An Iterative Label-based Algorithm for Graph Isomorphism

M. D. V. Baroni, M. C. S. Boeres, M. C. Rangel[a], L. S. Buriol[b]

[a]*Departamento de Informática, Centro Tecnológico, Universidade Federal do Espírito Santo, Av. Fernando Ferrari, 514, Vitória, ES, Brazil*
[b]*Instituto de Informtíca, Universidade Federal do Rio Grande do Sul, Av. Bento Gonçalves, Porto Alegre, 9500, RS, Brazil*

Abstract

Graph invariants as spectrum and eigenvector centrality, have been used in graph theory for identification of relevant information related to graphs adjacencies. This paper proposes an Iterated Vertex Label-based (IVL) algorithm for the Graph Isomorphism Problem. Eigenvector centrality labels attached to each graph vertex are iteratively updated with adjacency information, in order to discriminate isomorphic matches.

One relevant feature of the IVL algorithm is that it can be applied to any kind of graph, including regular graphs. Computational tests over random, regular and strongly regular graphs, considering isomorphic and nonisomorphic test sets were performed and comparison results with the most well known algorithms of the literature for this problem are presented.

Eigenvector centrality was already used in our previous proposed spectral algorithm to identify isomorphic graphs, with competitive results when applied to random non regular graphs and compared to GIP state-of-art algorithms.

Keywords: Graph Isomorphism, Labeling Algorithm, Invariant property, Eigenvector Centrality

Email addresses:
marcosdaniel.baroni@gmail.com,boeres@inf.ufes.br,crangel@inf.ufes.br (M. D. V. Baroni, M. C. S. Boeres, M. C. Rangel), buriol@inf.ufrgs.br (L. S. Buriol)

1. Introduction

The Graph Isomorphism Problem (GIP) can be applied in many real-life problems. We can highlight those involving pattern and image processing, information security in social networks and identification of structural similarities in chemical compounds.

From the mentioned applications, the one with most GIP contributions in the literature is indeed the first. A known crucial operation in pattern and image recognition is the comparison between two objects or an object and a given model to which the object could be related. Graph based representations are widely used in this context. For example, see [1, 2, 3, 4, 5, 6]. Vertices of the graphs usually represent the objects in the scenes, while their edges represent the relationships between them. Relevant information for the recognition is extracted from the scene and represented as atributes of vertices and edges. *Graph matching* is one of the important problems to be solved whenever such representations are used. The goal pursued is to establish the correct association between vertices sets, according to structural similarities and objects characteristics. It can be performed for matching interests on two whole objects, an object and a subpart of another or an object and a set of models. There is a huge literature on this subject, e.g. [7, 8, 9, 10, 11, 12, 13]. According to [7], matching methods can be divided into two classes: exact and inexact methods. Exact methods requires strict constrains between the objects being compared or among subparts of them. Usually in these cases, graph and subgraphs isomorphism approaches beyond graph edit distances are applied [14, 15, 16, 17, 8]. In inexact graph matching methods, a matching can occur even if the two graphs being compared are not totally structurally identical [5, 18, 12, 9]. In [19], fingerprint identification is performed by inexact graph matching. In this work structural peculiarities of fingerprints, called *minutiae*, are mapped onto images. Each *minutiae* represent a vertex of a graph where the edges represent neighborhood relations between them. GIP approaches can be also considered for segmentation purposes [2, 1, 20]. In [2], for instance, the images are segmented into sectors considering the shapes found in the images. Each sector is represented by a vertex of a graph and the edges are labeled by their distances. Similar images are represented by all isomorphic graphs to each other. Analogous, an application of iris recognition can be found in [1].

Recent GIP application can be found in literature for information security in social networks [21, 22]. The outstanding growth of social netwoks

popularity, with its huge accumulation of user data can cause serious problems as, for example, accidentally leaking individual identifiable information in sharing tasks or public release of data anonymized. This type of data treatment should enable publication of part of information, while guaranting the privacy of identifiable information against a variety of attacks [23]. In last years, several techniques for data anonymization have been investigated by reseachers for many kinds of structured data, including graphs [24]. As an example, a private identifiable network could be viewed as a graph where each individual is a vertice and their links are represented as the graph edges. If there is exactly one isomorphism between a sharing graph of anonymous data and a public one, it could associate the anonymous information to that extracted from the public graph [21].

For the last GIP mentioned aplication, identification of structural similarities in chemical compound, it is necessary to determine whether or not a molecule has the same structure of another before giving it an exclusive name [25]. A possible way of doing it is representing the molecules as graphs where each vertex represents an atom and the edges represent its chemical bonds. By doing that we can say that the molecules have a similar structure if their graphs are isomorphic.

Classified as a decision problem, the GIP is to determine if two graphs are isomorphic [26, 27], i.e., if there is at least a mapping between vertices for which graph adjacency structures are preserved. Formally two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ of same order and size are isomorphic if there is a bijection $f : V_1 \to V_2$ such that $\forall a, b \in V_1, \{a, b\} \in E_1 \Leftrightarrow \big\{f(a), f(b)\big\} \in E_2$. We are interested in the isomorphism problem for undirected graphs. Although necessary, the conditions of same order and size are not sufficient to answer if two graphs are isomorphic.

GIP is one of the few problems that belongs to the class NP, but it is not known whether it belongs to class P or NP-complete, though it is not a co-NP problem [25, 28]. The commonly accepted assumption is that it is strictly between the two classes [29].

There are several exact algorithms in the literature to solve the GIP we can highlight for their efficiency: the Ullmann algorithm [30], VF2 [31], Nauty [32] and Bliss [33], among others. All of them improve the search for the GIP solution using different filters.

There are also examples of polynomial time algorithms dedicated to specific classes of graphs [34, 35, 36, 37]. Moreover, we can cite [38] for a Simulated Annealing approach to solve GIP.

We are motivated with the study of filters for GIP algorithms. Given a graph $G = (V, E)$, we understand a filter as a G property that enables the partition of V in groups of vertices which are equivalent with respect to the given property. Thus, isomorphism checking is restrict to groups of vertices which share the same property instantiation. In previous work, we started this investigation with the most natural, but naive filter for graph isomorphism: only vertices of same degree can be mapped by a isomorphism function [39]. As expected with this approach, best performance was achieved for denser graphs with low cardinality sets of same degree vertices. Further research in this topic lead us to investigate spectral filters.

Spectral Graph Theory (SGT) is a field of Discrete Mathematics which deals with graph properties by their matrix representation (adjacency, Laplacian and unsigned Laplacian) and spectrum [40, 41]. Concepts such as spectrum of a graph and eigenvector centrality have been used in graph theory for identification of relevant information related to graphs adjacencies.

Two different filters for isomorphism detection using eigenvector centrality are presented in [42, 43]. These filters have been stated as theoretical results and included in a tree search GIP procedure. However this procedure requires a highly eigenvector computing, at first performed by available functions of a linear algebra package, CLAPACK 3.2.1 [44] increasing significantly the algorithm CPU time. Considering the nature of adjacency matrix, we replaced these library functions by a well known and simpler method from the literature for computing eigenvectors, the power method [45]. Adopting this method, we managed to improve the whole algorithm CPU time, reducing it over 90%. The spectral algorithm and its computational results can be seen in details in [43] and are briefly described in Section 5.

In this work we propose an Iterated Vertex Labeling Algorithm for the Graph Isomorphism Problem, originated from the idea of eigenvector centrality utilization for discrimination purposes. For this, the algorithm uses invariant properties, normally derived from known invariant ones, but with more discriminative power. This algorithm can be applied even for regular graphs, a famous GIP bottleneck.

Computational tests are performed with several graph classes instances (random, regular, strongly regular, isomorphic and nonisomorphic) and results are compared with Nauty [46], VF2 [31], Bliss [33] and Saucy [47, 48].

The next section discuss some of the main algorithms for GIP, including the above mentioned ones and also highlight some theoretical aspects. Sections 3 and 4 summarize some definitions and theoretical results respectively

about spectra graph and graph invariant properties. The spectral algorithm we proposed in a previous work is briefly discussed in Section 5. The Iterative Vertex Labeling Algorithm is fully explained in Section 6. Section 7 presents computational results obtained and conclusions and future work can be found in Section 8.

2. Related work

essa seção ainda será melhor trabalhada, tentando explicar melhor o que é rotulação canônica. Buscar mais referências

Graph Isomorphism Problem is widely studied for several research groups as mathematicians, chemists and computer scientists, with relevant developments both in theory and practical applications. This section is dedicated to a briefly survey of the principal techniques and theoretical results found in the literature for the problem.

Roughly speaking, GIP algorithms can be classified into two major classes, according to the type of resolution strategy: direct approach or canonical labeling [49]. In direct approach, the algorithm searches at least for one isomorphism between two input graphs, in general performing a classical depth-first backtrack algorithm, with different techniques for prunning. As the subjacent search tree is exaustively explored, if a isomorphism is not found, then the algorithm concludes that the graphs are not isomorphic.

In canonical labeling, given two graphs G_1 and G_2, the algorithm generates a labeling $H(G_1)$ of vertices such that $H(G_1) = H(G_2)$ if and only if $G_1 \cong G_2$ [50]. Nauty [32, 46] is one of the most efficient algorithms for solving GIP. It builds iteratively a canonical vertices labeling, producing new labeling versions according to the application of a set of invariants. The implemented invariants allow the generation of ner partitions, in order to speed-up the computation for suitable classes of graphs. The algorithms Saucy [47, 48] and Bliss [33] work similarly but the implementations are different. Saucy computes the automorphism group for the input graphs, without considering the canonical labeling problem. Bliss uses efficient data structures and new heuristics for computing partitions and refinements to traverse the search space. Saucy is focused to sparse graphs and Bliss can be applied on regular graphs.

In spite of implementation aspects, each one of the mentioned algorithms are based on the same idea, summarize in the following four steps: (a) the strategy for building and pruning the search tree (depth-first), (b) the absence

of specic tools for manipulating information coming from the group of detected automorphisms, (c) the refinement procedure, known as 1-dimensional Weisfeiler-Lehman algorithm [51] or vertex classification and, (d) the target cell selector, based on local properties of vertices adjacencies.

Traces [52] is an algorithm based on different designs for dealing with these four of the aforementioned issues. A new algorithmic design computes a canonical form for a colored graph and/or a set of generators for its automorphism group. This algorithm does not use backtrack to traverse the search space. The central innovation of Traces over other canonical labeling algorithms concerns the search space construction strategy. Only part of the whole backtrack tree is actually generated. The other parts are either shown to be equivalent to those already generated, or are pruned by means of invariant information computed during the tree traverse. Automorphisms which are detected are manipulated in Traces by means of the Schreier-Sims algorithm [53]. Some information about the group structure can be also used by the refinement procedure to eliminate redundant computations. renement functions tp produce ner partitions are designed for more complex graphs.

Nishe is another canonical labeling algorithm for graphs focused in more complex GIP instances [54, 55]. This algorithm outputs generators of the graph automorphism group like canonical labeling algorithms (nauty, bliss and saucy). It can treat the exponential-family of graphs for nauty, bliss and saucy in polynomial time. It has enhancements similar to Traces, like refinement techniques to solve more complex GIP instances.

[56] proposed an algorithm with the objective of improving the basic paradigm of individualization and refinement used in canonical labeling algorithms (nauty and similar tools) with new search space pruning techniques: conflict propagation based on recorded failure information and recursion over non uniformly joined components. The proposed pruning techniques do not increase the runtime when no pruning is produced.

3. Graph Spectral Features

Spectral Graph Theory (SGT) focuses on the study of graph spectrum and other concepts via their matrix representation, which can be conceived in different ways (adjacency, Laplacian and unsigned Laplacian) [40, 41]. Eigenvector centrality is a concept concerning adjacency matrices. In this section we present definitions and results related to eigenvector centrality, a key concept to the algorithms described in the following.

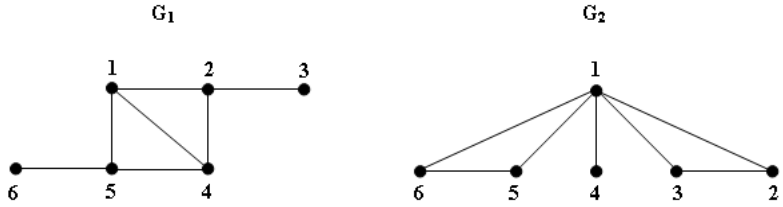


Figure 1: Cospectral graphs that are not isomorphic.

Let $G = (V, E)$ a graph with n vertices. The adjacency matrix $A(G)$ is the square matrix of order n whose entries are

$$a_{ij} = \begin{cases} 1, & \text{if } \{v_i, v_j\} \in E, \ \ \forall v_i, v_j \in V \\ 0, & \text{otherwise.} \end{cases}$$

Let v be a vertice in graph G. The set $\Gamma(v)$ consists of all adjacent vertices of v and is called neighborhood or adjacency of v. Yet, we define $\Gamma_d(v)$ the set of vertices that are distant d of v.

A multiset is a list of elements where order and multiplicities are considered. Two different multisets can have the same elements but in different order or with repeated occurrences. The notation $\{\!\{\cdot\}\!\}$ is adopted to represent a multiset in this work.

The polynomial $p_G(\lambda) = det\big(\lambda I - A(G)\big)$ is denoted characteristic polynomial of G, where λ is an eigenvalue of G when λ is root of $p_G(\lambda)$.

Consider $\lambda_1 > \ldots > \lambda_s$ the s distincts eigenvalues of G and $m(\lambda_1), \ldots,$ $m(\lambda_s)$ their respective multiplicities. The graph spectrum of G, named by $spect(G)$ is defined as the matrix

$$spect(G) = \begin{bmatrix} \lambda_1 & \cdots & \lambda_s \\ m(\lambda_1) & \cdots & m(\lambda_s) \end{bmatrix}.$$

The greatest eigenvalue of G is called index of G and denoted by $ind(G)$.

The eigenvector centrality x_i of a vertex v_i is defined as the i-th component of the nonnegative vector x associated with $ind(G)$.

Cospectral graphs are those with same spectrum. Isomorphic graphs are always cospectral, but the opposite may not be true. Figure 1 shows an example of cospectral graphs that are not isomorphic.

Wilson and Zhu [57] are concerned with the effectiveness of the spectrum of graphs as a measure of similarity between them. According to their work,

the graph spectrum is highly dependent on the form of its matrix representation. Haemers and Spence [58] have investigated the cospectrality of graphs up to order 11 and concluded that the adjacency matrix seems to be the worst in terms of producing a large number of cospectral graphs. In this sense, cospectrality seems not to be a good choice as a filter in a GIP algorithm.

Eigenvector centrality is a measure of the importance of a vertex in a graph, with respect to its conectivity. Vertices with high centralities values are conected to vertices with the same characteristic.

Thus, we started to investigate the utilization of eigenvector as a filter to isomorphism, persuaded of the importance of adjacency information. In a previous work, we proposed two theoretical results related to eigenvector centrality and isomorphism. The results are reproduced in this section and their proofs can be found in [59].

Theorem 1. *If two graphs are isomorphics to each other their eigenvector centralities are proportional.*

The theorem 1 makes possible to conclude that two graphs are not isomorphic to each other after computing their eigenvector centralities and observed that these are not proportional.

Theorem 2. *If two graphs have their eigenvector centralities components proportional and distinct from each other then the graphs are isomorphic.*

4. Graph Invariant Properties

Graph topologies own several properties which are unaffected by vertices labeling. These properties are called invariants and are shared by isomorphic graphs. Graph degrees, number of vertices and edges, distances between two vertices and diameter of a graph are some of numerous examples.

Definition 1 formalizes the concept of graph invariant property.

Definition 1. *Let P be a graph property. P is called invariant if $P(G_1) = P(G_2)$ for any pair of graphs G_1 and G_2 such that $G_1 \cong G_2$.*

Invariant properties can be stated with respect to vertices as well. In this case we call vertex invariant. Definition 2 establishes this concept.

Definition 2. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs such that $G_1 \cong G_2$, p a vertex property and $f : V_1 \mapsto V_2$ an isomorphism labeling. Property p is vertex invariant if $p(v) = p\big(f(v)\big)$, $\forall v \in V_1$.*

Vertex invariant properties can be quite useful for GIP resolution, because they may be used to guide the search for an isomorphism, since vertices must share the same invariant properties with their atributed images by an isomorphism function.

Therefore, in this work, the strategy we adopt to search for an isomorphism between a given pair of graphs, is to check vertices associations only between those which share a same given vertex invariant property. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ as input, with same number of vertices and edges and same sorted degree sequence, we organize vertices from each graph by groups with the same value of a given vertex invariant property p. More formally, we define $grp(G, p, c) = \big\{v | p(v) = c, v \in V\big\}$ and $grps(G, p) = \big\{grp(G, p, c) \big| c \in \{p(v) | v \in V\}\big\}$, respectively, as the group of vertices of a given graph $G = (V, E)$ which share the same value c of property p ($grp(G, p, c)$) and the union of all groups $grp(G, p, c)$ for a given p ($grps(G, p)$).

However, an important aspect of property p we have to take in account to avoid charging the performance of an isomorphism search, is that it must be the most effortless to compute and discriminative as possible. Furthermore, when this invariant property is complete, we manage to solve GIP more efficiently. Because of its stronger effect, some research approaches denote p as the *certificate* of the graph [49]. Definition 3 formalizes this concept.

Definition 3. *An invariant property P is called* **complete** *if, and only if $P(G_1) = P(G_2)$ implies $G_1 \cong G_2$.*

The next two theorems (theorems 3 and 4) provide mechanisms to deal with invariant properties in a graph and are used in the IVL algorithm explained in Section 6. Theorem 3 allows to determine the neighborhood of $f(v)$ in G_2 just evaluating the neighborhood of v in graph G_1. Theorem 4 guarantees that a derived property from a vertex invariant property remains vertex invariant.

Theorem 3. *Let $u \in V_1$, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ isomorphic graphs and $f : V_1 \mapsto V_2$ their labeling function. Then $\Gamma\big(f(u)\big) = \big\{f(v) | v \in \Gamma(u)\big\}$.*

Proof:
Let us assume two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ isomorphic with a labeling function $f : V_1 \mapsto V_2$ and $v \in V_1$. So $f(u) \in V_2$ is the f image of vertice $u \in V_1$. By definition (Section 3), the adjacency of $f(u)$ in G_2 is given by equation 1.

$$\Gamma\big(f(u)\big) = \big\{w\big|\big\{w, f(u)\big\} \in E_2\big\} \tag{1}$$

As f is an one-to-one mapping function, its inverse, f^{-1}, exists. Thus, as $w \in V_2$ and f maps the vertices from V_1 to V_2, we call v the vertex associated to w in graph G_1. So:

$$f^{-1}(w) = v, \text{ for some } v \in V_1 \Leftrightarrow f \circ f^{-1}(w) = f(v) \Leftrightarrow w = f(v), v \in V_1 \tag{2}$$

Replacing w by $f(v)$ (equation 2) in equation 1 we obtain:

$$\Gamma\big(f(u)\big) = \big\{f(v)\big|\big\{f(u), f(v)\big\} \in E_2\big\} \tag{3}$$

As f is an isomorphism, each $\big\{f(u), f(v)\big\} \in E_2$ has an unique correspondent edge $\{u, v\} \in E_1$. In this way, we can rewrite equation (3) as:

$$\Gamma\big(f(u)\big) = \big\{f(v)\big|\{u, v\} \in E_1\big\} \tag{4}$$

The set described in equation 4 also represents $\Gamma(u)$. Therefore, this equation can be rewritten as

$$\Gamma\big(f(u)\big) = \big\{f(v)\big|v \in \Gamma(u)\big\}$$ ■

Definition 4. *Given a graph $G = (V, E)$, $u, v \in V$ and properties p and p', we say that p' is derived from p if $p'(v) = \{p(u)|u \in \Gamma(v)\}$.*

Theorem 4. *If p is a vertex invariant property then the derived property p' is also vertex invariant.*

Proof:
Let us consider two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ isomorphic with a labeling function $f : V_1 \mapsto V_2$, $v \in V_1$ and p a vertex invariant property over G. We aim to show that p' is vertex invariant. For this, according to Definition 2, we have just to show that $p'(v) = p'(f(v))$, $\forall v \in V_1$. So,

applying definition 4 for p' over a given vertice $v \in V$ we obtain

$$p'\big(f(v)\big) = \big\{p(w)\big|w \in \Gamma\big(f(v)\big)\big\} \tag{5}$$

Similarly, we can apply the same definition to $p'(f(v))$, resulting in equation 6.

$$p'\big(f(v)\big) = \big\{p(w)\big|w \in \{f(u)|u \in \Gamma(v)\}\big\} \tag{6}$$

By theorem 3 we know that

$$\Gamma\big(f(v)\big) = \big\{f(u)\big|u \in \Gamma(v)\big\} \tag{7}$$

As $w \in \Gamma\big(f(v)\big)$, we have

$$w = \big\{f(u)\big|u \in \Gamma(v)\big\} \tag{8}$$

Replacing 8 into 6 we obtain 9:

$$p'\big(f(v)\big) = \big\{p(f(u))\big|u \in \Gamma(v)\big\} \tag{9}$$

By hypothesis, p is vertex invariant. So, $p(u) = p(f(u)$. Performing this replacement into 9, we have

$$p'\big(f(v)\big) = \big\{p(u)\big|u \in \Gamma(v)\big\} \tag{10}$$

As the respective sets of equations 5 and 10 are identical, we conclude that

$$p'\big(f(v)\big) = p'(v) \qquad \blacksquare$$

This theorem allows us to easily derive a more discriminative property from a slightly discriminative one.

5. The spectral algorithm

In this section, we summarize the main aspects related to the last version of the spectral algorithm based on eigenvector centralities that we developed and published in previous work [42, 43]. The choice of this filter to detect isomorphism was motivated mainly for two reasons: it is a graph invariant property and is more discriminative than graph degrees, since two vertices of same degree in a graph may have distinct centrality values [60]. Thus, both theoretical results presented in Section 3, theorems 1 and 2 are used as filters

in the algorithm explained here. The former is used to identify if graphs have proportional eigenvector centralities. If not, they are not isomorphic. The latter concerns graphs associated to proportional eigenvector centralities with all components distinct from each other. If so, the graphs are isomorphic. The algorithm contains three phases: (a) the graphs eigenvector centralities are computed. The purpose of computing these eigenvectors is to gather the vertices in groups of centrality, thus the associations are restricted to vertices of the same group. Until in this stage, theorem 2 is checked; (b) theorem 1 is verified and in (c) the centrality groups produced in phase 1 are used as a filter to the solutions tree built according to the exact backtracking algorithm presented in [39]. When the tree search ends in a leaf, a feasible association is found and the algorithm halts concluding the graphs are isomorphic. Otherwise, the solution tree search is exhausted ending in the tree root, no feasible association is found and the algorithms halts concluding the graphs are not isomorphic. The algorithm pseudocode is reproduced in Algorithm 1.

As already commented in the introduction of this paper, the use of an adapted version of the Power Method [45] to GIP features reduced significantly the CPU time of the eigenvector centrality calculation in Phase 1 (line 2) of Algorithm 1. The adapted Power Method is explained in Section 5.1.

5.1. The Adapted Power Method

The power method is a simple iterative method that computes the eigenvector associated with the greatest eigenvalue of a matrix [45] through a sequence of k vectors $v_i = A^i v_0$ where v_0 is an initial vector solution and A a $n \times n$ matrix.

The method receives an input matrix and a vector $\vec{v}_0$ as an initial solution and performs k iterations. In each one, a new approximated eigenvector $\vec{w}_i$ is computed multiplying the input matrix by the vector $\vec{v}_{i-1}$ computed in the previous iteration. Then the greatest component of $\vec{w}_i$ is selected and used to generate a normalized vector $\vec{v}_i$, whose greatest component is 1. The value of α_k is the approximated eigenvalue associated with eigenvector $\vec{v}_i$. After k iterations the method outputs the eigenvalue α_k and its associated eigenvector $\vec{v}_k$.

Usually the halting criterion adopted in this method is the evaluation of a residue value $r_i = |\lambda_i - \lambda_{i-1}|$ for each iteration i. If $r_i < \varepsilon \in \mathbb{R}$, for a given value ε, the algorithm halts.

Algorithm 1: **The Spectral Algorithm**

Input: Adjacency matrices $A(G_1)$ and $A(G_2)$ for G_1 and G_2
Output: true ($G_1 \simeq G_2$) or false (otherwise)

```
1  begin
2      Compute the centrality eigenvectors x⃗¹, x⃗² of G1, G2 ;  // Phase 1
3      Order x⃗¹ and x⃗² components in ascending order;
4      if x⃗¹ ≠ k x⃗², k ∈ ℝ* then
5          return false
6      else
7          if x⃗^i = (x_1^i, ..., x_n^i), x_j^i ≠ x_k^i, j,k = 1,...,n, j ≠ k, i = 1,2;
           // Phase 2
8          then
9              return true
10         else
11             Execute backtracking using vertices grouped by centrality ;
               // Phase 3
12             if found a feasible solution then
13                 return true
14             else
15                 return false
16             endif
17         endif
18     endif
19 end
```

So, after k iterations the eigenvector $\vec{v}_k$ is, as Equation 11, the initial vector v_0 multiplied by a power of A and a normalization factor.

$$v_k = \frac{1}{\alpha_k} A v_{k-1} = \frac{1}{\alpha_k} \frac{1}{\alpha_{k-1}} A^2 v_{k-2} = \underbrace{\frac{1}{\alpha_1 \alpha_2 \cdots \alpha_{k-1} \alpha_k}}_{\text{normalization factor}} A^k v_0 \tag{11}$$

This method is quite relevant to the spectral graph theory. Besides computing the eigenvector centrality of a graph together with its index, allow us to deduce the meaning of eigenvector centrality. The $v0$ vector represents a initial centrality value for each vertice, that is iteratively updated by the sum of its adjacents vertices previous centralities (performed by the product of vector v_i at iteration i by matrix A). The method achieves convergence when the vertices centrality is exactly the sum of the normalized centrality of its adjacents vertices.

However, in the context of Algorithm 1, the normalization operation is not necessary because the index value is not used, but only its associated eigenvector. Discarding this operation, we can work with the correspondent integer vectors, making the product matrix-vector faster. Moreover, improvements can be made regarding the specificity of the matrix considered (adjacency matrix) and number of iterations. The first improvement is achieved with matrix storage by lists of adjacency and vectors components represented by integers, reducing the number and time of product matrix-vector operations. The number of iterations can be reduced, considering as stopping criteria the evaluation of the difference between the number of groups of distinct vector components computed in consecutive iterations. If it does not chance, the algorithm halts. The Adapted Power Method is present in Algorithm 2.

The algorithm receives as input the graph $G = (V, E)$ and returns its integer eigenvector centrality. In line 2, the initial vector v_0 is initialized with the graph degrees. The vertices are organized in groups by their centrality value. The number of groups distincts related to v_0 is atributed to ng_0 (line 3). First iteration is performed (line 4) with the product of v_0 by the graph adjacency matrix, represented by lists of adjacency. In line 5, the number of groups related to v_1 is computed in the same way and the iterations counter k is set to one (line 6). The while loop of lines 7 to 11 computes new eigenvectors and their respectives number of distinct components groups until no improvement is achieved in the number of groups or it is equal to

Algorithm 2: The Adapted Power Method

Input: $G = (V, E)$
Output: eigenvector centrality: $\vec{v_k}$

```
1  begin
2      v_0 ← degrees(G) ;
3      ng_0 ← number of groups(v_0) ;
4      v_1 ← L(G) ⊙ v_0 ;
5      ng_1 ← number of groups(v_1) ;
6      k ← 1 ;
7      while ng_{k-1} < ng_k e ng_k < |V| do
8          v_{k+1} ← L(G) ⊙ v_k ;
9          ng_{k+1} ← number of groups(v_k) ;
10         k ← k + 1 ;
11     endw
12 end
```

$|V|$.

6. The Iterative Vertex Labeling Algorithm

In this section we propose an algorithm for solving the Graph Isomorphism Problem, called Iterative Vertex Labeling (IVL). Eigenvector centrality as a isomorphism filter (Section 5) is expanded to the use of any other suitable invariant property.

An iterative update of invariant properties defines groups of vertices with same labels for which isomorphism checking should be performed. This idea is based on the eigenvector centrality calculation procedure (adapted power method): each vertice value is updated according to the sum of the respective values of its adjacents, starting from the vertices degrees.

Actually, the adapted power method is an application of theorem 4: from an initial invariant property (vertices degrees), a new invariant property (sum of adjacent vertices degrees) is derived, using adjacency information. The sum operation is the only difference between an iteration of the method and the theorem 4 application. As an example, we can observe two different ways of labeling hypothetical vertices v_1, v_2 e v_3. First, we compute the sum of their adjacent vertices degrees (Equation 12). After, Equation 13 shows the

application of theorem 4 over the same vertices, obtaining multisets (defined in Section 3).

$$\begin{aligned} p_1(v_1) &= \sum_{u \in \Gamma(v_1)} p_0(u) = 1 + 2 + 3 = 6 \\ p_1(v_2) &= \sum_{u \in \Gamma(v_2)} p_0(u) = 1 + 1 + 4 = 6 \\ p_1(v_3) &= \sum_{u \in \Gamma(v_3)} p_0(u) = \quad 3 + 3 = 6 \end{aligned} \tag{12}$$

$$\boxed{p_1(v_1) = p_1(v_2) = p_1(v_3)}$$

$$\begin{aligned} p_1(v_1) &= \{\!\{ p_0(u) \big| u \in \Gamma(v_1) \}\!\} = \{\!\{1, 2, 3\}\!\} \\ p_1(v_2) &= \{\!\{ p_0(u) \big| u \in \Gamma(v_2) \}\!\} = \{\!\{1, 1, 4\}\!\} \\ p_1(v_3) &= \{\!\{ p_0(u) \big| u \in \Gamma(v_3) \}\!\} = \quad \{\!\{3, 3\}\!\} \end{aligned} \tag{13}$$

$$\boxed{p_1(v_1) \neq p_1(v_2) \neq p_1(v_3)}$$

Equations 12 and 13 ilustrates an example where we have loss of discriminative power when using sum operation. This justify our choice of using multisets rather than their summed values elements as vertices labels.

Briefly, the algorithm considers as input an initial invariant property. Tipically the chosen property is the degree of vertices. Then a new labeling is generated for each of the vertices from their neighbors based on Theorem 4. The vertices are then grouped by their labels and the number of generated groups is used as stopping criteria. The labeling process ends when the number of groups is equal to the total number of vertices or number of groups generated by the invariant property of the previous iteration. While the halt criteria is not achieved, the algorithm continues generating more discriminative labelings in the following iterations. The pseudocode of the IVL Algorithm is presented in Algorithm 3.

The method has as input a graph $G = (V, E)$ and an initial invariant property p_0. As output, an invariant property p_k with more discriminative power should be obtained. The **for loop** of lines 2 to 4 is performed to generate a new invariant property p_1 derived from p_0 by the application of

Algorithm 3: **The Iterative Labeling Method.**

Input: Graph: $G = (V, E)$; Initial Invariant: p_0
Output: Invariant: p_k

```
1  begin
2      for v ∈ V do
3          p_1(v) ← {p_0(u) | u ∈ Γ(v)}
4      endfor
5      k ← 1
6      while |grps(G, p_{k-1})| < |grps(G, p_k)| do
7          for v ∈ V do
8              p_{k+1}(v) ← {p_k(u) | u ∈ Γ(v)}
9          endfor
10         k ← k + 1
11     endw
12 end
```

Theorem 4 (line 3). The k iteration counter is initialized in line 5. The **while loop**, performed in lines 6 to 11 for vertices labeling purpose, is carried out until no improvement on the number of generated groups occurs from one iteration to the next. A similar **for loop** to the initial one is performed from lines 7 to 9, applying again theorem 4 in line 8 to obtain a new invariant property.

This method generates distinguished labels even to vertices which, eventually, have same eigenvector centrality values but different degrees. With this idea, we hope to find groups of vertices in a smaller number of iterations and consequentely, in a shorter execution time. An example of an iteration of the algorithm can be followed in Figure 2.

The method starts from an initial labeling of the vertices, in this case the degree of vertices (Table 1). In the first iteration, for each of the vertices, a multiset is constructed based on the degrees of its adjacents (Table 2). Then the multisets are labeled (Table 3) and the vertices grouped according to the labels generated (Table 4). We can see that in this case, all vertices are grouped into distinct sets (highlighted by red circumferences) in just one algorithm iteration.

Both the spectral (Section 5) and IVL algorithms seem to be very efficient for non regular graphs, especially for those with different centrality values

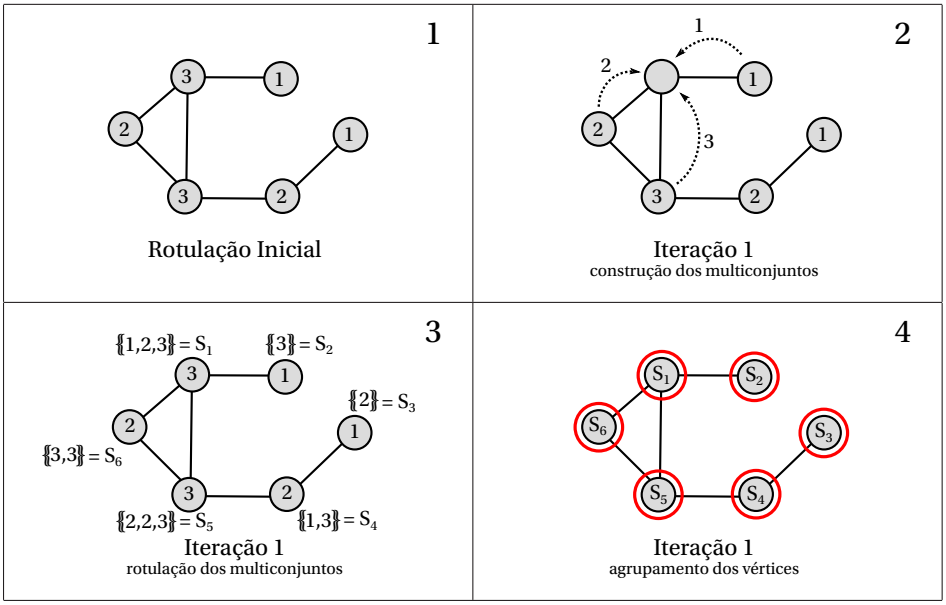


Figure 2: The Iterative Vertex Labeling Method example.

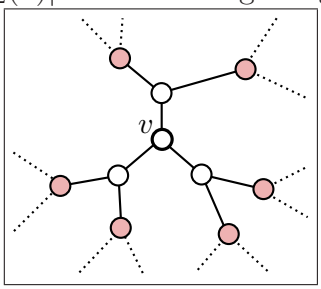
for their vertices. However, both techniques are inefficient for regular graphs. The former is inefficient because all vertices have the same spectral centrality value and the latter, based on degree, is also inefficient since all vertices have the same number of neighbors with the same degree, generating always the same labels for all vertices. A function to choice a suitable initial invariant property to IVL algorithm, according to the graph nature (regular or not) is proposed in Section 6.1.

6.1. Handling with regular graphs

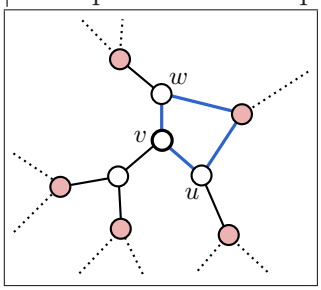
A challenge imposed in the regular graphs case is to find an invariant property with respect to the vertices, easy to be calculated and discriminative, even among vertices of same degree.

We observe that $\Gamma_d(v)$ can be a good choice for the initial invariant property, when dealing with regular graphs. For instance, Figure 3 shows that $\Gamma_2(v)$ cardinality can vary even in regular graphs. If $\Gamma_2(v)$ cardinality remains unchangeable, we compute $\Gamma_3(v)$. Given a vertex v in a 3-regular graph, Figure 3-(a) shows $\Gamma_2(v)$ consists of v neighbors which do not share any other neighbor to each other. In this case, its cardinality is equal to 6. In Figure 3-(b), we can observe the vertices w and u have a common

(a) $|\Gamma_2(v)| = 6$ in a 3-regular graph.



(b) $|\Gamma_2(v)| = 5$: presence of a square cycle.



(c) $|\Gamma_2(v)| = 4$: presence of a triangular cycle.

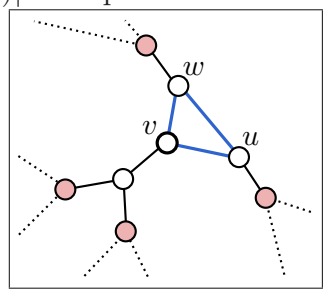


Figure 3: Three different $\Gamma_2(v)$ cardinalities for a 3-regular graph. The dotted lines represent the remaining edges of the graph.

adjacent vertex, in addition to v, so $|\Gamma_2(v)|$ is decremented by one unit. In Figure 3-(c), vertice v is the only shared neighbor of w and u. So, as they are 1-distant from v, they do not belong to $\Gamma_2(v)$, reducing its size to 4 vertices.

Therefore, we need to set p_0 according to the nature of the input graph. If it is regular, we use $\Gamma_d(v)$ as initial invariant property. Otherwise, we consider the vertices degrees. Algorithm 4 presents the pseudocode of the function that selects the invariant initial property for IVL algorithm. It receives as input a graph and selects the initial invariant property to be used in the iterative labeling method. In line 2 it is checked whether the graph

is regular. If the answer is positive, lines 4 to 5 calculate, for each vertex, the neighborhood size $\Gamma_2(v)$. In line 6 it is checked whether the size of this neighborhood is the same for all vertices. If so, lines 7 to 10 calculate the neighborhood size $\Gamma_3(v)$. In the case the graph is not regular, the degree of vertices is used as the initial invariant (lines 11 to 15).

Regarding implementation purposes, another challenge we have to deal with is to find a consistent and efficient way to label a multiset, particularly, a multiset of integers. A proposal is presented in Section 6.2.

Algorithm 4: Selection of the initial invariant property (*inv_inicial*).

```
   Input: G = (V, E)
   Output: p_0
 1 begin
 2   if G regular then
 3     for v ∈ V do
 4       p_0(v) ← |Γ_2(v)| ;
 5     endfor
 6     if p_0(u) = p_0(v), ∀u, v ∈ V, u ≠ v then
 7       for v ∈ V do
 8         p_0(v) ← |Γ_3(v)| ;
 9       endfor
10     endif
11   else
12     for v ∈ V do
13       p_0(v) ← d(v) ;
14     endfor
15   endif
16 end
```

6.2. An efficient computational representation for a multiset

A vector of integers is an intuitive way to represent a multiset computationally. However, basic operations as comparison become costly as, for multisets of size n, we must ensure the vectors are ordered and then perform n comparisons between its elements for each vector position. A robust way to label a multiset is a function which maps it to a single integer, allowing labels efficient comparisons. This function have to be injective (different

multisets are not mapped to the same label) in order to not impair on the discriminative power of labels. Also it must be commutative[1] because if a pair of vertices is mapped by an isomorphism in an iteration of the method, their respective neighbors may be visited in a different order, as the graphs are labeled in different ways. However the final labels of these two vertices must be the same.

Binary logic operations such as *OR*, *AND* and *XOR* are efficient commutative operations. In order to illustrate the performance of these operations on the binary representation (all with 8 bits) of their respective integer values and then choice the one with the best discriminative feature, we evaluate them on the examples of Figure 4. To show their commutative aspects, each operation is applied twice (left and right) over the same values arranged in different order. In Figure 4-(a) AND operation is performed. The result is an integer described by a binary representation with several null positions. This type of result reflects the AND operation characteristic: answer 1 is achieved only if all its operands are also 1. In the example this occurs in only one position. Similar deficiency can be observed with OR operation (Figure 4-(b)). In this case, a position is 0 if all its operands are 0. However this is not a characteristic of the XOR operation, as we can see in the example (Figure 4-(c)).

Unfortunately, XOR application directly on small integers (usual representation of vertices degrees) restricts the possibilities of different labels, since the leftmost bits are always zero.

Consider for example the labeling operation on the vertices of a graph $G = (V, E)$, $d(v) < 8$, $\forall v \in V$ of Figure 5. The application of XOR operation directly on the values result in the same label to all multisets.

One way of solving this problem is applying a *dispersing function* f_{dsp}[61] on the labels before XOR application. A dispersing function maps integers into a small range to a larger range, thereby decreasing the chances of associating a same label to different multisets. This artifice can be understood as an heuristic to transform XOR in an injective function. A random mapping between integers is enough to define a dispersing function. It is important however to guarantee the construction of a wide interval as the function image set in order to achieve its injective characteristic (for $x, y \in \mathbb{Z},\ x \neq y,\ f_{dsp}(x) \neq f_{dsp}(y)$).

[1] an operation $\cdot$ is commutative if $a \cdot b = b \cdot a$.

10010110	10010110
10010110	01110101
01110101	10010110
AND 01010011	AND 01010011
00010000	00010000

(a) AND comutativity.

10010110	10010110
10010110	01110101
01110101	10010110
OR 01010011	OR 01010011
11110111	11110111

(b) OR comutativity.

10010110	10010110
10010110	01110101
01110101	10010110
XOR 01010011	XOR 01010011
00100110	00100110

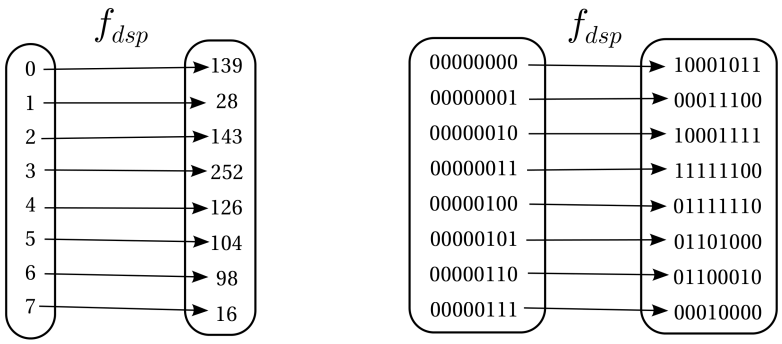
(c) XOR comutativity.

Figure 4: Binary operations applyed to a same set of values.

$\{\!\{2,2,3,4,5\}\!\}$	$\{\!\{1,3\}\!\}$	$\{\!\{2,3,3\}\!\}$	$\{\!\{2,3,4,7\}\!\}$	$\{\!\{5,7\}\!\}$
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
010				
010			010	
011		010	011	
100	001	011	100	101
XOR 101	**XOR** 011	**XOR** 011	**XOR** 111	**XOR** 111
010	010	010	010	010

Figure 5: XOR application on multisets generated to a graph.

A dispersing function example which maps integers of the range $[0, 7]$ into integers of the range $[0, 255]$ is showed in Figure 6. Figura 6-(a) presents



(a) Decimal representation. (b) Binary representation.

Figure 6: A dispersing function example with domain in $[0, 7]$.

$\{\!\{2,2,3,4,5\}\!\}$	$\{\!\{1,3\}\!\}$	$\{\!\{2,3,3\}\!\}$	$\{\!\{2,3,4,7\}\!\}$
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
10001111			
10001111			10001111
11111100		10001111	11111100
01111110	00011100	11111100	01111110
XOR 01101000	**XOR** 11111100	**XOR** 11111100	**XOR** 00010000
11101010	11100000	10001111	00011101

Figure 7: XOR application over values of multisets of Figure 5, using the dispersing function of Figure 6.

the chosen numbers decimal representations and Figura 6-(b), its respective binary representation.

The application of the dispersing function exemplified in Figure 6 for the multisets of Figure 5 results in the labels of Figure 7. This was applied on the values before applying the XOR operation. With the use of the dispersing function, all multisets have been associated to different labels. In this work the dispersion function is implemented via a static vector of size 10000, populated with random values.

The whole IVL algorithm, with all the aspects discussed in this section embedded is presented in the pseudocode of Algorithm 5. The algorithm takes as input a graph and output, a invariant property as discriminative as possible. Line 2 initializes the vector that represents the dispersing function.

Algorithm 5: IVL based on centrality measures.

Input: $G = (V, E)$
Output: p_k

begin
  $f_{dsp} \leftarrow \{139, 28, 143, \ldots, 53\}$;
  $p_0 \leftarrow inv\_inicial(G)$;
  for $u \in V$ **do**
    $p_1(u) \leftarrow 0$;
    for $v \in \Gamma(u)$ **do**
      $p_1(u) \leftarrow p_1(u)$ **XOR** $f_{dsp}\big(p_0(v)\big)$;
    endfor
  endfor
  $k \leftarrow 1$;
  while $|grps(G, p_{k-1})| < |grps(G, p_k)|$ **do**
    for $u \in V$ **do**
      $p_{k+1}(u) \leftarrow 0$;
      for $v \in \Gamma(u)$ **do**
        $p_{k+1}(u) \leftarrow p_{k+1}(u)$ **XOR** $p_k(v)$;
      endfor
    endfor
    $k \leftarrow k + 1$;
  endw
end

In line 3 it is selected the initial invariant to be considered, using Algorithm 4. In lines 4 to 9 is performed the first labeling iteration. The current label of each vertex is initialized with zero (line 5). Then, for each one of its neighbors labels, the XOR operation is applied (line 6) and the resultant label, mapped by the dispersing function (line 7). On line 10, the iteration counter is set to one. Lines 11 to 19 is the while loop for labeling, which is performed until no improvement occurs from one iteration to the next.

7. Computational Results

Nesta seção só coloquei os gráficos de resultados. Falta escrever o texto, escolher os gráficos que vão permanecer e inserir o gráfico dos testes com grafos fortemente regulares

8. Concluding Remarks and Future Work

References

[1] R. M. Farouk, Iris recognition based on elastic graph matching and gabor wavelets, Computer Vision and Image Understanding 115 (8) (2011) 1239 – 1244. doi:10.1016/j.cviu.2011.04.002.

[2] C. Martins, R. Cesar, L. Jorge, A. Freitas, Segmentation of Similar Images Using Graph Matching and Community Detection, Vol. 6658 of Graph-Based Representations in Pattern Recognition, Springer Berlin / Heidelberg, 2011.

[3] T. B. Sebastian, P. N. Klein, B. B. Kimia, Recognition of shapes by editing their shock graphs, IEEE Trans. Pattern Anal. Mach. Intell. 26 (2004) 550–571.

[4] G. Liu, Z. Lin, X. Tang, Y. Yu, Unsupervised object segmentation with a hybrid graph model (hgm), IEEE Trans. Pattern Anal. Mach. Intell. 32 (5) (2010) 910–924.

[5] J. Keustermans, D. Seghers, W. Mollemans, D. Vandermeulen, P. Suetens, Image segmentation using graph representations and local appearance and shape models, in: Proceedings of the 7th IAPR-TC-15 International Workshop on Graph-Based Representations in Pattern Recognition, GbRPR '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 353–365.

[6] S. Wachenfeld, K. Broelemann, X. Jiang, A. Krüger, Graph-based registration of partial images of city maps using geometric hashing, in: Proceedings of the 7th IAPR-TC-15 International Workshop on Graph-Based Representations in Pattern Recognition, GbRPR '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 92–101.

[7] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, IJPRAI 18 (3) (2004) 265–298.

[8] X. Gao, B. Xiao, D. Tao, X. Li, A survey of graph edit distance, Pattern Anal. Applic. 13 (2010) 113 – 129. doi:10.1007/s10044-008-0141-y.

[9] E. Bengoetxea, P. Larranaga, I. Bloch, A. Perchant, C. Boeres, Inexact Graph Matching by Means of Estimation of Distribution Algorithms, Pattern Recognition 35 (2002) 2867–2880.

[10] L. Zheng, X. He, Classification techniques in pattern recognition, in: Proceedings of the 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, 2005, pp. 77–79.

[11] A. Noma, A. B. V. Graciano, L. A. Consularo, R. M. Cesar-Jr, R. M., I. Bloch, A graph distance metric based on the maximal common subgraph, Pattern Recognition Letters 19 (1998) 255–259.

[12] R. M. CESAR-JR, E. Bengoetxea, I. Bloch, P. Larraaga, Inexact graph matching for model-based recognition: Evaluation and comparison of optimization algorithms, Pattern Recognition 38 (2005) 2099–2113.

[13] A. D. Cross, E. R. Hancock, Graph matching with a dual-step em algorithm, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (1998) 1236–1253. doi:http://doi.ieeecomputersociety.org/10.1109/34.730557.

[14] O. Sammoud, C. Solnon, K. Ghédira, An ant algorithm for the graph matching problem, in: 5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005), LNCS 3448, Springer:213-223, Springer, 2005, pp. 213–223.

[15] Y. El-Sonbaty, M. A. Ismail, A new algorithm for subgraph optimal isomorphism, Pattern Recognition 31 (1998) 205–218.

[16] A. Cross, R. Wilson, E. Hancock, Inexact graph matching using genetic search, Pattern Recognition 30 (1997) 953–970.

[17] E. Wong, Model matching in robot vision by subgraph isomorphism, Pattern Recognition 25 (1992) 287–303.

[18] A. Noma, L. Velho, R. M. Cesar, Jr, A computer-assisted colorization approach based on efficient belief propagation and graph matching, in: Proceedings of the 14th Iberoamerican Conference on Pattern Recognition: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, CIARP '09, 2009.

[19] D. K. Isenor, S. G. Zaky, Fingerprint identification using graph matching, Pattern Recognition 19 (2) (1986) 113 – 122.

[20] A. Noma, A. B. V. Graciano, L. A. Consularo, R. M. Cesar-Jr, R. M., I. Bloch, A new algorithm for interactive structural image segmentation, CoRR abs/0805.1854.

[21] P. Pedarsani, M. Grossglauser, On the privacy of anonymized networks, in: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '11, ACM, New York, NY, USA, 2011, pp. 1235–1243. doi:10.1145/2020408.2020596.
URL http://doi.acm.org/10.1145/2020408.2020596

[22] A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, B. Y. Zhao, Measurement-calibrated graph models for social network experiments, in: Proceedings of the 19th international conference on World wide web, WWW '10, ACM, New York, NY, USA, 2010, pp. 861–870. doi:10.1145/1772690.1772778.
URL http://doi.acm.org/10.1145/1772690.1772778

[23] G. Wondracek, T. Holz, E. Kirda, C. Kruegel, A practical attack to de-anonymize social network users, in: IEEE Symposium on Security and Privacy, 2010, pp. 223–238.

[24] G. Cormode, D. Srivastava, Anonymized data: generation, models, usage, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, SIGMOD '09, ACM, New York, NY, USA, 2009, pp. 1015–1018. doi:10.1145/1559845.1559968.
URL http://doi.acm.org/10.1145/1559845.1559968

[25] S. Fortin, The graph isomorphism problem, Tech. rep., University of Alberta, Edmonton, Alberta, Canada (1996).

[26] R. Diestel, Graph Theory (3 Ed.), Springer, 2006.

[27] E. Dalcumune, Algoritmos qunticos para o problema do isomorfismo de grafos, Master's thesis, Laboratrio Nacional de Computao Cientfica, Petrpolis (2008).

[28] B. Jenner, J. K. P. McKenzie, J. Torn, Completeness results for graph isomorphism, Journal of Computer and System Sciences 66 (3) (2003) 549–566.

[29] V. Arvind, J. Torn, Isomorphism testing: Perspective and open problems, Bulletin European Association of Theoretical Computer Science 86 (2005) 66–84.

[30] J. Ullmann, An algorithm for subgraph isomorphism, Journal of the Association for Computing Machinery 23 (1) (1976) 31–42.

[31] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, An improved algorithm for matching large graphs, in: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, 2001, pp. 149–159.

[32] B. D. McKay, Practical graph isomorphism, in: Congressus Numerantium, Vol. 30, 1981, pp. 45–87.

[33] T. Junttila, P. Kaski, Engineering an efficient canonical labeling tool for large and sparse graphs, in: D. Applegate, G. S. Brodat, D. Panario, R. Sedgewick (Eds.), Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics, SIAM, 2007.
URL http://www.siam.org/proceedings/alenex/2007/alx07_013junttilat.pdf

[34] S. Sorlin, C. Solnon, A global constraint for graph isomorphism problems, in: Springer-Verlag (Ed.), the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004), 2004, pp. 287–301.

[35] R. Uehara, S. Toda, T. Nagoya, Graph isomorphism completeness for chordal bipartite graphs and strongly chordal graphs, Discrete Applied Mathematics 145 (3) (2005) 479–482.

[36] L. Zager, Graph similarity and matching, Master's thesis, Department of Electrical Engineering and Computer Science. Massachusetts Institute of Technology (2005).

[37] A. Dharwadker, J. Tevet, The graph isomorphism algorithm, in: Proceedings of the Structure Semiotics Research Group, Eurouniversity Tallinn, 2009.

[38] G. Xiutang, Z. Kai, Simulated annealing algorithm for detecting graph isomorphism, Journal of Systems Engineering and Electronics 19 (4) (2008) 52–57.

[39] L. Lee, Reformulao do problema de isomorfismo de grafos como um problema quadrtico de alocao, Master's thesis, Programa de Ps-Graduao em Informtica - UFES, Vitria, ES (2007).

[40] D. Cvetković, M. Doob, I. Gutman, A. Torgašer (Eds.), Recent Results in the Theory of Graph Spectra, Vol. 36, Annals of Discrete Mathematics, 1988.

[41] L. Hogben, Spectral graph theory and the inverse eigenvalue problem of a graph, Chamchuri Journal of Mathematics 1 (1) (2009) 51–72.

[42] P. L. F. dos Santos, M. C. Rangel, M. C. S. Boeres, Teoria espectral de grafos aplicada ao problema de isomorfismo de grafos, in: ILTC (Ed.), Proceedings of XLII Simpósio Brasileiro de Pesquisa Operacional (SBPO 2010), Bento Gonalves, RS, 2010, pp. 1–12.

[43] P. L. F. Santos, M. C. Rangel, M. C. S. Boeres, An adapted power method for eigenvector computing applied to a graph isomorphism algorithm, in: ILTC (Ed.), Proceedings of XLIII Simpsio Brasileiro de Pesquisa Operacional (SBPO 2011), Ubatuba, SP, 2011, pp. 1–12.

[44] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, , D. Sorensen, Lapack user's guide, `http://www.netlib.org/lapack/lug/` (August 1999).

[45] Y. Saad, Numerical Methods for Large Eigenvalue Problems, Manchester University Press, Manchester, UK, 1992.

[46] B. D. McKay, The nauty page (1984).
URL http://cs.anu.edu.au/ bdm/nauty/

[47] P. T. Darga, H. Katebi, M. Liffiton, I. Markov, K. Sakallah, Saucy 2.0, http://vlsicad.eecs.umich.edu/BK/SAUCY/ (2008).

[48] P. T. Darga, K. A. Sakallah, I. L. Markov, Faster symmetry discovery using sparsity of symmetries, in: Proceedings of the 45th annual Design Automation Conference, DAC '08, ACM, New York, NY, USA, 2008, pp. 149–154.

[49] J. L. L. Presa, Efficient algorithms for graph isomorphism testing, Ph.D. thesis, Universidad Rey Juan Carlos (2009).

[50] D. Cook, L. Holder, Mining graph data, Wiley-Interscience, 2007.

[51] B. Weisfeiler, On construction and identification of graphs, in: B. Springer-Verlag (Ed.), Lecture Notes in Mathematics, Vol. 558, 1976.

[52] A. Piperno, Search space contraction in canonical labeling of graphs (preliminary version), CoRR abs/0804.4881.

[53] C. C. Sims, Computation with permutation groups, in: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, SYMSAC '71, ACM, New York, NY, USA, 1971, pp. 23–28. doi:10.1145/800204.806264.
URL http://doi.acm.org/10.1145/800204.806264

[54] G. D. Tener, Attacks on difficult instances of graph isomorphism: sequential and parallel algorithms, Ph.D. thesis, Orlando, FL, USA, aAI3401107 (2009).

[55] G. Tener, N. Deo, Attacks on hard instances of graph isomorphism., JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing 64 (2008) 203–225.

[56] T. Junttila, P. Kaski, Conflict propagation and component recursion for canonical labeling, in: Proceedings of the First international ICST

conference on Theory and practice of algorithms in (computer) systems, TAPAS'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 151–162.
URL http://dl.acm.org/citation.cfm?id=1987334.1987350

[57] R. Wilson, P. Zhu, A study of graph spectra for comparing graphs and trees, Pattern Recognition 41 (2008) 2833–2841.

[58] W. Haemers, E. Spence, Enumeration of cospectral graphs, Eur. J. Combinatorics 25 (2004) 199–211.

[59] P. L. F. dos Santos, Teoria espectral de grafos aplicada ao problema de isomorfismo de grafos, Master's thesis, Programa de Pós-Graduação em Informática, UFES, Vitória, ES (2010).

[60] R. Grassi, S. Stefani, A. Torriero, Some new results on the eigenvector centrality, Journal of Mathematical Sociology 31 (3) (2007) 237–248.

[61] D. Knuth, The art of computer programming: Sorting and searching, The Art of Computer Programming, Addison-Wesley, 1998.

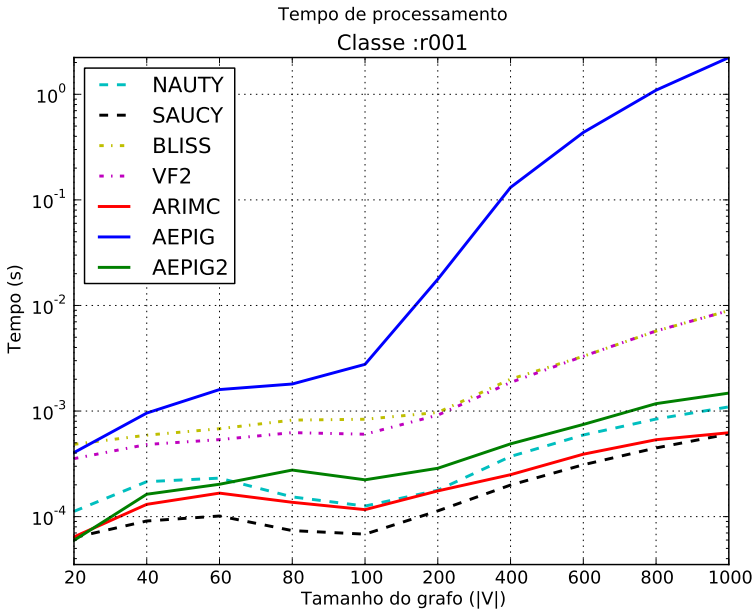


Figure 8: CPU time for class $r001$ isomorphic graphs.

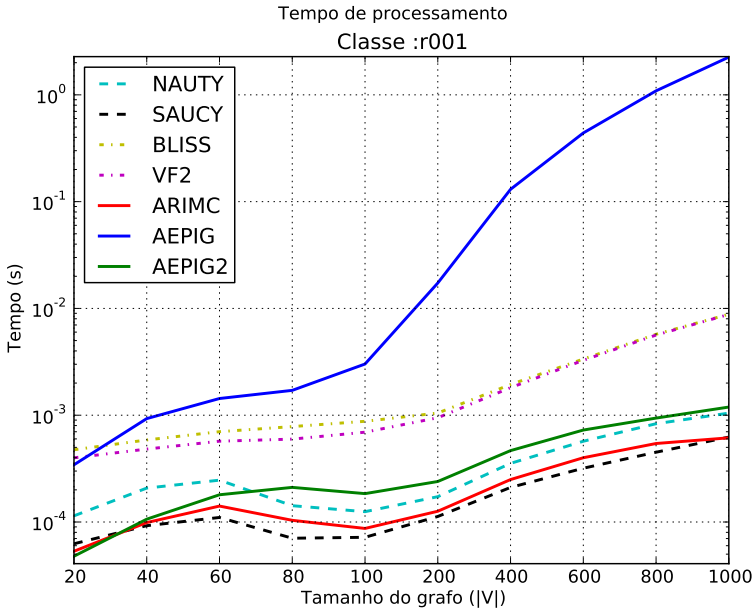


Figure 9: CPU time for class $r001$ non isomorphic graphs.

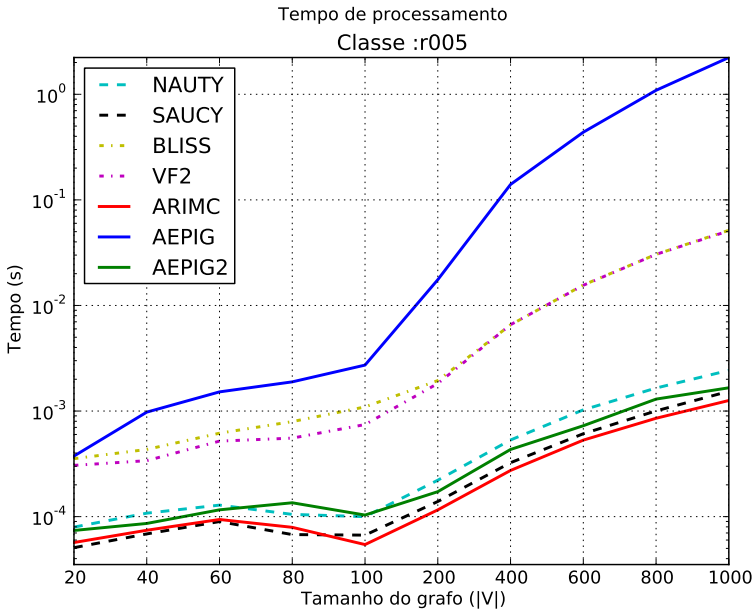


Figure 10: CPU time for class $r005$ isomorphic graphs.

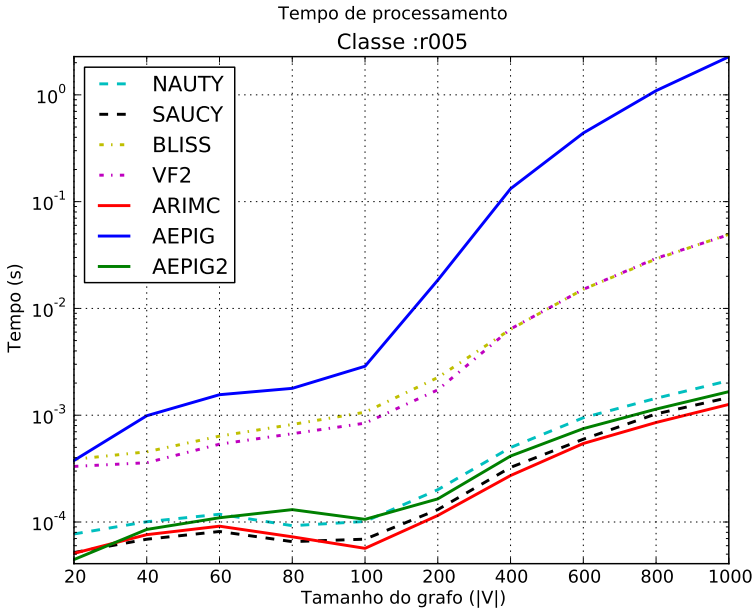


Figure 11: CPU time for class $r005$ non isomorphic graphs.

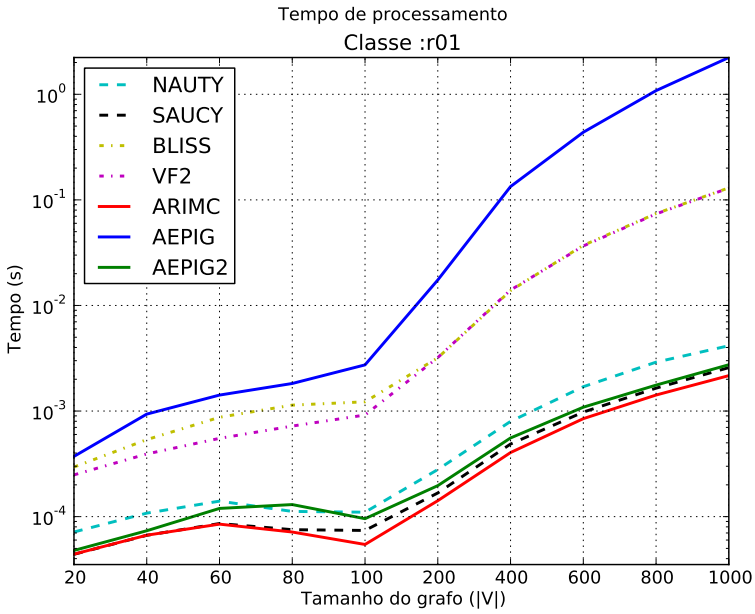


Figure 12: CPU time for class $r01$ isomorphic graphs.

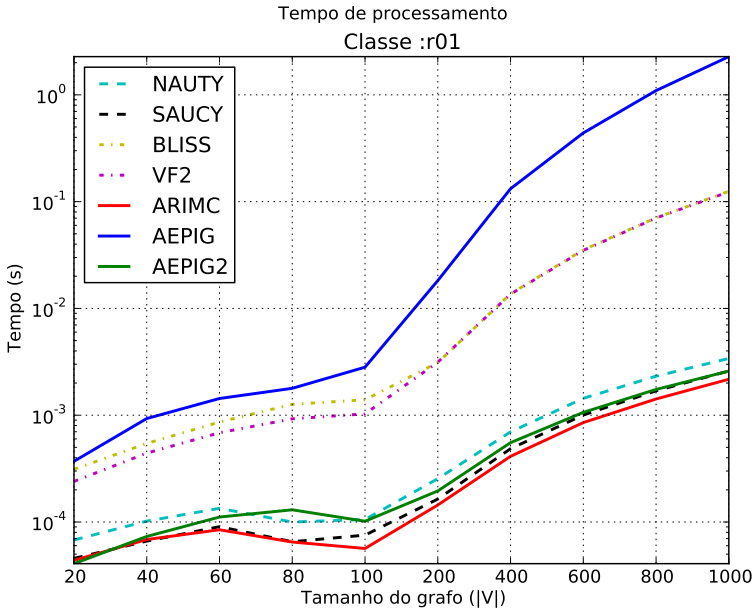


Figure 13: CPU time for class $r01$ non isomorphic graphs.

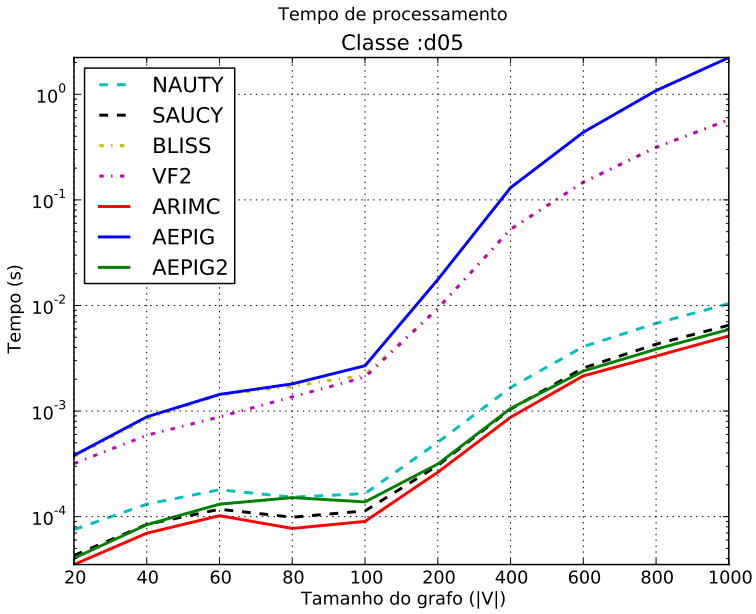


Figure 14: CPU time for class $d05$ isomorphic graphs.

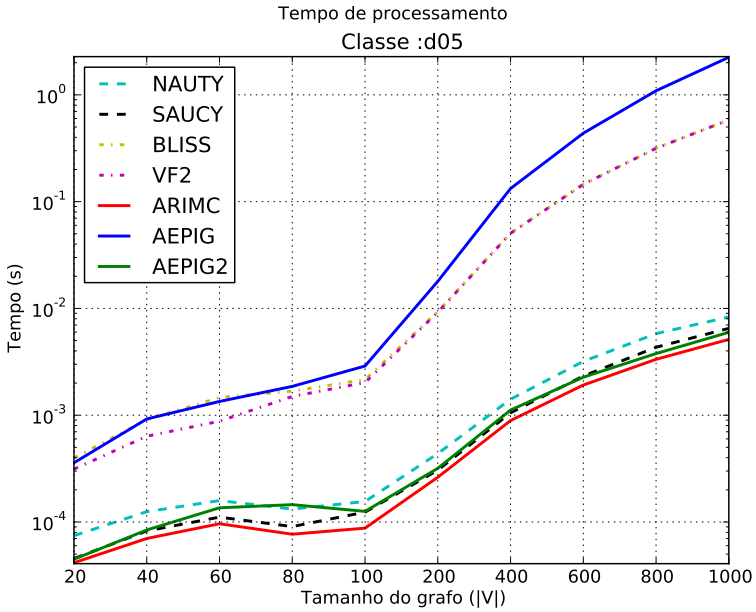


Figure 15: CPU time for class $d05$ non isomorphic graphs.

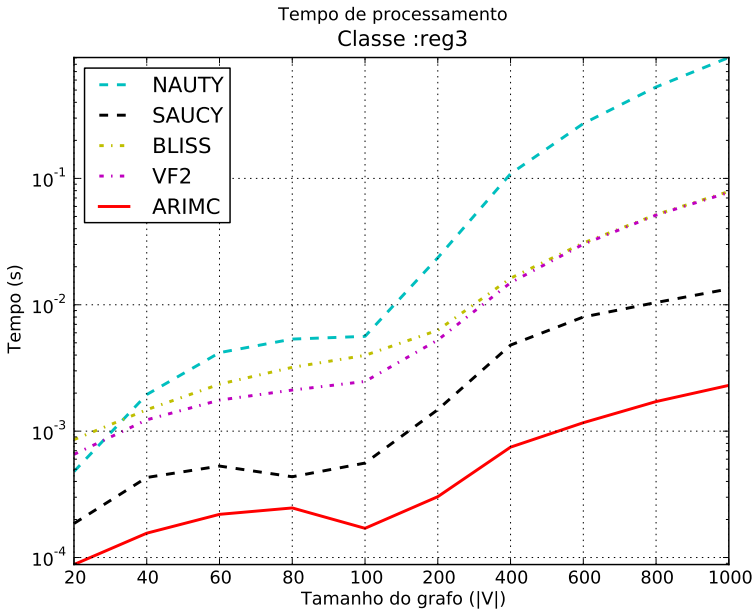


Figure 16: CPU time for class $reg3$ isomorphic graphs.

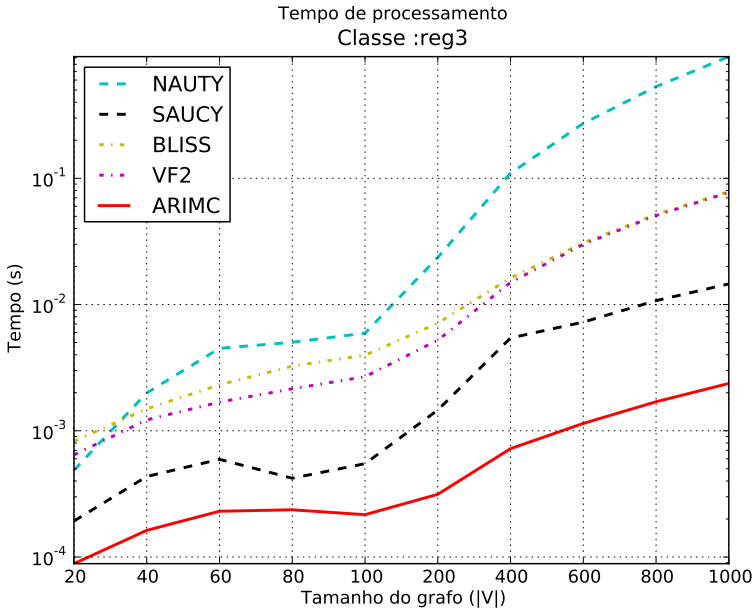


Figure 17: CPU time for class $reg3$ non isomorphic graphs.

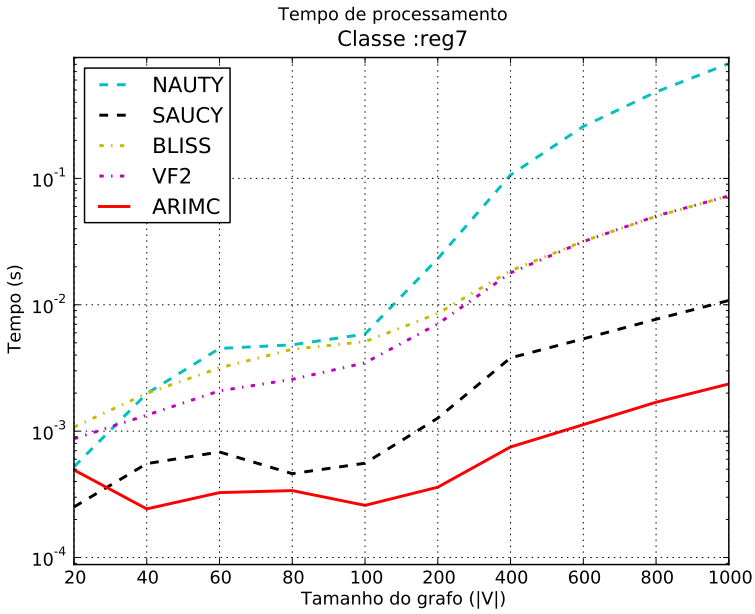


Figure 18: CPU time for class $reg7$ isomorphic graphs.

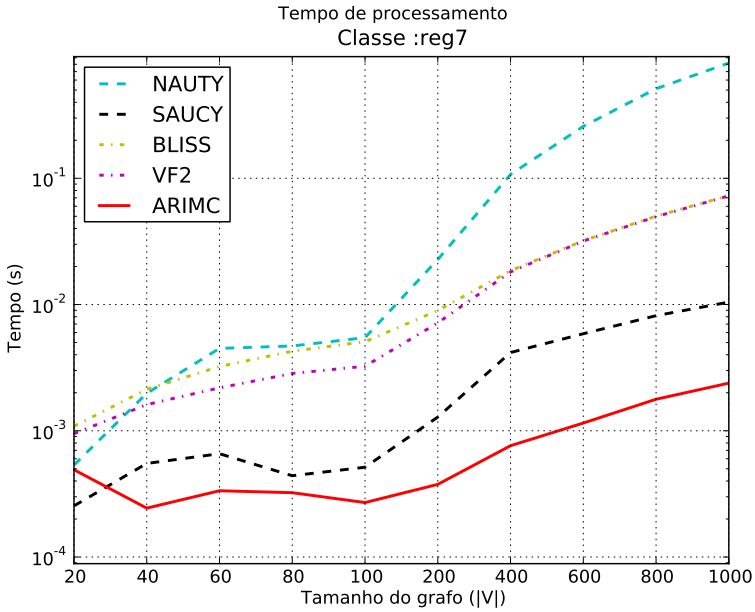


Figure 19: CPU time for class $reg7$ non isomorphic graphs.

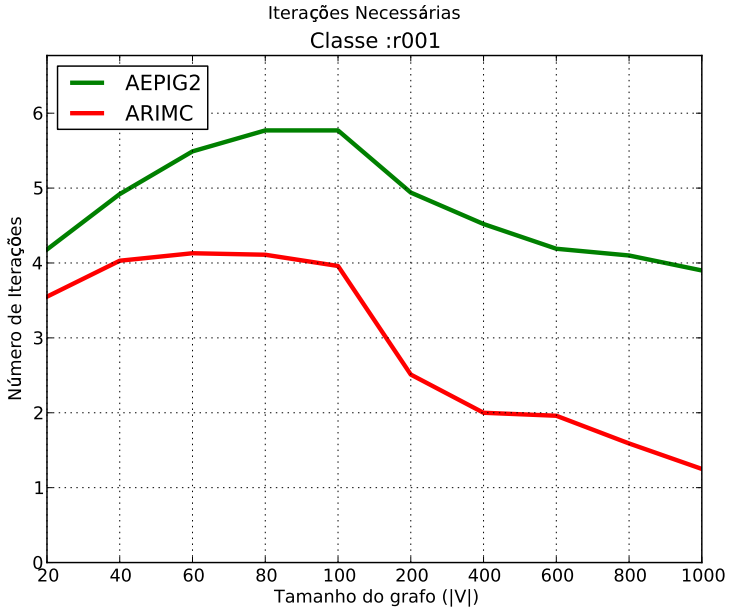


Figure 20: Average number of iterations for class $r001$.

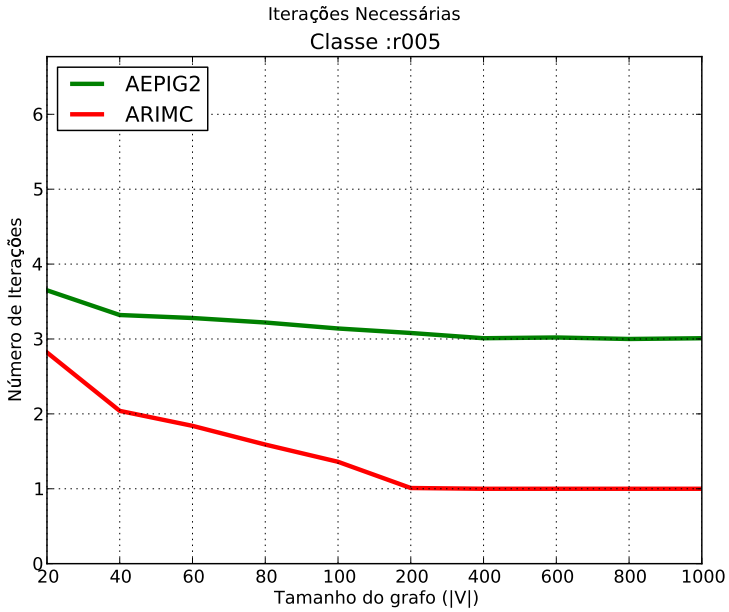


Figure 21: Average number of iterations for class $r005$.

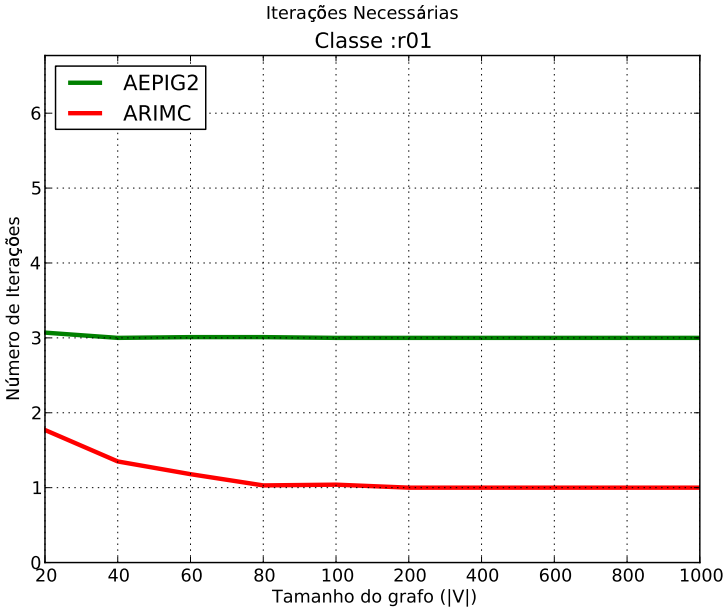


Figure 22: Average number of iterations for class $r01$.

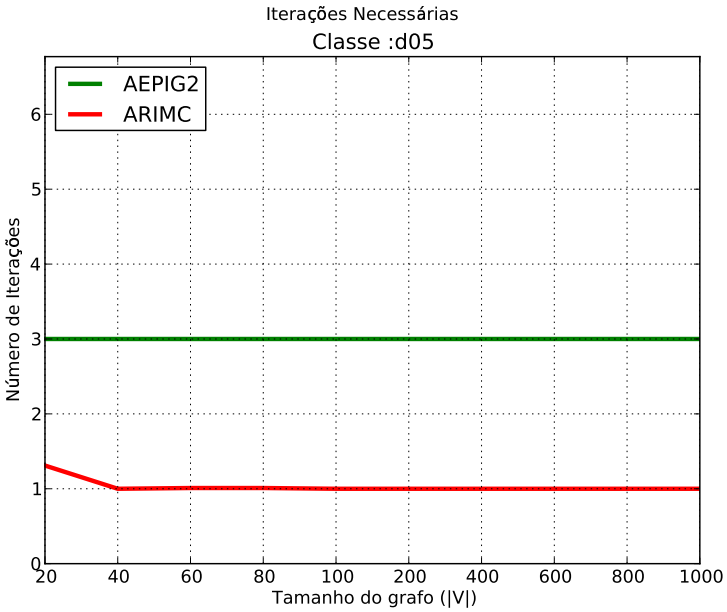


Figure 23: Average number of iterations for class $d05$.

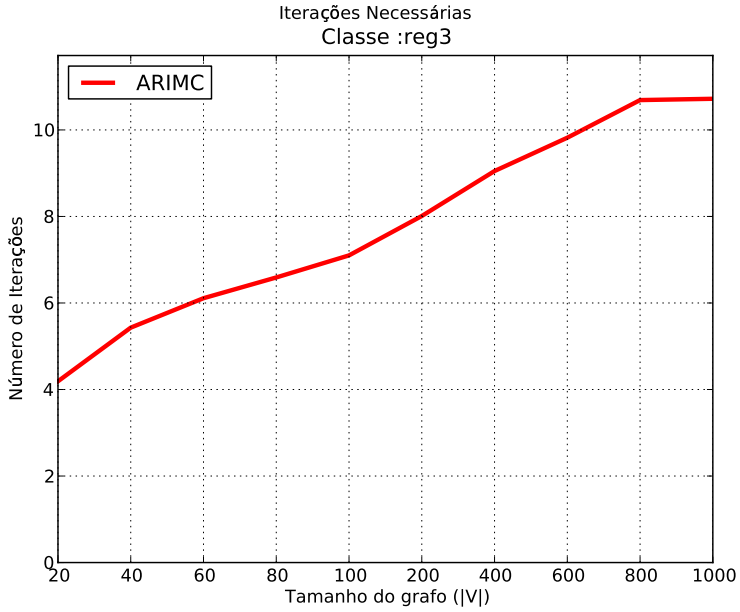


Figure 24: Average number of iterations for class $reg3$.

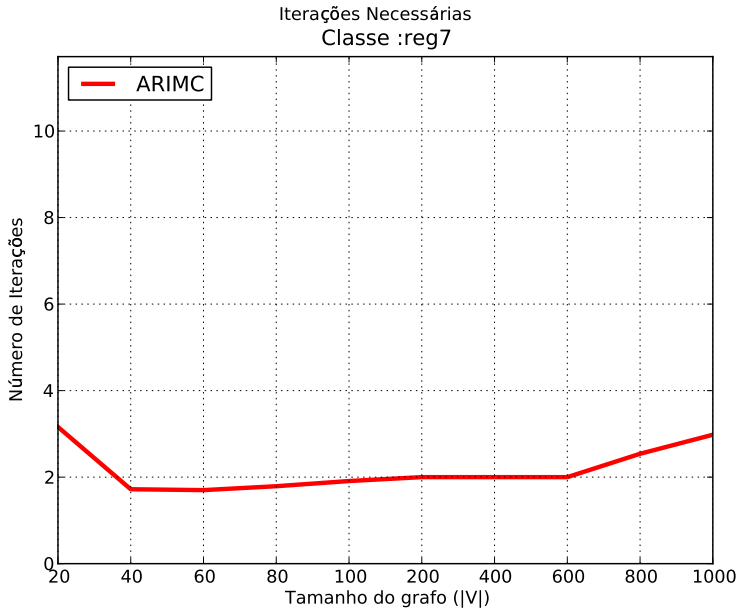


Figure 25: Average number of iterations for class $reg7$.