede on the preconditioner’s effectiveness. For demonstrating it, let us compute an approximate factorization for three randomly chosen, albeit compatible sizes $\mathbf{n}_i$ for $i = 1, 2, 3$ and compare them to the sparsity-informed guess $\mathbf{n} = (31, 12, 12, 12)$. As shown in Table 5.1, the approximation errors in the Frobenius norm are about 10 times larger than for the sparsity-informed guess. Approximating K_s and M_s each with a single length 3 factorization and plugging the result in (5.1) also produces a valid approximation but the error (0.2855) is still significantly larger than for the sparsity-informed length 4 factorization. Smaller approximation errors are expected to produce better preconditioners. For illustrating it, we solve the linear system $B\mathbf{x} = \mathbf{e}_1$, where $\mathbf{e}_1$ is the first vector of the canonical basis of $\mathbb{R}^n$. In this experiment, the nonsymmetric linear system is solved iteratively with a right preconditioned GMRES method [28], restarted every 30 iterations until reaching an absolute residual norm of 10^{-8} . The method converged after 109, 13 and 19 iterations for $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_3$ whereas only 8 iterations were required for $\mathbf{n}$. The iteration count increases to 52 when separately approximating K_s and M_s . This example further highlights the value of approximate Kronecker factorizations for building preconditioners and is in this context one among many other strategies proposed for space-time discretizations; see e.g. [20, 24, 29].

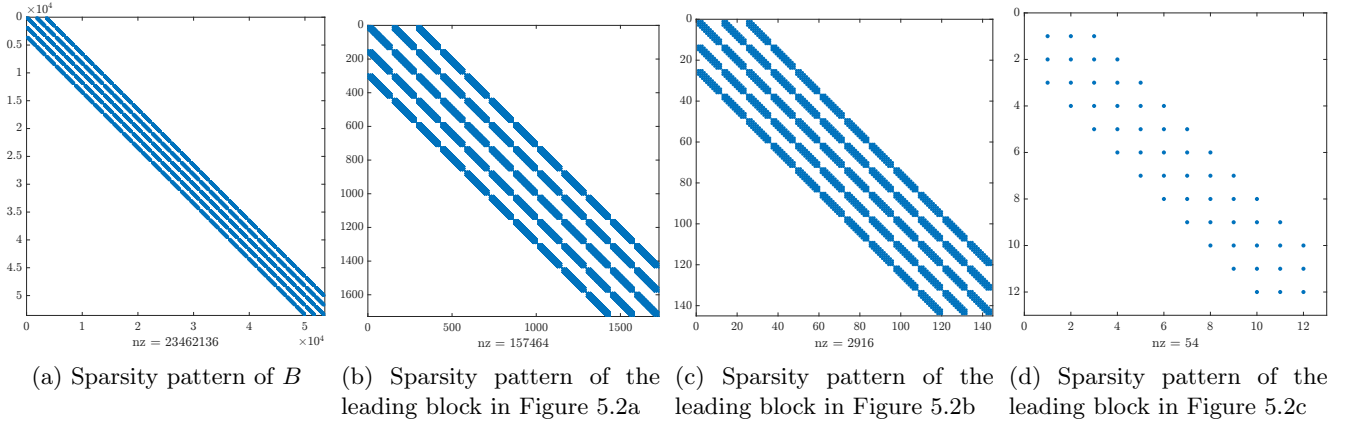


Figure 5.2: Hierarchical block structure of B

Size	$\ B - \tilde{B}\ _F$
$\mathbf{n} = (31, 12, 12, 12)$	0.0489
$\mathbf{n}_1 = (31, 144, 6, 2)$	0.5561
$\mathbf{n}_2 = (31, 12, 6, 24)$	0.3832
$\mathbf{n}_3 = (62, 6, 6, 24)$	0.4973

Table 5.1: Sizes of the factor matrices and resulting approximation errors

Remark 5.1. In the example of Section 5.1, the sparsity pattern had a unique prime factorization. In fact, from the point of view of preconditioning, the existence of multiple prime factorizations is undesirable since it still leaves multiple candidate sizes. In the extreme case of dense matrices, the sparsity pattern is maximal and does not help at all for choosing candidate sizes. In such cases, exhaustive search strategies described in [17] might become necessary, unless additional information on the problem’s origin is known.

5.2 Decomposition and visualization of Kronecker graphs

Our second application pertains to the representation and visualization of networks as Kronecker graphs. Given two graphs G_1 and G_2 with vertex set $V(G_i)$ and edge set $E(G_i)$ for $i = 1, 2$, their Kronecker product $G_1 \otimes G_2$ (sometimes also called direct, cardinal, categorical or tensor product) is a larger graph with vertex set defined as the Cartesian product

$$V(G) = V(G_1) \times V(G_2) = \{(u, v) : u \in V(G_1), v \in V(G_2)\}$$

and edge set

$$E(G) = \{((u, v), (u', v')) : (u, u') \in E(G_1), (v, v') \in E(G_2)\}.$$

Up to a relabeling of the vertices, the adjacency matrix A of a Kronecker graph is the Kronecker product of the adjacency matrices A_i [8]; i.e. there exists a permutation matrix P such that

$$P^T A P = A_1 \otimes A_2.$$

This permutation matrix is a major hurdle for identifying Kronecker graphs and a key difference with the matrix case. Nevertheless, Kronecker graphs are attractive for multiple reasons. Firstly, the structure and properties of Kronecker graphs are easier to analyze and often deduced from the properties of the smaller graphs constituting them. Thus, they effectively condense information. Secondly, many of those properties accurately model real networks. For instance, in the machine learning community, Leskovec et al. [2, 3] realized that synthetic Kronecker graphs could somewhat reproduce the degree distribution, diameter and spectrum of real networks, a set of properties earlier models often failed to mimic. Finally, the inherent structure of Kronecker graphs also helps visualizing them. Here we will especially focus on this last point, which is largely independent of any application. Despite a late surge of interest in Kronecker graphs, attempts at visualizing them are rather sparse. As a matter of fact, [4, Chapter 11] is the only noteworthy contribution we are aware of. Therein, apart from visualizing the sparsity pattern of the adjacency matrix, the authors propose a 3D visualization of Kronecker graphs by projection them onto the surface of a sphere. However, the figures produced are somewhat confusing and difficult to interpret. In this section, we present a new visualization technique that exploits the Kronecker structure of the (reordered) adjacency matrix A . Assuming we have found an $(n_1, n_2, \dots, n_d)$ factorization of A , we first label the vertices and identify them with tuples $\mathbf{v} = (v_1, v_2, \dots, v_d)$, where $1 \leq v_i \leq n_i$ for $i = 1, \dots, d$. This mindset allows modeling short, medium or long distance interactions between vertices belonging, say, to different communities. If only the last few indices of two connected vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ differ, then their link represents relatively short-distance interactions, within the same community. On the contrary, if any of the first few indices differ, then they might represent long-distance inter-community interactions. The construction described below conforms with such intuitive understanding of the network by explicitly defining the position of the graph's vertices. In order to model communities on different levels (e.g. local, regional, continental, inter-continental, etc), we first draw a circle of unit radius r_1 centered at the origin of the complex plane and uniformly place n_1 nodes along the circle. These nodes are simply the n_1 roots of unity $z_k = e^{\frac{2\pi(k-1)i}{n_1}}$ for $k = 1, \dots, n_1$ and model the centers of long-distance (inter-continental) communities. For moving down to the continental scale, we draw circles around each point z_k with a smaller radius r_2 and place n_2 nodes along each of these circle for modeling the centers of continental hubs. We then repeat the process until reaching the local scale. At this stage, the points placed on the smallest circles are the position of the vertices of the graph. The construction process is illustrated in Figure 5.3a for $d = 3$ and $(n_1, n_2, n_3) = (4, 3, 2)$. Note that it only depends on the sizes of the factor matrices, not on the connectivity of the network.

The algorithm combines rotations and translations to produce a visually pleasing and interpretable representation of the network. More formally, for each level $j = 1, \dots, d$, we first place equidistance points on a circle in the complex plane

$$z_k^{(j)} = r_j e^{\frac{2\pi(k-1)i}{n_j}} \quad k = 1, \dots, n_j, \quad j = 1, \dots, d.$$

where $(r_j)_{j=1}^d$ is a decreasing sequence of radii and i denotes the imaginary unit. Additionally, we denote

$$\theta_k^{(j)} = \arg(z_k^{(j)}) + \frac{\pi}{2} \quad \text{and} \quad r_k^{(j)} = e^{\theta_k^{(j)}i}$$

the (shifted) phase angle and rotation, respectively. The shift of $\frac{\pi}{2}$ is purely for aesthetic reasons as it produced clearer figures. The position of the vertices in the graph is then defined recursively, starting from the bottom of the hierarchy, moving upwards and augmenting the index set at each step. We first initialize $g_{k_d}^{(d)} = z_{k_d}^{(d)}$. Then, assuming $g_{(k_{j+1}, \dots, k_d)}^{(j+1)}$ is known, we define

$$g_{(k_j, k_{j+1}, \dots, k_d)}^{(j)} = z_{k_j}^{(j)} + r_{k_j}^{(j)} g_{(k_{j+1}, \dots, k_d)}^{(j+1)} \quad j = 1, \dots, d-1.$$

At the end of the recursion, $p_{\mathbf{k}} = g_{\mathbf{k}}^{(1)}$ defines the position of vertex $\mathbf{k} = (k_1, k_2, \dots, k_d)$ and its coordinate values are finally retrieved as

$$x_{\mathbf{k}} = \text{Re}(p_{\mathbf{k}}), \quad y_{\mathbf{k}} = \text{Im}(p_{\mathbf{k}}).$$

Let us visualize the output of this algorithm on the Kronecker graph whose adjacency matrix is

$$A = A_1 \otimes A_2 \otimes A_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \quad (5.2)$$

The $(4, 3, 2)$ factorization of this matrix leads to the skeleton shown in Figure 5.3a. The final step simply consists in connecting the vertices (i.e. the black dots in Figure 5.3a) according to the connectivity encoded in A . The off-diagonal entries in A_1 , A_2 and A_3 may model long, medium and short-distance interactions between continental, regional and local communities, respectively. Figure 5.3b faithfully captures this interpretation. Indeed, Figure 5.3b reveals interactions between the first, second and third continental communities, in agreement with the off-diagonal entries of A_1 . We also easily identify a connection between the first and second regional communities within each continental community, which results from the off-diagonal entry in A_2 . Overall, the algorithm satisfactorily uncovers the salient features of the network. The same cannot be said of Matlab’s built-in visualization algorithms, which tend to isolate connected components or pick up other important properties of the network, albeit less relevant in this context.

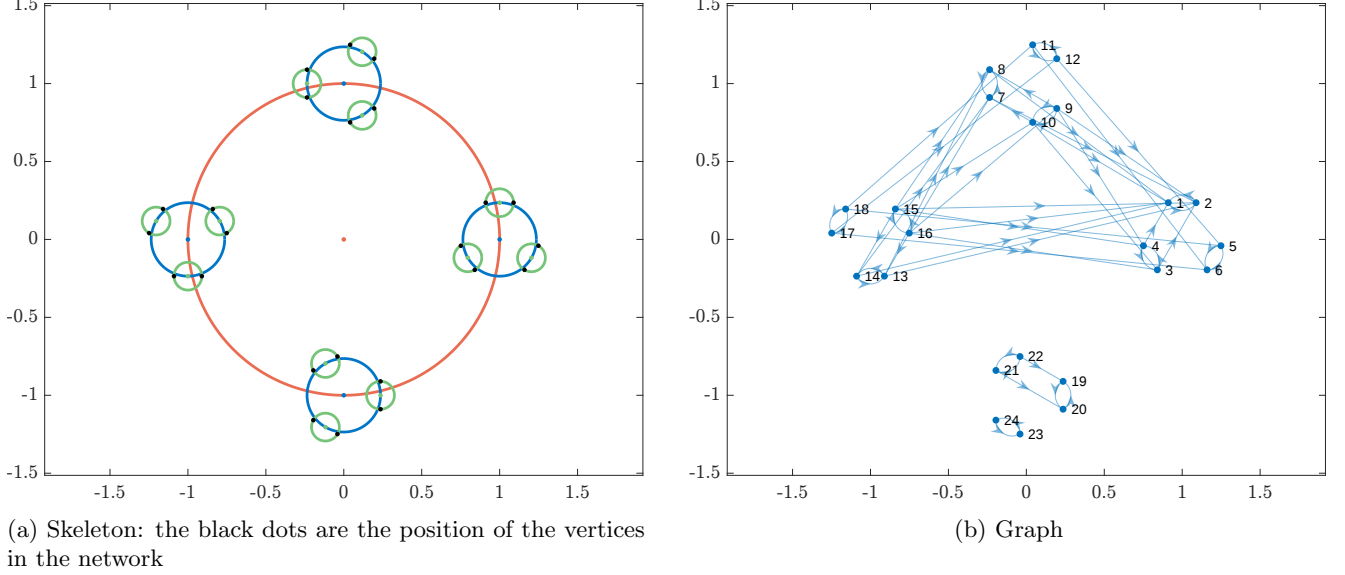


Figure 5.3: Kronecker graph visualization for (5.2)

To further highlight the capabilities of our algorithm, we test it on one of the examples presented in [3, Figure 3]. The adjacency matrix is defined as $A = \bigotimes_{i=1}^3 A_i$, with the arrowhead matrices

$$A_i = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \quad i = 1, 2, 3.$$

Figure 5.4 visualizes the graph by exploiting its Kronecker structure. Once again, the layout helps identify inter-community interactions. By construction, the edges are concentrated along specific directions, which allows easily extracting useful patterns. For large networks, it might be useful to alter the color shading or transparency of the edges depending on the nature of the interactions they model (e.g. long, medium or short-distance interactions). In a computer graphics tool, one may also zoom in as much as needed to view interactions within local communities.

In fact, our visualization algorithm is applicable regardless of the graph structure, provided the sizes are given. However, it is usually ill-suited unless the graph is “close” to a Kronecker graph. The method described in Section 3 allows identifying a Kronecker structure for a *fixed* adjacency matrix but does not immediately tackle the graph factorization problem. Nevertheless, it may serve as a building block within other strategies, as for example in [9].

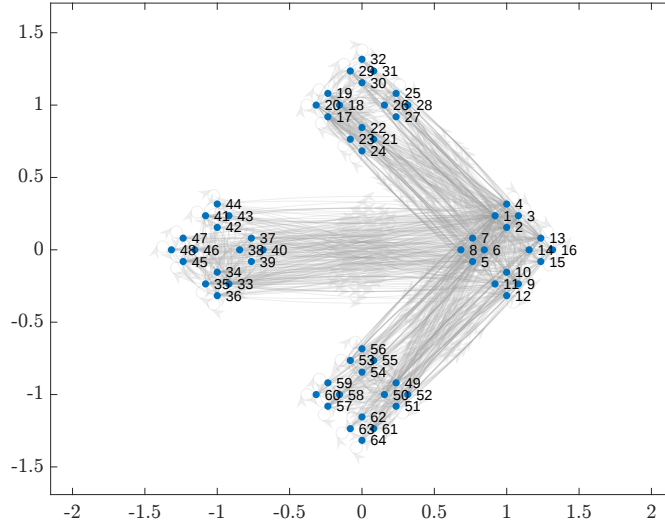


Figure 5.4: Kronecker graph for the first example in [3, Figure 3]

5.3 Quantum computing

Quantum computing is a fairly recent computational paradigm built from concepts in quantum mechanics. Contrary to classical computing, its basic unit of information, the qubit, may live in a state of superposition, a linear combination of two states. Its greatest promise lies in solving complex problems that would otherwise be infeasible on a classical computer in any reasonable time. One prominent example is the groundbreaking work of Shor [30, 31] for integer factorizations and the potential implications in cryptography. In recent years, quantum computing has gained considerable momentum, hoping for similar achievements in solving other problems. For an introduction to quantum computing, readers may consult [32, 33], among many other references. For a d -qubit system, quantum operations are defined through unitary operators (or matrices) $U \in U(2^d)$, called *gates*, acting on state vectors $|\psi\rangle \in \mathbb{C}^{2^d}$. Hereafter, $U(n)$ denotes the group of unitary matrices of size n and we introduce the normalized Hilbert–Schmidt (or trace) inner product $\langle A, B \rangle = \frac{1}{n} \text{trace}(A^* B)$ with induced norm $\|A\|_H = \sqrt{\langle A, A \rangle}$. A quantum gate is called separable if its matrix representation is decomposable and is called entangling otherwise. The most obvious instance of an entangling gate is the controlled-not (or CNOT) gate

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

In the context of quantum computing, entanglement is a desirable property for reproducing quantum phenomena and many authors have introduced measures for quantifying and maximizing it [34, 35]. A related question is to identify separable gates. Exact separability is rather uncommon in quantum systems since it boils down to local operations on independent subsystems. However, quantum gates are commonly defined through sequences of unitary operations such that $U = U_1 U_2 \dots U_k$, where $U_i \in U(2^d)$ for all $i = 1, \dots, k$. Although U is rarely separable, the gates U_i constituting the product often are. Thus, one may try identifying separability within a subsequence of operations. For a general unitary matrix, an analytical method, related to the so-called Schmidt decomposition [33], is to first decompose U in the Pauli basis. For single qubit systems, the Pauli basis is $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, where

$$\sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \sigma_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_3 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_4 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

For 2-qubit systems, $\{\sigma_i \otimes \sigma_j\}_{i,j=1}^4$ forms an orthonormal basis of $U(4)$ for the normalized Hilbert–Schmidt inner product. Consequently, any $U \in U(4)$ may be decomposed as

$$U = \sum_{i,j=1}^4 \alpha_{ij} (\sigma_i \otimes \sigma_j)$$

with coefficients $\alpha_{ij} = \langle \sigma_i \otimes \sigma_j, U \rangle$. It follows that U is separable if and only if $\alpha_{ij} = a_i b_j$. In other words, the coefficient matrix $(\alpha_{ij})_{i,j=1}^4$ is rank-1. Generalizing this method to d -qubit systems with $d \geq 2$ is straightforward.

However, it is analogous to Van Loan’s algorithm with the sizes $\mathbf{n} = (2, 2, \dots, 2) \in \mathbb{N}^d$ and merely tests separability in the Pauli basis instead of the canonical one. Moreover, even if the test fails, the gate may still be separable for different sizes of the factor matrices. In this section, we apply our algorithm to the matrix representation of the operator to classically determine whether it is separable. Similarly to the example in Section 5.1, our algorithm operates on the sparsity pattern for determining candidate sizes before attempting a tensor decomposition. We consider the synthetic gate V encoding the circuit shown in Figure 5.5. For an introduction to quantum circuits, interested readers may refer to [32, 33]. The symbols in the circuit represent single-qubit or multi-qubit gates but their definition is irrelevant here.

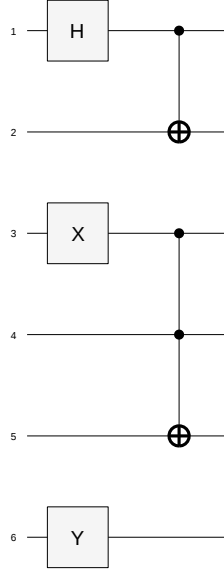


Figure 5.5: Synthetic quantum circuit encoded in V

Clearly, some of the wires in the circuit (representing qubits) do not interact and the underlying gate must have a Kronecker factorization. Indeed, our algorithm finds a unique $(4, 8, 2)$ prime factorization for the sparsity pattern. After supplying those sizes to a tensor decomposition algorithm, we obtain the Kronecker factorization

$$V = V_1 \otimes V_2 \otimes V_3.$$

Since V is unitary, the factor matrices V_i have orthogonal but not necessarily orthonormal columns. However, normalizing them as $V_i / \|V_i\|_H$ ensures they become unitary. Nevertheless, we must stress that this and other analytical methods can only solve relatively small problem sizes, much smaller than those quantum computing is expected to handle. Beyond analytical methods, in the quantum computing literature, Harrow and Montanaro [36] proposed the product-state test, a probabilistic test for determining whether a gate is separable.

6 Conclusion

In this article, we have presented a theory and algorithm for finding all possible Kronecker factorizations of a large sparse binary matrix. When encoding the sparsity pattern of real or complex matrices, factorizing binary matrices implicitly suggests suitable sizes for the factor matrices in (approximate) Kronecker factorizations. Such sparsity-informed guesses may produce exceedingly good approximations, particularly for system matrices stemming from PDE discretizations. In other applications, sparse binary matrices may represent graph adjacency matrices and factorizing them may uncover some latent structure in a network. We have subsequently proposed a graph visualization algorithm that exploits this structure to faithfully depict the nature of the interactions within the network. Although most real networks are certainly not factorizable, some are nevertheless accurately modeled as Kronecker graphs.

Regardless of their origin, binary matrices may admit multiple distinct Kronecker factorizations. For visualizing them, we have constructed a decomposition graph depicting the number of factorizations, their length and the sizes of the factor matrices entering each decomposition. Extending our framework to rectangular factor matrices is

possible but quite irrelevant to the applications considered in this work, where decompositions with square factor matrices were always sought. Nevertheless, more general decompositions might have applications elsewhere and remain an interesting problem from a theoretical perspective. Even in the square case, the theoretical possibilities are way ahead of the practical reality. For instance, we are not even aware of a real network admitting multiple factorizations, although we cannot a priori exclude it.

References

- [1] C. F. Van Loan, The ubiquitous Kronecker product, *Journal of computational and applied mathematics* 123 (1-2) (2000) 85–100.
- [2] J. Leskovec, C. Faloutsos, Scalable modeling of real graphs using Kronecker multiplication, in: *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 497–504.
- [3] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, Z. Ghahramani, Kronecker graphs: an approach to modeling networks., *Journal of Machine Learning Research* 11 (2) (2010).
- [4] J. Kepner, J. Gilbert, *Graph algorithms in the language of linear algebra*, SIAM, 2011.
- [5] W. Imrich, Factoring cardinal product graphs in polynomial time, *Discrete Mathematics* 192 (1-3) (1998) 119–144.
- [6] R. H. Hammack, W. Imrich, S. Klavžar, *Handbook of product graphs*, Vol. 2, CRC press Boca Raton, 2011.
- [7] P. M. Weichsel, The Kronecker product of graphs, *Proceedings of the American mathematical society* 13 (1) (1962) 47–52.
- [8] L. Calderoni, L. Margara, M. Marzolla, Direct product primality testing of graphs is GI-hard, *Theoretical Computer Science* 860 (2021) 72–83.
- [9] L. Calderoni, L. Margara, M. Marzolla, A heuristic for direct product graph decomposition, *Journal of Graph Algorithms and Applications* 27 (7) (2023) 581–601.
- [10] C. F. Van Loan, N. Pitsianis, Approximation with Kronecker products, in: *Linear algebra for large scale and real-time applications*, Springer, 1993, pp. 293–314.
- [11] A. N. Langville, W. J. Stewart, A Kronecker product approximate preconditioner for SANs, *Numerical Linear Algebra with Applications* 11 (8-9) (2004) 723–752.
- [12] J. G. Nagy, M. E. Kilmer, Kronecker product approximation for preconditioning in three-dimensional imaging applications, *IEEE Transactions on Image Processing* 15 (3) (2006) 604–613.
- [13] Y. Voet, Preconditioning techniques for generalized Sylvester matrix equations, *Numerical Linear Algebra with Applications* 32 (2) (2025) e70020.
- [14] C. Hofreither, A black-box low-rank approximation algorithm for fast matrix assembly in isogeometric analysis, *Computer Methods in Applied Mechanics and Engineering* 333 (2018) 311–330.
- [15] Y. Voet, E. Sande, A. Buffa, Mass lumping and outlier removal strategies for complex geometries in isogeometric analysis, *Mathematics of Computation* (2025).
- [16] S. Egner, M. Püschel, T. Beth, Decomposing a permutation into a conjugated tensor product, in: *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, 1997, pp. 101–108.
- [17] C. Cai, R. Chen, H. Xiao, Kopa: Automated Kronecker product approximation, *Journal of machine learning research* 23 (236) (2022) 1–44.
- [18] R. A. Horn, C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, 1991.
- [19] X. Wei, H. Li, G. Zhao, Kronecker product decomposition of Boolean matrix with application to topological structure analysis of Boolean networks, *Math. Model. Control* 3 (2023) 306–315.
- [20] G. Loli, M. Montardini, G. Sangalli, M. Tani, An efficient solver for space–time isogeometric Galerkin methods for parabolic problems, *Computers & Mathematics with Applications* 80 (11) (2020) 2586–2603.

- [21] G. Loli, G. Sangalli, P. Tesini, High-order spline upwind for space–time Isogeometric Analysis, *Computer Methods in Applied Mechanics and Engineering* 417 (2023) 116408.
- [22] T. J. Hughes, J. A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, *Computer methods in applied mechanics and engineering* 194 (39-41) (2005) 4135–4195.
- [23] J. A. Cottrell, T. J. Hughes, Y. Bazilevs, *Isogeometric analysis: toward integration of CAD and FEA*, John Wiley & Sons, 2009.
- [24] E. McDonald, J. Pestana, A. Wathen, Preconditioning and iterative solution of all-at-once systems for evolutionary partial differential equations, *SIAM Journal on Scientific Computing* 40 (2) (2018) A1012–A1033.
- [25] A. Mantzaflaris, B. Jüttler, B. N. Khoromskij, U. Langer, Low rank tensor methods in Galerkin-based isogeometric analysis, *Computer Methods in Applied Mechanics and Engineering* 316 (2017) 1062–1085.
- [26] F. Scholz, A. Mantzaflaris, B. Jüttler, Partial tensor decomposition for decoupling isogeometric Galerkin discretizations, *Computer Methods in Applied Mechanics and Engineering* 336 (2018) 485–506.
- [27] B. W. Bader, T. G. Kolda, *Tensor Toolbox for MATLAB*, Version 3.6 (2023).
- [28] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2003.
- [29] D. Kressner, S. Massei, J. Zhu, Improved ParaDiag via low-rank updates and interpolation, *Numerische Mathematik* 155 (1-2) (2023) 175–209.
- [30] P. W. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: *Proceedings 35th annual symposium on foundations of computer science*, Ieee, 1994, pp. 124–134.
- [31] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM review* 41 (2) (1999) 303–332.
- [32] P. Kaye, R. Laflamme, M. Mosca, *An introduction to quantum computing*, OUP Oxford, 2006.
- [33] M. A. Nielsen, I. L. Chuang, *Quantum computation and quantum information*, Cambridge university press, 2010.
- [34] B. Kraus, J. I. Cirac, Optimal creation of entanglement using a two-qubit gate, *Physical Review A* 63 (6) (2001) 062309.
- [35] S. Balakrishnan, R. Sankaranarayanan, Entangling power and local invariants of two-qubit gates, *Physical Review A—Atomic, Molecular, and Optical Physics* 82 (3) (2010) 034301.
- [36] A. W. Harrow, A. Montanaro, Testing product states, quantum Merlin-Arthur games and tensor optimization, *Journal of the ACM (JACM)* 60 (1) (2013) 1–43.