The graph isomorphism problem and the structure of walks

Péter Madarasi
Applied Mathematics MSc
Thesis

Supervisor:
Alpár Jüttner
senior research fellow
ELTE Institute of Mathematics,
Department of Operations Research

Eötvös Loránd University
Faculty of Science
Budapest, 2018

Contents

1 Introduction **4**

2 Preliminaries **8**

2.1 Notation . . . 8

2.2 Linear algebra . . . 9

2.3 Hashing . . . 10

3 Artificial labels **11**

3.1 Walk-labeling . . . 12

3.1.1 Getting rid of infinite labels . . . 13

3.1.2 Walk labels in practice . . . 15

3.2 Strong walk-labeling . . . 17

3.2.1 Strong walk labels in practice . . . 19

4 Some theoretical results **20**

4.1 Walk-labeling . . . 21

4.2 Strong walk-labeling . . . 28

4.3 Polynomial time graph isomorphism algorithm for certain graph classes . . . 30

5 Applications **32**

5.1 Node-labeling . . . 32

5.1.1 Biological graphs . . . 33

5.1.2 Regular graphs . . . 36

5.2 A sophisticated backtracking algorithm . . . 39

5.3 The (induced) subgraph isomorphism problem . . . 40

5.4 Graph fingerprints 41
5.4.1 Strong walk fingerprint 42
5.4.2 Walk fingerprint 43
5.4.3 Generalization of strong walk fingerprint 44

6 Conclusion and future work **44**

7 Acknowledgement **45**

Abstract

This thesis presents the concept of *walk-labeling* that can be used to solve the graph isomorphism problem in polynomial time combinatorially under certain conditions — which hold for a wide range of the graph pairs. It turns out that all non-cospectral graph pairs can be differentiated with a combinatorial method, furthermore, even non-isomorphic co-spectral graphs might be distinguished by combinatorially verifying certain properties of their eigenspaces. New polynomial time graph isomorphism algorithms are given under certain spectral conditions and in the case of large graph diameter.

The concept of *strong walk-labeling* is a refinement of the aforementioned labeling, which has important theoretical and practical applications. Its applications include speeding up any backtracking graph matching algorithm, and the generation of graph fingerprints, which uniquely identify all the graphs in the considered databases — including all strongly regular graphs on at most 64 vertices. They are also proved to identify all trees up to isomorphism, which — as a byproduct — gives a new isomorphism algorithm for trees. The practical importance of this fingerprint lies in significantly speeding up searching in graph databases and graph matching algorithms, which are commonly required in biological and chemical applications.

In addition to the theoretical results, computational tests have been carried out on biological, strongly regular and random graphs to practically evaluate the proposed methods.

1 Introduction

In the last decades, combinatorial structures and especially graphs have been considered with ever increasing interest, and applied to the solution of several new and revised questions. The expressiveness, the simplicity and the deep theoretical background of graphs make it one of the most useful modeling tool, which appears constantly in several seemingly independent fields, such as bioinformatics and chemistry.

Getting acquainted with the structure of complex biological systems at the molecular level is of primary importance, since protein-protein interaction, DNA-protein interaction, metabolic interaction, transcription factor binding, neuronal networks, and hormone singling networks may be better understood this way.

Many chemical and biological structures can easily be modeled this way, for instance, a molecular structure can be considered as a graph, whose nodes and edges correspond to atoms and chemical bonds, respectively. The similarity and dissimilarity of objects corresponding to nodes may be incorporated to the model by *node labels*. Understanding such networks basically requires finding specific subgraphs, thus it calls for efficient graph matching algorithms.

Other real-world fields related to some variants of graph matching include pattern recognition and machine vision [7], symbol recognition [12], and face identification [20].

While this work focuses on the graph isomorphism problem in the first place, some of the proposed methods can be applied in the case of subgraph and induces subgraph matching problems, as well.

Subgraph and induced subgraph matching problems are known to be NP-Complete [9], while the graph isomorphism problem is one of the problems in NP neither known to be in P nor NP-Complete. At the same time, polynomial-time isomorphism algorithms are known for various graph classes, like trees and planar graphs [17], bounded valence graphs [22], interval graphs [21] or permutation graphs [8]. Furthermore, an FPT algorithm has also been recently presented for the colored hypergraph isomorphism problem in [2].

Some algorithms which do not need any restrictions on the graphs are summarized below. Even though, an overall polynomial behavior may not be expected from such an alternative, they may often have good practical performance. In fact, they might be the best choice in practice even on a graph class for which polynomial algorithm is known.

The first practically efficient approach was due to *Ullmann* [33], which is a commonly used algorithm based on depth-first search with a complex heuristic for reducing the number of visited states. A major problem is its $\Theta(n^3)$ space complexity, which makes it impractical for big sparse graphs. In a recent paper, Ullmann [18] presents an improved version of this algorithm based on a bit-vector solution for the binary Constraint Satisfaction Problem.

The *Nauty* algorithm [25] transforms the two graphs to a canonical form before starting to look for an isomorphism. It has been considered as one of the fastest graph isomorphism algorithms, although graph categories were shown in which it takes exponentially many steps. This algorithm handles only the graph isomorphism problem.

The *LAD* algorithm [31] uses a depth-first search strategy and formulates the matching as a Constraint Satisfaction Problem to prune the search tree. The constraints are that the mapping has to be injective and edge-preserving, hence it is possible to handle new matching types as well.

The *RI* algorithm [5] and its variations are based on a state space representation. After reordering the nodes of the graphs, it uses some fast to execute heuristic checks without using any complex pruning rules. It seems to run really efficiently on graphs coming from biology, and won the International Contest on Pattern Search in Biological Databases [35].

Currently, one of the most commonly used algorithm is *VF2* [11], an improved version of *VF* [10], which was designed for solving pattern matching and computer vision problems, and has been one of the best overall algorithms for more than a decade. Although, it is not as fast as some of the new specialized algorithms, it is still widely used due to its simplicity and space efficiency. VF2 uses a state space representation and checks specific conditions in each state to prune the search tree.

Another variant called *VF2++* [19],[23] has recently been published. It is one of the most efficient graph matching algorithms on biological and chemical graph. This method is at least as efficient as the RI or other VF2-like methods, and it has strictly better behavior on large graphs. The main idea of VF2++ is to precompute a heuristic node order of the graph to be embedded, on which VF2 works more efficiently, and apply stronger cutting rules - which are easier to verify at the same time.

In addition to checking whether two given graphs are (sub)graph isomorphic, in many cases a given graph G is searched in a database. Instead of solving the graph isomorphism problem for G and for each graphs in the database, one might generate so called fingerprints for all graphs s.t. if two fingerprints are different, then the corresponding graphs can not be isomorphic. Now, one has to solve the graph isomorphism problem only for graphs having the same fingerprint as G does.

Graph fingerprints are widely used, and several schemes have been proposed to generate them. In [30], graph fingerprints were generated by considering the node labels of short paths. This type of fingerprints is also useful in the case of subgraph matching.

Fingerprints constructed by combining the Laplacian spectrum and the heat kernels associated to the graph Laplacian is described in [27]. In general, the strength of graph spectrum as signature is theoretically studied in [34] and [36]. The developed tool apparently works only for graph having special structures, such as distance-regular graphs. They conclude that it seems to be out of reach to answer which graphs are determined by their adjacency and Laplacian spectrum. However, the number of graphs determined by their spectrum was numerically examined up to 12 nodes in [15], and around 80% of the graphs were found to be determined by their spectrum. Note that it is conjectured that almost all graphs are determined by their spectrum.

Recently, various algorithms have been developed based on discrete time quantum walks (DTQW) or continuous time quantum walks (CTQW), aiming at distinguishing non-isomorphic graph pairs. It is well known that neither standard single-particle DTQW nor CTQW can distinguish a pair of Strongly regular graphs (SRG) of the same parameters, furthermore a constant-particle CTQW without interaction can distinguish no SRG pairs of the same parameters, see [14] and [28]. However, the distinguishing power of a variant of single-particle DTQW presented in [14] turned out to be larger than that of a standard DTQW. Namely, it generates different signatures for certain non-isomorphic SRG pairs of the same parameters, but there are SRG pairs that it fails to distinguish. In [24], CTQW were shown to be less powerful than DTQW as far as the graph isomorphism is concerned.

On the other hand, a state-of-the-art quantum walk method using interacting bosons turned out to distinguish all SRG's on at most 64 vertices [16]. In this light, it is especially precious that the fingerprint introduced in Section 5 distinguishes all the mentioned SRG's, in addition, it provides a compact description of the graphs.

This work presents the concept of walk-labeling, which can be used to solve the graph isomorphism problem combinatorially in polynomial time under certain conditions - which hold for a wide range of the graph pairs. All non-cospectral graph pairs are proved to be distinguished by the proposed combinatorial method (without computing the graph spectra). Furthermore, even if the graphs are cospectral and non-isomorphic, various conditions are shown to ensure that the graphs are distinguished. Polynomial time isomorphism algorithm will be given for graphs of large diameter.

For practical purposes, a refinement of the aforementioned labeling called strong walk-labeling is also introduced. Its applications include speeding up any backtracking-based graph matching algorithm, which will be tested by counting the number of backtracks the VF2 algorithm takes on different graph classes. Another important application is a fingerprint generation method based on strong walk-labeling, which was able to uniquely identify all the graphs in the considered graph databases — including all the known strongly regular graphs. Therefore, it is competitive with the state-of-the-art quantum walk algorithms and compress all information about the graph in a short fingerprint, as well. Note that the strongly regular graphs are well known as possibly the hardest instances of the graph isomorphism problem.

The rest of the thesis is structured as follows. Section 2 introduces the most important notations and results that will be used throughout this work. Section 3 defines the artificial labels, the fundamental concept of this work. Theoretical results demonstrating the strength of the proposed artificial labels are described in Section 4. The most important applications are presented in Section 5, including the generation of unique graph identifiers. The thesis is concluded by a brief summary of the open questions.

2 Preliminaries

This section introduces the notations, and some of the main results that will be used throughout this thesis.

2.1 Notation

As usual, sets are described in curly brackets, and multisets are described in curly brackets followed by a superscript hash character. For example, $\{1, 2, 3\}$ denotes the set consisting of the numbers 1,2,3, and $\{1, 1, 2, 3\}^{\#}$ denotes the multiset consisting of numbers 1,1,2,3.

Let $\mathbb{N}$ denote the non-negative integer numbers.

For a positive integer n, let $[n]$ denote the set $\{i \in \mathbb{N} : 1 \leq i \leq n\}$.

Throughout the thesis $G = (V, E)$, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ denote three arbitrary loop-free undirected graphs with $n > 1$ nodes, where V, V_1, V_2 denotes the node sets and E, E_1, E_2 the edge sets, respectively. For the sake of simplicity, the node sets are assumed to be $[n]$, that is $V = V_1 = V_2 = [n]$. The adjacency matrices of these graphs are $A, A_1, A_2 \in \{0, 1\}^{n \times n}$, respectively.

Unless stated otherwise, the presented results apply to graphs having loops, as well. Note that node labeled graphs can be modeled by adding loops, and clearly, even if the graph has both loops and node labels, there is a compact way to encode them using loops only. Therefore node labels can be omitted likewise.

Let $\Gamma_G(i)$ denote the set of the neighbors of node i in graph G.

Finally, let $\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$ denote the Kronecker delta.

2.2 Linear algebra

For later reference, some of the well-known results in linear algebra are summarized here.

Theorem 2.2.1. *The adjacency matrix of a graph has n eigenvalues over* $\mathbb{R}$.

Theorem 2.2.2. *Let* G *be a simple graph with adjacency matrix* A*, and let* $\lambda_1 \geq \lambda_2 \geq .. \geq \lambda_n$ *denote the eigenvalues of* A*. There are* $u_1, .., u_n \in \mathbb{R}^n$ *orthonormal eigenvectors s.t.* $Au_i = \lambda_i u_i$ *for all* $i \in [n]$.

The adjacency matrices of G_1 and G_2 are denoted by A_1 and A_2, respectively. Let $\lambda_1 \geq \lambda_2 \geq .. \geq \lambda_n$ and $\mu_1 \geq \mu_2 \geq .. \geq \mu_n$ denote their eigenvalues. G_1 and G_2 are **cospectral** if the multisets of the eigenvalues of A_1 and A_2 are the same, that is $\{\lambda_i : i \in [n]\}^{\#} = \{\mu_i : i \in [n]\}^{\#}$. Let $U, V \in \mathbb{R}^{n\times n}$ orthogonal matrices (i.e. $U^T U = I$ and $V^T V = I$) s.t. $A_1 U = U \operatorname{diag}(\lambda_1, \lambda_2, .., \lambda_n)$ and $A_2 V = V \operatorname{diag}(\mu_1, \mu_2, .., \mu_n)$. U and V are called the **eigenmatrices** of G_1 and G_2, respectively. Let $u_1, u_2, ..u_n$ and $v_1, v_2, ..v_n$ denote the column vectors of U and V, respectively.

Note that V denotes both the eigenmatrix of G_2 and the node set of G, but this will not cause ambiguity.

Please note that u_{ij} denotes the j^{th} entry of eigenvector u_i, i.e. it is the entry of U in the j^{th} row and i^{th} column, where $i, j \in [n]$. Similarly, $v_{ij} = V_{ji}$ for $i, j \in [n]$.

Theorem 2.2.3. *The minimal polynomial of a real symmetric matrix* A *is* $m_A(x) = \prod_{i=1}^{p}(x - \tilde{\lambda}_i)$*, where* $\tilde{\lambda}_1, \tilde{\lambda}_2, ..\tilde{\lambda}_p$ *are the distinct eigenvalues of* A.

Theorem 2.2.4 (Perron-Frobenius)**.** *If a real square matrix has nonnegative entries, then it has a nonnegative real eigenvalue* λ *which has maximum absolute value among all eigenvalues. This eigenvalue has a nonnegative real eigenvector. If, in addition, the matrix does not contain a* $k \times (n-k)$ *block of 0-s disjoint from the diagonal, then* λ *has multiplicity 1 and the corresponding eigenvector can be chosen positive.*

The following theorem is an immediate consequence of Theorem 2.2.4.

Theorem 2.2.5. *Graph G is connected and has at least two nodes. The largest eigenvalue of the adjacency matrix of G is positive and has multiplicity one.*

The positive normalized eigenvector corresponding to the largest positive eigenvalue in Theorem 2.2.5 will be referred to as the **Perron-Frobenius eigenvector** of G.

The following lemma and its corollary will play fundamental role in the proofs of the theoretical results presented in Section 4.

Lemma 2.2.6. *For all $i, j \in [n]$ and for all $l \geq 1$, $(A^l)_{ij} = \sum\limits_{k=1}^{n} u_{ki}u_{kj}\lambda_k^l$ holds, where $\lambda_1, \lambda_2, ..\lambda_n$ are the eigenvalues of G. The right-hand side of this equation will be referred to as the **eigen decomposition**.*

Proof: $U \in \mathbb{R}^{n \times n}$ is an orthonormal matrix s.t. $AU = U\,\mathrm{diag}(\lambda_1, \lambda_2, ..\lambda_n)$. Clearly, $U^{-1}A^lU = \mathrm{diag}(\lambda_1^l, \lambda_2^l, ..\lambda_n^l)$ holds, hence $A^l = U\,\mathrm{diag}(\lambda_1^l, \lambda_2^l, ..\lambda_n^l)U^{-1}$. Therefore, $(A^l)_{ij} = \sum\limits_{k=1}^{n} u_{ki}u_{kj}\lambda_k^l$ for any node pair $i, j \in [n]$. ■

The following observation is an immediate consequence of Lemma 2.2.6.

Corollary 2.2.7. *For all $i, j \in [n]$ and $l \geq 1$, there exist $\beta_1^{ij}, \beta_2^{ij}, ..\beta_p^{ij} \in \mathbb{R}$ s.t. $(A^l)_{ij} = \sum\limits_{m=1}^{p} \beta_m^{ij}\tilde{\lambda}_m^l$, where $\tilde{\lambda}_1, \tilde{\lambda}_2, ..\tilde{\lambda}_p$ are the distinct non-zero eigenvalues of G. The the right-hand side of this equation will be referred to as the **aggregated eigen decomposition**.*

Proof: By Lemma 2.2.6, $(A^l)_{ij} = \sum\limits_{k=1}^{n} u_{ki}u_{kj}\lambda_k^l$ for $l \geq 1$ and $i, j \in [n]$. Clearly, $\beta_m^{ij} := \sum\limits_{k:\lambda_k = \tilde{\lambda}_m} u_{ki}u_{kj}$ is a proper choice, where $i, j \in [n]$ and $m \in [p]$. ■

2.3 Hashing

A function $\mathcal{H}$ mapping bitsequences to a set $U_{\mathcal{H}}$ is called **hash function**. By definition, if $\mathcal{H}(b_1) \neq \mathcal{H}(b_2)$, then $b_1 \neq b_2$ for all bitsequences b_1, b_2. The point is that there are hash functions that (practically) satisfy the other direction, and at the same time, $U_{\mathcal{H}}$ is efficient to work with — even if then the original bitsequences are exponentially long.

Assume that $U_{\mathcal{H}}$ is the set of all bitsequences of length c for a given constant $c \in \mathbb{N}$, unless otherwise stated.

For a multiset $S \subseteq U_{\mathcal{H}}$, let $\mathcal{H}(S)$ denote the bitsequence $\mathcal{H}(b_S)$, where b_S denotes the bitsequence gained by concatenating the bitsequences of set S in non-decreasing order (b_S is of length $c|S|$). Let $\mathcal{H}(a_1, a_2)$ denote $\mathcal{H}(b_{12})$, where a_1 and a_2 are s.t. $\mathcal{H}(a_1)$ and $\mathcal{H}(a_2)$ are defined, and b_{12} is the concatenation of $\mathcal{H}(a_1)$ and $\mathcal{H}(a_2)$ in this order.

Note that long bitsequences are mapped to short bitsequences, yet the injectivity of the function is required. However contradictory this seems, huge practical experience shows that the SHA512 function fulfills this requirement. On one hand, the incidental hashing errors will not influence the exactness of the proposed methods, on the other hand, a perfect hash function will be developed later for the specific applications.

3 Artificial labels

This section introduces a concept to classify the nodes of graphs by assigning different labels to them such that if two nodes can be assigned to each other in an isomorphism, then the assigned labels are equal. The aim is to find node-labelings for which the reverse direction (practically) holds, as well, i.e. if the labels of two nodes are equal, then there is an isomorphism assigning them to each other.

Definition 3.0.1. *A function $l_G : V_G \longrightarrow \mathcal{L}$ is called* ***node-labeling****, where G is a graph and $\mathcal{L}$ denotes the set of labels.*

Definition 3.0.2. *A family $\{l_G : G \text{ graph}\}$ of node-labelings is called* ***artificial labeling*** *if the following holds for all G_1, G_2 graphs. If there exist an isomorphism mapping node i to node i', then $l_{G_1}(i) = l_{G_2}(i')$ for all $i \in V_1, i' \in V_2$.*

Three simple examples are shown to demonstrate the concept of artificial labeling. The first example shows that assigning to each node its degree is an artificial labeling.

Example 3.0.1. *Let $\mathfrak{L}^1 = \{l^1_G : V_G \longrightarrow \mathbb{N} \mid G$ graph$\}$ a family of* ***node-labelings****, where $l^1_G(v) = deg_G(v), \ (v \in V_G)$.*

The following examples strengthen the previous one.

Example 3.0.2. *Let $\mathfrak{L}^2 = \{l^2_G : V_G \longrightarrow \mathbb{N} \mid G$ graph$\}$ a family of node-labelings, where $l^2_G(v) = deg(v) + \sum\limits_{w \in \Gamma(v)} deg_G(v), \ (v \in V_G)$.*

Example 3.0.3. *Let $\mathfrak{L}^3 = \{l^3_G : V_G \longrightarrow (\mathbb{N} \times 2^{\mathbb{N}}) \mid G$ graph$\}$ a family of node labels, where $l^3_G(v) = (deg(v), \{deg(w) : w \in \Gamma(v)\}), \ (v \in V_G)$.*

Observe that the artificial labelings given in the previous examples are not very strong in the sense that they assign the same label to any two nodes of a d-regular graph.

The following two sections describe two efficient artificial labelings, the main concepts of this work.

3.1 Walk-labeling

This section introduces a special artificial labeling called walk-labeling.

Let $\mathcal{L}^w = \{Q : Q \in \mathbb{N}^{n \times \mathbb{N}}\}$ be the set of labels, i.e. $\mathcal{L}^w$ consists of matrices having n rows and infinitely many columns. Two such labels, Q_1 and $Q_2 \in \mathcal{L}^w$ are said to be **permutation-equal** if there exists a permutation matrix P for which $PQ_1 = Q_2$. This relation is denoted by $Q_1 \overset{\mathrm{P}}{=} Q_2$.

Claim 3.1.1. *$\overset{\mathrm{P}}{=}$ is an equivalence relation.*

The following artificial labeling, besides its practical relevance, plays a primary role when analysing the so-called strong walk-labeling, see Section 3.2.

Notation 3.1.1. *Let $\ell_G : V_G \longrightarrow \mathbb{N}^{n \times \mathbb{N}}$ be s.t. $\ell_G(i)_{jl}$ denotes the number of walks of length l between node i and node j for $l \geq 0$. In other words, the l^{th} column of matrix $\ell_G(i)$ is $A^{l-1}e_i$, where e_i is the incidence vector of node $i \in V_G$ and $l \geq 1$. The function ℓ_G will be referred to as* ***(infinite) walk-labeling****.*

The following claim easily follows from the definition of the walk-labeling.

Claim 3.1.2. *The family* $\{\ell_G \mid G \text{ graph}\}$ *is an artificial labeling if labels are compared using* $\overset{\mathrm{p}}{=}$.

The definition of walk-isomorphism follows, which plays an important role in the studies of Section 4.

Definition 3.1.1. G_1 *and* G_2 *are* ***walk-isomorphic*** *if the nodes can be relabeled s.t.* $\ell_{G_1}(i) \overset{\mathrm{p}}{=} \ell_{G_2}(i)$ *for all node* i.

Claim 3.1.3. *If two graphs are isomorphic, then they are walk-isomorphic.*

Later on, it will be shown in important special cases, that the reverse direction holds as well. Observe that it can be checked in polynomial time whether to graphs are walk-isomorphic.

3.1.1 Getting rid of infinite labels

With the matrices in $\mathcal{L}^w$ being infinite long, there is no straightforward way of checking whether two such labels are permutation-equal or not. This section reveals that it is sufficient to consider the first $n+1$ column of the label matrices.

Definition 3.1.2. *For given column vectors* $q_0, q_1, ..$ *over a field, let* $\operatorname{span}(q_0, q_1, ..)$ *denote the linear subspace spanned by the column vectors* $q_0, q_1, ...$

The following lemma will be useful in the proof of Theorem 3.1.6.

Lemma 3.1.4. *For an arbitrary real square matrix* $M \in \mathbb{R}^{n\times n}$ *and* $q_0 \in \mathbb{R}^n$ *column vector,* $\operatorname{span}(q_0, q_1, q_2, ..) = \operatorname{span}(q_0, q_1, .., q_{n-1})$, *where* $q_i := M^i q_0$ *for all* $i \geq 0$.

Proof:

Claim 3.1.5. $\operatorname{span}(q_0, q_1, ..q_i) = \operatorname{span}(q_0, q_1, ..q_{i+1}) \implies \operatorname{span}(q_0, q_1, ..q_i) = \operatorname{span}(q_0, q_1, q_2, ..)$ *for all* i.

Proof: By induction, it is sufficient to show that $q_{i+2} \in \text{span}(q_0, q_1, ..q_i)$. There exist $\alpha_0..\alpha_i$ coefficients s.t. $q_{i+1} = \sum\limits_{j=0}^{i} \alpha_j q_j$, thus $q_{i+2} = Mq_{i+1} = M(\sum\limits_{j=0}^{i} \alpha_j q_j) = \sum\limits_{j=0}^{i} \alpha_j M q_j = \sum\limits_{j=0}^{i} \alpha_j q_{j+1} \in \text{span}(q_0, q_1, ..q_{i+1}) = \text{span}(q_0, q_1, ..q_i)$. ■

By the previous claim, the first few columns of the sequence $q_0, q_1, q_2, ..$ form a basis B_k of $\text{span}(q_0, q_1, q_2, ..)$. Since the columns of B_k are all present in $q_0, q_1, ..q_{n-1}$, indeed $\text{span}(q_0, q_1, q_2, ..) = \text{span}(q_0, q_1, ..q_{n-1})$. ■

The following theorem shows that it is sufficient to consider the first few columns of the labels, i.e. only the number of short walks matters.

Notation 3.1.2. *Let $\ell_G|_k(i)$ denote the first k columns of matrix $\ell_G(i)$.*

Theorem 3.1.6. *For every graph pair G_1, G_2 and for all $i_1 \in V_1, i_2 \in V_2$*

$$\ell_{G_1}(i_1) \overset{\text{p}}{=} \ell_{G_2}(i_2) \Longleftrightarrow \ell_{G_1}|_{n+1}(i_1) \overset{\text{p}}{=} \ell_{G_2}|_{n+1}(i_2),$$

where $n = |V_1| = |V_2|$.

Proof: Let Q_1, Q_2, Q_1' and Q_2' denote the matrices $\ell_{G_1}(v_1), \ell_{G_2}(v_2), \ell_{G_1}|_{n+1}(v_1)$ and $\ell_{G_2}|_{n+1}(v_2)$, respectively.

If $Q_1 \overset{\text{p}}{=} Q_2$, then, by definition, there exists a permutation matrix P for which $PQ_1 = Q_2$. Clearly, $PQ_1 = Q_2 \Rightarrow PQ_1' = Q_2'$.

To show the other direction, suppose that $Q_1' \overset{\text{p}}{=} Q_2'$, and the columns of Q_1 and Q_2 are $q_0, q_1, q_2..$ and $q_0', q_1', q_2'..$, respectively.

Let A_1, A_2 denote the adjacency matrices of G_1 and G_2, respectively.

Since $Q_1' \overset{\text{p}}{=} Q_2'$, there exists a permutation matrix P s.t. $PQ_1' = Q_2'$, thus it is sufficient to prove that $Pq_i = q_i'$ hols for all $i \geq n+1$.
By induction, suppose that $k < i \Longrightarrow Pq_k = q_k'$ for all k. The existence of coefficients $\alpha_0, ..\alpha_{n-1}$ s.t. $q_{i-1} = \sum\limits_{j=0}^{n-1} \alpha_j q_j$ and $q_{i-1}' = \sum\limits_{j=0}^{n-1} \alpha_j q_j'$ is an immediate consequence of Lemma 3.1.4. Therefore,

$$
\begin{aligned}
Pq_i = PA_1q_{i-1} = P\sum_{j=0}^{n-1}\alpha_j A_1 q_j = \sum_{j=0}^{n-1}\alpha_j P q_{j+1} = \sum_{j=0}^{n-1}\alpha_j q'_{j+1} = \\
= \sum_{j=0}^{n-1}\alpha_j A_2 q'_j = A_2 q'_{i-1} = q'_i
\end{aligned} \tag{1}
$$

holds for all $i \geq n+1$, which had to be shown.

■

The following example shows that the previous theorem is tight in the sense that it is not always sufficient to consider the first n columns of the walk labels.

Example 3.1.1. *Let P_n denote the path of n nodes, and let P'_n denote the path of n nodes with a loop on one of its endpoints. To distinguish two loop-free endpoints of the two graphs, indeed $n+1$ columns are necessary, since their labels do not turn out to be different earlier.*

In the rest of the thesis, ℓ_G might refer to $\ell_G|_{n+1}$ or the infinite walk label.

Note that the walk label $\ell_G|_{n+1}(i)$ of a given node i can be computed in $O(nm)$ operations using a simple dynamic programming method. Furthermore, one might prove that the occurring numbers consist of polynomial many bits. Therefore it takes $O(n^2m + n^3log(n))$ steps to decide whether two graphs are walk-isomorphic by sorting the labels of both graphs.

3.1.2 Walk labels in practice

In the previous section, it has been shown that the first $n+1$ columns already provide the distinguishing power of the infinite walk-labels – and that they are necessary in some sense. This section revises the number of columns to be generated, since in a particular node, it might be sufficient to consider significantly less columns than $n+1$. Firstly, observe that Theorem 3.1.6 holds even in the following stronger form, which informally states that instead of the first $n+1$ columns of the walk label of node i, it suffices to generate the first $r(\ell_G(i))+1$ columns.

Theorem 3.1.7. *For every graph pair G_1, G_2 and for all $i_1 \in V_1, i_2 \in V_2$*

$$\ell_{G_1}(i_1) \overset{\mathrm{p}}{=} \ell_{G_2}(i_2) \iff \ell_{G_1}|_{s(i_1)}(i_1) \overset{\mathrm{p}}{=} \ell_{G_2}|_{s(i_2)}(i_2),$$

where $s(i_1) = r(\ell_{G_1}(i_1)) + 1$ and $s(i_2) = r(\ell_{G_2}(i_2)) + 1$.

The proof is similar to that of Theorem 3.1.6, therefore it is omitted. Note that the permutation equality of two matrices implies that they are of the same dimension, thus if two labels are the same in the previous theorem, then their ranks are the same, as well.

The hardest instances of the graph isomorphism problem typically only have a few distinct adjacency eigenvalues. For example, strongly regular graphs only have three distinct eigenvalues only. Intuition suggests that if a walk-label have low rank, then it has low distinguishing power, as well. Therefore it is natural to investigate the connection between the number of distinct eigenvalues and the rank of the labels. To this end, a few more notations are necessary. First of all, let $\mathrm{diam}(G, i)$ denote the longest shortest path starting from node i, i.e. $\mathrm{diam}(G, i) := \max\{\mathrm{dist}(i, j) : j \in V_G\}$, where $\mathrm{dist}(i, j)$ is the distance of nodes i and j in G. Let $D, p, \mathrm{diam}(G)$ denote the largest rank of the node labels, the number of distinct eigenvalues and the diameter of graph G (i.e. $\max\{\mathrm{diam}(G, i) : i \in V_G\}$), respectively.

The following theorem states that the number of distinct eigenvalues is an upper bound on the rank of the walk labels.

Theorem 3.1.8. $p \geq D \geq \mathrm{diam}(G) + 1$

Proof: Let Q denote a node label having the largest rank, i.e. $r(Q) = D$. By Claim 3.1.5, the first D columns of Q are linearly independent, which implies that $I, A, A^2, .., A^{D-1}$ are linearly independent.

By Theorem 2.2.3, $p = deg(m_A)$, hence $I, A, A^2, .., A^p$ are linearly dependent, thus indeed $p \geq D$.

To prove that $D \geq \mathrm{diam}(G) + 1$, observe that $r(\ell_G(i)) > \mathrm{diam}(G, i)$ at any node i, that is, the rank of $\ell_G(i)$ is larger than the length of the longest shortest path from node i. Applying this to a node i that realizes the diameter (i.e. $\mathrm{diam}(G) = \mathrm{diam}(G, i)$), one gets that $D \geq r(\ell_G(i)) > \mathrm{diam}(G, i) = \mathrm{diam}(G)$. ■

Theorem 3.1.7 and Theorem 3.1.8 together give that in general, it is sufficient to calculate the first $p+1$ columns of the walk labels, but the proof of Theorem 3.1.8 also implies that at a given node i, at least $\operatorname{diam}(G,i)+2$ columns are necessary to obtain the power of infinite walk labels.

Even when no upper bound on the rank of a certain walk label is available, one can generate the columns one by one and stop as soon as the current column is linearly dependent form the previous ones. The correctness of this method easily follows from Theorem 3.1.7 and Claim 3.1.5.

As it has already been pointed out, the logarithm of numbers occurring in the label matrices is polynomial in the size of the graph, which ensures that the walk labels are polynomially large in the size of the graph and can be calculated in polynomial time. In practice, the entries of these matrices might be calculated modulo M, where M is a reasonably large number. This simplification reduces both the running time and the memory usage, yet it does not reduce the *practical* efficiency of the node labels - provided that M is large enough.

3.2 Strong walk-labeling

This section introduces an improved version of the walk-labeling, which has a significant practical importance, and it is the basis of the graph fingerprints described in Section 5.4.

Notation 3.2.1. *Let $\mathfrak{s}_G(i)$ be an $n \times \mathbb{N}$ matrix consisting of multisets for all $i \in [n]$. Informally, the multiset in the (j,l) position of $\mathfrak{s}_G(i)$ describes the structure of the walks of length l between nodes i and j. Formally, let*

$$\mathfrak{s}_G(i)_{jl} := \begin{cases} \delta_{ij}, & \textit{if } l = 0 \\ \{\mathfrak{s}_G(i)_{i'l-1} : i' \in \Gamma_G(j)\}^{\#}, & \textit{otherwise} \end{cases} \tag{2}$$

for all $i,j \in [n]$ and for all $l \geq 0$.

$\mathfrak{s}_G(i)$ will be referred to as the ***(infinite) strong walk label*** *of node i.*

By the definition of $\mathfrak{s}_G$, the following claim easily follows.

Claim 3.2.1. *The family $\{\mathfrak{s}_G \mid G \text{ graph}\}$ is an artificial labeling.*

Definition 3.2.1. *G_1 and G_2 are **strongly walk-isomorphic** if the nodes can be relabeled s.t. $\mathfrak{s}_{G_1}(i) \overset{\text{p}}{=} \mathfrak{s}_{G_2}(i)$ for all node i.*

Claim 3.2.2. *If G_1 and G_2 are strongly walk-isomorphic, then they are walk-isomorphic.*

If the number of columns were set to $n + 1$, it would be ensured that strong walk-labeling is at least as strong as walk-labeling. The question arises naturally whether generating more than $n + 1$ columns gives a finer node-labeling. At first sight, the conjecture seems to be hard to prove, because linear algebra tools are out of reach - unlike in the case of walk-labels. Luckily, an elementary proof can be given to the following claim, which states that it is sufficient to consider the first $n+1$ columns in the case of strong walk-labels, as well.

Claim 3.2.3. *For every graph pair G_1, G_2 and for all $i_1 \in V_1, i_2 \in V_2$*

$$\mathfrak{s}_{G_1}(i_1) \overset{\text{p}}{=} \mathfrak{s}_{G_2}(i_2) \iff \mathfrak{s}_{G_1}|_{n+1}(i_1) \overset{\text{p}}{=} \mathfrak{s}_{G_2}|_{n+1}(i_2),$$

where $n = |V_1| = |V_2|$.

Proof: Let $i_1 \in V_1$ and $i_2 \in V_2$ be two nodes. Clearly, if their infinite strong walk labels are permutation equal, then the first $n + 1$ columns are too.

To prove the other direction, assume that $\mathfrak{s}_{G_1}|_{n+1}(i_1) \overset{\text{p}}{=} \mathfrak{s}_{G_2}|_{n+1}(i_2)$. Observe that as the number of the considered columns increases, one gets finer and finer node partitions, where two nodes are in the same class, iff their rows are the same (considering the first few columns).

Observe that if the node partition determined by the first $k \geq 0$ columns is not strictly finer (that is the number of classes is not larger) than in the partition given by the first $k - 1$ columns, then the partition remains the same forever. On the other hand, the number of classes of a node partition is at most n, therefore there are two consecutive columns among the first $n+1$ columns which give the same node partition. Consequently, the partition given by the first n columns is final.

Let $k_1 \leq n+1$ and $k_2 \leq n+1$ the first column indeces s.t. the partitions do not get finer in $\mathfrak{s}_{G_1}(i_1)$ and $\mathfrak{s}_{G_2}(i_2)$, respectively. Observe that $k_1 = k_2$, otherwise $\mathfrak{s}_{G_1}|_{n+1}(i_1)$ and $\mathfrak{s}_{G_2}|_{n+1}(i_2)$ would not be permutation equal. Let

k denote this value. It is easy to see, that the multisets of the labels of the neighbors are the same, otherwise the partition would get finer if the $(k+1)^{th}$ column is considered too (see recursion (2)). Therefore, if the first k columns of $\mathfrak{s}_{G_1}(i_1)$ and $\mathfrak{s}_{G_2}(i_2)$ are permutation equal, then the first $k+1$ columns are permutation equal, as well. By induction, it easily follows that any column prefixes are permutation equal, as well.

Consequently, if $\mathfrak{s}_{G_1}|_{n+1}(i_1) \overset{\mathrm{P}}{=} \mathfrak{s}_{G_2}|_{n+1}(i_2)$, then $\mathfrak{s}_{G_1}(i_1) \overset{\mathrm{P}}{=} \mathfrak{s}_{G_2}(i_2)$. ■

In the rest of the work, $\mathfrak{s}_G$ might refer both to $\mathfrak{s}_G|_{n+1}$ and the infinite strong walk label.

Example 3.1.1 shows that the previous claim is tight in the sense that considering the first n columns would not be sufficient.

Note that the size of multiset $\mathfrak{s}_G(i)_{jl}$ may be exponentially large in n. The next section addresses this issue.

3.2.1 Strong walk labels in practice

To reduce the time and space complexity of the computation of matrix $\mathfrak{s}_G(i)$, a hash function $\mathcal{H}$ is used which maps any bit sequence to a constant long bitsequence. In the implementations, the SHA512 hash function was used, which maps to bitsequences of length 512.

Suppose that $\mathcal{H}$ is a practically perfect hash function, i.e. if the hash values of two arbitrarily long bitsequence are the same, then the sequences are practically the same, as well.

With such a hash function $\mathcal{H}$, for all $i, j \in [n]$ and for $l = 0..n$, let

$$\mathfrak{s}_G(i)_{jl} := \begin{cases} \mathcal{H}(\delta_{ij}), & \text{if } l = 0 \\ \mathcal{H}(\mathfrak{s}_G(i)_{jl-1}, \mathcal{H}(\{\mathfrak{s}_G(i)_{i'l-1} : i' \in \Gamma_G(j)\}^{\#})), & \text{otherwise} \end{cases} \tag{3}$$

Informally, the dynamic program counting the number of walks is hashed, so that a description of the structure of the walks is gained.

Observe that using this definition, $\mathfrak{s}_G(i)_{jn}$ describes the whole j^{th} row of $\mathfrak{s}_G(i)$, therefore to compare two rows of strong walk labels, it is sufficient to compare (and store) the last elements of the rows only.

In Section 3.1.2, it has been remarked that if a column of a walk-label linearly depends on the previous columns, then considering more columns does no increase the distinguishing power of the labels. Likewise, one can stop generating the columns of a strong walk-label as soon as the current column does not refine the current partition. The correctness of this method easily follows from the proof of Theorem 3.2.3. Note that the running time also can be reduced by calculating uniformly fewer columns possibly at the expense of the distinguishing power of the labels.

Also note that $\mathfrak{s}_G(i)$ denotes a matrix consisting of multisets, and a matrix consisting of bitsequences, as well. Keeping in mind that $\mathcal{H}$ is practically perfect, this should cause no misunderstanding.

Note that SHA512 is — in theory — not a perfect hash function. However, one can easily develop a real perfect hash function for strong walk labels by storing and gradually extending the set of the current label-key pairs. Let the key of the empty set be 0, and the key of a new occurring label (while computing recursion (3)) is the smallest free natural number. Observe that the number and the size of label-key pairs to be stored is polynomial in the size of the graphs if each occurrence of known labels are replaced by their keys. The label-key set can be stored in a binary search tree, since a total order can be defined on the labels which allows easy-to-compute comparison. An other option is to store them in a hash table.

4 Some theoretical results

The results presented in this section show the strength of the walk labels and that of the strong walk labels. It will be shown that if two graphs have different spectra, then they are not walk-isomorphic. In spectral graph theory it is conjectured that most non-isomorphic graph pairs are not cospectral, therefore, if the conjecture holds, most non-isomorphic graph pairs can be distinguished by verifying whether they are walk-isomorphic - without computing their spectra.

To investigate the case of cospectral but non-isomorphic graphs, the walk-isomorphism will be proved to be capable of recognizing certain differences between the eigenmatrices of the graphs.

In addition, graph classes will be shown in which the graph isomorphism is equivalent to the walk-isomorphism.

The strong walk-isomorphism will turn out to be equivalent to the graph isomorphism in case of trees.

Finally, polynomial time graph isomorphism algorithms will be described for graph classes fulfilling certain spectral conditions, for trees and for graphs having large diameter.

4.1 Walk-labeling

A simple observation follows for later reference.

Claim 4.1.1. *If $\ell_{G_1}(i) \overset{\mathrm{P}}{=} \ell_{G_2}(i')$, then the number of closed walks of length l starting from $i \in V_1$ and $i' \in V_2$ are the same for all $l \geq 0$.*

Proof: By definition, there exists a permutation matrix P s.t. $P\ell_{G_1}(i) = \ell_{G_2}(i')$. Notice that the first column of $\ell_{G_1}(i)$ and $\ell_{G_2}(i')$ enforces that P maps the i^{th} row of $\ell_{G_1}(i)$ to the i'^{th} row of $\ell_{G_2}(i')$, which means that the number of closed walks from $i \in V_1$ and $i' \in V_2$ are the same for all $l \geq 0$. ■

The following theorem shows that any two non-cospectral graphs are not walk-isomorphic.

Theorem 4.1.2. *If G_1 and G_2 are walk-isomorphic, then the spectra of G_1 and G_2 are the same.*

Proof: The proof consists of two steps.

Step 1: We prove that the set of non-zero eigenvalues of G_1 and G_2 are the same.

Lemma 4.1.3. *If $\tilde{\lambda}_k$ is an eigenvalue of exactly one of G_1 and G_2, then the coefficient β_k^{ii} in the aggregated eigen decomposition is zero for all $i, k \in [n]$.*

Proof: Let $\tilde{\lambda}_1, \tilde{\lambda}_2, .., \tilde{\lambda}_r, \tilde{\theta}_{r+1}, ..\tilde{\theta}_p$ be the distinct non-zero eigenvalues of G_1, and let $\tilde{\lambda}_1, \tilde{\lambda}_2, .., \tilde{\lambda}_r, \tilde{\mu}_{r+1}, ..\tilde{\mu}_q$ be the distinct non-zero eigenvalues of G_2. Note that $\tilde\lambda_1, \tilde\lambda_2, .., \tilde\lambda_r$ are the mutual non-zero eigenvalues and $\tilde\theta_{r+1}, ..\tilde\theta_p, \tilde\mu_{r+1}, ..\tilde\mu_q$ are pairwise distinct.

For the sake of simplicity, the nodes are reindexed s.t. the identity mapping is a walk-isomorphism, i.e. $\ell_{G_1}(i) \overset{\mathrm{p}}{=} \ell_{G_2}(i)$ for all node i.

By Corollary 2.2.7, for a given pair i, j there exist coefficients $\alpha_1, \alpha_2, .., \alpha_p$, $\beta_1, \beta_2, ..\beta_q$ s.t.

$$(A_1^l)_{ij} = \sum_{k=1}^{r} \alpha_k \tilde\lambda_k^l + \sum_{k=r+1}^{p} \alpha_k \tilde\theta_k^l \tag{4}$$

and

$$(A_2^l)_{ij} = \sum_{k=1}^{r} \beta_k \tilde\lambda_k^l + \sum_{k=r+1}^{q} \beta_k \tilde\mu_k^l \tag{5}$$

for all $l \geq 1$.

The two graphs being walk-isomorphic, one gets that

$$\sum_{k=1}^{r} \alpha_k \tilde\lambda_k^l + \sum_{k=r+1}^{p} \alpha_k \tilde\theta_k^l = (A_1^l)_{ii} = (A_2^l)_{ii} = \sum_{k=1}^{r} \beta_k \tilde\lambda_k^l + \sum_{k=r+1}^{q} \beta_k \tilde\mu_k^l \tag{6}$$

holds for all $i \in [n]$ and $l \geq 1$, where the second equality follows from Claim 4.1.1.

Subtracting the right-hand side, the following equations are gained for all $l \geq 1$.

$$\sum_{k=1}^{r} (\alpha_k - \beta_k) \tilde\lambda_k^l + \sum_{k=r+1}^{p} \alpha_k \tilde\theta_k^l - \sum_{k=r+1}^{q} \beta_k \tilde\mu_k^l = 0 \tag{7}$$

The following linear equations will be examined for all $l \in [m]$, where $m := p + q - r$.

$$\sum_{k=1}^{r} x_k \tilde\lambda_k^l + \sum_{k=r+1}^{p} x_k \tilde\theta_k^l + \sum_{k=r+1}^{q} x_{p+k-r} \tilde\mu_k^l = 0, \tag{8}$$

where for all $s \in [m]$

$$x_s := \begin{cases} \alpha_s - \beta_s, & \text{if } 1 \leq s \leq r \\ \alpha_s, & \text{if } r+1 \leq s \leq p \\ -\beta_{r+s-p}, & \text{if } p+1 \leq s \leq p+q-r \end{cases} \tag{9}$$

The matrix of this equality system is

$$M := \begin{bmatrix} \tilde{\lambda}_1^1 & \dots & \tilde{\lambda}_r^1 & \tilde{\theta}_{r+1}^1 & \dots & \tilde{\theta}_p^1 & \tilde{\mu}_{r+1}^1 & \dots & \tilde{\mu}_q^1 \\ \tilde{\lambda}_1^2 & \dots & \tilde{\lambda}_r^2 & \tilde{\theta}_{r+1}^2 & \dots & \tilde{\theta}_p^2 & \tilde{\mu}_{r+1}^2 & \dots & \tilde{\mu}_q^2 \\ \tilde{\lambda}_1^3 & \dots & \tilde{\lambda}_r^3 & \tilde{\theta}_{r+1}^3 & \dots & \tilde{\theta}_p^3 & \tilde{\mu}_{r+1}^3 & \dots & \tilde{\mu}_q^3 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \tilde{\lambda}_1^m & \dots & \tilde{\lambda}_r^m & \tilde{\theta}_{r+1}^m & \dots & \tilde{\theta}_p^m & \tilde{\mu}_{r+1}^m & \dots & \tilde{\mu}_q^m \end{bmatrix}. \tag{10}$$

Observe that $M = M' \operatorname{diag}(\tilde{\lambda}_1^1, \dots, \tilde{\lambda}_r^1, \tilde{\theta}_{r+1}^1, \dots, \tilde{\theta}_p^1, \tilde{\mu}_{r+1}^1, \dots, \tilde{\mu}_q^1)$, where M' denotes the following Vandermonde matrix.

$$M' := \begin{bmatrix} 1 & \dots & 1 & 1 & \dots & 1 & 1 & \dots & 1 \\ \tilde{\lambda}_1^1 & \dots & \tilde{\lambda}_r^1 & \tilde{\theta}_{r+1}^1 & \dots & \tilde{\theta}_p^1 & \tilde{\mu}_{r+1}^1 & \dots & \tilde{\mu}_q^1 \\ \tilde{\lambda}_1^2 & \dots & \tilde{\lambda}_r^2 & \tilde{\theta}_{r+1}^2 & \dots & \tilde{\theta}_p^2 & \tilde{\mu}_{r+1}^2 & \dots & \tilde{\mu}_q^2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \tilde{\lambda}_1^m & \dots & \tilde{\lambda}_r^m & \tilde{\theta}_{r+1}^m & \dots & \tilde{\theta}_p^m & \tilde{\mu}_{r+1}^m & \dots & \tilde{\mu}_q^m \end{bmatrix} \tag{11}$$

Therefore $\det(M) = \det(M') \prod_{k=1}^{r} \tilde{\lambda}_k \prod_{k=r+1}^{p} \tilde{\theta}_k \prod_{k=r+1}^{q} \tilde{\mu}_k \neq 0$, thus the only solution is $x \equiv 0$, that is for all $s \in [m]$

$$\begin{cases} \alpha_s = \beta_s, & \text{if } 1 \leq s \leq r \\ \alpha_s = 0, & \text{if } r+1 \leq s \leq p \\ \beta_{r+s-p} = 0, & \text{if } p+1 \leq s \leq p+q-r \end{cases} \tag{12}$$

■

Let $\lambda^* \neq 0$ denote an eigenvalue which corresponds to one of the graphs only, say to G_1.

Claim 4.1.4. *There is a node $i \in V_1$ s.t. λ^* has non-zero coefficient in the decomposition given by Corollary 2.2.7 for $(A_1^l)_{ii}$.*

Proof: Let $\tilde{m}$ denote the unique index for which $\tilde{\lambda}_{\tilde{m}} = \lambda^*$. By Corollary 2.2.7, the coefficient of $\tilde{\lambda}_{\tilde{m}}$ in the case of the number of closed walks from node i is $\beta_{\tilde{m}}^{ii} := \sum_{k:\lambda_k = \tilde{\lambda}_{\tilde{m}}} u_{ki}u_{ki}$. Let m be an index s.t. $\lambda_m = \tilde{\lambda}_{\tilde{m}}$. Let index i be s.t. $u_{mi}u_{mi} > 0$. It exists , since $u_m u_m = 1$. In addition, $\beta_{\tilde{m}}^{ii} \geq u_{mi}u_{mi} > 0$ holds, therefore node i meets the requirements. ■

Let $i \in V_1$ be a node provided by Claim 4.1.4 to λ^*. If λ^* would indeed correspond to exactly one of the graphs, then, by Lemma 4.1.3, the coefficient of λ^* in the decomposition of closed walks would be 0 contradicting to Claim 4.1.4.

Step 2: We show that the multiplicities of the eigenvalues are the same in G_1 and G_2. It is sufficient to show that the multiplicities of the non-zero eigenvalues are the same. Let $\tau_i^{(k)}$ denote the multiplicity of $\tilde{\lambda}_i$ in G_k $(k = 1, 2)$, where $\tilde{\lambda}_1, .., \tilde{\lambda}_p$ are the mutual eigenvalues of G_1 and G_2.

As a consequence of Lemma 2.2.6, the sum of the numbers of closed walks of G_k of length l is $\sum_{j=1}^{p} \tau_j^{(k)} \tilde{\lambda}_j^l$, $(l \geq 1)$. Since G_1 and G_2 are walk-isomorphic, Claim 4.1.1 applies, thus the sum of the numbers of closed walks of length l in the two graphs are the same for all l, i.e. $\sum_{j=1}^{p} \tau_j^{(1)} \tilde{\lambda}_j^l = \sum_{j=1}^{p} \tau_j^{(2)} \tilde{\lambda}_j^l$ for all $l \geq 1$. Subtracting the right-hand side provides for all $l \geq 1$ that

$$\sum_{j=1}^{p} (\tau_j^{(1)} - \tau_j^{(2)}) \tilde{\lambda}_j^l = 0 \tag{13}$$

Consider these equations for $l \in [p]$, and let $x_j := \tau_j^{(1)} - \tau_j^{(2)}$ for all $j \in [p]$. Similarly to step 1, the matrix of this equation system has non-zero determinant, thus the only solution is $x \equiv 0$, i.e. $\tau_j^{(1)} = \tau_j^{(2)}$ for all $j \in [p]$. Therefore each non-zero eigenvalue has the same multiplicities in the two graphs, which implies that the multiplicities of eigenvalue 0 is the same, as well. This means that the multisets of the eigenvalues are indeed equal.

■

An immediate consequence follows.

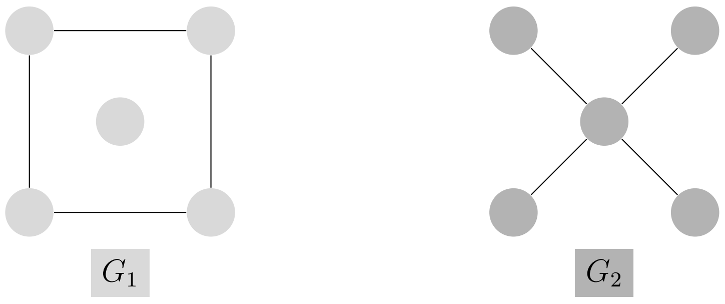
Corollary 4.1.5. *If G_1 is bipartite and G_2 is not, then they are not walk-isomorphic.*

Proof: Recall that a graph is bipartite if and only if its spectrum is symmetric about the origin, see [4, p. 53], therefore the spectra of G_1 and G_2 are different.

Note that an alternative proof could refer to Claim 4.1.1. ■

The following example shows that using walk-isomorphism, strictly more non-isomorphic graph pairs can be differentiated than by comparing their spectrum.

Example 4.1.1. *One may show that the following non-isomorphic graphs are cospectral, yet they are clearly not walk-isomorphic.*



The following theorems examine different properties of the eigenmatrices that can be verified by comparing solely the walk labels of the graphs.

Theorem 4.1.6. *If G_1 and G_2 are walk-isomorphic graphs and the nodes of G_2 are reindexed s.t. $\ell_{G_1}(i) \overset{\text{p}}{=} \ell_{G_2}(i)$ for all $i \in [n]$, then for any single eigenvalue, the corresponding normalized eigenvectors in the two graphs are element-wise the same up to sign.*

Proof: By definition, $\ell_{G_1}|_{n+1}(i) \overset{\text{p}}{=} \ell_{G_2}|_{n+1}(i)$ implies that

$$\sum_{k=1}^{n} u_{ik}u_{ik}\lambda_k^l = (A_1^l)_{ii} = (A_2^l)_{ii} = \sum_{k=1}^{n} v_{ik}v_{ik}\lambda_k^l, \tag{14}$$

thus $\forall i \in [n] : u_{ik}u_{ik} = v_{ik}v_{ik}$. That is, $|u_{ik}| = |v_{ik}|$ for all $i, k \in [n]$. Notice that this proof works even if 0 is an eigenvalue. ■

An immediate consequence follows.

Corollary 4.1.7. *If G_1 and G_2 are walk-isomorphic graphs and the nodes of G_2 are reindexed s.t. $\ell_{G_1}(i) \overset{\text{p}}{=} \ell_{G_2}(i)$ for all $i \in [n]$, and every eigenvalue is single, then the eigenmatrices are element-wise the same up to sign.*

The following perspective of the previous theorem will be useful later on.

Theorem 4.1.8. *If G_1 and G_2 are walk-isomorphic, then $\{\tilde{u}_i : i \in [n]\}^{\#} = \{\tilde{v}_i : i \in [n]\}^{\#}$ or $\{\tilde{u}_i : i \in [n]\}^{\#} = \{-\tilde{v}_i : i \in [n]\}^{\#}$, where $\tilde{u}$ and $\tilde{v}$ denote two normalized eigenvectors of G_1 and G_2, respectively, corresponding to the same single eigenvalue.*

Proof: By the definition of walk-isomorphism, the nodes of G_2 can be reindexed s.t. $\ell_{G_1}|_{n+1}(i) \overset{\mathrm{p}}{=} \ell_{G_2}|_{n+1}(i)$ for all $i \in [n]$. That is, for all $i^* \in [n]$ there exists permutation $\pi^* : V_1 \longrightarrow V_2$ s.t. $\pi^*(i^*) = i^*$ and $(A_1^l)_{i^*j} = (A_2^l)_{i^*\pi^*(j)}$ for all $j \in [n]$ and $l \geq 1$. Let i^* be chosen s.t. $\tilde{u}_{i^*} \neq 0$. Now, $\sum\limits_{k=1}^{n} u_{ki^*}u_{kj}\lambda_k^l = \sum\limits_{k=1}^{n} v_{ki^*}v_{k\pi^*(j)}\lambda_k^l$ for all $j \in [n]$, thus $\tilde{u}_{i^*}\tilde{u}_j = \tilde{v}_{i^*}\tilde{v}_{\pi^*(j)}$ for all $j \in [n]$, regardless of whether 0 is an eigenvalue or not.

Clearly, $|\tilde{u}_{i^*}| = |\tilde{v}_{i^*}|$ holds, which implies that $\tilde{v}_{i^*} \neq 0$.
Let $\sigma := \operatorname{sgn}(\tilde{u}_{i^*})\operatorname{sgn}(\tilde{v}_{i^*}) \in \{-1, 1\}$. Since $\tilde{u}_{i^*}\tilde{u}_j = \tilde{v}_{i^*}\tilde{v}_{\pi^*(j)}$ holds for all $j \in [n]$, it follows that $\tilde{u}_j = \sigma\tilde{v}_{\pi^*(j)}$ for all $j \in [n]$, which implies the theorem. ■

Theorem 4.1.9. *Let G_1 and G_2 be cospectral with single eigenvalues. If one of the eigenmatrices has a row which contains non-zero elements only, then the walk-isomorphism is equivalent to the graph isomorphism.*

Proof: Clearly, it suffices to show that if G_1 and G_2 are walk-isomorphic, then they are isomorphic.

Finding an isomorphism consists in finding a permutation matrix Π s.t. $\Pi A_1 \Pi^T = A_2$. Recall that $U = (u_1, u_2, .., u_n)$ and $V = (v_1, v_2, .., v_n)$ denote the eigenmatrices of G_1 and G_2, respectively, i.e. $A_1 = U diag(\lambda_1, .., \lambda_n)U^T$ and $A_2 = V diag(\lambda_1, .., \lambda_n)V^T$. A permutation matrix Π corresponds to a proper bijection between the nodes if and only if $\Pi U diag(\lambda_1, .., \lambda_n)U^T\Pi^T = V diag(\lambda_1, .., \lambda_n)V^T$, which holds if and only if $\Pi U = VS$ for some matrix $S = diag(\sigma_1, .., \sigma_n)$, where $\sigma_i \in \{-1, 1\}$. Therefore it is sufficient to show such matrices Π and S.

Without loss of generality, assume that the $i^{*\,th}$ row of U consists of non-zero elements.

By the definition of walk-isomorphism, there is a permutation π s.t. $(A_1^l)_{i^*j} = (A_2^l)_{\pi(i^*)\pi(j)}$, thus $u_{ki^*}u_{kj} = v_{k\pi(i^*)}v_{k\pi(j)}$ for all $j \in [n]$. Clearly, the $\pi(i^*)^{th}$ row of V consists of non-zero elements. Let $S := \operatorname{diag}(\sigma_1, .., \sigma_n)$, where $\sigma_k := \operatorname{sgn}(u_{ki^*})\operatorname{sgn}(v_{k\pi(i^*)}) \in \{-1, 1\}$, and let

$$\Pi = \begin{cases} 1, & \text{if } \pi(j) = i \\ 0, & \text{otherwise} \end{cases}. \tag{15}$$

Having said that, the following claim implies the theorem.

Claim 4.1.10. $\Pi U = VS$

Proof: The values in position (j, k) of the left and the right side are $u_{k\pi^{-1}(j)}$ and $\sigma_k v_{kj}$, respectively. $\forall j, k \in [n] : u_{k\pi^{-1}(j)} = \sigma_k v_{kj} \iff \forall j, k \in [n] : u_{kj} = \sigma_k v_{k\pi(j)} \iff \forall j, k \in [n] : u_{ki^*}u_{kj} = v_{k\pi(i^*)}v_{k\pi(j)}$, where the last equivalence holds because $u_{ki^*} = \sigma_k v_{k\pi(i^*)}$ and $\sigma_k^2 = 1$, and indeed, π was chosen s.t. $u_{ki^*}u_{kj} = v_{k\pi(i^*)}v_{k\pi(j)}$ for all $j, k \in [n]$. ∎

∎

Theorem 4.1.11. *Let G_1 and G_2 be cospectral with single eigenvalues. If $\{u_{ik} : k \in [n]\}^\# \neq \{-u_{ik} : k \in [n]\}^\#$ and $\{v_{ik} : k \in [n]\}^\# \neq \{-v_{ik} : k \in [n]\}^\#$ for all $i \in [n]$, then the walk-isomorphism is equivalent to the graph isomorphism.*

Proof: If G_1 and G_2 are isomorphic, then they are clearly walk-isomorphic.

On the other hand, let $w_1, w_2 \in \mathbb{R}^n$ be arbitrary vectors, s.t. $\{w_{1k} : k \in [n]\}^\# \neq \{w_{2k} : k \in [n]\}^\#$. Let $w_1 \overset{\text{L}}{\succ} w_2$ mean that after non-increasingly ordering their coordinates, w_1 is lexicographically strictly larger than w_2.

Without loss of generality, it can be assumed that $u_i \overset{\text{L}}{\succ} -u_i$ and $v_i \overset{\text{L}}{\succ} -v_i$ holds for all $i \in [n]$.

Assume that there is a walk-isomorphism realized by $\pi : V_1 \longrightarrow V_2$. By Theorem 4.1.6, $|u_{ki}| = |v_{k\pi(i)}|$ holds for all $k \in [n]$ and $i \in [n]$. By contradiction, suppose that there is an index k^* and i^* s.t. $u_{k^*i^*} \neq v_{k^*\pi(i^*)}$. Clearly, $u_{k^*i^*} = -v_{k^*\pi(i^*)}$. Let π^* denote the bijection of i^*, i.e. $u_{ki^*}u_{kj} = v_{k\pi^*(i^*)}v_{k\pi^*(j)}$ holds for all $j, k \in [n]$, even if 0 is an eigenvalue. π^* can be prescribed to satisfy $\pi^*(i^*) = \pi(i^*)$. Thus one gets that $u_{k^*i^*}u_{k^*j} = v_{k^*\pi^*(i^*)}v_{k^*\pi^*(j)}$ for all $j \in [n]$, which implies $-u_{k^*} = \pi v_{k^*}$. But $u_{k^*} \overset{\text{L}}{\succ} -u_{k^*} = \pi^* v_{k^*} \overset{\text{L}}{\succ} -v_{k^*} = \pi^{*-1} u_{k^*}$. Therefore G_1 and G_2 are indeed isomorphic. ∎

Definition 4.1.1. *A graph is friendly if it has single eigenvalues and $\mathbb{1}U$ has no zero coordinates, where U is the eigenmatrix of the graph.*

Corollary 4.1.12. *If G_1 and G_2 are friendly, then the walk-isomorphism is equivalent to the graph isomorphism.*

Proof: For all $i \in [n] : \mathbb{1}u_i \neq 0$ and $\mathbb{1}v_i \neq 0$ implies that $\{u_{ik} : k \in [n]\}^{\#} \neq \{-u_{ik} : k \in [n]\}^{\#}$ and $\{v_{ik} : k \in [n]\}^{\#} \neq \{-v_{ik} : k \in [n]\}^{\#}$, thus Theorem 4.1.11 can be applied. ■

Theorem 4.1.13. *Let G_1 and G_2 be connected cospectral graphs on at least two nodes. If the Perron-Frobenius eigenvectors of G_1 and G_2 are different, then G_1 and G_2 are not walk-isomorphic.*

Proof: Let λ_1 denote the unique largest eigenvalue, and let u_1, v_1 denote the corresponding Perron-Frobenius eigenvectors in G_1 and G_2, respectively. Theorem 2.2.5 implies u_1 and v_1 are strictly positive.

Clearly, $\lambda_1 > 0$, since the sum of the eigenvalues is the number of closed walks of length one, thus it is non-negative, therefore $\lambda_1 \leq 0$ would imply that each eigenvalue is zero. It is easy to see that a graph having zero eigenvalues only must be the empty graph, which contradicts the assumption of the theorem.

In any walk-isomorphism, $\sum\limits_{k=1}^{n} u_{ki}u_{ki}\lambda_k^l = \sum\limits_{k=1}^{n} v_{ki}v_{ki}\lambda_k^l$ holds for all $l \geq 1$ after reindexing the nodes. With the multiplicity of λ_1 being one, $u_{1i}u_{1i} = v_{1i}v_{1i}$ for all $i \in [n]$. Since both u_1 and v_1 are strictly positive, indeed $u_1 = v_1$. ■

Based on computational tests, the following conjecture holds for trees with at most 22 nodes.

Conjecture 4.1.1. *The walk-isomorphism is equivalent to the graph isomorphism on trees.*

4.2 Strong walk-labeling

First notice that since the strong walk labels are stronger than the walk labels, the results described in the previous section apply to strong walk labels, as well.

The following theorem suggests that the strength of strong walk labels do not confine to spectral properties, since it was shown in [29] that almost all trees are cospectral.

Theorem 4.2.1. *The strong walk-isomorphism is equivalent to the graph isomorphism on trees.*

Proof: Given are two strongly walk-isomorphic trees $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. For an edge $(r, p) \in E_i$, let $T_i(r, p) = (V_i(r, p), E_i(r, p))$ denote the subtree of G_i obtained as the connected component of $(V, E_i \setminus \{(r, p)\})$ containing node r.

By induction, we prove that for any edges $(r_1, p_1) \in E_1$ and $(r_2, p_2) \in E_2$ if $\mathfrak{s}_{G_1}(r_1)|_k \overset{\mathrm{p}}{=} \mathfrak{s}_{G_2}(r_2)|_k$ and $\mathfrak{s}_{G_1}(p_1)|_k \overset{\mathrm{p}}{=} \mathfrak{s}_{G_2}(p_2)|_k$ for $k = |V_1(r_1, p_1)|$, then $T_1(r_1, p_1)$ and $T_2(r_2, p_2)$ are isomorphic. Clearly, if $k = 1$ — i.e. r_1 is a leaf node in G_1 — then r_2 must also be a leaf node in G_2.

Otherwise, one gets that $\{\mathfrak{s}_{G_1}(i)|_{k-1} : i \in \Gamma_{G_1}(r_1)\}^{\#} = \{\mathfrak{s}_{G_2}(i)|_{k-1} : i \in \Gamma_{G_2}(r_2)\}^{\#}$. Thus r_1 and r_2 have the same number of neighbors and there is a one-to-one mapping $\phi : \Gamma_{G_1}(r_1) \longrightarrow \Gamma_{G_2}(r_2)$ so that v and $\phi(v)$ have same label up to the first $k-1$ columns for each $v \in \Gamma_{G_1}(r_1)$. Therefore, from the induction hypothesis, $T_1(v, r_1)$ and $T_1(\phi(v), r_2)$ are isomorphic subtrees for all $v \in \Gamma_{G_1}(r_1) \setminus p$. The isomorphism of $T_1(r_1, p_1)$ and $T_2(r_2, p_2)$ follows from this immediately.

In order to complete the proof of the theorem, let us choose an arbitrary leaf node $r_1 \in V_1$ and a node $r_2 \in V_2$ with $\mathfrak{s}_{G_1}(r_1) \overset{\mathrm{p}}{=} \mathfrak{s}_{G_2}(r_2)$. Node r_2 is also a leaf node and $\mathfrak{s}_{G_1}(p_1)|_{|V_1|-1} \overset{\mathrm{p}}{=} \mathfrak{s}_{G_2}(p_2)|_{|V_1|-1}$ for their neighbors $p_1 \in V_1$ and $p_2 \in V_2$. Applying the above claim to r_1, p_1, r_2, p_2 proves the isomorphism of G_1 and G_2. ■

Note that the proof above also provides a polynomial time algorithm for finding the isomorphism between trees. In fact, there exists a linear time algorithm to decide whether two trees are isomorphic or not, see [17].

Theorem 4.2.2. *The walk-isomorphism is equivalent to the graph isomorphism on cacti graphs that contain no intersecting cycles.*

If there is only one cycle, then the poof is similar to the case of trees - but technically more difficult. If there are at least two disjoint cycles, a fairly simple induction works. The details are omitted.

Finally, a conjecture follows which generalizes the previous two theorems.

Conjecture 4.2.1. *Two cacti graphs are isomorphic if and only if they are walk-isomorphic.*

4.3 Polynomial time graph isomorphism algorithm for certain graph classes

In this section, different graph classes are considered in which the graph isomorphism problem can be solved in polynomial time. Note that non of the described methods involves the calculation of eigenvalues or eigenvectors.

Theorem 4.3.1. *The graph isomorphism problem can be solved in $O(n^2m + n^3log(n) + n^{c+1}m)$ steps, where $c := n - \operatorname{diam}(G_1) - 1$.*

Proof: First, generate the walk labels of the two graphs in $O(n^2m)$ steps and sort the rows of each label in lexicographical order in $O(n^3log(n))$ steps. Find a node $i_1 \in V_1$ for which $\ell_{G_1}(i_1)$ has at least $\operatorname{diam}(G_1) + 1$ different rows. Such i_1 exists based on Theorem 3.1.8, in addition, it is easy to find in $O(n^3)$ steps by finding a node having walk label with as many different rows as possible. For the sake of simplicity, reorder the rows of $\ell_{G_1}(i_1)$ s.t. the last $n - c$ rows are different. This can be done in $O(n^2)$ steps.

For each node $i_2 \in V_2$, if $\ell_{G_1}(i_1) \overset{\mathrm{p}}{=} \ell_{G_2}(i_2)$, then try every permutation matrix P for which $\ell_{G_1}(i_1) = P\ell_{G_2}(i_2)$. Observe that there are at most $\prod_{k=0}^{c-1}(n - k) = O(n^c)$ different proper permutations, since if P' is s.t. $\ell_{G_1}(i_1)^T\Big|_c = P'\,\ell_{G_2}(i_2)^T\Big|_c$, then there is a unique P permutation matrix for which $P^T\Big|_c = P'^T\Big|_c$ and $\ell_{G_1}(i_1) = P\ell_{G_2}(i_2)$.

All these permutations can be enumerated in amortized constant time, in addition, to decide whether $\ell_{G_1}(i_1) = P\ell_{G_2}(i_2)$ holds for a certain permutation P takes amortized $O(n)$ steps, since only amortized constant row comparisons are needed. If P is s.t. $\ell_{G_1}(i_1) = P\ell_{G_2}(i_2)$, then, representing each permutation with an array, it can be tested in $O(m)$ time whether $A_1 = PA_2P^T$, i.e. whether P corresponds to a graph isomorphism or not. To sum up, generating and checking a permutation takes amortized $O(n+m)$ time and there are at most nn^c of them altogether.

It is easy to see that this algorithm finds each and every isomorphism between the two graphs.

The total number of steps taken $= O(n^2m + n^3log(n) + n^{c+1}m)$. ■

Theorem 4.3.2. *Let G_1 and G_2 be cospectral with single eigenvalues. If one of the eigenmatrices has a row which contains non-zero elements only, then the graph isomorphism problem can be solved is polynomial time.*

Proof: Recall that because of Theorem 4.1.9, the graph isomorphism is equivalent to the walk-isomorphism. The latter can be verified in $O(n^3log(n)+ n^2m)$ steps. ■

The next theorem generalizes Theorem 4.1.9.

Theorem 4.3.3. *Let G_1 and G_2 be cospectral with single eigenvalues. If $\exists c \in \mathbb{N}$ constant : $\exists B \in \mathbb{R}^{c\times n}$ consisting of c rows of the eigenmatrix of G_1 s.t. each column of B has exactly one non-zero element, then the graph isomorphism problem can be solved in polynomial time.*

Proof: Without loss of generality, assume that B_1 and B_2 consists of the first c rows of U and V, respectively, and that both contain exactly one non-zero entry in each of their columns.

Observe that it can be verified whether there is an isomorphism mapping node $i \in V_1$ and $i \in V_2$ together for all $i \in [c]$. To do so, it is sufficient to map the rest of the nodes together s.t. for any pair $(i_1, i_2) \in V_1 \times V_2$, the number of walks from node $i =\in [c]$ to i_1 and to i_2 are the same in G_1 and G_2 for length $l = 0..n$. If such a mapping is found then it is an isomorphism, because one can construct the matrices Π and S used in the proof of Theorem 4.1.9.

On the other hand, there are at most n^{2c} different choices for B_1 and B_2, therefore the described method for a given B_1 and B_2 can be applied for all possible combinations.

It is easy to see that if there exists an isomorphism, then it will be found. ■

5 Applications

This section describes and analyses various practical applications of the (strong) walk labels. In the implementations, a cryptographic hash function called SHA512 is used, which maps any bitsequence to a bitsequence of length 512 in linear time. The algorithms were implemented in C++ using the open-source LEMON graph and optimization library [13].

Note that one can use the perfect hash function described is Section 3.2.1 instead of SHA, even in graph databases. However, if the size and the number of the graphs are large, the set of label-key pairs to be stored –although polynomial in the size and the number of graphs– might be practically too large.

5.1 Node-labeling

Most of the state-of-the-art graph matching algorithms are backtracking algorithms, i.e. they start with an empty mapping and gradually extend it with respect to the definition of the graph isomorphism and the node labels. Typically, they use different cutting rules to prune some of the unfruitful branches. The idea is to strengthen these cutting rules using the proposed artificial labels.

Recall that a node-labeling was artificial labeling if the labels of any two nodes that can be mapped together in an isomorphism are equal. Therefore, any branch of the search tree can be pruned that maps together two nodes having different artificial labels. Any graph isomorphism algorithm that handles node-labeled graphs can easily make use of such a label by simply generating a third label of pairs which consists of the original and the (compressed) artificial label of v in a given node v.

This section analyses the practical efficiency of supporting a common graph matching algorithm, called VF2 [11], with strong walk labels. Tests have been executed on a recent biological dataset created for the International Contest on Pattern Search in Biological Datasets [35] and on random regular graphs, as well.

Note that (strong) walk-labels can be compressed using hashing to a short bitsequence s.t. two hash values are the same if and only if the corresponding matrices are permutation equal. Therefore, after preprocessing, permutation equality can be tested by comparing two small numbers.

5.1.1 Biological graphs

In this test, the graph isomorphism problem was solved on biological graphs using VF2, a common graph matching algorithm. First, all instances were solved with VF2, and afterwards, the strong walk labels were generated and given to the VF2 algorithm as additional node labels. The blue plot shows the numbers of backtracks taken by VF2, and the red one shows that when the strong walk labels were used too.

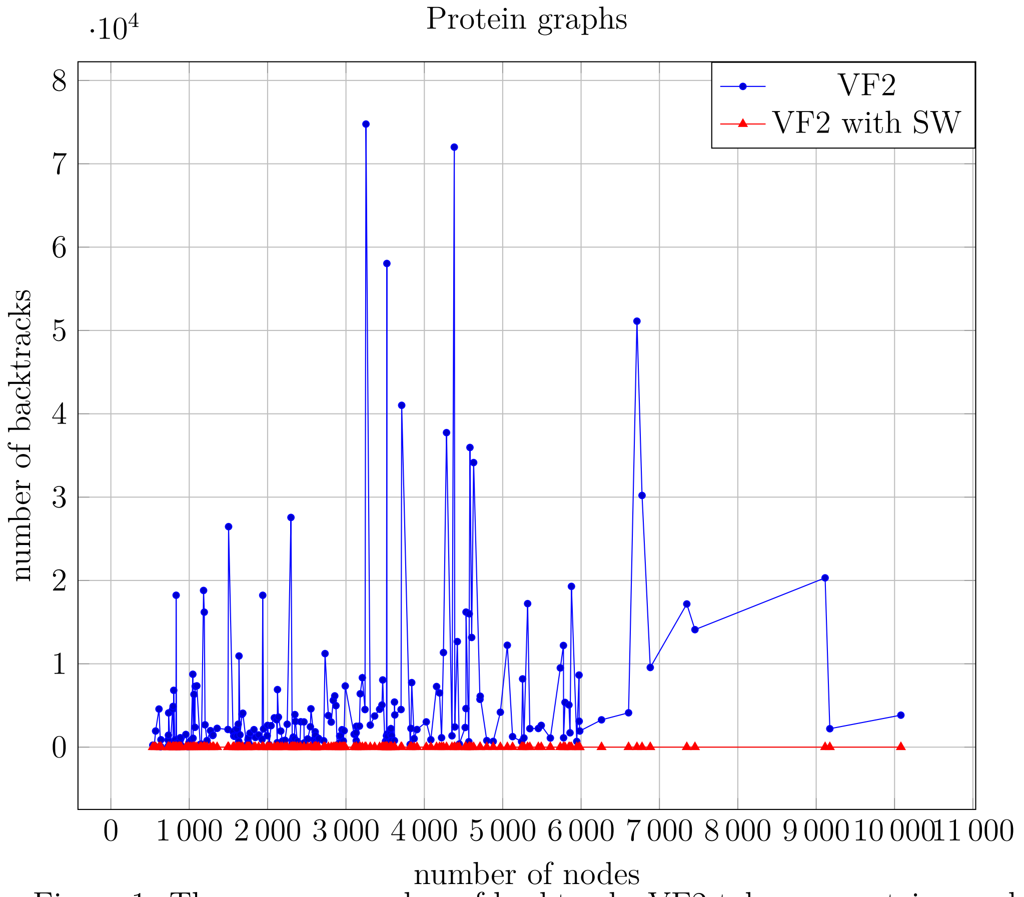


Figure 1: The average number of backtracks VF2 takes on protein graphs is 246352.6, while if it is given the strong walk labels, it never steps back. Note that for the sake of perspicuity, the following blue points were not plotted: (1312, 54910220),(3483, 1438982).

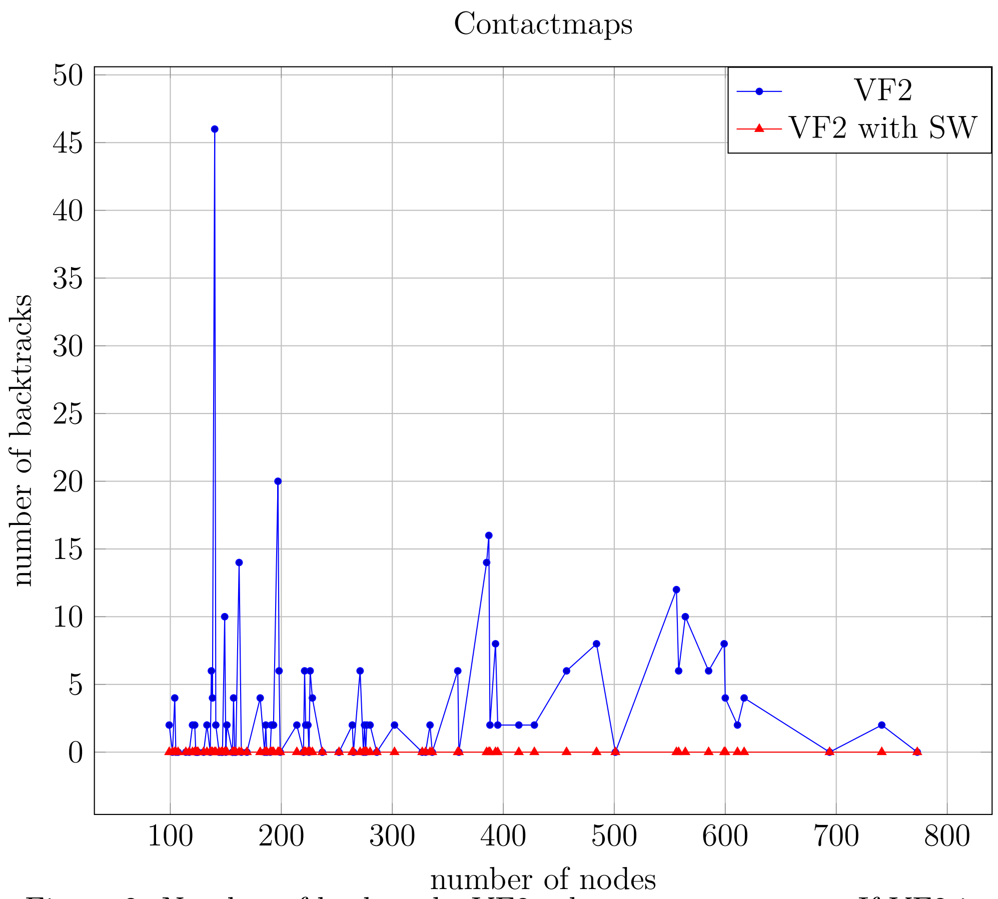


Figure 2: Number of backtracks VF2 takes on contactmaps. If VF2 is given the strong walk labels, it never steps back.

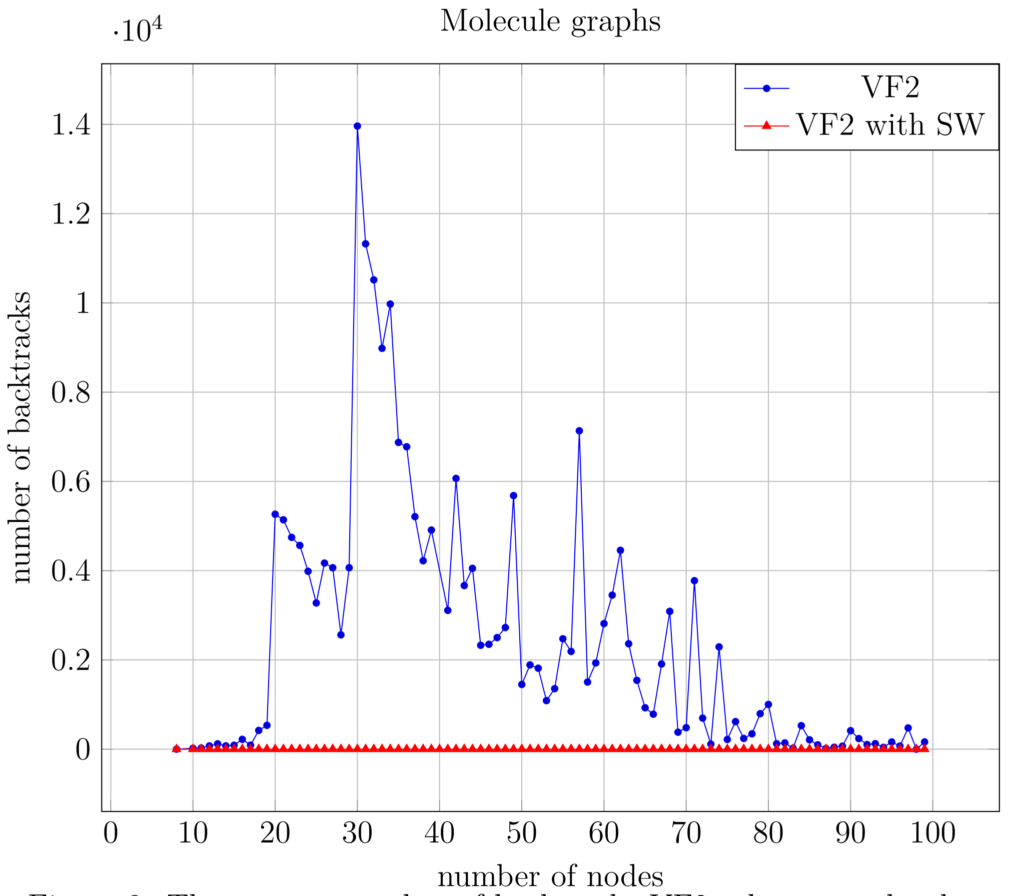


Figure 3: The average number of backtracks VF2 takes on molecule graphs is 3433.1, while if it is given the strong walk labels, it never steps back. For the sake of perspicuity, the following blue point was not plotted: (40, 95504).

5.1.2 Regular graphs

For a number $d \in \mathbb{N}$ and for each $n = 100, 110, 120, ..500$, generate 15 d-regular graphs on n nodes, and randomly permute the nodes of each graph. For each of the 15 graphs, find an isomorphism between the mixed and the original graphs. The plots show the average number of backtracks while performing the 15 searches for each n. Figure 4, 5 and 6 show the $d = 5, 10, 20$ cases, respectively.

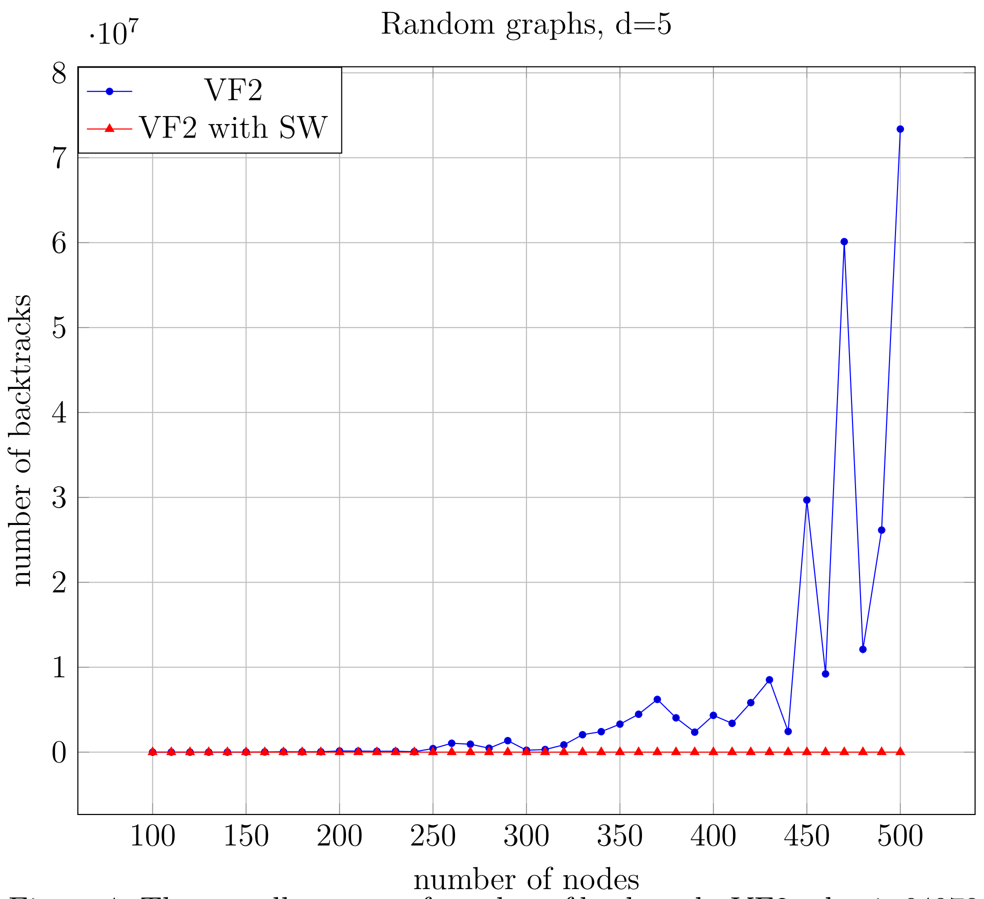


Figure 4: The overall average of number of backtracks VF2 takes is 6497890.9, while if it is given the strong walk labels, it never steps back.

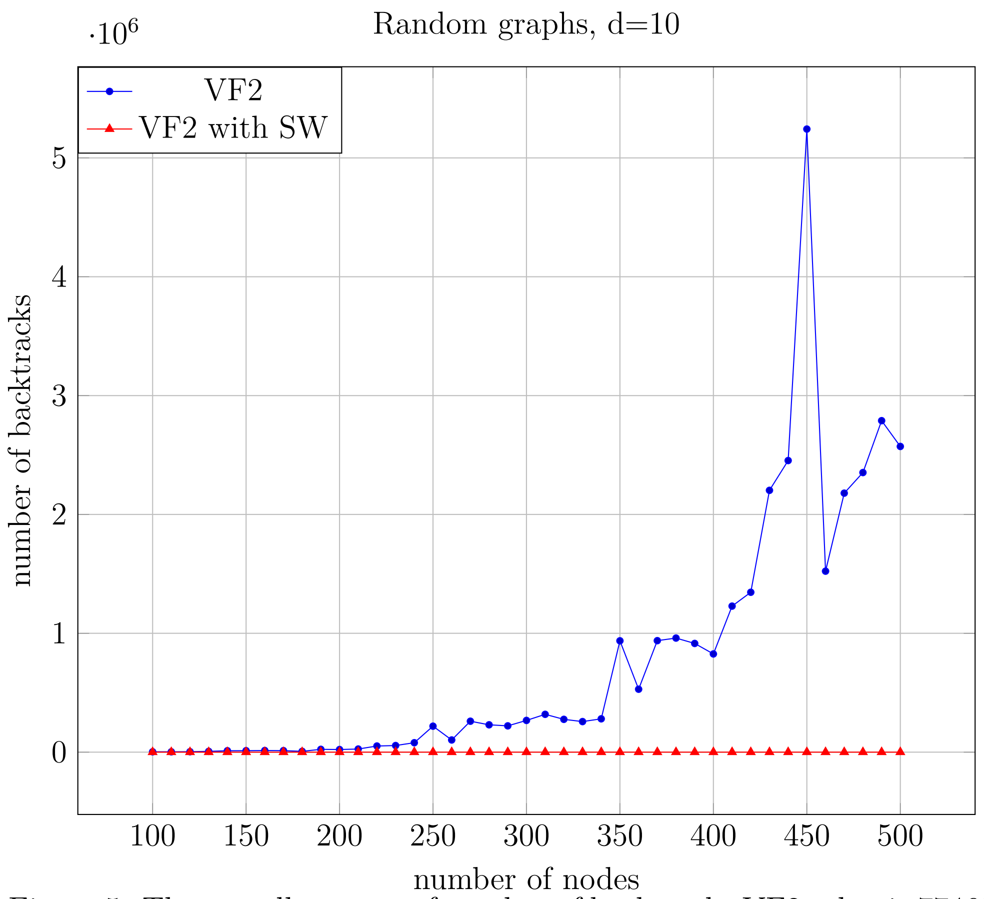


Figure 5: The overall average of number of backtracks VF2 takes is 774635.6, while if it is given the strong walk labels, it never steps back.

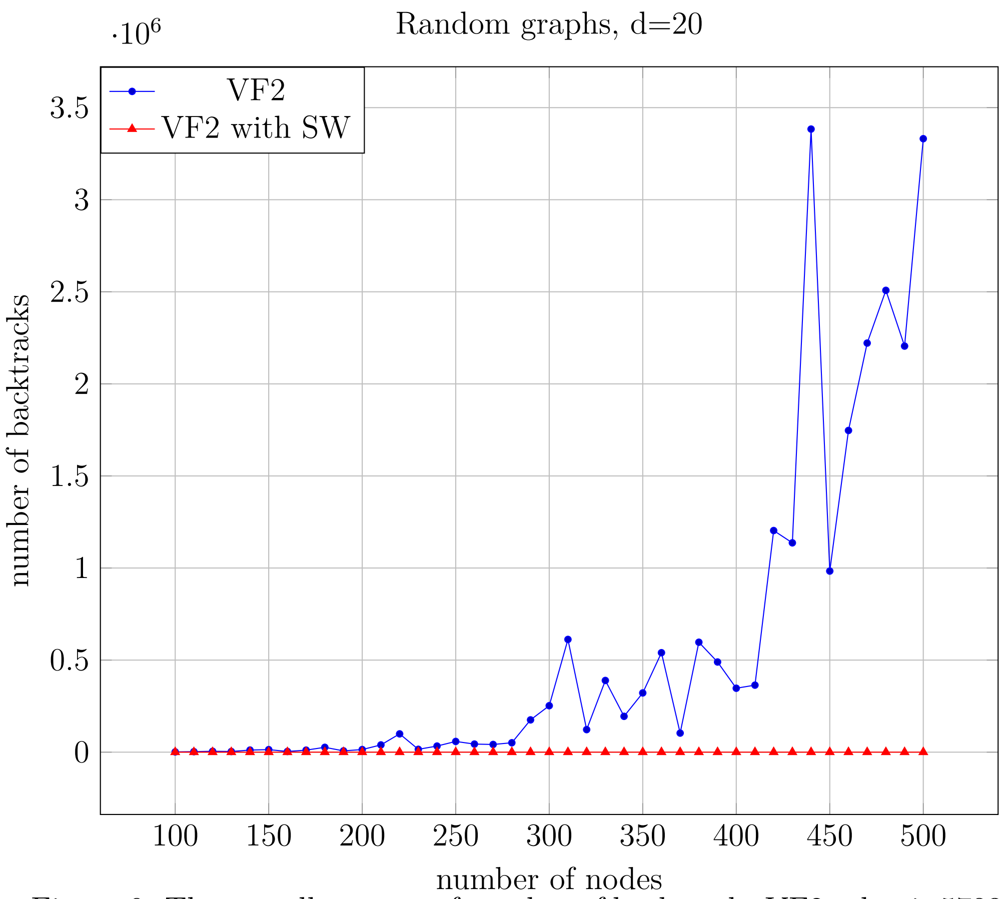


Figure 6: The overall average of number of backtracks VF2 takes is 578315.1, while if it is given the strong walk labels, it never steps back.

To sum up, VF2 never steps back if it uses the strong walk labels.

5.2 A sophisticated backtracking algorithm

Section 5.1 demonstrated that comparing the strong walk labels as an extension of the cutting rule in a backtracking algorithm significantly reduces the size of the search space on certain graph classes. However, only a small fraction of the information stemming from strong walk labels is utilized. This section introduces a refined backtracking graph isomorphism algorithm exhaustively using the strong walk labels.

Assume that the algorithm has already made a few assignments, and it is about to match node $i_1 \in V_1$ and node $i_2 \in V_2$. Based on Section 5.1, if

$\mathfrak{s}_{G_1}(i_1) \overset{p}{\neq} \mathfrak{s}_{G_2}(i_2)$, then i_1 and i_2 can be *never mapped together*, therefore the current branch can be pruned.

Observe that even if $\mathfrak{s}_{G_1}(i_1) \overset{p}{=} \mathfrak{s}_{G_2}(i_2)$, it might be the case that there are nodes $j_1 \in V_1$ and $j_2 \in V_2$ s.t. node $j_1 \in V_1$ is already mapped to $j_2 \in V_2$ and the row of $\mathfrak{s}_{G_1}(j_1)$ corresponding to node i_1 is different from the row of $\mathfrak{s}_{G_2}(j_2)$ corresponding to i_2. Clearly, the existence of such nodes j_1 and j_2 ensures that i_1 can not be mapped to i_2 *in the current branch*.

In other words, if the backtracking algorithm has already mapped certain nodes, then in this branch the permutation-equality of two node labels can be defined in a more strict way. By definition, $\mathfrak{s}_{G_1}(i_1) \overset{p}{=} \mathfrak{s}_{G_2}(i_2)$ means that there exist a permutation matrix P s.t. $P\mathfrak{s}_{G_1}(i_1) = \mathfrak{s}_{G_2}(i_2)$. Now, the permutation matrix can be prescribed to satisfy $P_{j_2 j_1} = 1$ for all $(j_1, j_2) \in \{(j_1, j_2) \in V_1 \times V_2$: j_1 *is mapped to* j_2 *in the current branch*$\}$.

It is an open question whether there exists a graph pair for which this algorithm can be executed s.t. it steps back. Computational tests have been carried out on various graph classes including distance regular graphs, strongly regular graphs and other symmetric graph constructions, but the algorithm has never stepped back. Investigating this question will be the basis of a future research.

Note that if the algorithm never stepped back under depth $O(logn)$, it would solve the graph isomorphism problem in polynomial time. To prove that it does not solve the graph isomorphism problem, a graph sequence should be constructed s.t. the maximum number of backtracks the algorithm might take tends to infinity asymptotically faster than $logn$.

The next section outlines an (induced) subgraph isomorphism algorithm along the same idea.

5.3 The (induced) subgraph isomorphism problem

Observe that if G_1 is searched in G_2 and node $i_1 \in V_1$ can be mapped to node $i_2 \in V_2$, then there exists a permutation matrix P s.t. $P\ell_{G_1}(i_1)$ is element-wise not larger than $\ell_{G_2}(i_2)$. This immediately suggest a node-labeling for the (induced) subgraph isomorphism problem.

In addition, the sophisticated backtracking algorithm described in Section 5.2 is easy to adapt to this case.

Note that there is no straightforward way to use the strong walk labels instead of the walk labels. However, it is possible to support subgraph isomorphism algorithms similarly to the method developed for isomorphism algorithms in Section 5.2. Based on preliminary computational tests, this methods works well if the sizes (and the densities) of the two graphs are comparable.

5.4 Graph fingerprints

Many practical graph matching algorithms are designed to solve the graph isomorphism problem between two graphs as fast as possible, although in typical applications, say in biology, this task is rarely needed. Instead, a graph is searched in a graph database, which involves solving the graph isomorphism problem many times. Supposing that each graph occurs only once in the database, most of the graph pairs to be compared are not isomorphic. The idea is to define a graph-labeling s.t. for any two graphs their labels are the same if the graphs are isomorphic, otherwise - apart from rare exceptions - the labels are different. Such graph-labelings are called fingerprints. Graph fingerprints can be used to reduce the number of graph isomorphism problems to be solved, since if the labels of the graphs are different, then they can not be isomorphic. In addition, if the fingerprints in a data base are precomputed, one can filter out efficiently all the graphs having the same label as the graph to be searched does by using binary search tree or hash table.

Sections 5.4.1 and 5.4.2 introduce two graph fingerprints using (strong) walk-labeling, and study their practical efficiency on graph databases containing biological, random and strongly regular graphs. Section 5.4.3 generalizes the graph fingerprint using strong walk labels.

The biological and the strongly regular graphs graphs were extracted from the Protein Data Bank [26] and from the homepage of Edward Spence [32], respectively.

5.4.1 Strong walk fingerprint

In the spirit of strong walk-labeling, matrix $\mathfrak{s}_G^k(i)$ corresponding to node i is defined such that

$$\mathfrak{s}_G^k(i)_{jl} := \begin{cases} \mathcal{H}(\delta_{ij} + 2\delta_{jk}), & \text{if } l = 0 \\ \mathcal{H}(\mathfrak{s}_G^k(i)_{jl-1}, \{\mathfrak{s}_G(i)_{i'l-1} : i' \in \Gamma_G(j)\}^{\#}), & \text{otherwise} \end{cases} \tag{16}$$

for all $i, k, j \in [n]$ and for all $l = 0..n$. Note that $\mathfrak{s}_G^k(i)$ is a strangely initialized version of $\mathfrak{s}_G(i)$.

Observe that the last column of $\mathfrak{s}_G^k(i)$ contains all information about the previous ones, therefore it is sufficient to store the last element of each row. Let $\tilde{\mathfrak{s}}_G^k(i)$ denote the multiset of the values of the last column of $\mathfrak{s}_G^k(i)$.

To define the strong walk fingerprint, let

$$\mathfrak{S}(G) := \mathcal{H}(\{\mathcal{H}(\{\mathcal{H}(\tilde{\mathfrak{s}}_G^k(i)) : i \in [n]\}^{\#}) : k \in [n]\}^{\#}) \tag{17}$$

By definition, the following claim holds.

Claim 5.4.1. *If G_1 and G_2 are isomorphic, then $\mathfrak{S}(G_1) = \mathfrak{S}(G_2)$.*

To test this graph-labeling, a graph database was considered, and the fingerprint was generated to every graph. The graph database consists of 10.000 biological graphs, 10.000 random 10-regular graphs on 1000 nodes and 43.753 strongly regular graphs.

It clearly shows the efficiency of fingerprint $\mathfrak{S}$ that all generated $\mathfrak{S}(G)$ labels were unique, i.e. two tested graphs are isomorphic if and only if their label are the same. These experiments suggest that in general, two graphs are isomorphic if and only if their labels are the same. Note that this is unlikely to hold, since, based on Section 3.2.1, this would imply a polynomial time graph isomorphism algorithm.

It is easy to see that if an oracle can decide in polynomial time whether two graphs are isomorphic, then an isomorphism can also be constructed between any to isomorphic graphs in polynomial time.

In the light of the following claim, the distinguishing power of strong walk fingerprint is especially precious.

Claim 5.4.2. *Any two strongly regular graphs with the same parameters are strongly walk-isomorphic.*

The relatively long yet straightforward proof is omitted.

To make sure of the correctness of the implementation, the following tests were carried out. For each graph of the database, generate 10 isomorphic graphs by permuting the nodes, and verify whether the 11 isomorphic instances have the same graph fingerprint.

Finally, observe that the following claim follows from Theorem 4.2.1.

Claim 5.4.3. *Two trees are isomorphic if and only if their strong walk fingerprints are equal.*

Note that the computation time of $\mathfrak{S}(G)$ labels can be reduced by considering shorter walks only – possibly at the expense of the distinguishing power. In addition, it is also easy to see that the length of considered walks can be also chosen depending on the number of vertices (or edges).

5.4.2 Walk fingerprint

To define a weaker - but easier to compute - analogue of strong walk fingerprint, let the walk fingerprint be defined as follows.

$$\mathfrak{L}(G) := \{\ell_{G_{ij}} : i, j \in [n],\ G_{ij} = (V, E \cup \{(i,i),(j,j)\})\} \tag{18}$$

The following claim is a direct consequence of the definition of the walk fingerprint.

Claim 5.4.4. *If G_1 and G_2 are isomorphic, then $\mathfrak{L}(G_1) = \mathfrak{L}(G_2)$.*

These fingerprints were calculated for each graph in the database described in Section 5.4.1, as well. All of the biological and regular graphs had unique fingerprints, but among the SRG's there were 120 non-isomorphic graph pairs having the same label.

5.4.3 Generalization of strong walk fingerprint

The equality of fingerprints given by $\mathfrak{S}(G)$ seems to be equivalent to the graph isomorphism – based on computational tests. Should it turn out to fail to distinguish two non-isomorphic graphs, a more general and stronger graph-labeling have been already developed. To describe this, first consider the following notation.

$$\mathfrak{s}_G^{k_1,..,k_q}(i)_{jl} := \begin{cases} \mathcal{H}(\delta_{ij} + \sum\limits_{r=1}^{q}(r+1)\delta_{jk_r}), & \text{if } l = 0 \\ \mathcal{H}(\mathfrak{s}_G^k(i)_{jl-1}, \{\mathfrak{s}_G(i)_{i'l-1} : i' \in \Gamma_G(j)\}^{\#}), & \text{otherwise} \end{cases} \tag{19}$$

where $q \in \mathbb{N}$ is a constant, $j, k_1, k_2, .., k_q \in [n]$ and $l = 0..n$ are such that $k_1 \leq k_2 \leq .. \leq k_q$.

Observe that the last column of $\mathfrak{s}_G^{k_1,..,k_q}(i)$ contains all information about the previous ones, therefore it is sufficient to store the last element of each row. Let $\tilde{\mathfrak{s}}_G^{k_1,..,k_q}(i)$ denote the multiset of the values of the last column of $\mathfrak{s}_G^{k_1,..,k_q}(i)$.

Finally, the new, improved graph fingerprint is defined as follows.

$$\mathfrak{S}^q(G) := \mathcal{H}(\{\mathcal{H}(\{\mathcal{H}(\tilde{\mathfrak{s}}_G^{k_1,..,k_q}(i)) : i \in [n]\}^{\#}) : k_1 \leq ... \leq k_q \in [n]\}^{\#}) \tag{20}$$

Claim 5.4.5. *If G_1 and G_2 are isomorphic, then $\mathfrak{S}^q(G_1) = \mathfrak{S}^q(G_2)$.*

6 Conclusion and future work

This thesis introduced the concepts of walk-labeling and strong walk-labeling. After working out their theoretical background, various theoretical and practical applications were presented. However, many interesting and important questions remain open. We conclude this work by presenting some of these questions and further ideas, which show the directions of the future work, as well.

- Is walk-isomorphism equivalent to graph isomorphism on graphs having single eigenvalues only?
- Is walk-isomorphism equivalent to graph isomorphism on trees?

- Does walk-isomorphism distinguish graphs having different combinatorial invariants, for instance k-connectivity?
- Which of the results hold for the Laplacian eigenvalues as well?
- There is no straightforward connection between quantum walks and the strong labeling. Does one of the two approaches dominate the other? Is there a way to combine them?
- The proposed sophisticated backtracking algorithm involves storing n^2 node labels for each graph in the data base. Is there a way to reduce the necessary space?
- Does there exist a graph sequence s.t. the proposed sophisticated backtracking algorithm might step back deeper that depth O(logn)?
- To test the distinguishing power of the fingerprints on graphs of even more graph classes.
- Disprove (or prove) that the generalized strong walk fingerprint solves the graph isomorphism problem.
- Develop sufficient conditions under which the generalized strong walk fingerprint solves the graph isomorphism problem. For example, one could consider node labeled graphs s.t. the given labels determine a "fine" partition on the node set.
- Is walk-isomorphism equivalent to graph isomorphism on cacti graphs?
- Is walk-isomorphism equivalent to graph isomorphism on planer graphs?
- Extend the methods to hypergraphs.
- The graph isomorphism problem can be modeled as an IP by searching a permutation matrix P s.t. $PA_1 = A_2P$. Using the proposed labels, one can strengthen the LP relaxation, in addition, cutting planes can be generated during a Branch & Bound method. It would be interesting to characterize the graphs for which the relaxed polytope is integer (or to develop sufficient conditions).

7 Acknowledgement

The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] QuantumBio Inc., http://www.quantumbioinc.com.

[2] V. Arvind, B. Das, J. Köbler, and S. Toda. Colored hypergraph isomorphism is fixed parameter tractable. *Algorithmica Volume 71, Pages 120-138*, January 2015.

[3] László Babai, D. Yu. Grigoryev, and David M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 310–324, New York, NY, USA, 1982. ACM.

[4] Norman Biggs. *Algebraic Graph Theory.* Cambridge University Press, 2 edition, 1994.

[5] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics. 14(Suppl 7): S13.*, April 2013.

[6] Richard A. Brualdi and Dragos Cvetkovic. *A Combinatorial Approach to Matrix Theory and Its Applications.* CRC Press, 2009.

[7] Horst Bunke. Graph matching: Theoretical foundations, algorithms, and applications. *In International Conference on Vision Interface, Pages 82-84,*, May 2000.

[8] Charles J. Colbourn. On testing isomorphism of permutation graphs. *Networks, Volume 11, Issue 1, Pages 13-21*, March 1981.

[9] S. A. Cook. The complexity of theorem-proving procedures. *Proc. 3rd ACM Symposium on Theory of Computing, Pages 151-158*, 1971.

[10] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Performance evaluation of the VF graph matching algorithm. *Proc. of the 10th ICIAP, IEEE Computer Society Press, Pages 1172-1177*, 1999.

[11] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence Volume 26 Issue 10, Page 1367-1372*, 2004.

[12] L. P. Cordella and M. Vento. Symbol recognition in documents: a collection of techniques? *International Journal on Document Analysis and Recognition Volume 3, Issue 2, Pages 73-88,*, December 2000.

[13] Balázs Dezső, Alpár Jüttner, and Péter Kovács. LEMON - an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23 – 45, jul 2011. Proceedings of the Second Workshop on Generative Technologies (WGT) 2010.

[14] Brendan L Douglas and Jingbo B Wang. A classical approach to the graph isomorphism problem using quantum walks. *Journal of Physics A: Mathematical and Theoretical*, 41(7):075303, 2008.

[15] A E. Brouwer and Edward Spence. Cospectral graphs on 12 vertices. *Electr. J. Comb.*, 16, 06 2009.

[16] John King Gamble, Mark Friesen, Dong Zhou, Robert Joynt, and S. N. Coppersmith. Two-particle quantum walks applied to the graph isomorphism problem. *Physical Review A*, 81:052313, May 2010.

[17] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs. *Proceeding STOC '74 Proceedings of the sixth annual ACM symposium on Theory of computing, Pages 172-184*, April 1974.

[18] J. R. Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *Journal of Experimental Algorithmics (JEA), Volume 15, Article No. 1.6, ACM New York, NY, USA*, 2010.

[19] Alpár Jüttner and Péter Madarasi. Vf2++—an improved subgraph isomorphism algorithm. *Discrete Applied Mathematics*, 2018.

[20] Jianzhuang Liu and Yong Tsui Lee. A graph-based method for face identification from a single 2D line drawing. *IEEE Transactions on Pattern Analysis and Machine Intelligence - Graph Algorithms and Computer Vision archive Volume 23 Issue 10, Pages 1106-1119*, October 2001.

[21] George S. Lue and Kellogg S. Booth. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM (JACM), Volume 26, Issue 2, Pages 183-195, 1979*, April 1979.

[22] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences, Volume 25, Issue 1, Pages 42-65*, August 1982.

[23] Péter Madarasi. Részgráf megfeleltetések keresése biológiai gráfokban. *OTDK*, 2017.

[24] A Mahasinghe, J A Izaac, J B Wang, and J K Wijerathna. Phase-modified ctqw unable to distinguish strongly regular graphs efficiently. *Journal of Physics A: Mathematical and Theoretical*, 48(26):265301, 2015.

[25] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium, Volume 30, Pages 45-87*, 1981.

[26] Protein Data Bank. `http://www.rcsb.org/pdb`.

[27] D. Raviv, R. Kimmel, and A. M. Bruckstein. Graph isomorphisms and automorphisms via spectral signatures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1985–1993, Aug 2013.

[28] Kenneth Rudinger, John King Gamble, Mark Wellons, Eric Bach, Mark Friesen, Robert Joynt, and Susan Coppersmith. Noninteracting multiparticle quantum random walks applied to the graph isomorphism problem for strongly regular graphs. *Phys. Rev. A*, 86, 08 2012.

[29] Allen Schwenk. Almost all trees are cospectral. In *New Directions in the Theory of Graphs*, pages 275–307, New York, NY, USA, 1973. Academic Press.

[30] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 39–52, New York, NY, USA, 2002. ACM.

[31] Christine Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artificial Intelligence 174, Pages 850-86*, 2010.

[32] Edward Spence. Strongly regular graphs on at most 64 vertices. `http://www.maths.gla.ac.uk/~es/srgraphs.php/`.

[33] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM), Volume 23 Issue 1, Pages 31-42*, January 1976.

[34] Edwin R. van Dam and Willem H. Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its Applications*, 373(Supplement C):241 – 272, 2003. Combinatorial Matrix Theory Conference (Pohang, 2002).

[35] Mario Vento, Xiaoyi Jiang, and Pasquale Foggia. International contest on pattern search in biological databases, http://biograph2014.unisa.it. June 2015.

[36] Richard C. Wilson and Ping Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41(9):2833 – 2841, 2008.