Title

Vertex analogy testing through comparison of ordered eigenprojections: an exact polynomial-time algorithm for potentially degenerate graph isomorphism tasks.

Corresponding author

Robert J. O'Shea
King's College London
5th floor, Becket House, 1 Lambeth Palace Rd, London, SE1 7EU, United Kingdom
robert.1.oshea@kcl.ac.uk

Abstract

The graph isomorphism problem is one of the most important open problems in complexity science. This study presents an exact method to identify analogous vertices in isomorphic graphs, through comparison of vertices' projections onto each eigenspace. It is demonstrated that an ordering transformation induces equality between analogous vertices eigenprojections, even in degenerate graphs. Furthermore, it is shown that equality of the ordered vertex eigenprojections is sufficient to demonstrate analogy of the corresponding vertices. A systematic perturbation strategy is proposed to address vertices with multiple analogies, which preserves isomorphism of perturbed graphs whilst inducing a unique analogy between perturbed vertices. Repeated iterations of analogy testing, and perturbation allow recovery of an isomorphic mapping in $\mathcal{O}(n^4)$ time. Analysis is provided to support claims and experimental performance is demonstrated in synthetic and biological data.

Section 1

Background

The graph isomorphism problem requires the determination of structural equivalence of two graphs, which may be demonstrated by inference of a quadratic permutation mapping one adjacency matrix to the other. It is considered one of the most important unresolved problems in computer science [1] arising in applications such as image recognition, systems biology and cheminformatics [1]. Although the problem's computational complexity has remained an open problem since its description in 1972 [1, 2], it has been demonstrated that it is in the low hierarchy of the class NP [3]. The graph isomorphism problem has been supposed to lie between P and NP-complete[4]. Polynomial time algorithms are available for some special cases of the problem. Babai proposed a spectral assignment algorithm for graphs with bounded eigenvalue multiplicity [5]. Luks proposed an algorithm for graphs of bounded degree [6] which was developed further by Babai to provide a quasipolynomial-time solution for the general case [7–9]. However, to date, no polynomial-time algorithm has been identified for the general case [1, 2, 10]. Klus developed a heuristic spectral assignment algorithm for degenerate graphs, in which vertex analogy is estimated by comparing vertex eigenpolytopes[10]. Klus employed a perturbation routine in which self-loops are applied to potentially analogous vertex pairs, with the aim of inducing unique analogies. However, the heuristic nature of the analogy testing procedure and the perturbation scheme necessitates occasional "backtracking". Consequently, polynomial complexity cannot be guaranteed[10].

Study Objectives

The objective of this study is to provide an $\mathcal{O}(n^4)$ time algorithm for the graph isomorphism problem where spectra are potentially degenerate. An exact method is developed to identify analogous vertex pairs by analysis of their eigenprojections. A perturbation strategy is developed to guarantee that each iteration preserves isomorphism whilst breaking symmetries. Theoretical analysis is provided to support these claims and experimental performance is demonstrated in real and synthetic data.

This paper is organised as follows: Section 2 introduces the notation employed in this article and formally defines the graph isomorphism problem and several relevant entities. Section 3 provides a textual overview of the algorithm. Section 4 provides the theoretical proofs supporting the algorithm. Section 5 provides the algorithm pseudocode. Section 6 describes experimental applications, results, and conclusions.

Section 2.

Preliminaries and notation

Let $\mathcal{S}_n$ the symmetric group of degree n and $\mathcal{P}_n$ denote the set of $n \times n$ dimensional permutation. Let $\pi \in \mathcal{S}_n$ and $P \in \mathcal{P}_n$ denote corresponding vector and matrix representations of the same permutation, such that:

$$p_{ij} = \begin{cases} 1, & \pi(i) = j \\ 0, & \pi(i) \neq j \end{cases}$$

Let $\mathcal{G}_A(\mathcal{V}, \mathcal{E}_A)$ and $\mathcal{G}_B(\mathcal{V}, \mathcal{E}_B)$ be two weighted undirected graphs, with adjacency matrices $A, B \in \mathbb{R}^{n \times n}$, where $\mathcal{V} = \{v_1, \dots, v_n\}$ is the vertex set and $\mathcal{E}_A$ and $\mathcal{E}_B$ are edge sets. The ith vertex in $\mathcal{G}_A$ is denoted $\mathcal{G}_A(v_i)$.

Definition. Graphs $\mathcal{G}_A$ and $\mathcal{G}_B$ are termed "isomorphic" if there exists some permutation $\pi \in \mathcal{S}_n$ such that:

$$(v_i, v_j) \in \mathcal{E}_A \leftrightarrow (v_{\pi(i)}, v_{\pi(j)}) \in \mathcal{E}_B$$

Equivalently, $\mathcal{G}_A$ and $\mathcal{G}_B$ are termed isomorphic if there exists some $P \in \mathcal{P}_n$, such that:

$$B = PAP^T$$

Isomorphism of $\mathcal{G}_A$ and $\mathcal{G}_B$ is denoted $\mathcal{G}_A \cong \mathcal{G}_B$.

Definition. A permutation $\pi \in \mathcal{S}_n$ is termed an "isomorphic mapping" from $\mathcal{G}_A$ to $\mathcal{G}_B$ if it satisfies:

$$(v_{\pi(i)}, v_{\pi(j)}) \in \mathcal{E}_A \leftrightarrow (v_i, v_j) \in \mathcal{E}_B$$

Likewise, $P \in \mathcal{P}_n$ is termed an isomorphic mapping from $\mathcal{G}_A$ to $\mathcal{G}_B$ if it satisfies $B = PAP^T$. This is denoted $\pi: \mathcal{G}_A \to \mathcal{G}_B$.

Definition. "Analogy" of $\mathcal{G}_A(v_i)$ to the jth vertex in $\mathcal{G}_B(v_j)$, implies that there exists some $\hat{\pi} \in \mathcal{S}_n$ such that $\hat{\pi}: \mathcal{G}_A \to \mathcal{G}_B$ and $\hat{\pi}(i) = j$. This is denoted $\mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B(v_j)$.

Definition. A "unique analogy" of the $\mathcal{G}_A(v_i)$ to $\mathcal{G}_B(v_j)$ implies that $\mathcal{G}_A(v_x) \leftrightarrow \mathcal{G}_B(v_y)$ if $x = i$ and $y = j$, otherwise $\mathcal{G}_A(v_x) \leftrightarrow \mathcal{G}_B(v_y)$. Unique analogy of $\mathcal{G}_A(v_i)$ to $\mathcal{G}_B(v_j)$ is denoted $\mathcal{G}_A(v_i) \dot{\leftrightarrow} \mathcal{G}_B(v_j)$.

Our method employs various objects derived from graph adjacency matrix eigendecomposition.

Let $\lambda_A = \left[\lambda_A^{(1)}, \dots, \lambda_A^{(n)}\right]$ denote the eigenvalues of $A \in \mathbb{R}^{n\times n}$ such that $\lambda_A^{(1)} \geq \dots \geq \lambda_A^{(n)}$. Let $\Lambda_A \in \mathbb{R}^{n\times n}$ denote $\mathrm{diag}(\lambda_A)$.

Let $\dot{\lambda}_A = \left[\dot{\lambda}_A^{(1)}, \dots, \dot{\lambda}_A^{(m)}\right]$ denote the $m \leq n$ distinct eigenvalues of A, such that $\dot{\lambda}_A^{(1)} > \dots > \dot{\lambda}_A^{(m)}$.

Let $\mu_A = \left[\mu_A^{(1)}, \dots, \mu_A^{(m)}\right]$ denote the eigenvalue multiplicities of A, such that:

$$\sum_{k=1}^{m} \mu_A^{(k)} = n$$

Let $V_A \in \mathbb{R}^{n\times n}$ be an eigenbasis of A. Let V_A be partitioned by eigenvalue, such that $V_A^{(k)} = \left[V_A^{(1)}, \dots, V_A^{(m)}\right]$, where $V_A^{(k)} \in \mathbb{R}^{n\times \mu_A^{(k)}}$.

Let $E_A^{(k)} \in \mathbb{R}^{n\times n}$ be the orthogonal projection of A onto its kth eigenspace, such that:

$$E_A^{(k)} = \dot{\lambda}_A^{(k)} V_A^{(k)} \left(V_A^{(k)}\right)^T$$

Let $E_A \in \mathbb{R}^{n\times n\times m}$ be the tensor eigenspace projections, such that $E_A = \left[E_A^{(1)}, \dots, E_A^{(m)}\right]$. Thus, we have:

$$A = \sum_{k=1}^{m} E_A^{(k)}$$

Let $E_A^{(k)}(v_i) \in \mathbb{R}^n$ denote the ith column of $E_A^{(k)}$, representing the projection of the ith vertex into the kth eigenspace.

Definition. The "vertex eigenprojection", is the matrix of the vertex's projections onto each eigenspace. $E_A(v_i) \in \mathbb{R}^{n\times m}$ denotes the ith vertex eigenprojection in A, such that:

$$E_A(v_i) = \left[E_A^{(1)}(v_i), \dots, E_A^{(m)}(v_i)\right]$$

Let s: $\mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m}$ be a row-ordering function where rows are sorted in ascending order - preferentially by the first, then second, then third columns, etc., resolving ties in earlier columns by latter columns. Thus, $s(X) = Q^{(X)}X$, where $X \in \mathbb{R}^{n\times m}$ is an input matrix and $Q^{(X)} \in \mathcal{P}_n$ is a row-permutation matrix satisfying:

$$\left(Q^{(X)}X\right)_{ik} < \left(Q^{(X)}X\right)_{jk} \ \forall \left\{i, j, k, l \in \mathbb{N}_{\leq n} \middle| i < j, k < l, \left(Q^{(X)}X\right)_{il} > \left(Q^{(X)}X\right)_{jl}\right\}$$

Definition. The row-permutation of a vertex eigenprojection under s is termed the "ordered vertex eigenprojection". The ith ordered vertex eigenprojection in A is given by $s\left(E_A(v_i)\right)$.

Let I_n denote the n-dimensional identity matrix. Let 1_n denote an $n \times 1$ column of ones and let $1_{n\times n}$ denote a matrix of ones. Let $\mathbb{I}(\cdot)$ denote the indicator function.

Section 3.

Algorithm Overview

This study presents an exact method to identify analogous vertices in isomorphic graphs. It is demonstrated that the ordered eigenprojections of analogous vertices are equal. Furthermore, it is demonstrated that equality of ordered vertex eigenprojections implies analogy of vertices. These results may be exploited to test analogy of any vertex pair. A perturbation strategy is devised to address instances where vertices have multiple analogies. In each iteration, one analogous vertex pair is perturbed by adding a characteristically weighted self loop to both vertices, inducing a unique analogy, whilst preserving isomorphism of the perturbed graphs. The processes of analogy testing and perturbation are iterated until no vertex in the perturbed graphs has more than one analogy. It is demonstrated that fewer than n iterations are required to infer an isomorphic mapping in this fashion. Therefore, the algorithm runs in $\mathcal{O}(n^4)$ time.

Section 3

In graph isomorphism problems, it is sufficient to assume that graphs are isomorphic whilst searching for a mapping. This assumption may be tested once the mapping has been recovered. Accordingly, in our problem setup, we let $\mathcal{G}_A \cong \mathcal{G}_B$.

Lemma 1 demonstrates that E_A is invariant to choice of eigenbasis, even when spectra are degenerate.

Lemma 1. If $B = PAP^T$ then $E_B^{(k)} = \hat{P} E_A^{(k)} \hat{P}^T$.

Proof

We have:

$$V_A^T A V_A = V_B^T B V_B = \Lambda$$

As Λ is diagonal, it may be decomposed by eigenspace, such that:

$$\left(V_A{}^{(k)}\right)^T A V_A^{(k)} = \left(V_B{}^{(k)}\right)^T B V_B^{(k)} = \lambda^{(k)} I_{\mu^{(k)}}$$

Substituting $B = PAP^T$, we have

$$\left(V_A{}^{(k)}\right)^T A V_A = \left(P^T V_B{}^{(k)}\right)^T A \left(P^T V_B{}^{(k)}\right)$$

Removing A, we have:

$$P \left(V_A{}^{(k)}\right)^T V_A P^T = V_B{}^{(k)} \left(V_B{}^{(k)}\right)^T$$

Substituting the definition of $E_A^{(k)}$ and $E_B^{(k)}$, we have:

$$PE_A^{(k)}P^T = E_B^{(k)}$$

End of lemma 1

Hence, the projections $E_A^{(k)}$ and $E_B^{(k)}$ are related by the same isomorphic mapping as $\mathcal{G}_A$ and $\mathcal{G}_B$, such that:

$$B = \hat{P}A\hat{P}^T \Leftrightarrow E_B^{(k)} = \hat{P}E_A^{(k)}\hat{P}^T$$

We have $E_B^{(k)} = PE_A^{(k)}P^T$ for all $k \in \{1, \dots, m\}$. Thus, we have:

$$E_B^{(k)}(v_i) = PE_A^{(k)}P^T(v_i)$$

$$E_B^{(k)}P(v_i) = PE_A^{(k)}(v_i)$$

$$E_B^{(k)}\left(v_{\pi(i)}\right) = PE_B^{(k)}(v_i)$$

Likewise, we have:

$$E_B\left(v_{\pi(i)}\right) = PE_A(v_i)$$

Lemma 2 demonstrates that eigenprojections of analogous vertices are related by a linear permutation. It is noted that this result holds under degenerate condition, providing a reliable starting point for our algorithm.

Lemma 2. $E_B\left(v_{\pi(i)}\right) = PE_A(v_i)$.

Proof

From Lemma 1 we have:

$$PE_A^{(k)}P^T = E_B^{(k)}$$

Extracting v_i from each eigenspace we have:

$$PE_A^{(k)}P^T(v_i) = E_B^{(k)}(v_i)$$

$$PE_A^{(k)}(v_i) = E_B^{(k)}P(v_i)$$

Substituting the vertex permutation, we have:

$$PE_A^{(k)}(v_i) = E_B^{(k)}\left(v_{\pi(i)}\right)$$

Applying to each eigenspace $k \in \{1, \dots, m\}$:

$$PE_A(v_i) = E_B\left(v_{\pi(i)}\right)$$

End of lemma 2

The linearly permuted relationship between analogous vertices' eigenprojections allows reframing of the analogy detection task as a linear assignment problem. However, as P may be underdefined, it may not be deducible from this equation alone. This issue may be circumvented by applying the row-sorting function s to the vertex eigenprojections, neutralising the effect of P. **Lemma 3** demonstrates that s induces equality between any two row-permutations of a matrix.

Lemma 3. Let $X \in \mathbb{R}^{\{n \times m\}}$ and $Q^{(x)} \in \mathcal{P}_n$ such that:

$$\left(Q^{(X)}X\right)_{ik} < \left(Q^{(X)}X\right)_{jk} \ \forall \left\{i, j, k, l \in \mathbb{N}_{\leq n} \middle| i < j, k < l, \left(Q^{(X)}X\right)_{il} > \left(Q^{(X)}X\right)_{jl}\right\}$$

Let $P \in \mathcal{P}_n$. Then $s(X) = s(PX)$.

Proof

We have:

$$\left(Q^{(X)}X\right)_{ik} < \left(Q^{(X)}X\right)_{jk} \ \forall \left\{i, j, k, l \in \mathbb{N}_{\leq n} \middle| i < j, k < l, \left(Q^{(X)}X\right)_{il} > \left(Q^{(X)}X\right)_{jl}\right\}$$

Multiplying internally by $P^T P$, we have:

$$\left(Q^{(X)}P^T PX\right)_{ik} < \left(Q^{(X)}P^T PX\right)_{jk} \ \forall \left\{i, j, k, l \in \mathbb{N}_{\leq n} \middle| i < j, k < l, \left(Q^{(X)}P^T PX\right)_{il} > \left(Q^{(X)}P^T PX\right)_{jl}\right\}$$

Factorising by $\left(Q^{(X)}P^T\right)$, it is evident that $Q^{(X)}P^T$ is a solution to $Q^{(PX)}$. Letting $Q^{(PX)} = Q^{(X)}P^T$, and multiplying from the right by PX, we have:

$$Q^{(PX)}PX = Q^{(X)}P^T PX$$

Thus:

$$Q^{(PX)}PX = Q^{(X)}X$$

Substituting for the definitions of $s(PX)$ and $s(X)$, we have:

$$s(PX) = s(X)$$

End of lemma 3

As analogous vertices eigenprojections are row-permutations of one another (lemma 2), s induces equality between them. Applying s to a vertex eigenprojection $E_A(v_i)$, we attain the ordered vertex eigenprojection, $s\left(E_A(v_i)\right)$. **Lemma 4a** demonstrates that analogous vertices have equal ordered eigenprojections, such that:

$$\mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B(v_j) \Rightarrow \ s\left(E_A(v_i)\right) = s\left(E_B(v_j)\right)$$

Lemma 4a. $s\left(E_A(v_i)\right) = s\left(E_B\left(v_{\pi(i)}\right)\right)$.

From Lemma 2 we have:

$$PE_A(v_i) = E_B\left(v_{\pi(i)}\right)$$

Therefore

$$s\left(PE_A(v_i)\right) = s\left(E_B\left(v_{\pi(i)}\right)\right)$$

By lemma 3 we may substitute $s\left(PE_A(v_i)\right)$ for $s\left(E_A(v_i)\right)$.

Thus

$$s\left(E_A(v_i)\right) = s\left(E_B\left(v_{\pi(i)}\right)\right)$$

End of lemma 4a

Thus, analogous vertices have equal ordered eigenprojections. **Lemma 4b** demonstrates that if $\mathcal{G}_A \cong \mathcal{G}_B$, then equality of ordered vertex eigenprojections implies analogy of the corresponding vertices, such that:

$$s\big(E_A(v_i)\big) = s\left(E_B\big(v_j\big)\right) \Rightarrow \mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B\big(v_j\big)$$

Lemma 4b. If $\mathcal{G}_A \cong \mathcal{G}_B$ and $s\big(E_A(v_i)\big) = s\left(E_B\big(v_j\big)\right)$ then $\mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B\big(v_j\big)$.

Proof

As $s\big(E_A(v_i)\big) = s\left(E_B\big(v_j\big)\right)$, there exists some $\hat{P} \in \mathcal{P}_n$ such that:

$$\hat{P}E_A(v_i) = E_B\big(v_j\big)$$

By Lemma 2, $\hat{P}E_A(v_i)$ may be substituted for $E_B\big(v_{\hat{\pi}(i)}\big)$. Hence $E_B\big(v_{\hat{\pi}(i)}\big) = E_B\big(v_j\big)$ and $\hat{\pi}(i) = j$.

Substituting $\hat{P}E_A(v_i)$ for $\hat{P}E_AP^T\big(v_{\hat{\pi}(i)}\big)$, we have:

$$\hat{P}E_AP^T\big(v_{\hat{\pi}(i)}\big) = E_B\big(v_j\big)$$

Substituting $\hat{\pi}(i)$ for j, we have:

$$\hat{P}E_AP^T\big(v_j\big) = E_B\big(v_j\big)$$

Likewise, for all $v_x \in \mathcal{V} \setminus v_j$, we have:

$$\hat{P}E_AP^T(v_x) = E_B(v_x)$$

Concatenating, we have:

$$\hat{P}E_AP^T = E_B$$

Therefore $\hat{P}A\hat{P}^T = B$ and $\hat{\pi}(i) = j$.

End of lemma 4b

Combining the results of lemmas 4a and 4b, it is evident that equality of ordered vertex eigenprojections determines analogy of the corresponding vertices, such that:

$$s\big(E_A(v_i)\big) = s\left(E_B\big(v_j\big)\right) \Leftrightarrow \mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B\big(v_j\big)$$

This deterministic relationship provides a cubic-time route to test analogy of vertex pairs without inferring P. Each pair may be tested in order, generating a precursor to P, which we will denote as $\check{P}$. Thus:

$$\check{p}_{ji} \coloneqq \mathbb{I}\left(s\big(E_A(v_i)\big) = s\left(E_B\big(v_j\big)\right)\right)$$

Equivalently,

$$\check{p}_{ji} \coloneqq \mathbb{I}\left(\mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B\big(v_j\big)\right)$$

If the solution to π is unique, each vertex has only one analogy and $\check{P} = P$. However, if multiple isomorphisms exist, some vertices will have multiple analogies and $\check{P} \notin \mathcal{P}_n$. To resolve this issue, analogy-preserving perturbations may be applied to A and B. Let $A^*, B^* \in \mathbb{R}^{n\times n}$ denote the

perturbed matrices. Let $\mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B(v_j)$. A and B may be perturbed at a_{ii} and b_{jj} to generate isomorphic matrices A^* and B^* with a unique isomorphism $\mathcal{G}_{A^*}(v_i) \dot{\leftrightarrow} \mathcal{G}_{B^*}(v_j)$.

Let $W^{(x)} \in \mathbb{R}^{n\times n}$ such that:

$$w_{ij}^{(x)} \coloneqq \mathbb{I}(i = j = x)$$

The perturbed matrices A^* and B^* are declared such that:

$$A^* \coloneqq A + W^{(i)} \cdot (1 + \max(A) - a_{ii})$$

$$B^* \coloneqq B + W^{(j)} \cdot \left(1 + \max(B) - b_{jj}\right)$$

This matrix perturbation is equivalent to the graph procedure of adding a characteristically weighted self loop to $\mathcal{G}_A(v_i)$ and $\mathcal{G}_B(v_j)$. **Lemma 5** demonstrates that perturbation of analogous vertices $\mathcal{G}_A(v_i)$ and $\mathcal{G}_B(v_j)$ induces a unique analogy $\mathcal{G}_{A^*}(v_i) \dot{\leftrightarrow} \mathcal{G}_{B^*}(v_j)$. Consequently, any other analogies $\mathcal{G}_A(v_i)$ and $\mathcal{G}_B(v_j)$ had are removed in the perturbed graphs. Thus, perturbation may be targeted to break graph symmetries and reduce degeneracy.

Lemma 5. Let $\mathcal{G}_A \cong \mathcal{G}_B$ and $\mathcal{G}_A(v_i) \leftrightarrow \mathcal{G}_B(v_j)$. Let $A^*, B^* \in \mathbb{R}^{n\times n}$, such that:

$$A^* = A + W^{(i)} \cdot (1 + \max(A) - a_{ii})$$
$$B^* = B + W^{(j)} \cdot \left(1 + \max(B) - b_{jj}\right)$$

Then $\mathcal{G}_{A^*}(v_i) \dot{\leftrightarrow} \mathcal{G}_{B^*}(v_j)$.

Proof

We have $\max(A) = \max(B)$. Hence:

$$\operatorname{diag}\left(A^{*i}\right)_i = \operatorname{diag}\left(B^{*j}\right)_j = \max(A) + 1$$

Hence, for all $v_x \in \mathcal{V} \setminus v_j$ we have:

$$\operatorname{diag}(A^*)_i > \operatorname{diag}(B^*)_x$$

Therefore $\hat{\pi}$ cannot satisfy $B^* = \hat{P}A^*\hat{P}^T$ unless $\hat{\pi}(i) = j$. Therefore $\mathcal{G}_{A^*}(v_i) \dot{\leftrightarrow} \mathcal{G}_{B^*}(v_j)$.

Lemma 6 demonstrates that if $\hat{\pi}$ satisfies the perturbed equality $B^* = \hat{P}A^*\hat{P}^T$ then it satisfies the original equality $B = \hat{P}A\hat{P}^T$. Thus, if $\hat{\pi}: \mathcal{G}_{A^*} \to \mathcal{G}_{B^*}$ then $\hat{\pi}: \mathcal{G}_A \to \mathcal{G}_B$. Consequently, the search for $\hat{\pi}$ may continue in the perturbed graphs.

Lemma 6. If $\mathcal{G}_{A^*}(v_i) \leftrightarrow \mathcal{G}_{B^*}(v_j)$ and $\hat{\pi}: \mathcal{G}_{A^*} \to \mathcal{G}_{B^*}$ then $\hat{\pi}: \mathcal{G}_A \to \mathcal{G}_B$.

We have:

$$B^* = B + W^{(j)} \cdot \left(1 + \max(B) - b_{jj}\right)$$

Factorising the left side of the equality we have:

$$\hat{P}A^*\hat{P}^T = B + W^{(j)} \cdot \left(1 + \max(B) - b_{jj}\right)$$

$$\hat{P}A\hat{P}^T + \hat{P}W^{(i)}\hat{P}^T \cdot \left(1 + \max(A) - a_{jj}\right) = B + W^{(j)} \cdot \left(1 + \max(B) - b_{jj}\right)$$

As $\mathcal{G}_{A^*}(v_i) \leftrightarrow \mathcal{G}_{B^*}(v_j)$ we can substitute $\hat{P}W^{(i)}\hat{P}^T$ for $W^{(j)}$, thus:

$$\hat{P}A\hat{P}^T + W^{(j)} \cdot \left(1 + \max(A) - a_{jj}\right) = B + W^{(j)}\left(1 + \max(B) - b_{jj}\right)$$

As $\max(A) = \max(B)$ and $a_{ii} = b_{jj}$, we have:

$$\hat{P}A\hat{P}^T = B$$

Therefore $\hat{\pi}: \mathcal{G}_A \to \mathcal{G}_B$.

End of lemma 6.

If a non-uniquely analogous vertex pair is perturbed in this way, a unique analogy is induced and the number of isomorphisms is thereby reduced. Vertex analogies may now be checked in the perturbed matrices A^* and B^*, such that:

$$\check{p}_{ji} \coloneqq \mathbb{I}\left(s\left(E_{A^*}(v_i)\right) = s\left(E_{B^*}(v_j)\right)\right)$$

Perturbations may be applied iteratively until unique analogies are induced for each vertex. Naively, each source vertex could be perturbed in turn. After $n-1$ iterative perturbations, each entry on the perturbed matrix diagonals would be distinct, implying that any solution to $\hat{\pi}: \mathcal{G}_{A^*} \to \mathcal{G}_{B^*}$ would be unique. However, if some vertices already have unique analogies, these need not be perturbed. Thus, fewer than n iterations are required to induce a unique isomorphism between $\mathcal{G}_{A^*}$ and $\mathcal{G}_{B^*}$. If $\mathcal{G}_{A^*}$ and $\mathcal{G}_{B^*}$ remain degenerate, other vertex pairs with multiple analogies may be perturbed iteratively. After $n-1$ iterations each diagonal entry of A^* and B^* will be unique, guaranteeing uniqueness of each vertex analogy.

Section 5

Algorithm pseudocode

Inputs

$A, B \in \mathbb{R}^{\{n \times n\}}, P \in \mathcal{S}_n$ such that $B = PAP^T$.

Outputs

$\hat{P} \in \mathcal{S}_n$ such that $B = \hat{P}A\hat{P}^T$

Algorithm

$P \coloneqq 1_{n \times n}$

$A^*, B^* \coloneqq A, B$

WHILE $\left(\min_i\left(\sum_{j=1}^{n} \check{p}_{ji}\right) > 1\right)$:

    Compute E_{A^*}, E_{B^*}

    FOR i in $\{1, \dots n\}$:

        Compute $s\left(E_{A^*}(v_i)\right),\ s\left(E_{B^*}(v_i)\right)$

    FOR i in $\{1, \dots n\}$:

        FOR j in $\{1, \dots n\}$:

$$\check{p}_{ji} \coloneqq \mathbb{I}\left(s\left(E_{A^*}(v_i)\right) = s\left(E_B(v_j)\right)\right)$$

IF $\left(\min_i\left(\sum_{j=1}^{n} \check{p}_{ji}\right) > 1\right)$:

$$i \coloneqq i \mid \left(\check{P}1\right)_i > 1$$

$$j \coloneqq j \mid \check{p}_{ji} = 1$$

$$a^*_{ii} \coloneqq \max(A^*) + 1$$

$$b^*_{jj} \coloneqq \max(B^*) + 1$$

ELSE:

$$\hat{P} \leftarrow \check{P}$$

RETURN $\hat{P}$

Section 6

Experiments

Algorithm performance was tested on real-word and synthetic graph dataset. Real world datasets consisted of the C-elegans network [11, 12] and the first 100 enzyme structure graphs in a cheminformatics repository [12]. As regularly structured synthetic graphs present some of the most challenging problems for traditional algorithms [5], Heawood graph [13] and Johnson graphs [14] with parameters (n=7, k=3) and (n=8, k=4) were also included. Graphs were represented as symmetric adjacency matrices. In each experiment, a random quadratic permutation was applied to one of the graph datasets, generating a tuple of isomorphic graphs. The algorithm was subsequently applied to infer a quadratic permutation function mapping the graphs. Five experiments were generated from different random permutation of each graph dataset. Analysis was performed with R and Rstudio [15, 16]. Numerical tolerances of 10^{-8} and 10^{-6}were applied when testing equality of eigenvalues and eigenvectors, respectively. All code required to reproduce the findings of this study is provided in **Supplementary Information 1**.

Results

The graph isomorphism algorithm identified a correct solution to $\hat{\pi}$ in each experiment. Algorithmic vulnerability to numerical instability was identified in some experiments, necessitating the imposition of numerical tolerances during equality checking operations.

Discussion

The algorithm described in this study solves all real-valued symmetric graph isomorphism problems. Neither positive definiteness nor eigenvalue multiplicity bounds are required conditions, differentiating this work from previous approaches. Where multiple isomorphic mappings exist, a single solution to $\hat{\pi}$ is returned, however the full set of solutions may be computed by changing the order of perturbations.

The complexity of the algorithm is dominated by the eigendecomposition procedure, which runs in $\mathcal{O}(n^3)$ time. As up to $n-1$ runs of eigendecomposition may be required, the algorithm will run in $\mathcal{O}(n^4)$ time in the worst case scenario (multiples of the identity matrix).

As the algorithm requires testing equality of eigenvalues and outer products of eigenspaces, issues may arise with regards to numerical precision. These problems may be addressed with appropriate rounding or numerical tolerance.

Schoning demonstrated that the polynomial hierarchy collapses to a finite level if the graph isomorphism problem is NP-complete [3]. The algorithm presented here demonstrates that any problem which may be reduced to a graph isomorphism problem may be solved in polynomial time. Thus, If the graph isomorphism problem is found to be NP-complete this algorithm may be used to solve all NP problems in polynomial time. Further research may investigate developments to the algorithm to solve the subgraph isomorphism problem, which is NP-complete [17], in polynomial time.

Funding

Authors acknowledge funding support from the UK Research & Innovation London Medical Imaging and Artificial Intelligence Centre; Wellcome/Engineering and Physical Sciences Research Council Centre for Medical Engineering at King's College London (WT 203148/Z/16/Z); National Institute for Health Research Biomedical Research Centre at Guy's & St Thomas' Hospitals and King's College London; Cancer Research UK National Cancer Imaging Translational Accelerator (A27066).

References

1. Grohe M, Schweitzer P (2020) The graph isomorphism problem. Commun ACM 63:128–134. https://doi.org/10.1145/3372123
2. Karp RM (1972) Reducibility among Combinatorial Problems
3. Schöning U (1988) Graph isomorphism is in the low hierarchy. J Comput Syst Sci 37:312–323. https://doi.org/10.1016/0022-0000(88)90010-4
4. Arvind V, Torán J (2005) Isomorphism testing: Perspective and open problems. Bull Eur Assoc Theor Comput Sci 86:66–84
5. Babai L, Grigoryev DY, Mount DM (1982) Isomorphism of graphs with bounded eigenvalue multiplicity. Proc Annu ACM Symp Theory Comput 310–324. https://doi.org/10.1145/800070.802206
6. Luks EM (1982) Isomorphism of graphs of bounded valence can be tested in polynomial time. J Comput Syst Sci 25:42–65. https://doi.org/10.1016/0022-0000(82)90009-5
7. Babai L (2016) Graph isomorphism in quasipolynomial time: [Extended abstract]. Proc Annu ACM Symp Theory Comput 19-21-June:684–697. https://doi.org/10.1145/2897518.2897542
8. Babai L (2015) Graph Isomorphism in Quasipolynomial Time
9. Helfgott HA, Bajpai J, Dona D (2017) Graph isomorphisms in quasi-polynomial time
10. Klus S, Sahai T (2018) A spectral assignment approach for the graph isomorphism problem. Inf Inference 7:689–706. https://doi.org/10.1093/imaiai/iay001

11. Kaiser M, Hilgetag CC (2005) Non-Optimal Component Placement, but Short Processing Paths, due to Long-Distance Projections in Neural Systems. PLoS Comput Biol preprint:e95. https://doi.org/10.1371/journal.pcbi.0020095.eor

12. Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. Proc Natl Conf Artif Intell 6:4292–4293

13. Heawood PJ (1890) Map-Colour Theorem. QJ Math First Seri:332–338

14. Holton DA, Sheehan J (1993) The Petersen Graph. The Petersen Graph. https://doi.org/10.1017/cbo9780511662058

15. R Core Team (2021) R: A Language and Environment for Statistical Computing. R Found. Stat. Comput.

16. RStudio Team (2021) RStudio: Integrated Development for R

17. Wegener I (2005) Complexity theory: Exploring the limits of efficient algorithms. Complex Theory Explor Limits Effic Algorithms 1–308. https://doi.org/10.1007/3-540-27477-4

```
#graph isomorphism algorithm

#libraries
if(T){
  library(abind)
  library(sand)
}

#permutation inference functions
if(T){

  #generate h(g(X))
  hgf <- function(X, tol=1e-8){
    n <- ncol(X)
    round_digits <- -log(tol, base=10)
    eigen_X <- eigen(X)
    evecs <- eigen_X$vectors
    evals <- eigen_X$values
    evals <- round(evals, digits=round_digits)
    unique_evals <- unique(evals)
    gX <- lapply(unique_evals, function(eval_i){
      eval_i*tcrossprod(evecs[,evals==eval_i, drop=F])
    })
    gX <- abind(gX, along=3)
    gX <- round(gX, digits=round_digits)
    hgX <- lapply(1:n, function(i){
      gX_i <- gX[,i,]
      order_i <- do.call(order, as.data.frame(gX_i))
      hgX_i <- gX_i[order_i,]
      return(hgX_i)
    })
    return(hgX)
  }

  #check equivalence of h(g(X)[:,:,i]) and h(g(Y)[:,:,j])
  x_eq_l <- function(hgX, hgY, tol=1e-6){
    sapply(hgX, function(hgX_i){
      sapply(hgY, function(hgY_i){
        all.equal(hgX_i, hgY_i,
                  check.attributes=F,
                  check.names = F,
                  tolerance=tol)==T
      })
    })
  }

  #permutation inference function
  perm_hat_f <- function(X, Y, verbose=F){

    #declare matrices for perturbation
    X_k <- X
    Y_k <- Y

    #compute h(g(X)) and h(g(Y))
```

```
    hgX <- hgf(X_k)
    hgY <- hgf(Y_k)

    #check for correspondences
    P_hat <- x_eq_l(hgX, hgY)

    #check for vertices with multiple correspondences
    v_j_remaining <- which(rowSums(P_hat)>1)

    #begin iterative loop
    iter <- 1
    while(length(v_j_remaining)>0){

      if(verbose){
        message(iter)
      }

      #find a vertex with multiple correspondences in X
      v_j <- v_j_remaining[1]

      #find a corresponding vertices in Y
      v_i <- which(P_hat[v_j,])[1]

      #perturb the selected vertices to induce a unique correspondence
      X_k[v_i,v_i]<- max(X_k)+1
      Y_k[v_j,v_j]<- max(Y_k)+1

      #recompute h(g(X)) and h(g(Y))
      hgX <- hgf(X_k)
      hgY <- hgf(Y_k)

      #recheck for for vertices with multiple correspondences
      P_hat <- x_eq_l(hgX, hgY)
      v_j_remaining <- which(rowSums(P_hat)>1)
      iter <- iter+1
    }
    return(P_hat)
  }
}

#download network data
if(F){

  #Heawood graph
  X_heawood <- as_adj(make_graph("heawood"),sparse=F)

  #Johnson graph
  make_johnson_graph <- function(nj,kj){
    all_subsets <- combn(1:nj,kj)
    n <- ncol(all_subsets)
    am <- apply(all_subsets, 2, function(a){
      apply(all_subsets, 2, function(b){
        sym_diff_ab <-  union(setdiff(a,b),
                              setdiff(b,a))
```

```
        length(sym_diff_ab)==2
      })
    })
    return(am)
  }
  X_johnson_7_3 <- make_johnson_graph(7, 3)
  X_johnson_8_4 <- make_johnson_graph(8, 4)

  #yeast protein protein interaction
  data("ppi.CC")
  X_yeast <- as_adjacency_matrix(ppi.CC, sparse = F, type="both")

  #c elegans
  c_elegans_url <- 'https://snap.stanford.edu/data/C-elegans-
frontal.txt.gz'
  c_elegans_filename <- "C-elegans-frontal.txt.gz"
  suppressMessages(download.file(c_elegans_url, destfile =
c_elegans_filename))
  e_list <- (read.csv(c_elegans_filename))[-(1:3),1]
  e_list <- do.call(rbind, strsplit(e_list, " "))
  class(e_list) <- "numeric"
  n <- max(e_list)
  X_celegans <- matrix(0,n,n)
  X_celegans[e_list]<- 1
  X_celegans <- X_celegans + t(X_celegans)

  #graph list
  X_list <- list(
    heawood=X_heawood,
    johnson_7_3=X_johnson_7_3,
    johnson_8_4=X_johnson_8_4,
    yeast=X_yeast,
    celegans=X_celegans
  )

}

#download network data
if(F){

  graph_data_dir <- "graph_data"
  if(!dir.exists(graph_data_dir)){
    dir.create(graph_data_dir)
  }

  #network repository
  repo_url <- "https://networkrepository.com/chem.php"
  repo_pg <- readLines(repo_url)
  all_repo_downloads <- grep(">Download<", repo_pg, value=T)
  all_enzyme_downloads <- grep("ENZYMES", all_repo_downloads, value=T)
  all_enzyme_downloads <- gsub(".*<a href=\"", "", all_enzyme_downloads)
  all_enzyme_downloads <- gsub("\".*", "", all_enzyme_downloads)

  all_els <- lapply(all_enzyme_downloads[1:10], function(file_i){
```

```
  basename_i <- basename(file_i)
  basename_i <- substr(basename_i, 1, nchar(basename_i)-4)
  tmp <- tempfile()
  download.file(file_i, tmp)
  el_i <- read.table(unz(tmp, paste0(basename_i, ".edges")))
  el_i$graph_name <- substr(basename_i, 9, nchar(basename_i))
  Sys.sleep(2)
  return(el_i)
})

all_els_2 <- do.call(rbind.data.frame, all_els)

for(file_i in all_enzyme_downloads){



  destfile_i <- file.path(graph_data_dir, paste0(basename_i, ".zip"))


  unzip(destfile_i, paste0(basename_i, ".edges"))
  read.delim(tmp)
  unlink(tmp)


  destfile_i <- file.path(graph_data_dir, basename_i)

  }



#Heawood graph
X_heawood <- as_adj(make_graph("heawood"),sparse=F)

#Johnson graph
make_johnson_graph <- function(nj,kj){
  all_subsets <- combn(1:nj,kj)
  n <- ncol(all_subsets)
  am <- apply(all_subsets, 2, function(a){
    apply(all_subsets, 2, function(b){
      sym_diff_ab <-  union(setdiff(a,b),
                            setdiff(b,a))
      length(sym_diff_ab)==2
    })
  })
  return(am)
}
X_johnson_7_3 <- make_johnson_graph(7, 3)
X_johnson_8_4 <- make_johnson_graph(8, 4)

#yeast protein protein interaction
data("ppi.CC")
X_yeast <- as_adjacency_matrix(ppi.CC, sparse = F, type="both")
```

```
  #c elegans
  c_elegans_url <- 'https://snap.stanford.edu/data/C-elegans-
frontal.txt.gz'
  c_elegans_filename <- "C-elegans-frontal.txt.gz"
  suppressMessages(download.file(c_elegans_url, destfile =
c_elegans_filename))
  e_list <- (read.csv(c_elegans_filename))[-(1:3),1]
  e_list <- do.call(rbind, strsplit(e_list, " "))
  class(e_list) <- "numeric"
  n <- max(e_list)
  X_celegans <- matrix(0,n,n)
  X_celegans[e_list]<- 1
  X_celegans <- X_celegans + t(X_celegans)

  #graph list
  X_list <- list(
    heawood=X_heawood,
    johnson_7_3=X_johnson_7_3,
    johnson_8_4=X_johnson_8_4,
    yeast=X_yeast,
    celegans=X_celegans
  )

}

#experiments
if(T){

  #set seed
  set.seed(1)

  #experiment parameters
  param_grid <- expand.grid(
    rep=1:10,
    graph=names(X_list)[5],
    stringsAsFactors = F
  )

  experiments <- lapply(1:nrow(param_grid), function(params_i){

    message(params_i)

    #select input graph
    X_name <- param_grid$graph[params_i]
    X <- X_list[[X_name]]
    n <- ncol(X)

    #generate random permutation
    perm <- sample(n)
    Y <- X[perm,perm]

    #infer permutation
    P_hat <- perm_hat_f(X=X, Y=Y, verbose=F)
```

```
    #test equality
    Y_hat <- P_hat %*% X %*% t(P_hat)
    test <- all.equal(Y, Y_hat,
                      check.attributes=F)==T
    message(test, "\n\n")
    output <- data.frame(
      name=X_name,
      test=test
    )
    return(output)
  })
  experiments <- do.call(rbind.data.frame, experiments)

}

#visualisation libraries
if(T){
  library(ggplot2)
  library(reshape2)
  library(ggpubr)
}

#visualisation functions
if(T){

  #generate h(g(X)) visualisation
  hgf_visualisation <- function(X,X_name, tol=1e-8){
    n <- ncol(X)
    round_digits <- -log(tol, base=10)
    eigen_X <- eigen(X)
    evecs <- eigen_X$vectors
    evals <- eigen_X$values
    evals <- round(evals, digits=round_digits)
    unique_evals <- unique(evals)
    gX <- lapply(unique_evals, function(eval_i){
      eval_i*tcrossprod(evecs[,evals==eval_i, drop=F])
    })
    gX <- abind(gX, along=3)
    gX <- round(gX, digits=round_digits)
    hgX <- lapply(1:n, function(i){
      gX_i <- gX[,i,]
      order_i <- do.call(order, as.data.frame(gX_i))
      hgX_i <- gX_i[order_i,]
      return(hgX_i)
    })
    m <- length(unique_evals)
    gX <- lapply(1:m, function(i){
      gX[,i,]
    })
    names(gX)<- paste0("g(",X_name,")[:,", 1:m, ",:]")
    names(hgX)<- paste0("h(g(",X_name,"))[:,", 1:m, ",:]")

    return(list(X=X, gX=gX, hgX=hgX))
  }
```

```
  make_heatmap <- function(X, title){
    X_df <- as.data.frame(melt(X))
    X_df$Var2 <- max(X_df$Var2)-X_df$Var2+1
    X_df$value <- rank(X_df$value)
    plot_i <- ggplot(X_df, aes(x=Var1,y=Var2,fill=value)) +
      geom_tile()+
      scale_fill_continuous(type = "viridis")+
      theme_void()+
      theme(legend.position = "none",
            plot.title = element_text(hjust = 0.5))+
      ggtitle(title)
    return(plot_i)
  }

}

#toy problem visualisation
if(T){
  n <- 3
  X <- crossprod(matrix(runif(n^2),n))
  perm <- c(3,1,2)
  Y <- X[perm,perm]
  perm_mat <- matrix(0, n,n)
  perm_mat[cbind(1:n, perm)] <- 1

  hgX_vis_list <- hgf_visualisation(X, "X")
  plot_X <- make_heatmap(hgX_vis_list$X, "X")
  plots_gX <- lapply(1:n, function(i){
    make_heatmap(hgX_vis_list$gX[[i]], names(hgX_vis_list$gX)[i])
  })
  plots_hgX <- lapply(1:n, function(i){
    make_heatmap(hgX_vis_list$hgX[[i]], names(hgX_vis_list$hgX)[i])
  })

  hgY_vis_list <- hgf_visualisation(Y, "Y")
  plot_Y <- make_heatmap(hgY_vis_list$X, "Y")
  plots_gY <- lapply(1:n, function(i){
    make_heatmap(hgY_vis_list$gX[[i]], names(hgY_vis_list$gX)[i])
  })
  plots_hgY <- lapply(1:n, function(i){
    make_heatmap(hgY_vis_list$hgX[[i]], names(hgY_vis_list$hgX)[i])
  })

  empty_plot <- ggplot()+theme_void()
  plotlist_X <- list(
    plot_X, plots_gX[[1]], plots_hgX[[1]],
    empty_plot, plots_gX[[2]], plots_hgX[[2]],
    empty_plot, plots_gX[[3]], plots_hgX[[3]]
  )
  plots_X <- ggarrange(plotlist=plotlist_X)
  ggsave(plots_X,
         filename = "images/plots_X.png",
         dpi=600,
```

```
        units="in",
        width=4,
        height=4)

  plotlist_Y <- list(
    plots_hgY[[1]],plots_gY[[1]], plot_Y,
    plots_hgY[[2]], plots_gY[[2]], empty_plot,
    plots_hgY[[3]],  plots_gY[[3]], empty_plot
  )
  plots_Y <- ggarrange(plotlist=plotlist_Y)
  ggsave(plots_Y,
        filename = "images/plots_Y.png",
        dpi=600,
        units="in",
        width=4,
        height=4)
}
```