

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

Maurício Machado Galvão<sup>1,2\*</sup>

Corresponding author(s). E-mail(s): [mauriciomgalvao@gmail.com](mailto: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 `graphiso` for determining graph isomorphisms, `graphautont` for non-trivial graph automorphisms.

## 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  $\{1, \dots, n\}$ , the associated permutation matrix  $P_\sigma$  satisfies:

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

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  $i, j = 1, \dots, n$ . 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:

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

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 given numerical tolerance. The default tolerance is set to  $10^{-10}$ , although the algorithm permits this value to be adjusted as needed.

The algorithm is founded on a sequence of linear algebraic properties of adjacency matrices and their spectral transformations, which are employed to determine graph isomorphism and to detect automorphisms. We begin by presenting the principal mathematical foundations.

**Theorem 1** (Isomorphism Theorem). *Let  $G_1$  and  $G_2$  be two graphs (or digraphs, multigraphs, with or without loops), and let  $A_{G_1}$  and  $A_{G_2}$  denote their respective adjacency matrices. Then,  $G_1$  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 a matrix similarity problem under permutation conjugation.

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

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

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

. Therefore:

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

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 exists.

The core idea of the algorithm is to track where the action of a permutation matrix leaves observable traces. The first such trace appears on the diagonal. From there, these traces can be followed column by column from  $B$  to  $A$ . While they 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. Some adjustments are necessary, but the core approach is preserved.

Furthermore, 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:

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

Now consider the matrix exponential:

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

which implies:

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

and hence:

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

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

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

$$P_\sigma v_1 = v_2$$

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

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

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

Another useful observation is that:

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

implies that for all  $i = 1, \dots, n$  and all  $k \geq 1$ , we have:

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

By the matrix exponential definition:

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

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

This identity holds for symmetric matrices  $A$ , and also for antisymmetric matrices  $A$  satisfying  $\|A\|_2 \leq \pi$ .

Hence, any real matrix  $M$  may 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 antisymmetric 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  $N = S_N + A_N$  for  $N = A, B$  since these matrices are normal and satisfy  $\ker(S_N) \cap \text{Im}(S_N) = \ker(A_N) \cap \text{Im}(A_N) = \{0\}$ , which is essential for the method.

We define:

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

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 [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  $i \in \{1, \dots, n\}$ , the following holds:

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

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

The Isomorphism Powers Theorem is the heart of the test, in it we test if the selected candidates  $(ES_A)_{i,i} \approx (ES_B)_{j,j}$  are valid for the powers for  $i1 \neq i$  and every candidate  $j1 \neq j$  we are left with only  $(S_A^k)_{i1,i} = (S_B^k)_{i1,i}$  and  $(A_A^k)_{i1,i} = (S_B^k)_{i1,i}$ , or equivalently in  $(ES_A)_{i1,i} \approx (ES_B)_{j1,j}$  and  $(EA_A)_{i1,j} \approx (EA_B)_{j1,j}$ , if any list of  $i1$  for some  $i1$ , or if we have  $(ES_A)_{i,i} \not\approx (ES_B)_{j,j}$  and  $(EA_A)_{i,i} \not\approx (EA_B)_{j,j}$ , there is no isomorphism passing through  $i$  and  $j$ , in another case there is at least one partial isomorphism, and  $A$  and  $B$  are isomorphic surely there is an isomorphism such that  $\sigma(i) = j$ .

We first verify  $\|S_A\| \approx \|S_B\|$  and  $\|A_A\| \approx \|A_B\|$ . It is crucial to employ norms that are invariant under permutations. In this case, we use the Frobenius norm, although the spectral norm (2-norm) is also acceptable.

We then check whether there exists  $P_\sigma$  such that  $\text{diag}(ES_A) = P_\sigma \text{diag}(ES_B)$ , by comparing the sorted vectors  $\text{diag}(ES_A)$  and  $\text{diag}(ES_B)$ . If the sorted vectors differ, then no isomorphism exists. Otherwise, we build a candidate list: for each  $i$ , include those  $j$  such that  $(S_A)_{i,i} \approx (S_B)_{j,j}$ .

For each index  $i$  and each candidate  $j$ , we perform the following test:

If  $(ES_A)_{i,i} \not\approx (ES_B)_{j,j}$ , then no isomorphism exists through  $i$  and  $j$ , and we discard  $j$  from the candidate list. Otherwise, for each  $i_1 \neq i$ , retain only  $j_1 \neq j$  such that:

$$(ES_A)_{i_1,i} \approx (ES_B)_{j_1,j}.$$

If any  $i_1$  has no valid  $j_1$  candidates, then no isomorphism exists through  $i$  and  $j$ .

If  $(EA_A)_{i,i} \not\approx (EA_B)_{j,j}$ , then no isomorphism exists through  $i$  and  $j$ , and we discard  $j$  from the candidate list. Otherwise, To complete the test, we also check:

$$(EA_A)_{i_1,i} \approx (EA_B)_{j_1,j}.$$

If any  $i_1$  results in an empty candidate set, then no isomorphism exists.

The test is said to fail if it returns an empty mapping. Otherwise, we determine a partial isomorphism between the neighborhoods  $Z(S_A, e_i)$  and  $Z(S_B, e_j)$ , and likewise for  $A_A$  and  $A_B$ . If an isomorphism exists, it must pass through  $i$  and  $j$ , as ensured by the test's correctness.

For non-trivial automorphisms, we replace  $ES_B$  and  $EA_B$  with  $ES_A$  and  $EA_A$ . For each  $i$ , we collect all  $j \neq i$  such that  $(ES_A)_{i,i} \approx (ES_A)_{j,j}$ . We traverse all  $i$ , and for any  $i$  with a nontrivial list, we forbid identity mappings ( $i \mapsto i$ ) and run the algorithm. If a non-trivial automorphism is found, we return it. Otherwise, we restore the list and continue.

As an alternative to the exponential, we may use:

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

for  $M = S_A, S_B, A_A, A_B$ , and the method remains valid.

### 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 3** (Isomorphism Theorem). *Let  $G_1$  and  $G_2$  be graphs (or digraphs, multi-graphs, 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 4** (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:*

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

*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:

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

This establishes the equivalence. □

**Theorem 5** (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:*

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

*where  $\text{diag}(X)$  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:

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

□

### 3.2 Permutation Similarity and Powers of Matrices

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

**Theorem 6** (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:*

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

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

*And*

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

*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.

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

Like this

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

□

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 7** (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.  $\square$

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 8** (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:

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

*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.  $\square$

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  $A = \log(e^A)$

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

**Theorem 9** (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  $i \in \{1, \dots, n\}$ , 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  $k \geq 1$ ,

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

by repeated application of the permutation similarity property.

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

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

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  $i \in \{1, \dots, n\}$ .

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  $B = R A R^{-1}$  (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 10** (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

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

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:

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

which implies:

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

since  $\|A\| = \|B\|$ . □

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

**Theorem 11** (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  $A_{i1,i} \approx B_{j1,j}$

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:

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

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

- (a)
- $$c2 \leftarrow \text{verij}(map, S_A, S_B, i, j)$$
- (b) If  $c2$  remains non-empty this indicates that a consistent mapping exists for this candidate at index  $i$ .
- (i)
- $$c2 \leftarrow \text{verij}(c2, A_A, A_B, i, j)$$
- (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  $EA_B$ ,  $A \in B$ .

**Procedure:**

We iterate over each vertex index:

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

For each  $i$ :

1.  $c2 \leftarrow \text{internalTest}(map, S_A, A_A, S_B, A_B, i)$
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: adjacency 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  $EA_B$ , then sorts  $sort(diag(ES_A)) \approx sort(diag(ES_B))$ . If these are not identical, it again returns null.

Otherwise, it initializes the candidate map for every  $i$  index of A we associate the list of  $j$  of B, such as  $(ES_A)_{i,i} \approx (ES_B)_{j,j}$ .

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.

The `graphiso` algorithm was presented in the introduction; here we will only discuss what guarantees its operation.

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 12** (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

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

or

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

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

so no isomorphism exists there.  $\square$

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

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

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 13** (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.  $\square$

**Proposition 14** (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.  $\square$

**Theorem 15** (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.* For every isomorphism  $\sigma$  such that  $\sigma(i) = j$ , we have

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

so

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

thus

$$A_B = P_\sigma A_A P_\sigma^t e_{\sigma(i)}, \quad S_B = P_\sigma S_A P_\sigma^t e_{\sigma(i)},$$

and the same goes for  $EA_A$  and  $EA_B$  and  $ES_A$  and  $ES_B$  so the test does not fail.

Conversely, if no isomorphism passes through  $i$  and  $j$ , then for every isomorphism  $\sigma$  from  $A$  to  $B$ ,

if

$$A_B = P_\sigma A_A P_\sigma^t e_{\sigma(i)} \quad \text{and} \quad S_B = P_\sigma S_A P_\sigma^t e_{\sigma(i)},$$

then by Lemma 1, swapping  $j$  and  $\sigma(i)$  would yield an automorphism of  $B$ , contradicting the hypothesis. Hence, for every  $\sigma$ ,

$$S_B e_j \neq S_B e_{\sigma(i)} = P_\sigma S_A P_\sigma^t e_{\sigma(i)} = P_\sigma S_A e_i,$$

or

$$A_B e_j \neq A_B e_{\sigma(i)} = P_\sigma A_A P_\sigma^t e_{\sigma(i)} = P_\sigma A_A e_i.$$

Since  $S_B$  and  $A_B$  are normal,

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

thus

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

for some  $k$ , so the test fails for some  $k$ . Therefore,

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

and no isomorphism from  $S_A$  to  $S_B$  passing through  $i$  and  $j$  exists, or from  $A_A$  to  $A_B$ . Hence the test fails, and the same goes for  $EA_A$  and  $EA_B$  and  $ES_A$  and  $ES_B$  so the test fail.  $\square$

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 16** (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 the list of possible isomorphisms is empty, there is no  $\sigma$  such that

$$\text{diag}(ne^{\frac{S_A}{\|S_A\|}}) = \text{diag}(ne^{\frac{S_B}{\|S_B\|}}),$$

so the isomorphism does not exist.

3. If for any  $i$ , we cannot find a valid match, according to the test theorem and its corollaries, there is no isomorphism between  $S_A$  and  $S_B$  or  $A_A$  and  $A_B$ , 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.  $\square$

**Theorem 17.** *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 compare  $\text{sort}(\text{diag}(ES_A))$  and  $\text{sort}(\text{diag}(ES_B))$ , the time complexity is:

$$2n + n \log n \leq 3n \log n.$$

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:

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

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:

$$2n^4 + 8n^3 + 8n^2 + 3n \log n \leq 21n^4,$$

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 matrix 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  $p \leftarrow \text{graphIsoPartFinal}(\text{pos}, S_A, A_A, S_A, A_A)$  if p is not a null permutation returns p
    - (iii) restores the original list at i
4. returns a null permutation.

**Theorem 18** (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.  $\square$

**Theorem 19.** *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**.

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:

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

and adding the initial  $11n^3$  terms, we get:

$$2n^5 + 11n^3 \leq 14n^5.$$

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:

$$10n^2 - 5n^2 + n(n + 1) \leq 13n^2,$$

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=I4kQAQAIAAJ>