# Polynomial-time algorithms for graph isomorphism and non-trivial graph automorphisms based on the exponential

Maurício Machado Galvão[1,2*]

Corresponding author(s). E-mail(s): mauriciomgalvao@gmail.com;

This paper presents new polynomial-time algorithms for determining graph isomorphisms, non-trivial graph automorphisms. Based on spectral properties and iterative refinement techniques, we introduce the algorithms `graphiso3` for determining graph isomorphisms, `graphautont3` for non-trivial graph automorphisms. These algorithms are based on matrix exponentials, These algorithms are based on the matrix exponential, but there are variation based on the inverse of normal matrices.

## 1 Notation

If $A$ is a real matrix, then $A^t$ denotes its transpose.

If $A^t = A$, we say that $A$ is *symmetric*.

If $A^t = -A$, we say that $A$ is *antisymmetric*.

For any real matrix $A$, there exist unique matrices

$$S_A = \frac{A + A^t}{2} \quad \text{and} \quad A_A = \frac{A - A^t}{2},$$

such that

$$A = S_A + A_A,$$

where $S_A$ is symmetric and $A_A$ is antisymmetric.

Let $e_i$ denote the $i$-th standard basis vector in $\mathbb{R}^n$, that is, a column vector with 1 in the $i$-th coordinate and 0 elsewhere. Note that $Ae_i = A_i$, the $i$-th column of $A$.

If $\sigma$ is a permutation of $\{1, \ldots, n\}$, the associated permutation matrix $P_\sigma$ satisfies:

$$P_\sigma e_i = e_{\sigma(i)} \quad \text{for all } i = 1, \ldots, n.$$

The Kronecker delta function is defined as:

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The identity matrix $I_n$ is the $n \times n$ matrix defined by $(I_n)_{i,j} = \delta_{i,j}$ for all $i, j = 1, \ldots, n$. When the dimension is clear from context, we simply write $I$.

If $A$ is an $n \times n$ matrix, the matrix exponential is defined as:

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}, \quad \text{where } A^0 := I.$$

Finally, for a vector $v$ and a matrix $A$, we define:

$$Z(A, v) = \text{span}\{v, Av, A^2v, A^3v, \ldots\},$$

that is, the linear span of the iterated images of $v$ under $A$. For more details, see [2].

# 2 Introduction

Due to the inherent limitations of floating-point arithmetic, all matrix equalities in this algorithm are interpreted as equalities up to a specified numerical tolerance. The default tolerance is set to $10^{-10}$, though the algorithm allows this threshold to be adjusted as necessary.

The algorithm is based on a sequence of linear algebraic properties of adjacency matrices and their spectral transformations, which are used to determine graph isomorphisms and to detect automorphisms. We begin by presenting the fundamental mathematical principles.

**Theorem 1** (Isomorphism Theorem)**.** *Let $G_1$ and $G_2$ be two graphs (possibly directed, with loops or multiple edges), and let $A_{G_1}$ and $A_{G_2}$ denote their respective adjacency matrices. Then $G_1$ is isomorphic to $G_2$ if and only if there exists a permutation matrix $P_\sigma$ such that*

$$A_{G_2} = P_\sigma A_{G_1} P_\sigma^t.$$

This reduces the graph isomorphism problem to that of matrix similarity under conjugation by permutation matrices.

From this point forward, $A$ and $B$ denote $n \times n$ real adjacency matrices, and $P_\sigma$ refers to the permutation matrix corresponding to $\sigma$, i.e., $P_\sigma e_i = e_{\sigma(i)}$ for $i = 1, \ldots, n$.

Note that $B = P_\sigma A P_\sigma^t$ if and only if, for all $i, j = 1, \ldots, n$, we have

$$B_{\sigma(i),\sigma(j)} = A_{i,j}.$$

Consequently,

$$\text{diag}(B) = P_\sigma \, \text{diag}(A),$$

where $\text{diag}(X)$ denotes the vector formed by the diagonal elements of matrix $X$.

Although the converse does not hold, this property enables an initial pruning step: if no permutation aligns the diagonals, then no isomorphism can exist.

The central idea of the algorithm is to identify where the action of a permutation matrix leaves detectable traces. The first such trace appears on the diagonal. From there, the traces can be followed column-by-column from $B$ to $A$. While these traces may not be visible in the raw adjacency matrices of simple graphs, they are preserved in matrix powers and, by extension, in the matrix exponential. Certain adjustments are required, but the core strategy is retained.

Moreover, if

$$B = P_\sigma A P_\sigma^t,$$

then for all $k \geq 1$,

$$B^k = P_\sigma A^k P_\sigma^t,$$

since $P_\sigma^t = P_\sigma^{-1}$. Consequently,

$$\operatorname{diag}(B^k) = P_\sigma \operatorname{diag}(A^k).$$

Now consider the matrix exponential:

$$e^A = \sum_{n=0}^{\infty} \frac{A^n}{n!},$$

which yields

$$e^B = P_\sigma e^A P_\sigma^t,$$

and hence,

$$\operatorname{diag}(e^B) = P_\sigma \operatorname{diag}(e^A).$$

If no permutation $P_\sigma$ satisfies $\operatorname{diag}(e^B) = P_\sigma \operatorname{diag}(e^A)$, then $A$ and $B$ are not isomorphic. To determine this, we rely on the following:

**Theorem 2** (Vector Sorting Theorem)**.** *Let $v_1$ and $v_2$ be vectors. There exists a permutation matrix $P_\sigma$ such that*

$$P_\sigma v_1 = v_2$$

*if and only if the sorted entries of $v_1$ and $v_2$ are identical.*

If $v_1$ contains repeated entries, with each distinct value $x_i$ occurring $k_i$ times for $i = 1, \ldots, s$, then the total number of such permutations satisfying $P_\sigma v_1 = v_2$ is $\prod_{i=1}^{s} k_i!$.

If there is no such $\sigma$ for which $\operatorname{diag}(e^B) = P_\sigma \operatorname{diag}(e^A)$, then no isomorphism exists between $A$ and $B$. Otherwise, we build an initial candidate mapping: for each index $i$ in $A$, we select indices $j$ in $B$ such that $\operatorname{diag}(e^B)(j) \approx \operatorname{diag}(e^A)(i)$. Any valid isomorphism must conform to this mapping.

An additional useful identity is:

$$B^k e_{\sigma(i)} = P_\sigma A^k e_i \quad \text{for all } k \geq 1.$$

By the definition of the matrix exponential:

$$e^B e_{\sigma(i)} = P_\sigma e^A e_i,$$

noting that $e^B e_{\sigma(i)} = (e^B)_{\sigma(i)}$ and $e^A e_i = (e^A)_i$, where $(M)_i$ denotes the $i$-th column of matrix $M$.

This identity holds for symmetric matrices $A$ and also for skew-symmetric matrices $A$ with $\|A\|_2 \leq \pi$.

Hence, any real matrix $M$ can be decomposed as

$$M = S_M + A_M,$$

where $S_M = \frac{M+M^t}{2}$ is the symmetric part, and $A_M = \frac{M-M^t}{2}$ is the skew-symmetric part.

Therefore,

$$B = P_\sigma A P_\sigma^t \iff S_B = P_\sigma S_A P_\sigma^t \quad \text{and} \quad A_B = P_\sigma A_A P_\sigma^t.$$

We decompose each $N = A, B$ as $N = S_N + A_N$ because these matrices are normal and satisfy $\ker(S_N) \cap \mathrm{Im}(S_N) = \ker(A_N) \cap \mathrm{Im}(A_N) = \{0\}$, which is crucial for the method.

We define:

$$EM = \begin{cases} n e^{\frac{M}{\|M\|}} & \text{if } \|M\| \not\approx 0, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } M = S_A, S_B, A_A, A_B.$$

Under these conditions:

$$S_B = P_\sigma S_A P_\sigma^t \iff ES_B = P_\sigma ES_A P_\sigma^t, A_B = P_\sigma A_A P_\sigma^t \iff EA_B = P_\sigma EA_A P_\sigma^t.$$

**Theorem 3** (Isomorphism Powers Theorem)**.** *Let $A$ and $B$ be $n \times n$ real matrices. Then $B = P_\sigma A P_\sigma^t$ for some permutation matrix $P_\sigma$ if and only if*

$$B^k e_{\sigma(i)} = P_\sigma A^k e_i$$

*for all integers $k \geq 1$ and all $i \in \{1, \ldots, n\}$.*

The Isomorphism Powers Theorem is the core of the test.

Given matrices $M$ and $M_1$, and an initial candidate mapping, we test whether $M_{i,i} \approx (M_1)_{j,j}$. If no match is found, we discard the candidate. Otherwise, for each $i_1 \neq i$, we seek $j_1 \neq j$ such that $M_{i_1,i} = (M_1)_{j_1,j}$. If this list is empty, we discard the candidate. If all $i_1$ find suitable $j_1$, the candidate remains viable.

We first test using $M = ES_A$ and $M_1 = ES_B$. If this test fails, the candidate is rejected. Otherwise, we continue the test using $M = EA_A$ and $M_1 = EA_B$ with the refined candidate map. A successful match in both confirms an isomorphism through $(i, j)$.

The initial diagonal filter is essential to prevent false positives, which may satisfy

$$(ES_B)_j = P_\sigma (ES_A)_i \quad \text{and} \quad (EA_B)_j = P_\sigma (EA_A)_i,$$

but fail to satisfy

$$\operatorname{diag}(ES_B) = P_\sigma \operatorname{diag}(ES_A), \quad \text{or} \quad \operatorname{diag}(EA_B) = P_\sigma \operatorname{diag}(EA_A),$$

thus invalidating the isomorphism.

We then proceed as follows:

For matrices $M$ and $M_1$ and an initial candidate mapping, we check whether $M_{i,i} \approx (M_1)_{j,j}$. If no such match exists, we return the empty map. Otherwise, we restrict the list of candidates to contain only $j$ for $i$.

Next, for each $i_1 \neq i$, we search for $j_1 \neq j$ such that $M_{i_1,i} \approx (M_1)_{j_1,j}$. If this set is empty for any $i_1$, the map is discarded. If all such $i_1$ yield at least one valid $j_1$, then a possible isomorphism may pass through $(i, j)$.

We first perform this test using $M = ES_A$ and $M_1 = ES_B$. If the resulting map is empty, we discard this candidate and move to the next $j$. Otherwise, we refine the test using $M = EA_A$ and $M_1 = EA_B$, providing the previously obtained map as a constraint. If this second test yields a non-empty map, then we have found a valid isomorphism through $(i, j)$.

The initial diagonal filtering is essential to avoid false positives. Without it, we might obtain a permutation $P_\sigma$ such that:

$$(ES_B)_j = P_\sigma (ES_A)_i \quad \text{and} \quad (EA_B)_j = P_\sigma (EA_A)_i,$$

but for which

$$\operatorname{diag}(ES_B) \neq P_\sigma \operatorname{diag}(ES_A) \quad \text{or} \quad \operatorname{diag}(EA_B) \neq P_\sigma \operatorname{diag}(EA_A),$$

which would invalidate the isomorphism.

We define that a test *fails* if it returns an empty map, and *does not fail* otherwise.

We iterate over all $i$ to build the candidate map. For each $i$, we collect indices $j$ such that both

$$(ES_A)_{i,i} \approx (ES_B)_{j,j} \quad \text{and} \quad (EA_A)_{i,i} \approx (EA_B)_{j,j}$$

hold. If any $i$ has an empty list, then isomorphism is impossible, and the algorithm returns a null permutation.

For each $i$ and each candidate $j$, we apply the described test. If the test fails, we discard the candidate and proceed to the next. If the test does not fail, we update the candidate map with the returned partial map and continue.

If, for any $i$, all candidate $j$ values are exhausted with the test failing in all cases, no isomorphism exists, and we return the null permutation.

Otherwise, we obtain a mapping where each $i$ corresponds to a unique $j$. We then convert this map into a permutation matrix and return it. This constitutes a valid isomorphism, provided the initial filtering is applied. Without the initial filter, false positives may occur, resulting in permutations $P_\sigma$ for which

$$(ES_B)_j = P_\sigma (ES_A)_i \quad \text{and} \quad (EA_B)_j = P_\sigma (EA_A)_i,$$

but

$$\operatorname{diag}(ES_B) \neq P_\sigma \operatorname{diag}(ES_A) \quad \text{or} \quad \operatorname{diag}(EA_B) \neq P_\sigma \operatorname{diag}(EA_A),$$

meaning that no valid isomorphism exists.

For detecting non-trivial automorphisms, we replace $ES_B$ and $EA_B$ by $ES_A$ and $EA_A$, respectively. For each $i$, we collect all $j \neq i$ such that

$$(ES_A)_{i,i} \approx (ES_A)_{j,j} \quad \text{and} \quad (EA_A)_{i,i} \approx (EA_A)_{j,j}.$$

We forbid identity mappings ($i \mapsto i$), and execute the algorithm as before. If a non-trivial automorphism is found, it is returned. Otherwise, we restore the candidate list and proceed.

As an alternative to the matrix exponential, we may use the following transformation:

$$IM = \left(I - \frac{M}{\|M\| + 1}\right)^{-1} = \sum_{k=0}^{\infty} \left(\frac{M}{\|M\| + 1}\right)^k,$$

where $M$ is any of $S_A, S_B, A_A, A_B$. This preserves the spectral structure required for the method and can be substituted with equivalent results.

# 3 Permutation Matrices and Graph Isomorphism

Before discussing graph isomorphism, we first review the formal definition and properties of permutation matrices.

## 3.1 Graph Isomorphism as Permutation Similarity

The entire algorithmic framework is based on the following fundamental theorem:

**Theorem 4** (Isomorphism Theorem)**.** *Let $G_1$ and $G_2$ be graphs (or digraphs, multigraphs, with or without loops), and let $A_{G_1}$ and $A_{G_2}$ denote their respective adjacency matrices. Then $G_1$ and $G_2$ are isomorphic if and only if there exists a permutation matrix $P_\sigma$ such that:*

$$A_{G_2} = P_\sigma A_{G_1} P_\sigma^t.$$

For reference, see [1].

**Theorem 5** (Permutation Similarity Entrywise Condition)**.** *Let $A, B \in \mathbb{R}^{n \times n}$ be square matrices. Then:*

$$B = P_\sigma A P_\sigma^t$$

*if and only if:*

$$B_{\sigma(i),\sigma(j)} = A_{i,j}, \quad \textit{for all } i, j = 1, \ldots, n.$$

*Proof.* By definition of permutation matrices:

$$P_\sigma e_i = e_{\sigma(i)} \quad \text{and} \quad e_i^t P_\sigma^t = e_{\sigma(i)}^t.$$

Thus, the following identities hold:

$$e_i = P_\sigma^t e_{\sigma(i)}, \quad e_i^t = e_{\sigma(i)}^t P_\sigma.$$

Now, observe that:

$$B_{\sigma(i),\sigma(j)} = e_{\sigma(i)}^t B e_{\sigma(j)} = e_{\sigma(i)}^t P_\sigma A P_\sigma^t e_{\sigma(j)}.$$

Using the identities above:

$$e_{\sigma(i)}^t P_\sigma = e_i^t, \quad P_\sigma^t e_{\sigma(j)} = e_j,$$

we get:

$$B_{\sigma(i),\sigma(j)} = e_i^t A e_j = A_{i,j}.$$

Therefore:

$$B_{\sigma(i),\sigma(j)} = A_{i,j}, \quad \forall i, j \in \{1, \ldots, n\}.$$

This establishes the equivalence. □

**Theorem 6** (Diagonal Permutation Theorem)**.** *Let $A, B \in \mathbb{R}^{n\times n}$ be matrices. If there exists a permutation matrix $P_\sigma$ such that:*

$$B = P_\sigma A P_\sigma^t,$$

*then their diagonal vectors satisfy:*

$$\operatorname{diag}(B) = P_\sigma \operatorname{diag}(A),$$

*where* $\operatorname{diag}(X)$ *denotes the vector in $\mathbb{R}^n$ whose $i$-th entry equals the $(i,i)$ entry of matrix $X$.*

*Proof.* The result follows directly from the entrywise property of permutation similarity, since for each $i$:

$$B_{\sigma(i),\sigma(i)} = A_{i,i}.$$

Thus, by definition of the action of $P_\sigma$ on vectors:

$$\operatorname{diag}(B) = P_\sigma \operatorname{diag}(A).$$

□

## 3.2 Permutation Similarity and Powers of Matrices

A key property of permutation similarity is its compatibility with matrix powers.

**Theorem 7** (Power Preservation under Permutation Similarity)**.** *Let $A, B \in \mathbb{R}^{n\times n}$ and let $P_\sigma$ be a permutation matrix such that:*

$$B = P_\sigma A P_\sigma^t.$$

*Then, for every integer $k \geq 1$:*

$$B^k = P_\sigma A^k P_\sigma^t,$$

*and consequently:*

$$\operatorname{diag}(B^k) = P_\sigma \operatorname{diag}(A^k),$$

*where* $\operatorname{diag}(X)$ *denotes the vector containing the diagonal entries of matrix* $X$*. And*

$$\operatorname{diag}(e^B) = P_\sigma \operatorname{diag}(e^A)$$

*Proof.* The result follows by induction on $k$, using the fact that multiplication of matrices is compatible with permutation similarity: As $P_\sigma^t = P_\sigma^{-1}$ we have

$$B^k = (P_\sigma A P_\sigma^t)^k = P_\sigma A^k P_\sigma^t,$$

for all $k \geq 1$. The property on the diagonal follows directly from the previous Diagonal Permutation Theorem.

$$e^B = \sum_{k=0}^{\infty} \frac{B^k}{k!} = \sum_{k=0}^{\infty} \frac{P_\sigma A^k P_\sigma^t}{k!} = \ P_\sigma (\sum_{k=0}^{\infty} \frac{A^k}{k!}) P_\sigma^t = P_\sigma e^A P_\sigma^t$$

Like this

$$diag(e^B) = P_\sigma diag(e^A)$$

□

This property is central to our algorithm, as it allows us to extract combinatorial invariants from the powers of the adjacency matrices. These invariants are used to restrict the search space by eliminating candidate vertex correspondences that violate necessary diagonal conditions for any $k$.

### 3.3 Vector Sorting and Permutation Existence

The following result formalizes a fundamental intuition used when comparing two multisets of values.

**Theorem 8** (Vector Sorting Theorem)**.** *Let $v_1, v_2 \in \mathbb{R}^n$. Then there exists a permutation matrix $P_\sigma$ such that:*

$$P_\sigma v_1 = v_2$$

*if and only if the sorted versions of $v_1$ and $v_2$ are identical, i.e., both vectors contain the same multiset of elements with the same multiplicities.*

*Proof.* If such a permutation $P_\sigma$ exists, then $v_2$ is a reordering of $v_1$, and both must contain the same elements, with the same frequencies. Thus, sorting both vectors will yield the same result.

Conversely, if the sorted vectors are equal, then $v_2$ can be obtained by permuting the elements of $v_1$, and hence there exists at least one permutation $P_\sigma$ such that $P_\sigma v_1 = v_2$.

Note that uniqueness of $\sigma$ is not guaranteed in the case where $v_1$ contains repeated elements. □

This principle will be used throughout the algorithm to reduce the candidate sets for possible isomorphisms.

Given that the elements of $v_1$ may repeat, the total number of distinct permutations $\sigma \in S_n$ satisfying $P_\sigma v_1 = v_2$ is given by:

$$\prod_{i=1}^{s} k_i!,$$

where $k_i$ denotes the number of times the distinct value $x_i$ appears in $v_1$, and $s$ is the number of distinct values in $v_1$.

### 3.4 Isomorphism Preservation under Matrix Decomposition

A further useful property relates permutation similarity to matrix decompositions into symmetric and antisymmetric parts.

**Theorem 9** (Isomorphism Decomposition Theorem)**.** *Let $A, B \in \mathbb{R}^{n\times n}$, and let $P_\sigma$ be a permutation matrix. Let $S_M$ and $A_M$ denote the symmetric and antisymmetric components of any square matrix $M$, respectively, i.e., for $M = A, B$:*

$$M = S_M + A_M,$$

*where:*

$$S_M = \frac{1}{2}(M + M^t), \quad A_M = \frac{1}{2}(M - M^t).$$

*Then:*

$$B = P_\sigma A P_\sigma^t$$

*if and only if:*

$$S_B = P_\sigma S_A P_\sigma^t \quad \textit{and} \quad A_B = P_\sigma A_A P_\sigma^t.$$

*Proof.* Suppose $B = P_\sigma A P_\sigma^t$. Then, substituting the decomposition of $A$:

$$B = P_\sigma(S_A + A_A)P_\sigma^t = P_\sigma S_A P_\sigma^t + P_\sigma A_A P_\sigma^t.$$

Since permutation similarity preserves symmetry type (i.e., symmetric matrices map to symmetric matrices, and antisymmetric to antisymmetric), it follows that:

$$S_B = P_\sigma S_A P_\sigma^t, \quad A_B = P_\sigma A_A P_\sigma^t.$$

Conversely, if the two conditions hold, then:

$$B = S_B + A_B = P_\sigma S_A P_\sigma^t + P_\sigma A_A P_\sigma^t = P_\sigma(S_A + A_A)P_\sigma^t = P_\sigma A P_\sigma^t.$$

Thus, the equivalence is established. □

This decomposition-based criterion further assists in splitting the graph isomorphism problem into symmetric and antisymmetric substructures, which can improve both theoretical analysis and computational efficiency. such as $\|A\|_2 \leq 1$ then $A = log(e^A)$

Now we define $EM = \begin{cases} ne^{\frac{M}{\|M\|}} & \text{if} \|M\| \not\approx 0, \\ 0 & \text{otherwise.} \end{cases}$ $M = S_A, S_B, A_A, A_B$

**Theorem 10** (Isomorphism Powers Theorem)**.** *Let $A$ and $B$ be $n \times n$ real matrices. Then $B = P_\sigma A P_\sigma^t$ for some permutation matrix $P_\sigma$ (associated with permutation $\sigma$) if and only if there exists a permutation matrix $P_\sigma$ such that, for all integers $k \geq 1$ and for all $i \in \{1, \ldots, n\}$, the following holds:*

$$B^k e_{\sigma(i)} = P_\sigma A^k e_i,$$

*where $e_i$ denotes the $i$-th standard basis vector.*

*Proof.* ($\Rightarrow$) Assume that $B = P_\sigma A P_\sigma^t$.

We know that for any $k \geq 1$,

$$B^k = \left(P_\sigma A P_\sigma^t\right)^k = P_\sigma A^k P_\sigma^t,$$

by repeated application of the permutation similarity property.

Then, for any $i \in \{1, \ldots, n\}$,

$$B^k e_{\sigma(i)} = P_\sigma A^k P_\sigma^t e_{\sigma(i)}.$$

Since $P_\sigma e_i = e_{\sigma(i)}$ then $e_i = P_\sigma^t P_\sigma e_i = P_\sigma^t e_{\sigma(i)}$, this becomes:

$$B^k e_{\sigma(i)} = P_\sigma A^k P_\sigma^t e_{\sigma(i)} = P_\sigma A^k e_i.$$

This proves the forward direction.

($\Leftarrow$) Conversely, suppose there exists a permutation matrix $P_\sigma$ such that:

$$B^k e_{\sigma(i)} = P_\sigma A^k e_i$$

for all $k \geq 1$ and all $i \in \{1, \ldots, n\}$.

In particular, for $k = 1$, we have:

$$B e_{\sigma(i)} = P_\sigma A e_i.$$

Using the property $P_\sigma e_i = e_{\sigma(i)}$, this can be rewritten as:

$$B P_\sigma e_i = P_\sigma A e_i,$$

which implies:

$$B P_\sigma = P_\sigma A.$$

Multiplying both sides on the right by $P_\sigma^t$, we obtain:

$$B = P_\sigma A P_\sigma^t.$$

Therefore, $A$ and $B$ are permutation similar.

□

By combining the previous theorems, we observe that the key difference between the case where $B = P_\sigma A P_\sigma^t$ and a more general similarity $B = RAR^{-1}$ (for some invertible matrix $R$ that is not a permutation matrix) is that the first preserves the entries of $A$ while the second generally does not, except in very rare circumstances.

This property was used to develop a test to determine whether an isomorphism passes through positions $i$ and $j$. This can be challenging to detect when $A$ and $B$ contain only zeros and ones, which is why we use powers of the adjacency matrix to expose structural differences. These are typically the most difficult cases when working with integer-entry matrices.

For automorphisms, this situation occurs only if the matrix $R$ belongs to the commutator set of $A$, that is, the set of matrices satisfying $AR = RA$. In the general isomorphism case, the condition becomes $BR = RA$. Of course, if $A$ is a scalar multiple of the identity, then $B$ must be the same scalar multiple, and $R$ can be any invertible matrix.

Originally, I tested all powers of $A$ and $B$, but later observed that the matrix exponentials $e^A$ and $e^B$ condense all this information. This led to the following result:

Finally, observe that if $A$ is antisymmetric, then all eigenvalues of $A$ are purely imaginary (of the form $ir$, with $r \in \mathbb{R}$), and thus $e^A$ will be an orthogonal matrix $O$. For antisymmetric $A$ with $\|A\|_2 \leq 1$, we have that all eigenvalues satisfy $|v_i| \leq 1 \leq \pi$, ensuring that:

$$\log(e^A) = A.$$

**Theorem 11** (Exponential of Permutations Theorem (Symmetric Case))**.** *Let $A$ and $B$ be nonzero symmetric matrices with $\|A\| = \|B\|$ (where $\|\cdot\|$ denotes either the 2-norm or the Frobenius norm). Then:*

$$B = P_\sigma A P_\sigma^t$$

*if and only if*

$$EB = ne^{\frac{B}{\|B\|}}, \quad EA = ne^{\frac{A}{\|A\|}}, \quad \textit{and} \quad ES_B = P_\sigma ES_A P_\sigma^t,$$

*where $ES_M$ denotes the scaled matrix exponential.*

*Proof.* Since $\|A\| = \|B\| > 0$, let us define:

$$A_1 = \frac{A}{\|A\|}, \quad B_1 = \frac{B}{\|B\|}.$$

If $B = P_\sigma A P_\sigma^t$, then:

$$B_1 = P_\sigma A_1 P_\sigma^t.$$

By properties of matrix exponentials:

$$e^{B_1} = \sum_{k=0}^{\infty} \frac{P_\sigma A_1^k P_\sigma^t}{k!} = P_\sigma e^{A_1} P_\sigma^t,$$

and therefore:

$$ne^{B_1} = P_\sigma (ne^{A_1}) P_\sigma^t.$$

For the reverse direction, using the fact that $A_1$ and $B_1$ are symmetric, the matrix logarithm satisfies $\log(e^{A_1}) = A_1$, giving:

$$B_1 = \log(e^{B_1}) = log(P_\sigma e_1^A P_\sigma^t) = P_\sigma log(e^{A_1}) P_\sigma^t = P_\sigma A_1 P_\sigma^t,$$

which implies:

$$B = P_\sigma A P_\sigma^t,$$

since $\|A\| = \|B\|$. □

For more details about $Z(A, v) = span\{k \geq 0, A^k v\}$, see [2].

**Theorem 12** (Persistence theorem)**.** *Let $A$ and $B$ be matrices. Suppose there exist vectors $e_i$ (a column vector of the standard basis) and $e_j$ such that for all $k \geq 1$:*

$$B^k e_j = P_\sigma A^k e_i = P_\sigma A^k P_\sigma^t e_j$$

*for some permutation matrix $P_\sigma$. Then:*

*1. The cyclic subspaces coincide:*

$$Z(B, e_j) = Z(P_\sigma A P_\sigma^t, e_j),$$

*and $P_\sigma A P_\sigma^t$ acts as $B$ on $Z(B, e_j)$, that is,*

$$P_\sigma A P_\sigma^t v = Bv, \quad \forall v \in Z(B, e_j).$$

*2. Moreover, the mapping $P_\sigma$ induces an isomorphism between the cyclic subspaces:*

$$B|_{Z(B,e_j}} = P_\sigma A|_{Z(A,e_i)} P_\sigma^t$$

*4. If $e_j$ is a cyclic vector for $B$ (i.e., $Z(B, e_j)$ is the whole space), then*

$$B = P_\sigma A P_\sigma^t.$$

*Proof.* From the hypothesis that for all $k$,

$$B^k e_j = P_\sigma A^k e_i = P_\sigma A^k P_\sigma^t e_j,$$

it follows that the cyclic subspaces generated by $e_j$ under $B$ and by $e_i$ under $A$ are related by conjugation with $P_\sigma$:

$$Z(B, e_j) = Z(P_\sigma A P_\sigma^t, e_j).$$

Since every vector $v \in Z(B, e_j)$ can be written as $v = p(B)e_j$ for some polynomial $p$, and polynomial functions of $B$ act as:

$$p(B)e_j = p(P_\sigma A P_\sigma^t)e_j = P_\sigma p(A) P_\sigma^t e_j = P_\sigma p(A) e_i,$$

it follows that $P_\sigma$ maps $Z(A, e_i)$ isomorphically onto $Z(B, e_j)$.

Moreover, the action of $B$ on $Z(B, e_j)$ corresponds to the conjugated action of $A$:

$$Bv = Bp(B)e_j = p_1(B)e_j = p_1(P_\sigma A P_\sigma^t)e_j = P_\sigma A P_\sigma^t p(B)e_j = P_\sigma A P_\sigma^t v.$$

Since $e_j$ is the standard basis vector corresponding to the index $j$, and $P_\sigma e_i = e_{\sigma(i)}$. Finally, if $e_j$ is a cyclic vector for $B$, then $Z(B, e_j)$ is the whole space, and the above equality extends to the entire space, implying

$$B = P_\sigma A P_\sigma^t.$$

□

# 4 Algorithm

## 4.1 Verify isomorphism: `verIsoTeste`

**verIso:** This function performs local consistency refinement on the candidate map.

***Input:***

- `map`: A current mapping where each vertex $i$ in graph $A$ is associated with a list of candidate vertices $j$ in graph $B$.
- $A$,$B$: Duas matrizes $S_A$ e $S_B$ ou $A_A$ e $A_B$ o
- $i, j$: The specific vertex pair currently under consideration, where $i$ is a vertex of $A$ and $j$ is a candidate vertex in $B$.

***Procedure:***

1. First, check if the diagonal entries match within tolerance:

$$A_{i,i} \approx B_{j,j}$$

   If this condition fails, the function immediately returns an empty map, signaling that no isomorphism is possible via this pairing.

   includes in the map i with only j in the list of possibilities.

   For all $i1 \neq i$ filters the list of candidates $j1 \neq j$ such that $A_{i1,i} \approx B_{j1,j}$ and $A_{i1,i1} \approx B_{j1,j1}$

   if filtered list is empty the function immediately returns an empty map, signaling that no isomorphism is possible via this pairing.

   otherwise put i1 in the return map with filtered list.
2. After completing all iterations over returns the possibly reduced map, since at least in i there are no other elements distinct from j.

***Return:***

## 4.2 Internal Consistency Test: `internalTest`

**Internal Test:** Given:

- `map`: A current mapping pos where each vertex $i$ in graph $A$ is associated with a list of candidate vertices $j$ in graph $B$.
- $S_A, A_A, S_B, S_A$
- The current index $i$.

***Procedure:***

1. Make a working copy of the map:

$$c2 \leftarrow \texttt{map}.$$

2. For each candidate $j$ in `map[`$i$`]`:

(a)

$$c2 \leftarrow \texttt{verij}(map, S_A, S_B, i, j)$$

(b) If $c2$ remains non-empty this indicates that a consistent mapping exists for this candidate at index $i$.

(i)

$$c2 \leftarrow \texttt{verij}(c2, A_A, A_B, i, j)$$

(ii) If $c2$ remains non-empty return $c2$.

3. If all candidates for $i$ have been tested and none yielded a non-empty $c2$, return the current (empty) c2, indicating that no isomorphism is possible given the current state.

***Return:***

## 4.3 Candidate Map Refinement: `graphIsoPartFinal`

**graphIsoPartFinal:** Given as input:

- A candidate `map`, where each vertex $i$ of graph $A$ is associated with a list of possible corresponding vertices $j$ in graph $B$.
- The matrices $ES_A$, $EA_A$, $ES_B$, and $EA_B$, $A$ e $B$.

***Procedure:***

We iterate over each vertex index:

$$i = 1, \ldots, n.$$

For each $i$:

1. $c2 \leftarrow \texttt{internalTest}(map, S_A, A_A, S_B, A_B, i)$
2. if $c2$ is empty returns null since there is no isomorphism.
3. Otherwise, `map` $\leftarrow c2$.

If all the indices were processed the map of possibilities for each i the list of possibilities has exactly one j, so we transform this into a permutation vector and return.

***Termination:***

### 4.4 Isomorphism Detection

**graphIso3:** This method determines if two graphs are isomorphic. input:
ajacency matrices A, B

***Procedure:***

First, it checks that:
it computes $S_A$,$A_A$,$S_B$,$A_B$

$$\|S_A\| \approx \|S_B\| \quad \text{and} \quad \|A_A\| \approx \|A_B\|.$$

If this fails, it returns null.

Next, it computes $ES_A$, $EA_A$, $ES_B$, and $EA_B$. It initializes the candidate map for every i index of A we associate the list of j of B, such as $(ES_A)_{i,i} \approx (ES_B)_{j,j}$ and $EA_A)_{i,i} \approx (EA_B)_{j,j}$, if for any i the set of $j$.

If any of the $i$ is left empty, we return the null permutation, indicating the non-existence of an automorphism.

Return's the calls `graphIsoPartFinal`$(pos, ES_A, EA_A, ES_B, EA_B)$

# 5 Test theorems, correctness and complexity of the graphiso algorithm

Just to understand the test.

In fact, the test is as follows: if for the $i$-th column of $A_i = S_A$ (or $A_A$) and the $j$-th column of $B_j = S_B$ (or $A_B$) and I test for all $k$, whether there is

$$B^k e_j = P_\sigma A^k e_i,$$

as I don't have $P_\sigma$, I test if, in the positions where I know possible isomorphisms could occur,

$$(A^k)_{i_1,i} = (B^k)_{j_1,j},$$

for some $i_1$, and keep only those that pass the test in the list. If the list is empty for some $i_1$, then there is the possibility that there is no permutation matrix $P_\sigma$ such that

$$B^k e_j = P_\sigma A^k e_i,$$

which means the test failed.

When we calculate $ES_A$, $ES_B$, $EA_A$, and $EA_B$, it is the same test but condensed.

In fact, this says whether there is a partial isomorphism connecting $i$ to $j$, because if this happens, the matrices satisfy

$$B = P_\sigma A P_\sigma^t$$

on the entire cyclic space $Z(B, e_j)$, which basically is the universe that, starting from $e_j$, the matrix $B$ can reach.

Before talking about the test failing on one matrix, it can fail on $S_A$ or $A_A$ or both, and that is what the test theorems are about: the failure in one or both parts.

**Theorem 13** (Test Theorem - Part 1)**.** *If the test fails on $i$ and $j$, then there is no isomorphism passing through $i$ and $j$.*

*Proof.* if the test fail to $S_A$ $S_B$ this mean there is no isomorphism transforming

$$Z(S_A, e_i) \to Z(S_B, e_j),$$

or

if the test fail to $EA_A$ $EA_B$ this mean there is no isomorphism transforming

$$Z(A_A, e_i) \to Z(A_B, e_j),$$

so no isomorphism exists there. □

The converse may not be true. A simple example is to take two isomorphic graphs $A$ and $B$ and two non-isomorphic graphs $C$ and $D$ that can be tested. Construct the graphs

$$A1 = \begin{pmatrix} A & 0 \\ 0 & C \end{pmatrix} \quad \text{and} \quad B1 = \begin{pmatrix} B & 0 \\ 0 & D \end{pmatrix}.$$

. Existem casos onde For all columns of $A$, the test passes, but when we get to $C$, it fails. What guarantees this works is that if the test works for a set of $n$ columns, we find the isomorphism.

**Proposition 14** (Corollary)**.** *When the test fails in all attempts to match column $i$, there is no isomorphism.*

*Proof.* By the Test Theorem Part 1, there is no isomorphism for this column of $A$ within the remaining possibilities, so the isomorphism does not exist. □

**Proposition 15** (Lemma 1)**.** *If $A$ is a symmetric or antisymmetric matrix, there exist $i$ and $j$ such that*

$$A^k e_i = A^k e_j$$

*for all $k \geq 1$. If $\sigma$ is a two-cycle swapping $i$ and $j$, then*

$$A = P_\sigma A P_\sigma^t.$$

*Proof.* Since

$$Ae_i = Ae_j,$$

we have

$$e_i^t A = e_j^t A.$$

Interchanging rows $i$ and $j$, then columns $i$ and $j$, returns the same matrix since $A$ is symmetric. This also holds for the rows of $A$.

In the antisymmetric case, since

and similarly for $j$, and since $Ae_i = Ae_j$, we have

$$e_i^t A = e_j^t A.$$

it follows that exchanging rows and columns $i$ and $j$ also returns the same matrix. Hence, the same automorphism applies. □

**Theorem 16** (Test Theorem – Part 2)**.** *If $A$ and $B$ are isomorphic, then there exists an isomorphism $\sigma$ such that $\sigma(i) = j$ if and only if the test does not fail on $i$ and $j$.*

*Proof.* Assume there exists an isomorphism $\sigma\colon V(A) \to V(B)$ such that $\sigma(i) = j$. Since $B = P_\sigma A P_\sigma^t$, it follows that

$$B^k e_j = B^k e_{\sigma(i)} = P_\sigma A^k P_\sigma^t e_{\sigma(i)} = P_\sigma A^k e_i,$$

so for all $k$,

$$A_B^k e_j = P_\sigma A_A^k e_i, \quad S_B^k e_j = P_\sigma S_A^k e_i.$$

As a result,

$$EA_B e_j = P_\sigma EA_A e_i, \quad ES_B e_j = P_\sigma ES_A e_i,$$

Thus for all $i1 \neq i$

$$(EA_B)_{\sigma(i1),j} = (EA_A)_{i1,i} \quad (ES_B)_{\sigma(i1),j} = (ES_A)_{i1,i},$$

and by taking diagonals,

$$(EA_B)_{j,j} = (EA_A)_{i,i}, \quad (ES_B)_{j,j} = (ES_A)_{i,i}.$$

Therefore, the test does not fail for the pair $(i, j)$, since all structural and diagonal conditions are preserved under the isomorphism.

Conversely, suppose the test fails for all if isomorphism $\sigma$ of A in B we have $\sigma(i) \neq j$. This implies that at least one of the following conditions fails:

- $(EA_A)_{i,i} \approx (EA_B)_{j,j}$,
- $(ES_A)_{i,i} \approx (ES_B)_{j,j}$,
- $EA_B e_j = P_\sigma EA_A e_i$,
- $ES_B e_j = P_\sigma ES_A e_i$.

The initial filter selects as candidates for $\sigma(i) = j$ only those vertices $j$ such that

$$(EA_A)_{i,i} \approx (EA_B)_{j,j} \quad \text{and} \quad (ES_A)_{i,i} \approx (ES_B)_{j,j},$$

within a given numerical tolerance. This step prevents false positives: even if a permutation $P_\sigma$ satisfies

$$EA_B e_j = P_\sigma EA_A e_i \quad \text{and} \quad ES_B e_j = P_\sigma ES_A e_i,$$

it must also satisfy

$$\operatorname{diag}(EA_B) = P_\sigma \operatorname{diag}(EA_A), \quad \operatorname{diag}(ES_B) = P_\sigma \operatorname{diag}(ES_A).$$

If these conditions are violated, then $P_\sigma$ is not an isomorphism between $EA_A$ and $EA_B$, or between $ES_A$ and $ES_B$, and thus cannot be an isomorphism between $A$ and $B$.

Moreover, if $A_B = P_\sigma A_A P_\sigma^t$ and $S_B = P_\sigma S_A P_\sigma^t$, then by Lemma 1, exchanging the roles of $j$ and $\sigma(i)$ would define an automorphism of $B$, contradicting the assumption that no such $\sigma$ passes through $(i, j)$. Therefore, for every such $\sigma$,

$$S_B e_j \neq P_\sigma S_A e_i \quad \text{or} \quad A_B e_j \neq P_\sigma A_A e_i.$$

Since both $A_B$ and $S_B$ are normal matrices, we have

$$\ker(A_B) \cap \mathrm{Im}(A_B) = \{0\}, \quad \ker(S_B) \cap \mathrm{Im}(S_B) = \{0\},$$

so

$$A_B^k(e_j - e_{\sigma(i)}) \neq 0 \quad \text{or} \quad S_B^k(e_j - e_{\sigma(i)}) \neq 0$$

for some $k$. Consequently,

$$EA_B(e_j - e_{\sigma(i)}) \neq 0 \quad \text{or} \quad ES_B(e_j - e_{\sigma(i)}) \neq 0,$$

and the test fails for $(i, j)$ for all isomorphisms $\sigma$ from $A$ to $B$. This completes the proof. □

If applied directly to the matrices $S_A$ and $S_B$, $A_A$ and $A_B$ this does not guarantee the existence of isomorphism between the respective cyclic spaces, or not.

Note that for two columns $A_i$ and $B_j$ of a simple graph or digraph, if the number of zeros in $A_i$ equals the number of zeros in $B_j$, it is always possible to find a permutation transforming one into the other. But if this does not hold across powers, then no isomorphism passes through there. This is very common in regular graphs and tends to be the most complex case for detecting isomorphisms. That's where we started and arrived on this path. There are cases where this also happens for power 2, but if the powers do not coincide up to the degree of the minimal polynomial of $A$, there will be a power where they differ by the persistence theorem.

In fact, two isomorphic matrices work as mirrors of each other, but in simple graphs and digraphs where you do not know the correspondences, you cause movements in both matrices and observe the "scene," then you can understand which points move identically even under the "crooked mirror" of permutation. In this case, the "movement" is the powers of the matrices.

Now, let's demonstrate the correctness of the algorithm.

**Theorem 17** (Correctness of the graphiso3 Algorithm)**.** *The* `graphiso3` *algorithm returns an isomorphism, or otherwise indicates no isomorphism exists. That is, given two matrices $A$ and $B$, it finds a permutation matrix $P_\sigma$ such that*

$$B = P_\sigma A P_\sigma^t,$$

*or returns the null permutation.*

*Proof.* 1. If $\|S_A\| \neq \|S_B\|$ or $\|A_A\| \neq \|A_B\|$, the isomorphism does not exist, because the norms used are invariant under conjugation by permutation matrices.
2. If for some i of A list of j such $(ES_A)_{i,i} \approx (ES_B)_{j,j}$ and $(EA_A)_{i,i} \approx (EA_B)_{j,j}$ there is no automorphism of $ES_A$ in $ES_B$, or of $EA_A$ in $EA_B$, then there is no isomorphism of $S_A$ in $S_B$ or $A_A$ in $A_B$, thus there is no automorphism of $A$ in $B$.
3. If for any $i$, we cannot find a valid match, according to the test theorem and its corollaries, there is no isomorphism between $ES_A$ and $ES_B$ or $EA_A$ and $EA_B$, so no isomorphism between $A$ and $B$.
4. Otherwise, the remaining isomorphism has been tested in all columns of our matrices. We have

$$ES_B = P_\sigma ES_A P_\sigma^t, \quad EA_B = P_\sigma EA_A P_\sigma^t,$$

thus

$$S_B = P_\sigma S_A P_\sigma^t, \quad A_B = P_\sigma A_A P_\sigma^t,$$

and

$$B = P_\sigma A P_\sigma^t,$$

in the algorithm according to the Graph Isomorphism Decomposition Theorem.

□

**Theorem 18.** *The* `graphiso3` *algorithm has time complexity* $\mathcal{O}(n^4)$ *and space complexity* $\mathcal{O}(n^2)$.

*Proof.* Computing the matrices $S_A$, $S_B$, $A_A$, and $A_B$ requires $4n^2$ operations.

Calculating the Frobenius norms of these matrices also takes $4n^2$ operations.

Computing the matrices $ES_A$, $ES_B$, $EA_A$, and $EA_B$ involves matrix exponentials, with total time less than:

$$4n^2 + 4n^3 \leq 8n^3.$$

To create the map of i and list of j such that $(ES_A)i, i = (ES_B)_{j,j}$ and $(EA_A)_{i,i} = (EA_B)_{j,j}$ $2n^2$

Within the main loop of the algorithm, at most $n^2$ tests are required for each pair of vectors. For each column $i$, we test against all $j$ not yet mapped. In the worst case, this yields:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad \text{tests.}$$

Each test may cost up to $2n^2$ operations (for the symmetric and antisymmetric components), resulting in:

$$\frac{n(n+1)}{2} \cdot 2n^2 = n^3(n+1) \leq 2n^4.$$

Summing all contributions:

$$2n^4 + 8n^3 + 8n^2 + 2n^2 \leq 20n^4,$$

so the total time complexity is $\mathcal{O}(n^4)$.

For memory usage, we store the matrices $A$, $S_A$, $A_A$, $ES_A$, and $EA_A$, and similarly five matrices for $B$, totaling $10n^2$ entries.

Additionally, the mapping structures may store up to $n(n+1)$ elements. At most three such maps are needed concurrently, so this adds at most:

$$3n(n+1) \leq 6n^2$$

entries, giving a total of $16n^2$, hence the space complexity is $\mathcal{O}(n^2)$. □

**Note:** If we replace the exponential transformation by the alternative:

$$IM = \left(I - \frac{M}{\|M\|+1}\right)^{-1} = \sum_{i=0}^{\infty} \left(\frac{M}{\|M\|+1}\right)^{i},$$

with $M \in \{S_A, S_B, A_A, A_B\}$, the algorithm remains correct and the overall time and space complexity are unchanged.

# 6 Non-trivial Automorphisms

The `graphautont` algorithm is described in the introduction.

## 6.1 Non-trivial graph automorphism detection-Algorithm

**graphautont:** This method searches for non-trivial automorphisms of a single graph $A$. Given as input:
a matriz A

***Procedure:***

1. creates IsoProblem pro with A
2. creates the map pos where for each index i of A we have the j of A such that $(S_A)_{i,i} \approx (S_A)_{j,j}$
3. for i=1..n
   (a) if the list of j associated with i has more than one element (someone distinct from i)
      (i) replaces the list at i with the same list without the i
      (ii) does $p \leftarrow graphIsoPartFinal(pos, S_A, A_A, S_A, A_A)$ if p is not a null permutation returns p
      (iii) restores the original list at i
4. returns a null permutation.

**Theorem 19** (Correctness Theorem of the `graphautont` Algorithm)**.** *The `graphautont3` algorithm returns an isomorphism that is a non-trivial automorphism, since some i does not map to itself. It returns the empty permutation if no non-trivial automorphism exists.*

*Proof.* If a non-empty permutation is returned, it corresponds to a non-trivial automorphism, by the same reasoning as in the correctness theorem of `graphiso`, and because there exists some $i$ such that $i$ does not map to itself.

If no automorphism is found, then no non-trivial automorphism exists, since for every $i$, if $i$ does not map to itself, there is no isomorphism. □

**Theorem 20.** *The `graphautont` algorithm has time complexity $\mathcal{O}(n^5)$ and space complexity $\mathcal{O}(n^2)$.*

*Proof.* In the initial phase, the complexity is $\mathcal{O}(11n^3)$, since all operations are performed on only half of the matrices used in `graphiso3`. Computing the matrices $S_A$, $A_A$ requires $2n^2$ operations.

Computing the matrices $ES_A$, $EA_A$, involves matrix exponentials, with total time less than:

$$2n^2 + 2n^3 \leq 4n^3.$$

To create the map of i and list of j such that $(ES_A)i, i = (ES_B)_{j,j}$ and $(EA_A)_{i,i} = (EA_B)_{j,j}$ $2n^2$

In the worst case, index $i$ is removed from the candidate list up to $n$ times. For each removal, the algorithm may invoke `graphIsoPartFinal`, which has complexity $\mathcal{O}(2n^4)$. Thus, the total worst-case time is:

$$n \cdot 2n^4 = 2n^5,$$

thus

$$2n^5 + 2n^2 + 4n^3 + 2n^2 \leq 10n^5.$$

Hence, the overall time complexity is $\mathcal{O}(n^5)$.

Regarding memory, the algorithm uses approximately half the number of matrices compared to `graphiso3`, saving five matrices. However, it adds one map to manage the list of forbidden identities. Therefore, the total memory usage is approximately:

$$16n^2 - 5n^2 + n(n+1) \leq 13n^2,$$

which gives space complexity $\mathcal{O}(n^2)$. □

# Use of Generative Artificial Intelligence Tools

Throughout the development of this work, I made extensive use of generative artificial intelligence tools, specifically OpenAI's ChatGPT, both for text drafting and language refinement.

There are several reasons that justify this choice:

- I am dyslexic, and therefore it is always necessary to carefully review and improve my written texts to avoid unintentional spelling, grammar, or structural mistakes.
- I am not fluent in English, which makes it difficult to write in a clear, precise, and formally appropriate style for scientific publications.
- My ability to produce aesthetically satisfactory documents and diagrams is limited, making AI assistance helpful in improving the presentation quality.

The primary generative AI tool used was ChatGPT (OpenAI, model `gpt-4o`, accessed via `chat.openai.com` between May and July 2025).

All mathematical definitions, algorithmic descriptions, and proofs presented in this article were conceived, designed, and validated by myself. The AI was used exclusively for language correction, formatting suggestions, clarity improvements, and LaTeX formatting support.

# References


[1] Godsil, C., Royle, G.: Algebraic Graph Theory. Springer, New York : (c2001.). http://www.loc.gov/catdir/enhancements/fy0816/00053776-d.html

[2] Hoffman, K., Kunze, R.A.: Linear Algebra. Prentice-Hall, ??? (1971). https://books.google.com.br/books?id=I4kQAQAAIAAJ