Marcos Daniel Valadão Baroni

Um Estudo da Eficiência da Autocentralidade no Problema de Isomorfismo de Grafos

Vitória - ES

Janeiro de 2012

Marcos Daniel Valadão Baroni

Um Estudo da Eficiência da Autocentralidade no Problema de Isomorfismo de Grafos

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do título de Mestre em Informática.

Orientadora:
Profª. Drª. Maria Claudia Silva Boeres

Co-orientadora:
Profª. Drª. Maria Cristina Rangel

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória - ES

Janeiro de 2012

Dissertação de Mestrado sob o título de *"Um Estudo da Eficiência da Autocentralidade no Problema de Isomorfismo de Grafos"*, defendida por Marcos Daniel Valadão Baroni em 27 de janeiro de 2012, aprovada pela banca examinadora constituída pelos doutores:

Profª. Drª. Maria Claudia Silva Boeres
Universidade Federal do Espírito Santo
Orientadora

Profª. Drª. Maria Cristina Rangel
Universidade Federal do Espírito Santo
Co-orientadora

Prof. Dr. Arlindo Gomes de Alvarenga
Universidade Federal do Espírito Santo

Profª. Drª. Claudia Marcela Justel
Instituto Militar de Engenharia

Resumo

Este trabalho trata da aplicação da autocentralidade na resolução do Problema de Isomorfismo de Grafos. Esta propriedade, retirada da teoria espectral de grafos, foi utilizada por Philippe Santos em [SANTOS 2010] para a proposta de um algoritmo espectral para resolução deste problema. Uma adaptação do método das potências é proposta para o cálculo das autocentralidades produzindo uma versão competitiva do algoritmo espectral proposto em [SANTOS 2010]. Baseado nesta adaptação, é feito um estudo da eficiência da autocentralidade na resolução do Problema de Isomorfismo. Além disso, é proposto um algoritmo de rotulação iterativa, denominado Algoritmo de Rotulação Iterativa Baseado em Medidas de Centralidades, que pode ser aplicado a qualquer tipo de grafo, inclusive grafos regulares. Uma bateria de testes computacionais foi realizada para comparar os dois algoritmos propostos com alguns bem conhecidos na literatura, como o Nauty.

PALAVRAS CHAVES: Teoria dos grafos, Isomorfismos, Algoritmos, Autovetores.

Abstract

This work treats the application of the eigenvector centrality in solving the Graph Isomorphism Problem. This property, taken from spectral graph theory, was used by Philippe Santos in [SANTOS 2010] to propose a spectral algorithm for solving this problem. An adaptation of the power method is proposed to compute the eigenvector centrality, producing a competitive version to the spectral algorithm of [SANTOS 2010]. Based on this adaptation, the efficiency of the eigenvector centrality in solving the problem is studied. In addition, it is proposed an iterative labeling algorithm, called Centrality Based Iterative Algorithm, which can be applied to any type of graph, including regular ones. Several tests are performed to compare the two proposed algorithms with some others well-known algorithms from literature, such as Nauty.

KEYWORDS: Graph theory, Isomorphims, Algorithms, Eigenvectors.

Agradecimentos

Agradeço a Deus, meu Senhor, pelo que Ele tem realizado em minha vida até aqui. Pela Sua Salvação e maravilhosa graça. Bendito seja o Seu nome eternamente.

Agradeço a minha família por todo o apoio e dedicação.

Agradeço às professoras Maria Claudia Boeres e Maria Cristina Rangel pela orientação, pelo ensino e atenção.

Sumário

Lista de Tabelas

Lista de Figuras

1 Introdução

O Problema de Isomorfismo de Grafos (PIG) pode ser aplicado a diversos problemas reais, como por exemplo, reconhecimento de padrões [CONTE ET AL. 2004], reconhecimento de imagens [FAROUK 2011, MARTINS ET AL. 2011], identificação de similaridades em estruturas químicas [OLIVEIRA , GREVE 2005, FORTIN 1996] e segurança de informação em redes sociais [PEDARSANI , GROSSGLAUSER 2011].

Formalmente dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ de mesma ordem e tamanho são isomorfos se existe uma bijeção $f : V_1 \mapsto V_2$ de forma que as adjacências de suas estruturas sejam preservadas, ou seja, $\{u,v\} \in E_1 \Leftrightarrow \{f(u), f(v)\} \in E_2, \ \forall u,v \in V_1$. O Problema de Isomorfismo de Grafos consiste em determinar se dois grafos são isomorfos [DIESTEL 2005, DALCUMUNE 2008]. Apesar de necessárias, as condições de mesma ordem e mesmo tamanho não são suficientes para concluir se dois grafos são isomorfos.

O PIG é um dos poucos problemas que pertencem a classe de problemas NP, mas não se sabe se é P ou NP-completo. O comumente aceito é que o PIG esteja estritamente entre as duas classes [ARVIND , TORÁN 2005].

A teoria espectral de grafos (TEG) é um campo da matemática discreta que estuda propriedades dos grafos utilizando suas representações matriciais (matriz de adjacência e laplaciana entre outras), autovalores e autovetores [HOGBEN 2009].

Em [SANTOS 2010] foi proposto um algoritmo que utiliza propriedades da teoria espectral de grafos para resolver o Problema de Isomorfismo de Grafos. Este algoritmo, denominado Algoritmo Espectral para o Problema de Isomorfismo de Grafos (AEPIG), utiliza um eficiente filtro espectral que, uma vez calculado, pode facilmente decidir sobre o possível isomorfismo entre um par de grafos. Porém na implementação apresentada em [SANTOS 2010] foi utilizada a função `dsyevr_` extraída da biblioteca CLAPACK para o cálculo deste filtro, a qual foi responsável por 90% do tempo de processamento. O trabalho de [SANTOS 2010] mostra a eficiência da centralidade de autovetor como um filtro porém com alto custo computacional.

Este trabalho tem por objetivo investigar a eficiência da aplicação da centralidade de

autovetor no problema em questão, além de propor a redução do tempo de cálculo da propriedade espectral utilizada e o aperfeiçoamento de sua aplicação.

Para reduzir o tempo de cálculo da centralidade de autovetor uma versão adaptada do método das potências é proposta neste trabalho, originando uma versão mais eficiente e competitiva do algoritmo espectral, denominada AEPIG2. Com base na utilização do método das potências e em dois teoremas apresentados relativos ao problema em questão, a eficiência da centralidade de autovetor é explicada formalmente e um método de rotulação iterativa é proposto como sendo o aperfeiçoamento das técnicas discriminativas implicitamente utilizadas pelo filtro espectral.

O método de rotulação iterativa calcula a partir de uma invariante pouco discriminativa, uma outra invariante bem mais discriminativa. Neste trabalho é proposta a aplicação deste método a algumas medidas de centralidade, originando o algoritmo de rotulação iterativa baseado em medidas de centralidade (ARIMC) que propõe resolver o PIG inclusive para casos de grafos regulares.

Para efeito de comparação do desempenho dos algoritmos AEPIG2 e ARIMC propostos, são realizados testes computacionais utilizando os seguintes algoritmos conhecidos da literatura: Nauty [McKay 1984], VF2 [Cordella et al. 2001], Bliss [Junttila , Kaski 2007] e Saucy [Darga et al. 2008a, Darga et al. 2008b].

No segundo Capítulo deste trabalho são apresentados alguns conceitos matemáticos que serão utilizados no decorrer do trabalho. No Capítulo 3 o Problema de Isomorfismo de Grafos é definido e discutido. No Capítulo 4 é apresentado o Algoritmo Espectral para o Problema de Isomorfismo de Grafos (AEPIG), proposto em [Santos 2010] e proposto o AEPIG2. No Capítulo 5 a eficiência do AEPIG é analisada e o Algoritmo de Rotulação Iterativa é proposto. No Capítulo 6 são tratados os testes computacionais e no Capítulo 7 são apresentadas as conclusões e os trabalhos futuros.

2 Conceitos Relacionados

Neste capítulo são apresentados alguns conceitos matemáticos relativos à teoria de grafos e da teoria espectral de grafos visando a melhor compreensão dos algoritmos para a resolução do PIG tratados neste trabalho. Além de conceitos básicos são também apresentados dois teoremas, provados em [SANTOS 2010], importantes para o algoritmo espectral para a resolução do PIG.

Apresentamos também neste capítulo o método das potências para o cálculo de autovetor e autovalor, que é de grande interesse neste trabalho por estarem intimamente relacionados com a teoria espectral de grafos.

As subseções são divididas em conceitos básicos em teoria dos grafos, em teoria espectral de grafos, o método das potências e os resultados teóricos apresentados em [SANTOS 2010]. Vários conceitos presentes neste capítulo são baseados em [DE ABREU ET AL. 2007] e [HORN , JOHNSON 1990].

2.1 Conceitos Básicos de Teoria dos Grafos e dos Conjuntos

Nesta seção são introduzidos alguns conceitos básicos de teoria de grafos. Grande parte destes conceitos foram retirados de [DIESTEL 2005].

Um **grafo simples** é um tupla $G = (V,E)$ de conjuntos tal que $E \subseteq V \times V$, onde os elementos de E são subconjuntos de V contendo exatamente dois elementos distintos. Os elementos do conjunto V são chamados de **vértices** e os elementos do conjunto E são chamados de **arestas**. Neste trabalho é tratado grafos simples apenas por **grafo**.

A forma mais comum de se representar um grafo graficamente é desenhando para cada vértice, um ponto (ou círculo) e para cada aresta, uma linha que liga os seus respectivos vértices.

O número $n = |V|$ de vértices do grafo é dito ser a **ordem** do grafo. Dizemos que u é **ad-**

jacente ao vértice v, ou ainda que u é **vizinho** de v se $\{u,v\} \in E$. O conjunto dos vértices adjacentes ao vértice $u \in V$ é denotado por $\Gamma(u)$, também chamado de **vizinhança** ou **adjacência** de u. O número de vértices adjacentes a u é chamado **grau** de u e denotado por $d(u)$. A **sequência de graus** é o vetor $seq_d(G) = \left(d(v_1)\ d(v_2)\ \ldots\ d(v_n) \right)$ onde $d(v_1) \leq d(v_2) \leq \ldots \leq d(v_n)$, $d_i \in V$, $i = 1,2,\ldots,n$.

Um grafo $G' = (V',E')$ é dito ser um **subgrafo** do grafo $G = (V,E)$ quando $V' \subseteq V$ e $E' \subseteq E$.

Um **caminho** $C = (V,E)$ é um grafo da forma $V = \{v_0, v_1, \ldots, v_r\}$, $\quad E = \big\{\{v_0,v_1\},\{v_1,v_2\},\ldots,$ $\{v_{r-1},v_r\}\big\}$ onde $v_i, i = 0,1,\ldots,r$ são distintos entre si. Neste caso dizemos que C é um caminho de v_0 a v_r. O número de arestas do caminho é o **tamanho** do caminho.

A **distância** entre dois vértices u e v é o tamanho do menor caminho existente entre u e v e é denotada por $dist(u,v)$. Definimos ainda como $\Gamma_d(u)$ o conjunto dos vértices que distam exatamente d do vértice u.

Um grafo **regular** é um grafo em que todos os seus vértices possuem o mesmo grau, ou seja, $d(v) = k, \forall v \in V, k \in \mathbb{Z}^+$. Neste caso dizemos que o grafo é k-regular.

Um multiconjunto é como um conjunto, mas que permite mais de um exemplar de um mesmo elemento. Para diferenciar um multiconjunto de um conjunto convencional são utilizados neste trabalho os símbolos $\{\!\{$ e $\}\!\}$ como delimitadores de um multiconjunto.

As definições de multiconjunto e de conjunto $\Gamma_d(u)$ são utilizadas no Capítulo 5 na proposta do algoritmo de rotulação iterativa.

2.2 Conceitos Básicos de Teoria Espectral de Grafos

Da mesma forma que a seção anterior, esta apresenta conceitos básicos de teoria espectral de grafos importantes para o desenvolvimento deste trabalho. Muitos dos resultados foram retirados de [DE ABREU ET AL. 2007].

Seja $G = (V,E)$ um grafo com n vértices. A **matriz de adjacência** $A(G)$ é a matriz quadrada de ordem n cujas entradas são

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

Pela definição a matriz $A(G)$ é real e simétrica (formada por uns e zeros). Como as arestas são conjuntos de dois elementos sempre distintos a diagonal principal desta matriz é sempre nula.

O polinômio $det\big(\lambda I - A(G)\big)$, é denominado **polinômio característico de** G e denotado por $p_G(\lambda)$ onde λ é dito **autovalor do grafo** G quando λ é raiz de $p_G(\lambda)$.

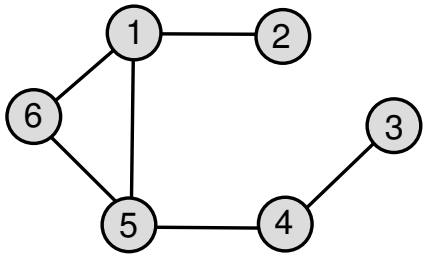
Considere $\lambda_1 > \ldots > \lambda_s$ como sendo os s autovalores distintos do grafo G sendo $m(\lambda_1), \ldots,$ $m(\lambda_s)$ suas respectivas multiplicidades. O **espectro do grafo** G, denotado por $spect(G)$ é definido como a matriz

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

Chamamos de **índice** de G o maior dos autovalores de G denotado também por $ind(G)$. Como o traço[1] de $A(G)$ é sempre zero a somatória de seus autovalores é também sempre zero [HORN , JOHNSON 1990]. Ressaltamos também que $A(G)$ possui sempre autovalores reais por ser uma matriz hermitiana[2] [HORN , JOHNSON 1990].

A **centralidade de autovetor** ou **autocentralidade** x_i de um vértice v_i é definida como a i-ésima componente do autovetor não-negativo x associado ao índice de G. A centralidade de autovetor é uma propriedade importante utilizada no desenvolvimento deste trabalho.

Na Figura 2.1 é apresentado o exemplo de um grafo G_1, sua matriz de adjacência $A(G_1)$, seu espectro $spect(G_1)$, o seu índice $ind(G_1)$ e o autovetor x associado ao índice (vetor de autocentralidade).



$$A(G_1) = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$spec(G_1) = \begin{bmatrix} 2.334 & 1.100 & 0.274 & -0.595 & -1.374 & -1.740 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$ind(G_1) = \lambda_1 = 2.334 \qquad x = \begin{pmatrix} 0.967 & 0.414 & 0.225 & 0.525 & 1.000 & 0.843 \end{pmatrix}$$

Figura 2.1: Exemplo de propriedades espectrais de um grafo.

A seguir são apresentados dois resultados teóricos relacionados com a centralidade de autovetor, retirados de [SANTOS 2010] e utilizados no algoritmo espectral para a resolução do PIG (Capítulo 4). As provas dos teoremas são encontradas em [SANTOS 2010].

[1] soma da diagonal principal.

[2] uma matrix quadrada H_n é hermitiana quando o elemento simétrico de cada uma das entradas da matrix é o seu conjugado complexo, ou seja, $h_{ij} = \overline{h}_{ji}$, onde $\overline{a + b\sqrt{-1}} = a - b\sqrt{-1}$.

Teorema 1. *Se dois grafos são isomorfos então suas centralidades de autovetor são proporcionais.*

O Teorema 1 torna possível concluir que dois grafos não são isomorfos entre si após calculados seus vetores de autocentralidade e constatados que estes não são proporcionais. Contudo, possuir as centralidades proporcionais, apesar de necessário, não é suficiente neste caso para dizer que dois grafos são isomorfos entre si.

Teorema 2. *Se dois grafos possuem centralidades de autovetor proporcionais e distintas entre si então os grafos são isomorfos.*

Em contrapartida com o Teorema 1, o Teorema 2 nos permite concluir que dois grafos são isomorfos entre si avaliando o autovetor de autocentralidade, caso estes possuam componentes distintas entre si.

Na seção seguinte é apresentado o método das potências

2.2.1 O Método das Potências

O método das potências é um método iterativo simples para se calcular um autovetor de uma matriz. Este método calcula de forma aproximada o autovetor associado ao maior autovalor de uma matriz [SAAD 1992]. A estratégia do método é calcular uma sequência de k vetores da forma $v_i = A^i v_0$ onde v_0 é um vetor de solução inicial e A uma matrix $n \times n$. O Algoritmo 1 descreve o método das potências.

Algoritmo 1: **O Método das Potências.**

Entrada: Matriz $n \times n$:A; Solução inicial: $\overrightarrow{v}_0$; k: Número de iterações
Saída: Autovalor: λ_1; Autovetor associado: $\overrightarrow{v}^{\lambda_1}$

```
1 início
2     para i ← 1 to k faça:
3         w_i = A v_{i-1}
4         α_i = max(w_i)
5         v_i = w_i / α_i
6     fim
7     λ_1 = α_k
8     v^{λ_1} = v_k
9 fim
```

Inicialmente o método recebe a matriz A da qual deseja-se calcular o autovetor onde o vetor v_0 é dado como solução inicial. O método então realiza k iterações das linhas 3 a 5. Na linha 3 é calculado um novo autovetor aproximado $\overrightarrow{w}_i$ multiplicando-se a matriz de entrada

A_n pelo autovetor v_{i-1} calculado na iteração anterior. Na linha 4 encontra-se a maior componente do autovetor $\overrightarrow{w}_i$ para que este seja normalizado na linha 5 gerando o novo vetor de solução v_i, cuja maior componente em módulo é 1. O valor α_i é a aproximação do autovalor associado ao autovetor v_i. Ao fim de k iterações o método tem em α_k o índice da matriz e em $\overrightarrow{v}_k$ o autovetor associado ao autovalor λ_1.

No pseudo-código apresentado no Algoritmo 1 não é descrito um critério de parada. Para isso geralmente é calculado um valor de resíduo $r_i = |\lambda_i - \lambda_{i-1}|$ para cada iteração i. Se este for menor que um valor $\varepsilon \in \mathbb{R}$ dado como entrada, o algoritmo interrompe as iterações e para.

Após k iterações o autovetor v_k calculado pelo método é, conforme Equação 2.1, o vetor inicial v_0 multiplicado por uma potência da matriz A e um fator de normalização.

$$v_k = \frac{1}{\alpha_k} A v_{k-1} = \frac{1}{\alpha_k}\frac{1}{\alpha_{k-1}} A^2 v_{k-2} = \underbrace{\frac{1}{\alpha_1 \alpha_2 \cdots \alpha_{k-1}\alpha_k}}_{\text{fator de normalização}} A^k v_0 \tag{2.1}$$

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{A}, \underbrace{\begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}}_{v_0} \underbrace{\begin{bmatrix} 1.000 \\ 0.333 \\ 0.333 \\ 0.667 \\ 1.000 \\ 0.667 \end{bmatrix}}_{v_1} \underbrace{\begin{bmatrix} 0.857 \\ 0.429 \\ 0.286 \\ 0.571 \\ 1.000 \\ 0.857 \end{bmatrix}}_{v_2} \underbrace{\begin{bmatrix} 1.000 \\ 0.375 \\ 0.259 \\ 0.563 \\ 1.000 \\ 0.813 \end{bmatrix}}_{v_3} \underbrace{\begin{bmatrix} 0.921 \\ 0.421 \\ 0.237 \\ 0.526 \\ 1.000 \\ 0.842 \end{bmatrix}}_{v_4} \cdots \underbrace{\begin{bmatrix} 0.967 \\ 0.414 \\ 0.225 \\ 0.525 \\ 1.000 \\ 0.843 \end{bmatrix}}_{v_k}$$

Figura 2.2: Exemplo de autovetores gerados durante a aplicação do método das potências.

A Figura 2.2 mostra alguns autovetores gerados durante a aplicação do método a uma matriz 6×6 tomando o vetor $\overrightarrow{1}$ como solução inicial. São apresentadas as primeiras quatro soluções parciais geradas e a solução final v_k.

Este método é bastante relevante para a teoria espectral de grafos pois, além de calcular o autovetor de centralidade de um grafo juntamente com o seu índice, ele nos permite deduzir o sentido da autocentralidade [DE FREITAS 2010].

Assumir uma solução inicial v_0 significa atribuir uma centralidade inicial v_0^i a cada um dos vértices $v_i \in V$. A cada iteração, multiplicar a matriz pelo vetor de centralidade v_{i-1} para gerar um novo vetor v_k significa gerar, para cada vértice, um novo valor de centralidade a partir da soma das centralidades anteriores dos seus vértices adjacentes.

O método alcança a convergência quando a centralidade dos vértices é exatamente a soma normalizada das centralidades dos seus vértices adjacentes. De acordo com a aplica-

ção do método das potências associada à matriz de adjacência do grafo, é possível concluir que a centralidade de autovetor de um vértice é uma combinação linear das centralidades dos vértices adjacentes.

A Figura 2.3 ilustra, passo a passo, a aplicação do método das potências no cálculo da centralidade dos vértices de um grafo com 6 vértices, considerando o vetor $\overrightarrow{1}$ como solução inicial (quadro 1). Dentro de cada vértice é escrito o valor de centralidade. A primeira iteração é iniciada calculando-se a soma das centralidades dos vértices adjacentes (quadro 2). As setas mostram esta soma de forma detalhada para um dos vértices do grafo. O segundo passo da primeira iteração é selecionar, dentre os vértices, a maior centralidade (quadro 3). Este valor passa a ser o índice λ_1 do grafo. O terceiro e último passo da primera iteração é normalizar as centralidades dividindo-as pelo valor de λ_1 (quadro 4). Mais algumas iterações são ilustradas nos quadros seguintes e no quadro 15 é apresentada a solução final, encontrada com a convergência dos valores após algumas iterações.

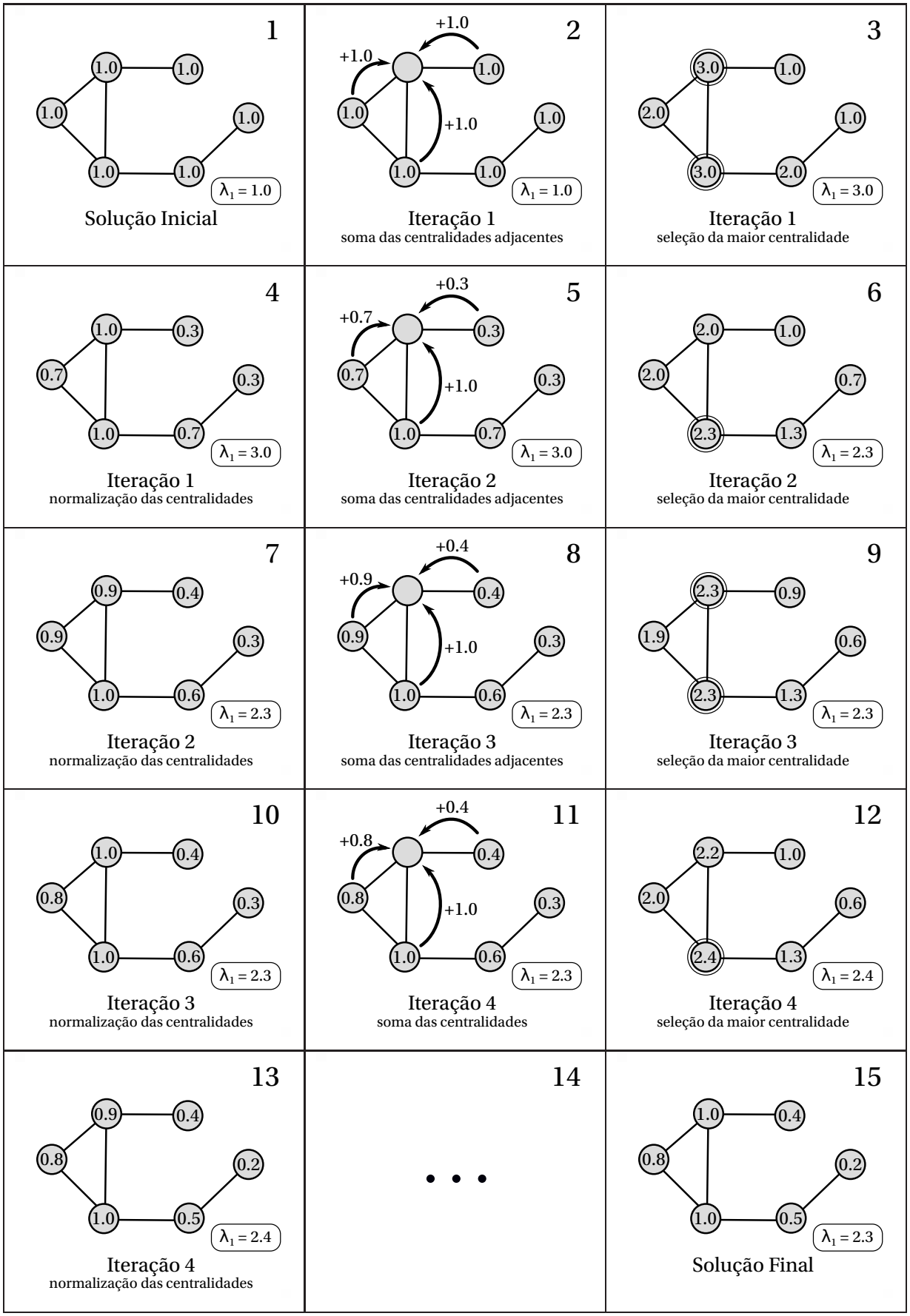


Figura 2.3: Exemplo de iterações do método das potências.

3 O Problema de Isomorfismo de Grafos

Neste capítulo são destacadas a definição do Problema de Isomorfismo de Grafos e alguns conceitos relacionados baseados em [DIESTEL 2005] e [MCKAY 1981]. São citadas também referências bibliográficas importantes deste problema apresentando alguns algoritmos existentes na literatura que propõem a sua resolução e algumas possíveis aplicações do problema.

3.1 Definição do Problema

Chamamos de isomorfismo entre dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ uma bijeção (ou mapeamento) dos vértices em V_1 nos vértices em V_2 de forma que as associações entre os vértices de G_1 sejam mantidas em G_2.

A Figura 3.1 ilustra um exemplo de isomorfismo entre dois grafos: em (a) estão dois grafos isomorfos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ e em (b) é apresentado o isomorfismo $f = \{(1,A), (2,B), (3,C), (4,D), (5,E), (6,F)\}$ entre eles, representado pela sobreposição dos vértices em V_1 sobre os vértices em V_2

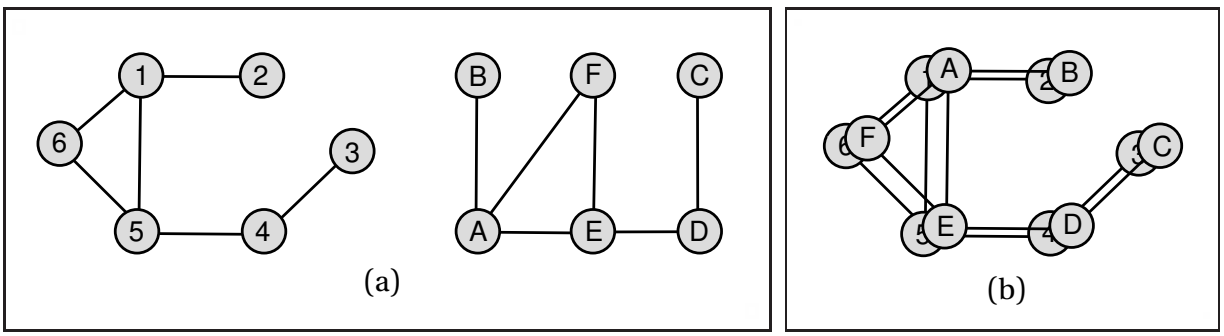


Figura 3.1: Exemplo de grafos isomorfos.

Definição 1. *Um* **isomorfismo** *entre dois grafos* $G_1 = (V_1, E_1)$ *e* $G_2 = (V_2, E_2)$ *é uma bijeção* $f : V_1 \to V_2$ *tal que* $\{a,b\} \in E_1 \Leftrightarrow \{f(a), f(b)\} \in E_2$, $a,b \in V_1$.

O isomorfismo entre dois grafos consiste na rerrotulação dos vértices de um dos grafos de forma a deixá-lo idêntico ao seu isomorfo, pois o que difere dois grafos isomorfos é apenas a forma com que seus vértice são rotulados (Figura 3.1).

Dois grafos podem ou não ser isomorfos. Se forem isomorfos, é possível ainda que exista mais de um isomorfismo entre eles. Ao longo do texto, quando necessário, notaremos por $F(G_1, G_2)$ o conjunto de isomorfismos existentes entre os grafos G_1 e G_2, e a notação $G_1 \cong G_2$ para dizer que dois grafos são isomorfos entre si. Neste caso, se f for o mapeamento que estabelece um isomorfismo, dizemos que os vértices u e v estão associados um ao outro, se $f(u) = v$, com $u \in V_1$ e $v \in V_2$.

A Figura 3.2 mostra dois isomorfismos distintos (f_1 e f_2), estabelecidos entre os grafos G_1 e G_2.

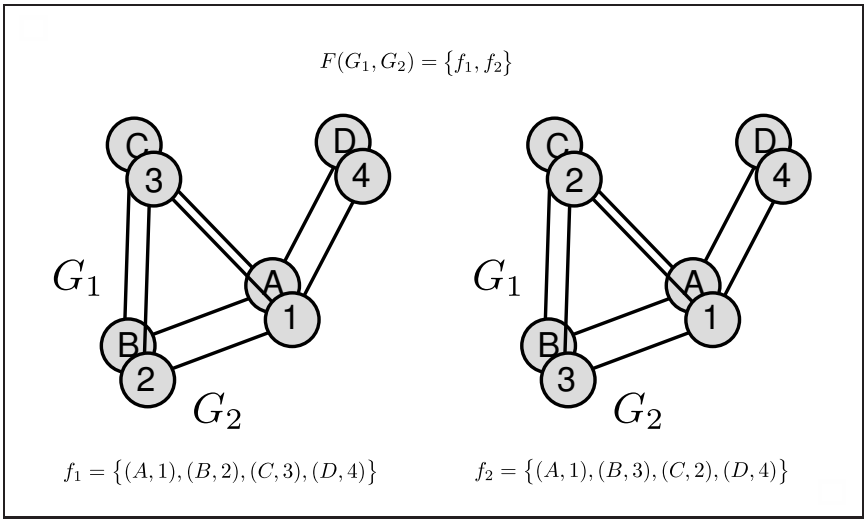


Figura 3.2: Dois isomorfismos distintos entre os grafos G_1 e G_2.

A Definição 2 estabelece o Problema de Isomorfismo de Grafos.

Definição 2. *O* **Problema de Isomorfismo de Grafos (PIG)** *é o problema de decisão que consiste em responder se dois dados grafos são isomorfos entre si.*

Para que dois grafos sejam isomorfos é preciso que algumas propriedades sejam compartilhadas entre eles, como por exemplo, número de vértices e número de arestas. Dois grafos com diferentes quantidades de vértices não podem ser isomorfos entre si. Propriedades como estas independem da forma com que os vértices são rotulados e são ditas propriedades invariantes ou simplesmente invariantes dos grafos. Formalmente o conceito de propriedade invariante com respeito a grafos é apresentada na Definição 3 e com respeito aos vértices de um mesmo grafo, na Definição 4.

Definição 3. *Seja P uma propriedade definida sobre grafo. P é dita* **invariante** *se $P(G_1) = P(G_2)$ para qualquer par de grafos G_1 e G_2 tal que $G_1 \cong G_2$.*

Definição 4. *Sejam $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ com $G_1 \cong G_2$, f um isomorfismo $f : V_1 \mapsto V_2$ e p uma propriedade sobre os vértices de um grafo. A propriedade p é dita* **invariante com respeito a um vértice** *se $p(v) = p(u)$ onde $f(v) = u$, com $v \in V_1$ e $u \in V_2$.*

As propriedades invariantes com respeito aos vértices são bastante úteis na resolução do PIG pois elas podem guiar a busca por uma função de isomorfismo entre dois grafos. Vértices que possuem propriedades invariantes diferentes não podem ser associados por um isomorfismo, isto é, se $v \in V_1, u \in V_2$ e $p(v) \neq p(u)$ então $f(v) \neq u$.

Considere dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ como grafos de entrada para o PIG. Considere ainda que G_1 e G_2 possuem o mesmo número de vértices, mesmo número de arestas e a mesma sequência de graus. A estratégia então é, em cada grafo, agrupar os vértices de acordo com o valor de invariante estabelecida p.

Dada uma propriedade invariante p sobre vértices de grafos e um grafo $G = (V, E)$ denota-se o grupo (ou conjunto) de vértices $v \in V$ com $p(v) = c$ por $grp(G, p, c)$ e $grps(G, p)$ todos os agrupamentos de vértices de V gerados pela propriedade p, onde c pode assumir um valor constante numérico ou c pode ser um conjunto (ou multiconjunto) de valores numéricos. Assim,

$$grp\big(G, p, c\big) = \big\{v | c = p(v), v \in V\big\}$$
$$grps\big(G, p\big) = \big\{grp(G, p, c) \big| c \in \{p(v) | v \in V\}\big\}$$

Porém, para que esta busca seja eficiente a propriedade invariante utilizada deve ser o mais discriminativa possível e ao mesmo tempo fácil de ser calculada.

Definição 5. *Uma propriedade invariante P é dita ser* **completa** *quando $P(G_1) \neq P(G_2)$ implica $G_1 \ncong G_2$.*

Encontrar uma propriedade invariante completa que seja fácil de ser calculada significa resolver o PIG de forma eficiente, pois esta propriedade invariante possui o máximo poder discriminativo.

Dados dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ isomorfos entre si através do isomorfismo f e um vértice $v \in V_1$, o teorema a seguir nos permite estabelecer quais os vértices de G_2 são adjacentes a $f(v)$ sem conhecermos necessariamente as arestas em E_2, mas apenas observando a vizinhança de v no grafo G_1.

Teorema 3. *Sendo G_1 e G_2 grafos isomorfos pelo isomorfismo $f : V_1 \mapsto V_2$ e $u \in V_1$, temos que* $\Gamma\big(f(u)\big) = \big\{ f(v) | v \in \Gamma(u) \big\}$.

Prova:
Considere $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ dois grafos isomorfos entre si através do isomorfismo $f : V_1 \mapsto V_2$ e $v \in V_1$. Pela definição de vizinhança temos:

$$\Gamma\big(f(u)\big) = \big\{ w \big| \big\{ w, f(u) \big\} \in E_2 \big\} \tag{3.1}$$

Como w é um vértice pertencente a V_2 e f mapeia os vértices de V_1 em vértices de V_2 chamamos de v o vértice associado a w no grafo G_1, ou seja:

$$w = f(v), v \in V_1 \tag{3.2}$$

Das equações (3.1) e (3.2) temos:

$$\Gamma\big(f(u)\big) = \big\{ f(v) \big| \big\{ f(u), f(v) \big\} \in E_2 \big\} \tag{3.3}$$

Pela definição de isomorfismo sabemos toda aresta $\big\{ f(u), f(v) \big\} \in E_2$ possui uma aresta equivalente $\{u, v\} \in E_1$. Assim a equação (3.3) pode ser reescrita da seguinte forma:

$$\Gamma\big(f(u)\big) = \big\{ f(v) \big| \{u, v\} \in E_1 \big\} \tag{3.4}$$

Observe que pela própria definição de vizinhança, condicionar u tal que $\{u, v\} \in E_1$ é semelhante a dizer que $v \in \Gamma(u)$. Assim reescrevemos a equação (3.4):

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

A seguir apresentamos o Teorema 4 sobre propriedades invariantes de vértices que é uma importante ferramenta na estratégia de resolução do PIG abordada neste trabalho.

Definição 6. *Considere um grafo $G = (V, E)$ e $p(v)$ uma propriedade sobre um vértice $v \in V$. A propriedade $p'(v) = \{p(u) | u \in \Gamma(v)\}$ é dita* **propriedade derivada** *de $p(v)$.*

Teorema 4. *Considere um grafo $G = (V, E)$. Se $p(v)$ é propriedade invariante com respeito ao vértice $v \in V$ então $p'(v)$ também é propriedade invariante com respeito ao vértice v.*

Prova:
Considere $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ dois grafos isomorfos entre si através do isomorfismo $f : V_1 \mapsto V_2$ e $v \in V_1$. Aplicando a definição da propriedade derivada p' sobre um vértice $f(v) \in V_2$ temos:

$$p'\big(f(v)\big) = \big\{ p(w) \big| w \in \Gamma\big(f(v)\big) \big\} \tag{3.5}$$

Aplicando o Teorema 3 ao lado direito da equação (3.5) temos:

$$\{p(w)|w \in \Gamma(f(v))\} = \{p(w)|w \in \{f(u)|u \in \Gamma(v)\}\} \tag{3.6}$$

Entretanto na composição do conjunto a direita da equação anterior existe uma instanciação ambígua entre os elementos w e $f(u)$. É possível ser mais direto substituindo w por $f(u)$:

$$\{p(w)|w \in \{f(u)|u \in \Gamma(v)\}\} = \{p(f(u))|u \in \Gamma(v)\} \tag{3.7}$$

Como p é propriedade invariante, ou seja, $p(u) = p(f(u))$ temos:

$$\{p(f(u))|u \in \Gamma(v)\} = \{p(u)|u \in \Gamma(v)\} \tag{3.8}$$

Por definição temos que:

$$\{p(u)|u \in \Gamma(v)\} = p'(v) \tag{3.9}$$

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

Este teorema nos permite derivar, a partir de uma invariante pouco discriminativa p, uma outra invariante p' com maior poder de discriminação.

A Figura 3.3 ilustra dois agrupamentos dos vértices do grafo G baseados em duas propriedades invariantes diferentes. Em (a) é apresentado o grafo G. Em (b) está representado o conjunto $grps(G,p)$ de agrupamentos gerados utilizando-se o grau do vértice como propriedade invariante p, definida na Tabela 3.1. Em (c) está representado o conjunto $grps(G,p')$ de agrupamentos gerados utilizando-se a propriedade invariante p' derivada do grau, definida na Tabela 3.2. É possível observar neste caso que p' é invariante mais discriminativa que p. Neste caso agrupando-se os vértices pelo grau foram gerados 3 agrupamentos, cada um possuindo 2 vértices. No entanto utilizando-se a propriedade derivada do grau p' foi gerado um total de 6 aprupamentos, cada um contendo um vértice.

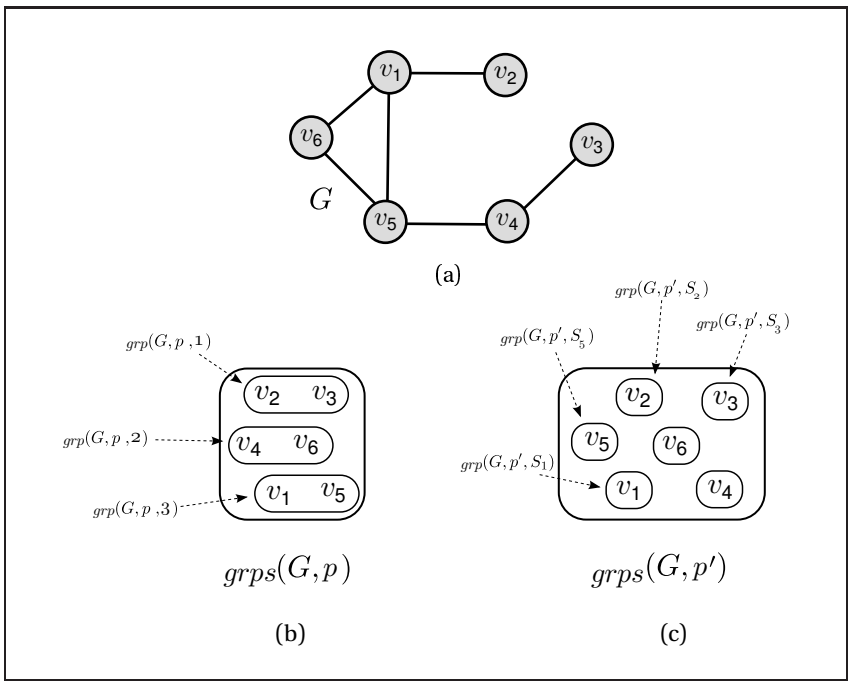


Figura 3.3: Exemplo de agrupamentos gerados pelo grau (a) e pela invariante derivada do grau (b).

propriedade inicial (grau)	agrupamentos gerados
$p(v_1) = d(v_1) = 3$	$grp(G,p,1) = \{v_2, v_3\}$
$p(v_2) = d(v_2) = 1$	$grp(G,p,2) = \{v_4, v_6\}$
$p(v_3) = d(v_3) = 1$	$grp(G,p,3) = \{v_1, v_5\}$
$p(v_4) = d(v_4) = 2$	
$p(v_5) = d(v_5) = 3$	
$p(v_6) = d(v_6) = 2$	

Tabela 3.1: Tabela com valores de invariante inicial (grau) para vértices do grafo da Figura 3.3 (a) e os grupos $grps(G,p)$ ilustrados na Figura 3.3 (b).

propriedade derivada	agrupamentos gerados
$p'(v_1) = \{\!\{p(v_2), p(v_5), p(v_6)\}\!\} = \{\!\{1,3,2\}\!\} = S_1$	$grp(G, p', S_1) = \{v_1\}$
$p'(v_2) = \{\!\{p(v_1)\}\!\} = \{\!\{3\}\!\} = S_2$	$grp(G, p', S_2) = \{v_2\}$
$p'(v_3) = \{\!\{p(v_4)\}\!\} = \{\!\{2\}\!\} = S_3$	$grp(G, p', S_3) = \{v_3\}$
$p'(v_4) = \{\!\{p(v_3), p(v_5)\}\!\} = \{\!\{1,3\}\!\} = S_4$	$grp(G, p', S_4) = \{v_4\}$
$p'(v_5) = \{\!\{p(v_1), p(v_4), p(v_6)\}\!\} = \{\!\{3,2,2\}\!\} = S_5$	$grp(G, p', S_5) = \{v_5\}$
$p'(v_6) = \{\!\{p(v_1), p(v_5)\}\!\} = \{\!\{3,3\}\!\} = S_6$	$grp(G, p', S_6) = \{v_6\}$

Tabela 3.2: Tabela com valores de invariante derivada do grau para vértices do grafo da Figura 3.3 (a) e os grupos $grps(G, p')$ ilustrados na Figura 3.3 (c).

Os enunciados dos Teoremas 3 e 4 apresentados nesta seção também são válidos para multiconjuntos. As respectivas provas se dão de forma semelhante.

3.2 Trabalhos Relacionados: aplicações e algoritmos para o PIG

O PIG é amplamente aplicado a problemas de reconhecimento de padrões sendo a subárea de reconhecimento de imagens provavelmente a de maior aplicação.

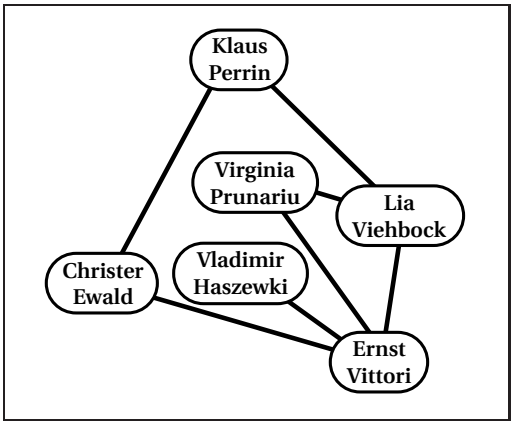
Em [MARTINS ET AL. 2011] o PIG é aplicado na segmentação de imagens semelhantes. Nesse trabalho as várias imagens comparadas são segmentadas em setores levando em consideração as formas encontradas nas imagens. Cada setor representa um vértice de um grafo e as proximidades entre os setores representam as arestas. As imagens representadas por grafos isomorfos entre si são então semelhantes.

Em [DE CÁSSIA NANDI 2006] o PIG é aplicado à comparação de impressões digitais. Nesse trabalho peculiaridades estruturais das impressões digitais, chamadas de minúcias, são mapeadas nas imagens. Cada uma das minúcias representam um vértice de um grafo onde as arestas representam relações de vizinhança entre as minúcias. A comparação entre as imagens é realizada então buscando-se um isomorfismo entre seus respectivos grafos. De forma um pouco semelhante em [FAROUK 2011] o PIG é aplicado no reconhecimento de íris.

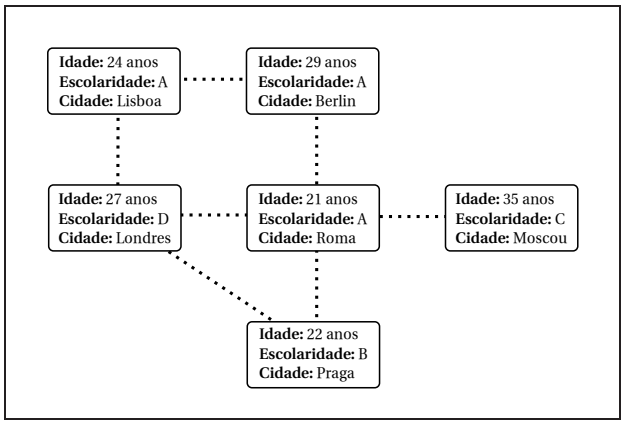
Na química, o PIG é aplicado na identificação de similaridades estruturais em componentes químicos. Antes de se conceder um nome exclusivo a uma molécula é necessário garantir que sua estrutura não é a mesma de nenhuma outra molécula já conhecida [CONTE ET AL. 2004]. Nesse caso, cada átomo da molécula representa um vértice do grafo e cada ligação química entre eles representa uma aresta do grafo. Assim as moléculas possuem a

mesma estrutura se os respectivos grafos forem isomorfos.

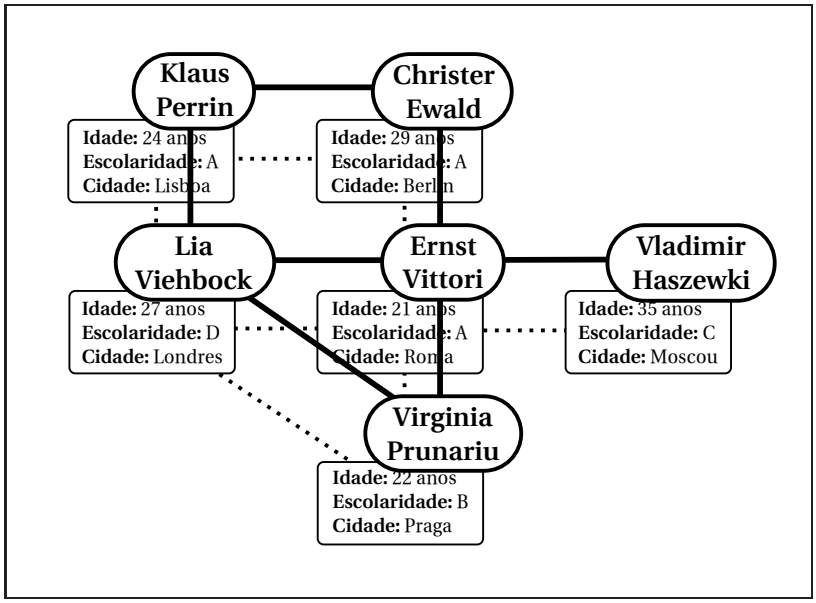
Recentemente o PIG tem sido estudado também na área de redes sociais. Com o crescimento e popularização de redes sociais aliado ao acúmulo centralizado de dados pessoais, surgem preocupações quanto à segurança e privacidade destas informações. Um desafio seria, por exemplo, compartilhar dados anônimos sem acidentalmente expor informações pessoais que identifiquem indivíduos na rede [PEDARSANI , GROSSGLAUSER 2011]. Nesse caso cada indivíduo na rede representaria um vértice e cada ligação entre indivíduos representaria uma aresta. Se entre o grafo compartilhado de informações anônimas e um grafo público não anônimo existir exatamente um isomorfismo, ao encontrá-lo associamos as informações anônimas às identidades extraídas do grafo público (Figura 3.4).



(a) Rede pública não anônima.



(b) Rede publicada com registros anônimos.



(c) O isomorfismo identifica os indivíduos da rede anônima.

Figura 3.4: Exemplo de aplicação do problema de isomorfismo em redes sociais.

Além de sua utilidade em aplicações importantes, o PIG desperta interesse de vários grupos de pesquisa devido à sua complexidade de solução. Atualmente o PIG é um dos poucos problemas que pertencem a classe NP mas que não se sabe ao certo se está na classe dos problemas P ou NP-completo. Sabe-se que não é um problema co-NP [FORTIN 1996, JENNER ET AL. 2003]. Entretanto aceita-se que esteja estritamente entre as duas classes [ARVIND , TORÁN 2005].

Na literatura os algoritmos de resolução do PIG podem ser classificados de duas formas segundo o tipo de estratégia de resolução: por abordagem direta ou por rotulação canônica. Na abordagem direta o algoritmo busca ao menos um isomorfismo entre os grafos de entrada. Não o encontrando declara-os como não isomorfos. Na rotulação canônica, dados dois grafos de entrada G_1 e G_2, o algoritmo gera uma rotulação $H(G_1)$ dos vértices tal que $H(G_1) = H(G_2)$ se, e somente se, $G_1 \cong G_2$ [COOK , HOLDER 2007].

Um dos algoritmos mais eficientes para a resolução do PIG é denominado Nauty [MCKAY 1981, MCKAY 1984], que produz uma rotulação canônica sobre os grafos de entrada através de rotulações iterativas onde um conjunto de invariantes de vértices é considerado. Os algoritmos Saucy [DARGA ET AL. 2008a, DARGA ET AL. 2008b] e Bliss [JUNTTILA , KASKI 2007] trabalham de forma semelhante mas possuem diferenças na forma de implementação. O Saucy, por exemplo, explora a esparsidade dos grafos para resolver o problema de forma mais eficiente.

O algoritmo espectral para o PIG, proposto em [SANTOS 2010], utiliza a autocentralidade dos vértices para auxiliar a resolução do problema. Vértices associados por um isomorfismo devem possuir a mesma autocentralidade. Assim, o algoritmo busca um isomorfismo testando associações apenas entre vértices que possuem a mesma autocentralidade, reduzindo o espaço de busca.

O AEPIG porém é ineficiente no caso de grafos regulares, pois seus vértices possuem a mesma autocentralidade. Em [RODRIGUES ET AL. 2011] é proposto contornar este problema buscando quebrar a regularidade dos grafos através de *modificações equivalentes*[1], possibilitando assim a utilização da autocentralidade dos vértices na busca pelo isomorfismo.

No capítulo seguinte será apresentado o AEPIG e proposta a adaptação do método das potências para o cálculo do vetor de autocentralidade. No Capítulo 5 a eficiência do AEPIG é analisada e explicada formalmente e o Algoritmo de Rotulação Iterativa Baseado em Medidas de Centralidade (ARIMC) é proposto apresentando-se o seu funcionamento, inclusive para grafos regulares. Tanto o AEPIG quanto o ARIMC são algoritmos de abordagem direta.

[1] É considerada modificação equivalente a aplicação do mesmo conjunto de operações de edição de grafos (inserção de vértice e de aresta) em ambos os grafos.

4 Um Algoritmo Espectral para o Problema de Isomorfismo de Grafos

Neste capítulo será apresentado o Algoritmo Espectral para a resolução do PIG (AEPIG) proposto em [SANTOS 2010]. Embora outros algoritmos que solucionam esse problema em seu caso geral já tenham sido propostos, o AEPIG resolve o PIG utilizando propriedades da teoria espectral de grafos descritas no Capítulo 2, em especial a centralidade de autovetor. O algoritmo é composto por três fases principais: cálculo dos vetores de autocentralidade associados aos grafos, verificação da distinção das autocentralidades dos autovetores e a descida na árvore de solução.

A intenção da utilização da autocentralidade é contribuir para a redução do espaço de soluções do problema. Este espaço pode ser descrito, para grafos de n vértices, como sendo o conjunto de permutações dos n vértices de um dos grafos de entrada do problema, fixando os vértices dos dois grafos de maneira a preservar suas adjacências. Deste modo, o tamanho do espaço de busca é igual a $n!$.

Avaliar todo este espaço de busca torna-se uma tarefa inviável devido ao seu tamanho. Entretanto é possível eliminar do espaço de busca grande parte das soluções inviáveis utilizando o que chamamos de **filtros**. Os filtros são, na verdade, propriedades invariantes avaliadas antes do processo de descida na árvore de solução com o objetivo de podar alguns de seus ramos.

Propriedades invariantes de grafos podem ser utilizadas preliminarmente como filtros decidindo se a árvore de busca deve ser ou não executada, uma vez que estas propriedades devem ser equivalentes em grafos isomorfos. Propriedades invariantes de vértices são utilizadas para gerar grupos de associações entre os vértices, assim a tentativa de associação se restringe a vértices de um mesmo grupo, conforme estratégia apresentada na Seção 3.1.

Antes do AEPIG ser aplicado são comparados o número de vértices, o número de arestas e a sequência de graus de ambos os grafos de entrada. Se alguma destas propriedades for

diferente entre os grafos, pode-se concluir que os mesmos não são isomorfos.

4.1 As Fases do AEPIG

A seguir são apresentadas as três fases do AEPIG. Inicialmente o AEPIG verifica se os grafos de entrada $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ possuem o mesmo número de vértices, o mesmo número de arestas e a mesma sequência ordenada de graus. Caso isto não aconteça o algoritmo pára concluindo que os grafos não são isomorfos. Caso contrário o algoritmo segue para a Fase 1.

4.1.1 Fase 1: Cálculo das Autocentralidades

Nesta primeira fase são calculados os autovetores $\overrightarrow{x_1}$ e $\overrightarrow{x_2}$ associados aos índices λ_1^1 e λ_2^1 que representam os vetores de autocentralidade dos grafos de entrada G_1 e G_2 respectivamente. Em seguida as componentes dos vetores são ordenadas e suas proporcionalidades comparadas. Caso os autovetores não sejam proporcionais, conclui-se então que os grafos não são isomorfos, segundo o Teorema 1. Sendo os autovetores proporcionais seguimos então para a próxima fase, uma vez que há possibilidade dos grafos serem isomorfos.

4.1.2 Fase 2: Verificação da distinção das autocentralidades

O objetivo desta fase é verificar se as autocentralidades de cada um dos grafos são distintas entre si. Se isto acontecer, pelo Teorema 2 concluímos que os grafos são isomorfos. Neste caso os grafos possuem um único isomorfismo entre si, que é a associação dos vértices em V_1 aos vértices V_2 que possuem o mesmo valor de autocentralidade, a menos de uma constante real. Se as autocentralidades dos grafos não forem distintas entre si o algoritmo segue para a última fase.

4.1.3 Fase 3: Descida na árvore de busca

Nesta fase é executada a árvore de busca pelo isomorfismo, onde os vértices em V_1 são associados a vértices em V_2 na tentativa de encontrar um isomorfismo. Para que esta busca seja mais eficiente são feitas tentativas de associação apenas entre vértices de V_1 e V_2 que tenham o mesmo valor de autocentralidade. Para isso os vértices em V_1 e em V_2 são agrupados de acordo com suas respectivas centralidades.

A cada nível *i* da árvore o vértice $v_i \in V_1$ é associado a um dos vértices de V_2. Feita a associação, a consistência do isomorfismo é verificada, levando-se em conta as arestas existentes em ambos os grafos. Não sendo consistente, a associação é desfeita e o vértice v_i é associado a um outro vértice livre[1] de V_2. Se forem esgotados todos os vértices livres em V_2 o algoritmo realiza o *backtracking*, subindo um nível na árvore, desfazendo a associação do vértice v_{i-1} e associando-o a um outro vértice livre em V_2.

Caso a associação feita ao vértice v_i seja consistente, o algoritmo desce mais um nível da árvore associando o vértice $v_{i+1} \in V_1$ a um outro vértice ainda não associado em V_2. Se o algoritmo alcançar uma folha da árvore, ou seja, chegar a finalmente associar o vértice $v_n \in V_1$ a outro vértice de V_2 de forma consistente, um isomorfismo é encontrado. O algoritmo então para e retorna o isomorfismo.

Se o algoritmo realizar *backtracking* até a raiz (vértice v_1) esgotando todas as possíveis associações, o algoritmo para, concluindo que não existe isomorfismo entre os grafos.

A seguir é apresentado no Algoritmo 2 o pseudo-código do AEPIG. Na linha 2 os vetores de autocentralidade dos grafos de entrada são calculados. Na linha 3 as componentes dos autovetores são ordenadas e na linha 4 é testada a proporcionalidade entre os vetores ordenados, caso não sejam proporcionais, os grafos não são isomorfos (linha 5) (Teorema 1). Caso contrário o algoritmo segue e verifica se as componentes são distintas entre si (linha 7 e 8). Se forem distintas os grafos são isomorfos (linha 10) (Teorema 2). Caso contrário é executada a árvore de busca na linha 12. Se encontrada uma solução viável os grafos são isomorfos

[1] que ainda não foi associado a nenhum outro vértice em V_1.

(linha 14), caso contrário não o são (linha 16).

Algoritmo 2: Algoritmo Espectral para o Problema de Isomorfismo de Grafos

Entrada: Grafos: $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$
Saída: Se G_1 e G_2 são isomorfos entre si.

```
1  início
2      calcula as autocentralidades x⃗¹, x⃗² de G1, G2
3      ordena as componentes de x⃗¹ e x⃗²
4      se x⃗¹ ≠ k x⃗², k ∈ ℝ* então
5          retorna falso
6      senão
7          se x⃗^i = (x_1^i, ..., x_n^i), tal que x_j^i ≠ x_k^i,
8              j,k = 1,...,n, j ≠ k e i = 1,2
9          então
10             retorna verdadeiro
11         senão
12             executada árvore de busca utilizando agrupamentos por autocentralidade
13             se solução viável encontrada então
14                 retorna verdadeiro
15             senão
16                 retorna falso
17             fim
18         fim
19     fim
20 fim
```

4.2 O Cálculo da Autocentralidade: O Método das Potências Adaptado

Na implementação do AEPIG em [SANTOS 2010], trabalho em que este algoritmo foi proposto, é utilizada para o cálculo do autovetor de centralidade a função `dsyevr_` da biblioteca CLAPACK. Esta função utiliza o algoritmo *dqds* para calcular autovetores para matrizes no caso geral também computando outras informações relacionadas ao seu espectro [ANDERSON ET AL. 1999], foi responsável por, em média, 90% do tempo de processamento do algoritmo.

Entretanto para a execução do AEPIG é necessário apenas o cálculo das componentes do autovetor associado ao índice da matriz. Outras informações relacionadas são descartadas pelo algoritmo. Desta forma o método das potências, apresentado na Seção 2.2.1, mostra ser o método apropriado para a realização deste cálculo, por calcular apenas o necessário.

Nesta seção é proposta a utilização do método das potências como ferramenta para o cálculo do vetor de autocentralidade, que representa a primeira fase do AEPIG. Em seguida

otimizações neste método são propostas, considerando o problema em questão, dando origem a uma versão otimizada do AEPIG proposta e implementada neste trabalho, chamada AEPIG2.

Algumas peculiaridades da matriz de adjacência e do problema em questão serão consideradas para que três técnicas de otimização sejam aplicadas ao método: remoção da fase de normalização, otimização da multiplicação matriz-vetor e redução do número de iterações. As três técnicas são descritas nas seções seguintes.

4.2.1 Remoção da Fase de Normalização

A cada iteração do método das potências o autovetor é normalizado dividindo-o pela sua componente de maior módulo. Esta etapa de normalização, além de garantir que as componentes dos autovetores gerados não venham crescer de forma ilimitada, também é útil para encontrar o autovalor associado ao autovetor que está sendo calculado. Como o AEPIG não utiliza o autovalor esta etapa de normalização pode ser omitida, possibilitando a utilização de vetores de inteiros como representação da solução. Isto reduz o esforço computacional do método, utilizando inteiros cujas operações aritméticas são mais eficientes e evitando que todo o vetor seja percorrido efetuando-se a divisão pelo fator de normalização.

A Equação 4.1 define o vetor solução gerado pelo método das potências sem a etapa de normalização. O que difere a Equação 4.1 da Equação 2.1 é a ausência do fator de normalização. A Figura 4.1 exemplifica os três primeiros vetores solução gerados por uma execução do método das potências sem a etapa de normalização para a mesma matriz de adjacência apresentada na Figura 2.1.

$$v_k = Av_{k-1} = A^2 v_{k-2} = A^k v_0 \tag{4.1}$$

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{A} , \quad \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}}_{v_0} \quad \underbrace{\begin{bmatrix} 2 \\ 3 \\ 3 \\ 1 \\ 2 \\ 1 \end{bmatrix}}_{v_1} \quad \underbrace{\begin{bmatrix} 6 \\ 7 \\ 6 \\ 3 \\ 4 \\ 2 \end{bmatrix}}_{v_2} \quad \underbrace{\begin{bmatrix} 13 \\ 16 \\ 16 \\ 6 \\ 9 \\ 4 \end{bmatrix}}_{v_3} \quad \underbrace{\begin{bmatrix} 32 \\ 38 \\ 35 \\ 16 \\ 20 \\ 9 \end{bmatrix}}_{v_4}$$

Figura 4.1: Exemplo de cinco primeiros autovetores gerados pelo método das potências sem a etapa de normalização.

4.2.2 Otimização da Multiplicação Matriz-Vetor

A multiplicação matriz-vetor necessária a cada iteração do método das potências (linha 3 do Algoritmo 1) é a parte mais custosa do método. Considere um grafo $G = (V,E)$ com $n = |V|$, $m = |E|$ e sua matriz de adjacência $A(G) = A$. Para multiplicar a matriz A por um vetor v são realizadas n^2 multiplicações entre os elementos da matriz e do vetor.

Entretanto é possível considerar peculiaridades da matriz de adjacência do grafo para otimizar a multiplicação matriz-vetor. A representação de matriz de adjacência pode ser substituída por uma lista de adjacência. Nesta representação a multiplicação matriz-vetor é feita iterando-se apenas sobre os elementos existentes na lista, reduzindo a complexidade da multiplicação de n^2 para $2m \leq \frac{n(n-1)}{2} < n^2$, como pode ser constatado através dos pseudo-códigos apresentados no Algoritmo 3 (linha 4) e Algoritmo 4 (linha 5).

Algoritmo 3: Multiplicação matriz-vetor convencional.

Entrada: Matriz: A_n; Vetor: $\overrightarrow{v}$
Saída: Vetor: $\overrightarrow{w}$

```
1 início
2     para i ∈ {1,2,...,n} faça:
3         w_i ← 0
4         para j ∈ {1,2,...,n} faça:
5             w_i ← w_i + a_ij * v_j
6         fim
7     fim
8 fim
```

Algoritmo 4: Multiplicação matriz-vetor utilizando lista de adjacências.

Entrada: Listas de adjacência: L; Vetor: $\overrightarrow{v}$
Saída: Vetor: $\overrightarrow{w}$

```
1 início
2     n ← |L|
3     para i ∈ {1,2,...,n} faça:
4         w_i ← 0
5         para j ∈ L_i faça:
6             w_i ← w_i + v_j
7         fim
8     fim
9 fim
```

A Figura 4.2 ilustra a multiplicação de uma matriz de adjacência por um vetor usando a forma convencional enquanto a Figura 4.3 ilustra a mesma multiplicação utilizando a representação de lista de adjacências.

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} v_2 + v_3 \\ v_1 + v_3 + v_5 \\ v_1 + v_2 + v_4 \\ v_3 \\ v_2 + v_6 \\ v_5 \end{bmatrix}$$

Figura 4.2: Exemplo de produto matriz-vetor.

$$\begin{pmatrix} \{2,3\} \\ \{1,3,5\} \\ \{1,2,4\} \\ \{3\} \\ \{2,6\} \\ \{5\} \end{pmatrix} \bigodot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} v_2 + v_3 \\ v_1 + v_3 + v_5 \\ v_1 + v_2 + v_4 \\ v_3 \\ v_2 + v_6 \\ v_5 \end{bmatrix}$$

Figura 4.3: Exemplo de produto matriz-vetor utilizando lista de adjacências.

Através da Figura 4.3 é possível observar nitidamente a redução do número de operações efetuadas ao utilizar a lista de adjacências.

4.2.3 Redução do Número de Iterações: o critério de parada

Convencionalmente o critério de parada utilizado no método das potências é definido baseado em um valor de resíduo, calculado utilizando-se a diferença entre a solução corrente e a solução da iteração anterior. Quando este resíduo é pequeno o bastante (menor que um dado ε bem pequeno) o método considera que o vetor alcançou a convergência e para. Porém o AEPIG não requer valores tão refinados de solução, uma vez que o interesse é apenas saber se um valor de autocentralidade difere ou não de outro. Com isto pode-se propor um critério de parada menos rígido.

Através do método das potências conclui-se que autocentralidade pode ser encontrada de forma iterativa. Em um passo inicial o grau do vértice é considerado como autocentralidade corrente para que em um segundo passo a soma dos graus dos vizinhos seja considerada como a autocentralidade corrente, e assim sucessivamente. Entretanto para vértices

associados por um isomorfismo o valor de grau deve ser o mesmo e, pelo Teorema 4, a soma dos graus dos vizinhos também deve ser a mesma. Isto também acontece para os passos sucessivos do método das potências, ou seja, para vértices associados por um isomorfismo a soma das autocentralidades dos vizinhos também deve ser a mesma. Dessa forma não é necessário chegar à convergência do autovetor, mas o método pode parar numa iteração arbitrária, retornando um valor de autocentralidade *parcial*. Mesmo não sendo o valor exato de autocentralidade, podemos utilizá-lo como um filtro válido na geração dos agrupamentos.

O critério de parada proposto neste trabalho considera o número de agrupamentos de centralidade gerados, ou seja, a quantidade de componentes distintas encontradas no vetor solução. Ao final de cada iteração o método calcula a quantidade de componentes distintas existentes no vetor solução corrente. Ao verificar que esta quantidade não aumentou de uma iteração para outra, o método para e retorna a solução corrente.

A Figura 4.4 ilustra a aplicação passo-a-passo do método das potências adaptado ao mesmo grafo da Figura 2.3. Como ainda não existe informação discriminativa sobre os vértices, é atribuido inicialmente 1 às suas centralidades, formando um só grupo de centralidade, representado pelo círculo vermelho (quadro 1). A primeira iteração é iniciada calculando-se a soma das centralidades dos vértices adjacentes (quadro 2). No segundo passo os vértices são reagrupados de acordo com as centralidades calculadas no passo anterior (quadro 3). Como o número de grupos formados foi maior que o anterior, a segunda iteração é executada (quadros 4 e 5). Na segunda iteração são formados 5 agrupamentos (quadro 5) e uma terceira iteração é executada (quadro 6 e 7). De acordo com o critério de parada considerado neste trabalho, o método pararia na iteração 3 (quadro 7), por não ter aumentado o número de agrupamentos. Porém é ilustrada ainda a quarta iteração, em que seriam agrupados todos os vértices em grupos distintos de centralidade.

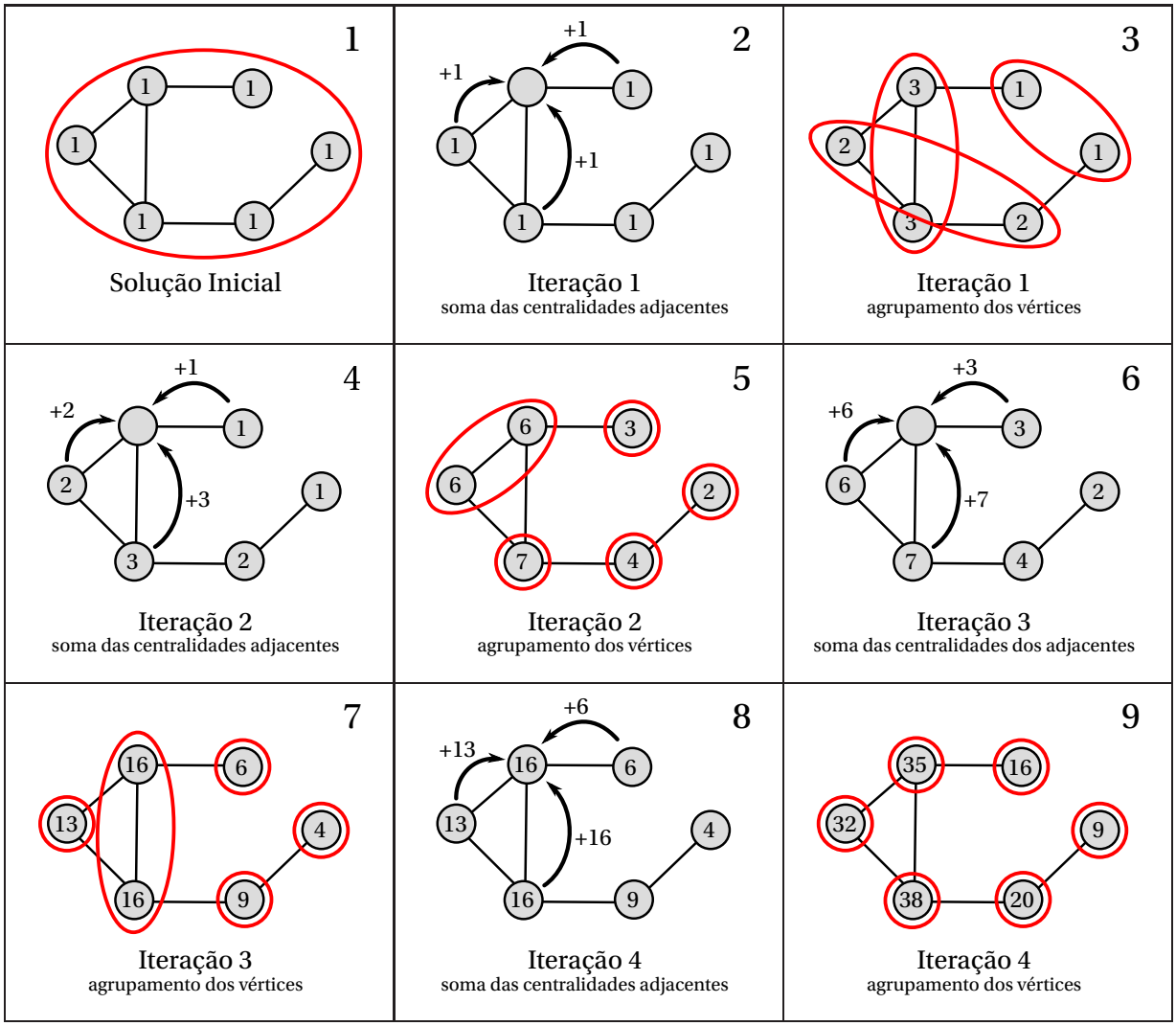


Figura 4.4: Exemplo de iterações do método das potências adaptado.

Apesar do critério de parada proposto não ser o ideal para o grafo do exemplo da Figura 4.4, nos testes computacionais realizados neste trabalho, utilizando grafos com no mínimo 20 vértices, não foi observado caso semelhante ao da figura, ou seja, que houvesse uma melhoria nos agrupamentos após uma iteração sem melhoria.

O Algoritmo 5 apresenta o pseudo-código do método das potências adaptado. Na linha 2 o algoritmo inicializa o vetor de solução inicial com a sequência de graus do grafo. Na linha 3 é calculada a quantidade de grupos de autocentralidade com o vetor inicial, através da função $conta\_grupos(\overrightarrow{v})$ que calcula a quantidade de componentes distintas existentes no vetor $\overrightarrow{v}$. Na linha 4 é realizada a primeira iteração multiplicando-se a matriz de adjacência do grafo G pelo vetor de solução inicial, utilizando-se a representação da matriz por lista de adjacências $L(G)$. Na linha 5 é calculada a quantidade de grupos de autocentralidade obtidos com o vetor $\overrightarrow{v_1}$. Na linha 6 o contador de iterações é inicializado e nas linhas 7 a 11 são realizadas as iterações onde são gerados os autovetores de solução até que não seja

observado melhoramento nos agrupamentos ou que a quantidade de agrupamentos seja igual a quantidade de vértices do grafo (linha 7).

Algoritmo 5: Método das Potências Adaptado

Entrada: Grafo: $G = (V,E)$
Saída: Vetor de autocentralidade: $\overrightarrow{v_k}$

1 **início**
2 $\quad \overrightarrow{v_0} \leftarrow graus(G)$
3 $\quad ng_0 \leftarrow conta\_grupos(\overrightarrow{v_0})$
4 $\quad \overrightarrow{v_1} \leftarrow L(G) \odot \overrightarrow{v_0}$
5 $\quad ng_1 \leftarrow conta\_grupos(\overrightarrow{v_1})$
6 $\quad k \leftarrow 1$
7 $\quad$ **enquanto** $ng_{k-1} < ng_k \; e \; ng_k < |V|$ **faça:**
8 $\quad\quad \overrightarrow{v_{k+1}} \leftarrow L(G) \odot \overrightarrow{v_k}$
9 $\quad\quad ng_{k+1} \leftarrow conta\_grupos(\overrightarrow{v_k})$
10 $\quad\quad k \leftarrow k+1$
11 $\quad$ **fim**
12 **fim**

O Algoritmo Espectral para o Problema de Isomorfismo de Grafos utilizando o método das potências Adaptado é denominado AEPIG2. Seu pseudo-código é semelhante ao Algoritmo 2, porém utiliza o método das potências adaptado para o cálculo das autocentralidades (linha 2).

5 Um Algoritmo de Rotulação Iterativa para o PIG baseado em Medidas de Centralidade

Nos capítulos anteriores a autocentralidade foi apresentada como uma invariante de apoio na busca por um isomorfismo entre dois grafos, conforme proposto em [SANTOS 2010]. Esse último trabalho mostrou empiricamente, através de extensos testes computacionais com o AEPIG, que esta invariante possui grande poder discriminativo.

Neste capítulo explicamos formalmente esta eficiência utilizando o Teorema 4 apresentado na Seção 3.1. Em seguida, uma vez compreendida a origem desta eficiência, propomos um novo algoritmo para a resolução do PIG propondo um aperfeiçoamento da técnica discriminativa implicitamente utilizada pelo AEPIG.

5.1 A Eficiência do AEPIG

No Capítulo 4 foi apresentado o AEPIG, algoritmo que utiliza o valor de autocentralidade dos vértices como propriedade invariante no auxílio da busca de um isomorfismo. Esta estratégia se mostrou bastante eficiente devido ao poder discriminativo da autocentralidade.

Nos casos gerais observou-se que dificilmente os vértices de um grafo possuem autocentralidades semelhantes entre si, fazendo desta propriedade uma invariante bastante discriminativa. Sendo assim, podemos utilizá-la para, facilmente, encontrar um isomorfismo entre dois grafos, caso exista. Nos casos em que os vértices possuam autocentralidades todas distintas, podemos já dizer que os grafos são isomorfos [SANTOS 2010].

Na Seção 2.2.1 o método das potências foi utilizado para mostrar que a autocentralidade de um vértice é encontrada realizando-se somas iterativas a partir do valor do grau de seus vértices adjacentes. Ao realizarmos esta soma iterativa, estamos na verdade aplicando o Teorema 4: calculando a partir de uma invariante inicial uma outra invariante utilizando informações da adjacência. O que difere uma iteração do método das potências de uma apli-

cação direta do Teorema 4 é a operação de soma que é realizada sobre os valores do conjunto de invariantes dos vizinhos após este ser construído.

Vamos observar esta diferença com um exemplo. Considere o processo de rotulação iterativa para hipotéticos vértices v_1, v_2 e v_3 primeiramente aplicando-se a soma sobre os graus dos vértices adjacentes, como é feito no método das potências:

$$\begin{aligned} p_1(v_1) &= \sum_{u \in \Gamma(v_1)} p_0(u) = 1+2+3 = 6 \\ p_1(v_2) &= \sum_{u \in \Gamma(v_2)} p_0(u) = 1+1+4 = 6 \\ p_1(v_3) &= \sum_{u \in \Gamma(v_3)} p_0(u) = \quad 3+3 = 6 \end{aligned} \tag{5.1}$$

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

Aplicado o Teorema 4 diretamente aos vértices v_1, v_2 e v_3 teríam-se os multiconjuntos:

$$\begin{aligned} p_1(v_1) &= \{\!\{ p_0(u) \,|\, u \in \Gamma(v_1) \}\!\} = \{\!\{1,2,3\}\!\} \\ p_1(v_2) &= \{\!\{ p_0(u) \,|\, u \in \Gamma(v_2) \}\!\} = \{\!\{1,1,4\}\!\} \\ p_1(v_3) &= \{\!\{ p_0(u) \,|\, u \in \Gamma(v_3) \}\!\} = \quad \{\!\{3,3\}\!\} \end{aligned} \tag{5.2}$$

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

De (5.1) e (5.2) não é difícil concluir que somar os valores de invariantes reduz o poder discriminativo que teríamos se aplicássemos o teorema diretamente, pois multiconjuntos cuja soma dos elementos são diferentes obrigatoriamente devem ser diferentes, mas podem existir multiconjuntos diferentes cujas somas de seus elementos sejam iguais, como podemos observar comparando os multiconjuntos apresentados em (5.2) e as respectivas somas dos seus elementos, apresentadas em (5.1).

Esperamos então ter um maior poder discriminativo se utilizado o próprio multiconjunto como novo rótulo do vértice, ao invés de somar seus elementos. Desta forma iríamos alcançar uma rotulação mais distinta de vértices em um menor número de iterações. Motivado por esta hipótese é proposto na seção seguinte um método de rotulação iterativa.

5.2 O Algoritmo de Rotulação Iterativa

Nesta seção será apresentado um novo algoritmo para a resolução do PIG, denominado Algoritmo de Rotulação Iterativa Baseado em Medidas de Centralidade (ARIMC). Este algo-

ritmo utiliza o método de rotulação iterativa, motivado pela hipótese apresentada na seção anterior.

Algoritmo 6 descreve o método de rotulação iterativa. O método tem por entrada um grafo $G = (V,E)$ e uma propriedade invariante inicial p_0, e tem como saída uma propriedade invariante p_k, com maior poder discriminativo. As linhas de 2 a 4 representam a primeira iteração do método, em que é gerada uma nova propriedade invariante p_1 derivada de p_0, através da aplicação do Teorema 4 (linha 3). Na linha 5 o contador de iterações k é inicializado. As linhas de 6 a 11 contêm o laço das iterações de rerrotulação, as quais são executadas até que nenhum melhoramento ocorra de uma iteração para outra. Na linha 8 está a aplicação do Teorema 4.

Algoritmo 6: O Método de Rotulação Iterativa

Entrada: Grafo: $G = (V,E)$; Invariante inicial: p_0
Saída: Invariante: p_k

```
1  início
2      para v ∈ V faça:
3          p_1(v) ← {{ p_0(u) | u ∈ Γ(v) }}
4      fim
5      k ← 1
6      enquanto |grps(G, p_{k-1})| < |grps(G, p_k)| faça:
7          para v ∈ V faça:
8              p_{k+1}(v) ← {{ p_k(u) | u ∈ Γ(v) }}
9          fim
10         k ← k + 1
11     fim
12 fim
```

A estrutura do ARIMC é a mesma do AEPIG mas utiliza os rótulos gerados pelo método de rotulação iterativa como invariantes para se definir os agrupamentos ao invés dos valores de autocentralidade resultantes do método de potências modificado.

Com isto esperamos encontrar os agrupamentos em um menor número de iterações e assim, em um menor tempo de execução. O novo método geraria rótulos mais discriminativos, diferenciando até mesmo aqueles vértices que eventualmente possuam mesma autocentralidade mas graus diferentes.

5.2.1 O Algoritmo de Rotulação Iterativa Baseado em Medidas de Centralidade

De maneira geral o algoritmo inicialmente considera o grau de cada vértice como a propriedade invariante inicial. Em seguida é gerada uma nova rotulação para cada um dos vér-

tices a partir dos rótulos de seus respectivos vizinhos com base no Teorema 4. Os vértices então são agrupados por rótulos e o número de agrupamentos gerados é utilizado como critério de parada. O processo de rotulação termina quando o número de agrupamentos gerados é igual ao número total de vértices ou igual ao número de agrupamentos da propriedade invariante da iteração anterior. Caso contrário o algoritmo realiza uma nova iteração de rerrotulação com o propósito de gerar uma rotulação mais discriminativa. Um exemplo pode ser observado na Figura 5.1.

A Figura 5.1 ilustra uma iteração do método de rotulação iterativa aplicado ao mesmo grafo das Figuras 2.3 e 4.4. O método inicia a partir de uma rotulação inicial dos vértices, neste caso o grau dos vértices foi utilizado (quadro 1). Dado início à primeira iteração, para cada um dos vértices um multiconjunto é construído com base nos graus dos adjacentes (quadro 2). Os multiconjuntos são rotulados (quadro 3) e os vértices agrupados de acordo com os rótulos gerados (quadro 4). Neste caso todos os vértices são agrupados em conjuntos distintos em apenas uma iteração.

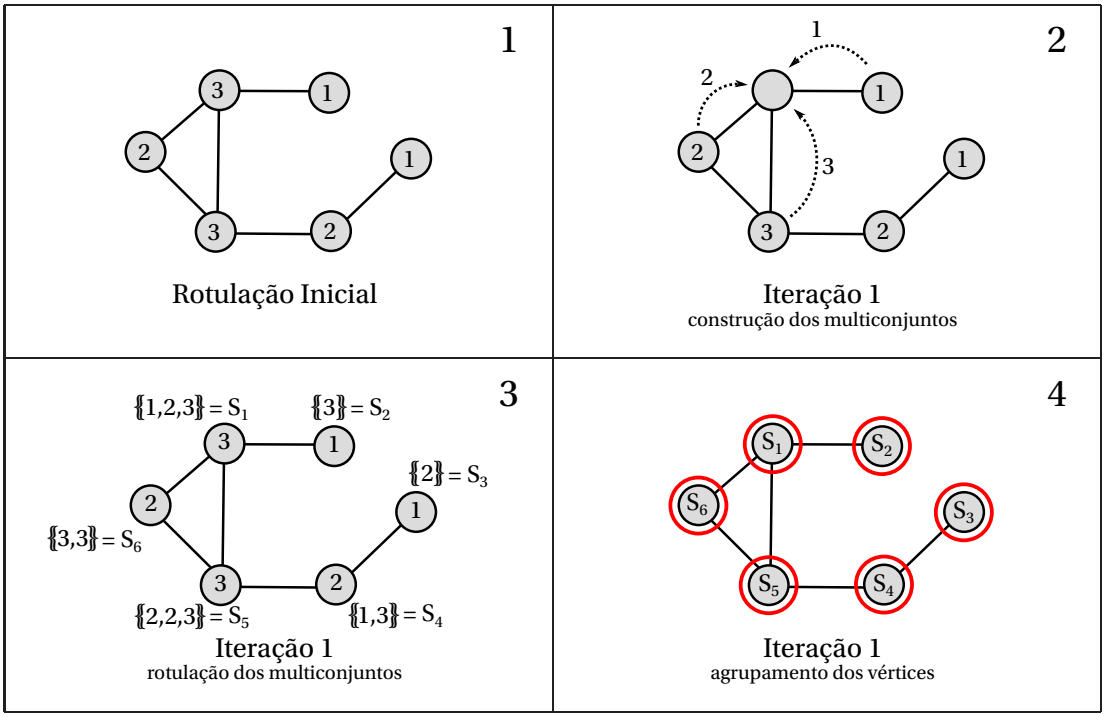


Figura 5.1: Exemplo de iterações do método de rotulação iterativa.

5.2.2 Tratando Grafos Regulares: A Vizinhança $\Gamma_2(v)$

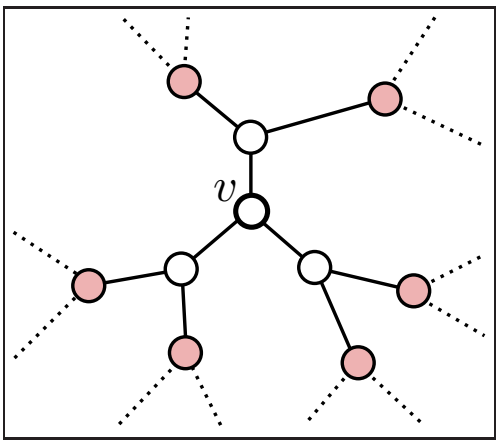
Como vimos no Capítulo 4 o filtro espectral é bastante eficiente para grafos não regulares, principalmente para uma grande quantidade de grafos, os quais possuem valores distintos de centralidade para cada um de seus vértices. A mesma eficiência é observada no método

de rotulação iterativa baseada nos graus.

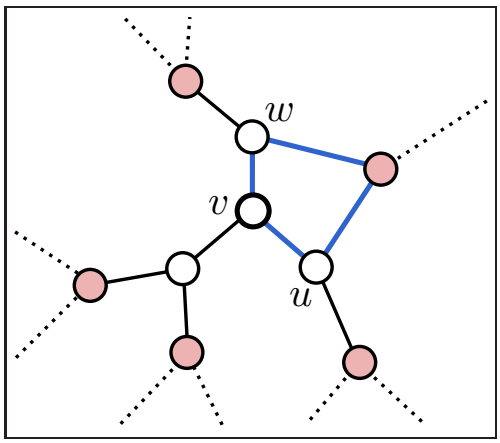
No entanto, ambas as técnicas são ineficientes para grafos regulares. O filtro espectral é ineficiente porque todos os vértices possuem a mesma autocentralidade e a rotulação iterativa, baseada no grau, também é ineficiente, uma vez que todos os vértices possuem o mesmo número de vizinhos com o mesmo grau, gerando sempre rótulos iguais para todos os vértices. Um desafio é encontrar uma invariante, com respeito aos vértices, fácil de ser calculada e que seja diferente mesmo entre os vértices de grafos regulares.

No Capítulo 2, definimos como $\Gamma_d(v)$ o conjunto de vértices que distam d de v. Explorando esta definição, observamos que o tamanho da vizinhança $\Gamma_2(v)$ pode variar mesmo em grafos regulares, como é possível observar na Figura 5.2.

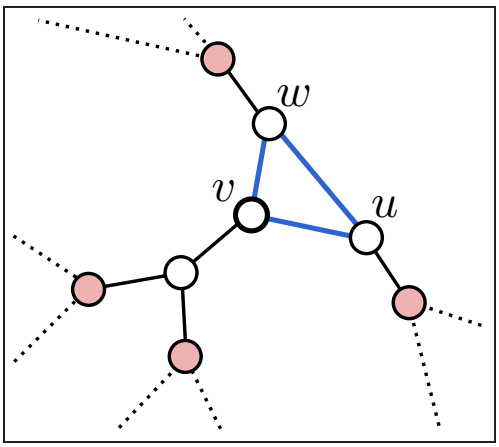
Podemos então utilizar o tamanho desta vizinhança como propriedade invariante inicial para o método de rotulação iterativa, no caso de grafos regulares. Caso o tamanho da vizinhança $\Gamma_2(v)$ seja ainda a mesma para todos os vértices é computado o tamanho da vizinhança $\Gamma_3(v)$.



(a) Vizinhança $\Gamma_2(v)$ de tamanho 6 em um grafo 3-regular.



(b) Presença de um quadrado diminuiu para 5 o tamanho da vizinhança $\Gamma_2(v)$.



(c) Presença de um triângulo diminuiu para 4 o tamanho da vizinhança $\Gamma_2(v)$.

Figura 5.2: Partes de um grafo 3-regular exemplificando vizinhanças $\Gamma_2(v)$. As linhas pontilhadas representam arestas restantes do grafo.

A Figura 5.2 ilustra três casos de vizinhança $\Gamma_2(v)$ num grafo 3-regular. Em (a) é apresentada uma vizinhança $\Gamma_2(v)$ em que os vértices adjacentes a v não compartilham nenhum outro vizinho entre si, contabilizando assim 6 vértices. Pode-se observar que em (b) as adjacências dos vértices w e u possuem um vértice em comum, além do próprio v, fazendo com que a vizinhança $\Gamma_2(v)$ contabilize 5 vértices. No caso (c) os vértices w e u, apesar de serem ambos vizinhos de um vizinho de v, distam 1 de v, o que os exclui da vizinhança $\Gamma_2(v)$ reduzindo seu tamanho para 4 vértices.

Uma vez proposto o método de rotulação iterativa o desafio é encontrar uma maneira coerente e eficiente de rotularmos um multiconjunto, particularmente, um multiconjunto de inteiros. Uma proposta é apresentada na Seção 5.3.

5.3 Estratégias de Implementação para Melhoria de Eficiência do Método

Uma forma intuitiva de rotularmos um multiconjunto de inteiros é utilizando um vetor de inteiros onde cada elemento do vetor representa um elemento do multiconjunto. Para realizarmos uma operação de comparação entre dois multiconjuntos de tamanho n é preciso, em primeiro lugar, certificar-se que os vetores estão ordenados e então realizar n operações de comparação entre os seus elementos par a par.

Porém esta abordagem é bastante custosa, principalmente quando a equiparamos com a forma utilizada no método das potências onde, mesmo perdendo em poder discriminativo, só é preciso realizar uma comparação entre dois inteiros, o que demanda tempo e espaço constante.

O ideal seria obter uma operação de rotulação que represente um multiconjunto como um só número inteiro, permitindo realizar comparações de forma eficiente. Deve ser também uma operação que não interfira no poder discriminativo dos rótulos, evitando ao máximo que multiconjuntos diferentes resultem em um mesmo rótulo. Finalmente, esta operação deve ser preferencialmente comutativa[1], pois durante a iteração do método de rotulação, para um par de vértices mapeados por um isomorfismo (caso exista), a vizinhança é visitada em ordem distinta, pois os grafos estão rotulados de formas diferentes. Porém os rótulos finais destes dois vértices devem ser os mesmos. Para uma operação não comutativa seria necessário ordenar os rótulos da vizinhança antes de realizarmos as operações, o que não é interessante.

A operação de multiplicação é comutativa, porém os valores cresceriam muito rápido após algumas iterações, o que computacionalmente provocaria um *overflow* das variáveis. Por outro lado operações lógicas binárias como *ou (OR)*, *e (AND)* e *ou exclusivo (XOR)* são eficientes operações binárias sobre inteiros, além de serem comutativas. Estas operações lógicas binárias são executadas sobre a representação binária dos respectivos valores inteiros conforme a Figura 5.3.

A Figura 5.3 ilustra a aplicação das operações lógicas binárias AND, OR e XOR sobre um mesmo conjunto de valores todos com 8 bits. Para mostrar a comutatividade das operações, cada uma delas é aplicada duas vezes (esquerda e direita) sobre os mesmos valores dispostos em ordens distintas. Em (a) é realizada a operação AND. É possível observar que esta operação teve como resultado um inteiro com 0 em muitas posições. Isto ocorre pois, para que uma posição resulte em 1 nesta operação é necessário que em todos os inteiros a serem ope-

[1] uma operação $*$ é comutativa se $a * b = b * a$.

rados os bits daquela posição sejam 1. No exemplo isto ocorre em apenas uma das posições. Isto tende a gerar um mesmo valor como resultado, com 0 em todas as posições. Deficiência semelhante ocorre com a operação OR (b), para que uma posição resulte em 0 é preciso que em todos os inteiros a serem operados os bits daquela posição sejam 0, tendendo a gerar um mesmo inteiro como resultado, com 1 em todas as posições. Entretanto a operação XOR não apresenta esta deficiência, como pode ser observado no exemplo, em (c).

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

(a) Comutatividade da operação AND.

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

(b) Comutatividade da operação OR.

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

(c) Comutatividade da operação XOR.

Figura 5.3: Operações binárias aplicadas a um mesmo conjunto de valores.

Porém aplicar a operação XOR diretamente sobre inteiros pequenos, em relação ao inteiro máximo suportado pela máquina (que é o caso dos graus dos vértices de um grafo) restringe as possibilidades de rótulos, uma vez que os bits mais a esquerda são sempre 0. Considere por exemplo a operação de rotulação sobre vértices de um grafo $G = (V,E)$, $d(v) < 8, \forall v \in V$. Neste caso vários multiconjuntos diferentes obteriam o mesmo rótulo. A Figura 5.4 ilustra o exemplo, aplicando a operação XOR sobre alguns multiconjuntos contendo valores

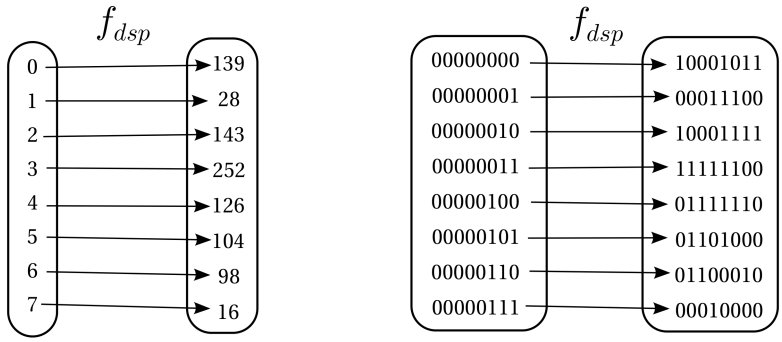
de graus de um grafo, cujo grau máximo é 7. A aplicação da operação XOR diretamente sobre os valores resultam em um mesmo rótulo para todos os multiconjuntos.

$\{\!\{2,2,3,4,5\}\!\}$	$\{\!\{1,3\}\!\}$	$\{\!\{2,3,3\}\!\}$	$\{\!\{2,3,4,7\}\!\}$	$\{\!\{5,7\}\!\}$
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
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

Figura 5.4: Aplicação da operação XOR sobre valores de multiconjuntos gerados por um grafo

Uma forma de solucionar este problema é aplicando uma *função de dispersão* f_{dsp} [KNUTH 1998] sobre os rótulos, antes de operarmos o XOR. O objetivo da função de dispersão é mapear valores inteiros de um pequeno intervalo em inteiros em um intervalo maior, tomando proveito dos demais bits suportados pela máquina, aumentando assim as possibilidades de rótulos evitando que multiconjuntos diferentes recebam o mesmo rótulo. Definir um mapeamento aleatório entre inteiros é o bastante para se definir uma função de dispersão. É importante porém que sua imagem varie por toda a extensão do conjunto de inteiros suportados pela máquina e que esta seja uma função injetora, ou seja, para $x, y \in \mathbb{Z},\ x \neq y,\ f_{dsp}(x) \neq f_{dsp}(y)$.

A Figura 5.5 apresenta um exemplo de função de dispersão que mapeia inteiros do intervalo $[0,7]$ em inteiros no intervalo $[0,255]$. Em (a) está sua representação decimal e em (b) sua representação binária.



(a) Representação decimal. (b) Representação binária.

Figura 5.5: Exemplo de função de dispersão com domínio definido para o intervalo $[0,7]$.

Na Figura 5.6 a função de dispersão da Figura 5.5 foi utilizada na rotulação dos multiconjuntos apresentados anteriormente na Figura 5.4. Esta foi aplicada sobre os valores antes de aplicar a operação XOR. É possível observar que, com a utilização da função de dispersão, todos os multiconjuntos obtiveram rótulos distintos.

$\{\!\{2,2,3,4,5\}\!\}$	$\{\!\{1,3\}\!\}$	$\{\!\{2,3,3\}\!\}$	$\{\!\{2,3,4,7\}\!\}$	$\{\!\{5,7\}\!\}$
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
10001111				
10001111			10001111	
11111100		10001111	11111100	
01111110	00011100	11111100	01111110	01101000
XOR 01101000	**XOR** 11111100	**XOR** 11111100	**XOR** 00010000	**XOR** 00010000
11101010	11100000	10001111	00011101	01111000

Figura 5.6: Aplicação do XOR sobre valores de multiconjuntos da Figura 5.4 utilizando a função de dispersão da Figura 5.5.

Neste trabalho a função de dispersão é implementada através de um vetor estático de 10000 posições populadas com valores aleatórios, definidos no próprio código.

A seguir são apresentados o pseudo-código da função que seleciona a invariante inicial utilizada pelo método de rotulação iterativa (Algoritmo 7) e do método de rotulação iterativa baseado em medidas de centralidade (Algoritmo 8) abordando alguns detalhes de implementação.

Algoritmo 7: Função de seleção de invariante inicial (*inv_inicial*).

Entrada: Grafo: $G = (V,E)$
Saída: Invariante: p_0

```
1  início
2      se G é regular então
3          para v ∈ V faça:
4              p_0(v) ← |Γ_2(v)|
5          fim
6          se p_0(u) = p_0(v), ∀u,v ∈ V, u ≠ v então
7              para v ∈ V faça:
8                  p_0(v) ← |Γ_3(v)|
9              fim
10         fim
11     senão
12         para v ∈ V faça:
13             p_0(v) ← d(v)
14         fim
15     fim
16 fim
```

O Algoritmo 7 recebe um grafo como entrada e seleciona a invariante inicial a ser utilizada no método de rotulação iterativa. Na linha 2 é verificado se o grafo é regular, se o for nas linhas 3 a 5 é calculado para cada vértice o tamanho da vizinhança $\Gamma_2(v)$. Na linha 6 é verificado se o tamanho desta vizinhança é o mesmo para todos os vértices. Se o for nas linhas 7 a 9 é calculado o tamanho da vizinhança $\Gamma_3(v)$. Caso o grafo não seja regular o grau dos vértices é utilizado como invariante inicial (linhas 11 a 15).

Algoritmo 8: O Método de Rotulação Iterativa Baseado em Medidas de Centralidade.

```
   Entrada: Grafo: G = (V,E)
   Saída: Invariante: p_k
1  início
2      f_dsp ← {139,28,143,...,53}
3      p_0 ← inv_inicial(G)
4      para u ∈ V faça:
5          p_1(u) ← 0
6          para v ∈ Γ(u) faça:
7              p_1(u) ← p_1(u) XOR f_dsp(p_0(v))
8          fim
9      fim
10     k ← 1
11     enquanto |grps(G,p_{k-1})| < |grps(G,p_k)| faça:
12         para u ∈ V faça:
13             p_{k+1}(u) ← 0
14             para v ∈ Γ(u) faça:
15                 p_{k+1}(u) ← p_{k+1}(u) XOR p_k(v)
16             fim
17         fim
18         k ← k+1
19     fim
20 fim
```

O Algoritmo 8 mostra o método de rotulação iterativa baseado em medidas de centralidade. O algoritmo tem como entrada um grafo e como saída uma propriedade invariante bastante discriminativa. Na linha 2 é inicializado o vetor que representa a função de dispersão. Na linha 3 é selecionada a invariante inicial a ser considerada, utilizando o Algoritmo 7. Nas linhas 4 a 9 é realizada a primeira iteração de rotulação. Para cada vértice do grafo é inicializado o rótulo corrente com 0 (linha 5) e então, para cada vizinho do vértice corrente (linha 6) é aplicada a operação XOR sobre o rótulo corrente e a invariante do respectivo vizinho mapeada, pela função de dispersão (linha 7). Na linha 10 o contador de iterações é inicializado. As linhas 11 a 19 contêm o laço das iterações de rerrotulação, as quais são executadas até que nenhum melhoramento ocorra de uma iteração para outra. Apesar de não estar descrito no código, a quantidade de agrupamentos gerados, utilizada na linha 11, é contada com o auxílio de uma *tabela de dispersão* que guarda os valores de rótulos gerados na iteração corrente.

6 Resultados Computacionais

Para efeito de comparação da desempenho dos algoritmos AEPIG2 e ARIMC propostos, foram realizados testes computacionais utilizando os seguintes algoritmos conhecidos da literatura: Nauty [McKay 1984], VF2 [Cordella et al. 2001], Bliss [Junttila , Kaski 2007] e Saucy [Darga et al. 2008a, Darga et al. 2008b]. O Nauty produz uma rotulação canônica sobre os grafos de entrada através de rotulações iterativas onde um conjunto de invariantes de vértices é considerado. Os algoritmos Saucy e Bliss trabalham de forma semelhante ao Nauty mas possuem diferenças na forma de implementação. O Saucy, por exemplo, explora a esparsidade dos grafos para resolver o problema de forma mais eficiente, sendo assim mais indicado para grafos esparsos. O algoritmo VF2 se fundamenta em uma estratégia de busca em profundidade. Esta busca é orientada por um conjunto de regras que impõem condições para o processo de mapeamento, tornando-a mais eficiente.

Todos os testes foram realizados sobre o mesmo conjunto de instâncias de teste utilizados em [Santos 2010], extraído de [SIVALab 2001]. As instâncias são compostas por 3000 pares de grafos conexos gerados aleatoriamente, divididos em três grupos de densidade: $\eta = 0.01$, $\eta = 0.05$ e $\eta = 0.1$, respectivamente denotados por $r001$, $r005$ e $r01$. Cada grupo contém 100 pares de grafos de tamanhos 20, 40, 60, 80, 100, 200, 400, 600, 800 e 1000 vértices, totalizando 1000 pares de grafos em cada grupo. De acordo com o valor de η, se n é o total de vértices no grafo, o número m de arestas é igual a $\eta \left[\frac{n(n-1)}{2}\right]$. Entretanto, se este número não é suficiente para obter um grafo conexo (ao menos $n-1$ arestas) arestas são adicionadas até que o grafo gerado seja conexo.

Além destes três grupos de instâncias, uma nova instância de grupo de densidade $\eta = 0.5$ foi gerada também com 100 pares de grafos isomorfos para cada um dos dez tamanhos considerados, utilizando a ferramenta de geração de grafos aleatórios do pacote Nauty, chamada *genrang*, a qual foi denotada por $d05$. Também foram geradas duas classes de grafos regulares utilizando a ferramenta *genrang* denominadas $reg3$ e $reg7$ de graus 3 e 7 respectivamente também com 100 pares de grafos isomorfos para cada um dos dez tamanhos considerados. Vale ressaltar que para a geração das classes de grafos regulares não foi mantida entre os grafos a densidade (relativa ao máximo de arestas possíveis), como feito com os grafos não regulares,

mas apenas o grau. Desta forma, para esta classe, os grafos maiores são mais esparsos, em relação aos menores.

Ainda sobre cada par de grafos isomorfos, das seis classes utilizadas, foi gerado um grafo não isomorfo com a mesma sequência de graus, sendo possível assim verificar o desempenho dos algoritmos sobre grafos não isomorfos, porém o comportamento dos algoritmos foram extremamente semelhantes ao caso de grafos isomorfos como pode ser observado nos gráficos das figuras 6.1 a 6.12.

Todos os algoritmos utilizados nos testes, inclusive os propostos neste trabalho, foram implementados em linguagem de programação C. O AEPIG utiliza para o cálculo do vetor de autocentralidade a função `dsyevr_` da biblioteca CLAPACK 3.2.1 [ANDERSON ET AL. 1999]. O AEPIG2 utiliza o método das potências adaptado proposto neste trabalho. As implementações dos algoritmos Bliss e VF2 foram retirados do pacote iGraph 0.5.1 [CSÁRDI , NEPUSZ 2010]. A implementação do Nauty foi utilizada em sua versão 2.4 e a do Saucy em sua versão 2.1. Para medir o tempo de execução dos algoritmos a função `gettimeofday` da biblioteca C `time.h` foi utilizada. Os testes foram realizados em uma máquina com processador Intel®Core™ i5 U470@1.33GHz (3MB cache) e 4Gb de RAM utilizando o sistema operacional Linux Ubuntu 11.04 kernel 2.6.38. Vale ressaltar que os algoritmos AEPIG e o AEPIG2 não tratam grafos regulares e que o algoritmo Saucy é dedicado a grafos esparsos [DARGA ET AL. 2008b].

Nas páginas a seguir são apresentados, para cada classe, o gráfico de tempo de processamento para os algoritmos tratados para os casos de grafos isomorfos e não isomorfos respectivamente.

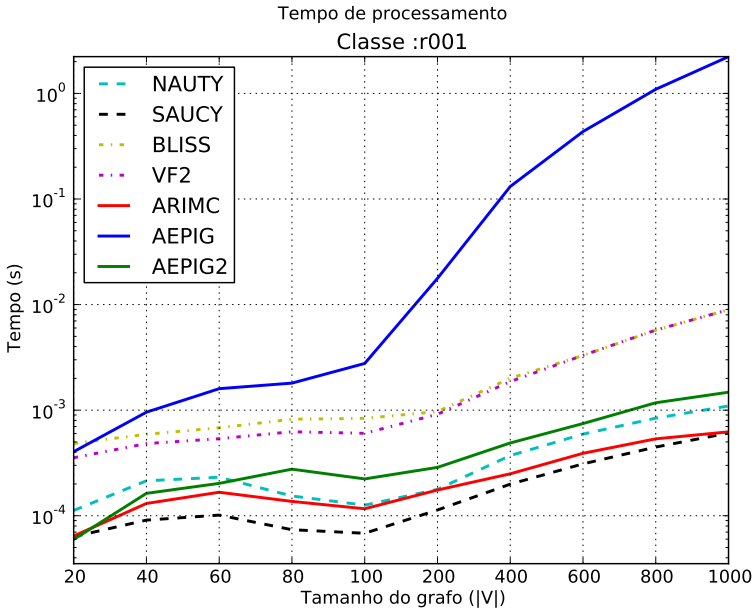


Figura 6.1: Tempo de processamento para pares de grafos isomorfos da classe *r*001.

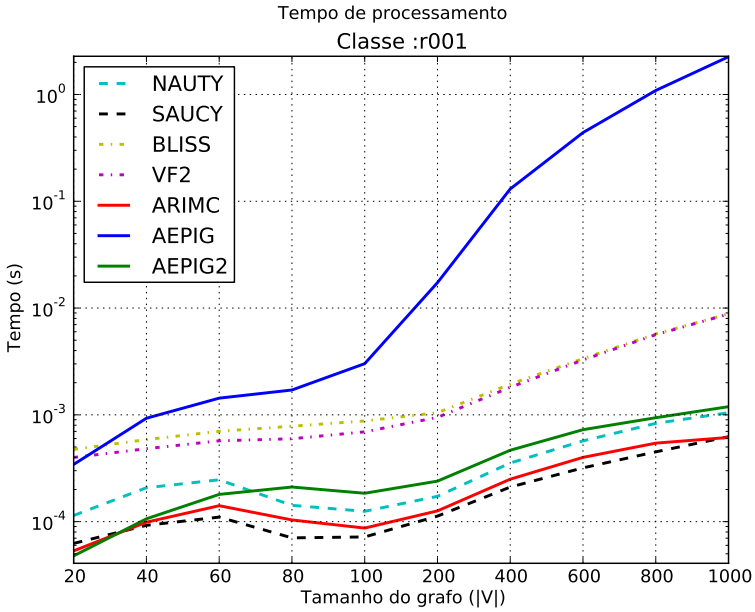


Figura 6.2: Tempo de processamento para pares de grafos não isomorfos da classe *r*001.

Nos gráficos das Figuras 6.1 e 6.2 pode-se observar que o algoritmo AEPIG executou em tempo bem maior que os demais. Os algoritmos Bliss e VF2 executaram em tempos semelhantes. Os demais algoritmos executaram em tempos relativamente próximos. O algoritmo SAUCY executou em menores tempos para todos os grafos a partir de 40 vértices, seguido do ARIMC. Os tempos de ambos porém se aproximam a medida que o tamanho do grafo aumenta se encontrando em grafos de 1000 vértices.

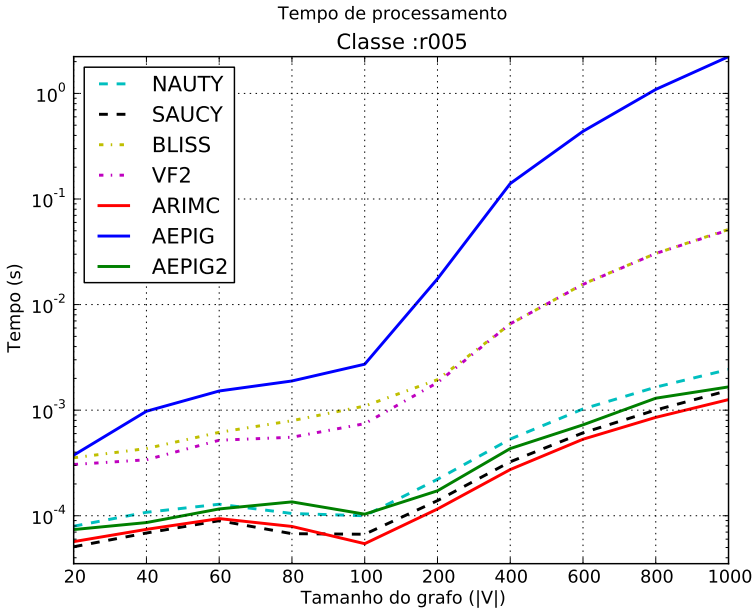


Figura 6.3: Tempo de processamento para pares de grafos isomorfos da classe *r*005.

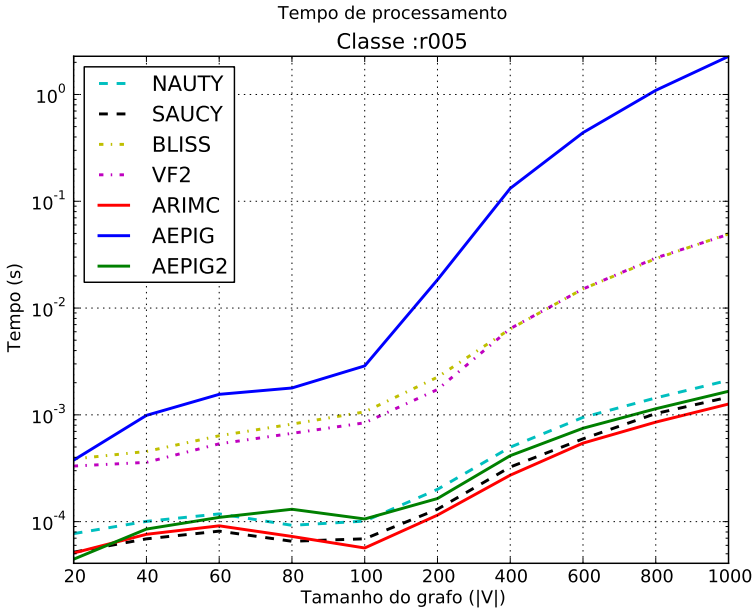


Figura 6.4: Tempo de processamento para pares de grafos não isomorfos da classe *r*005.

Nos gráficos das Figuras 6.3 e 6.4 pode-se observar com clareza que, devido a densidade da instância, os tempos obtidos pelos algoritmos se aproximaram uns dos outros de forma considerável quando comparados com os tempos obtidos para a classe *r*001. Porém o algoritmo AEPIG executou em maior tempo seguido pelo Bliss e VF2. Nos testes para esta classe de densidade o ARIMC passa a obter os melhores tempos para grafos a partir de 100 vértices. Exceto para os grafos com 80 vértices o algoritmo AEPIG2 obteve melhor tempo que o Nauty.

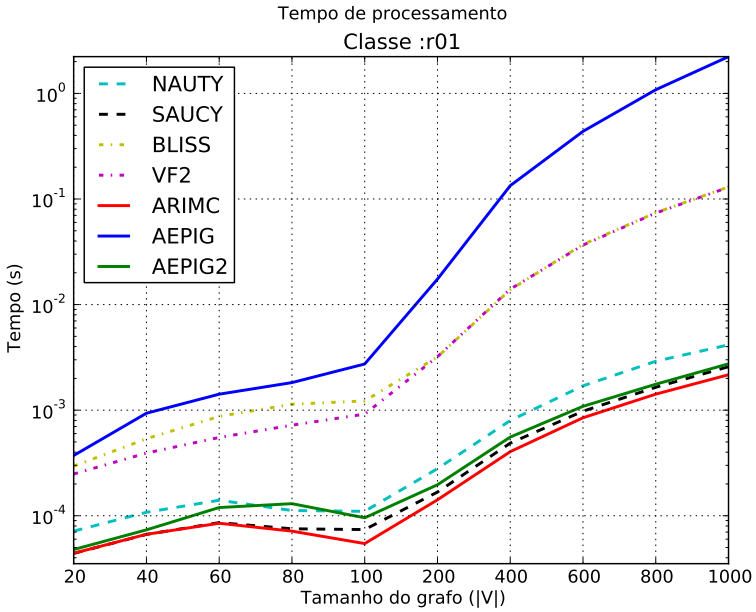


Figura 6.5: Tempo de processamento para pares de grafos isomorfos da classe *r*01.

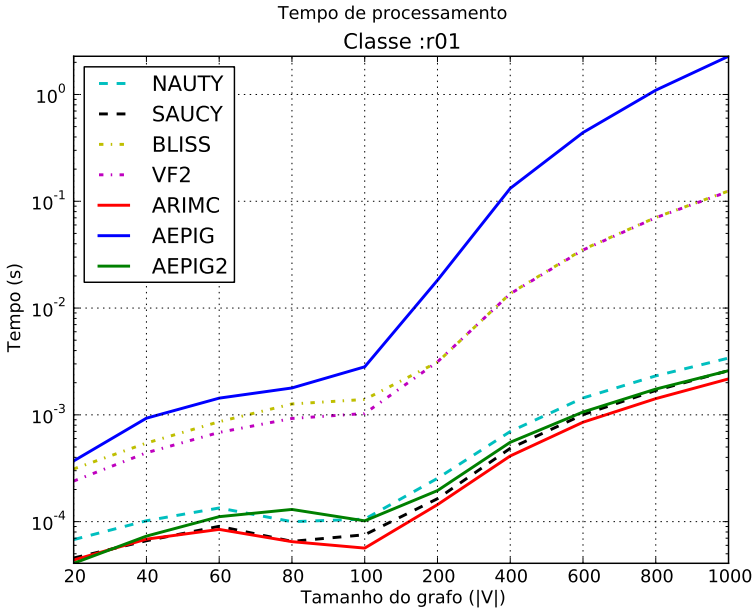


Figura 6.6: Tempo de processamento para pares de grafos não isomorfos da classe *r*01.

Os gráficos das Figuras 6.5 e 6.6 apresentam comportamentos semelhantes aos das Figuras 6.3 e 6.4. Os algoritmos Saucy e ARIMC obtiveram tempos ainda mais próximos.

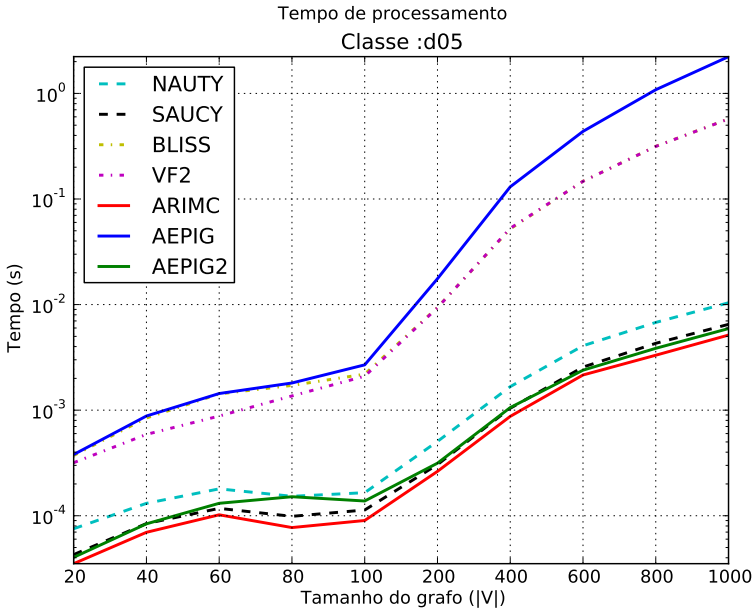


Figura 6.7: Tempo de processamento para pares de grafos isomorfos da classe *d*05.

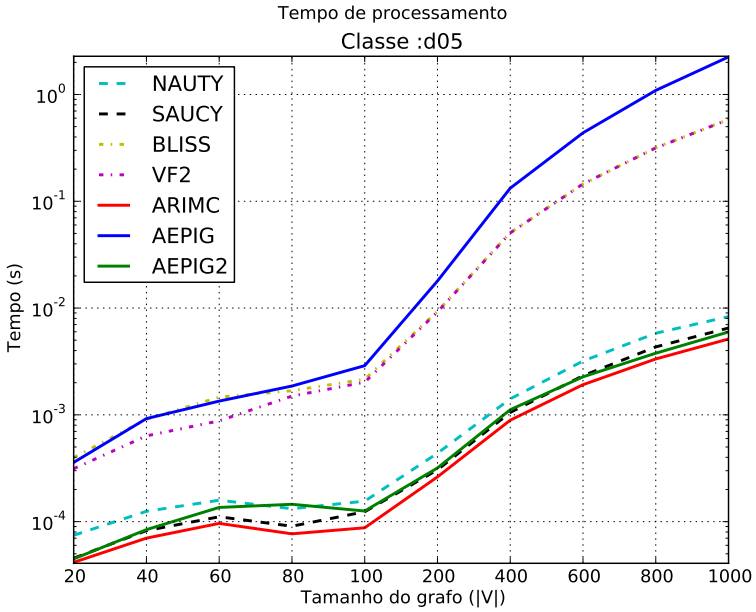


Figura 6.8: Tempo de processamento para pares de grafos não isomorfos da classe *d*05.

Os gráficos das Figuras 6.7 e 6.8 apresentam comportamentos semelhantes aos das Figuras 6.5 e 6.6. O ARIMC obteve tempos melhores que os outros em todos os tamanhos de grafos. O AEPIG2 executou em tempos menores do que o Nauty para a maioria dos grafos. O algoritmo Bliss executou em tempos melhores do que o VF2 até o tamanho de 100 vértices, a partir de 200 vértices ambos executaram em tempos semelhantes.

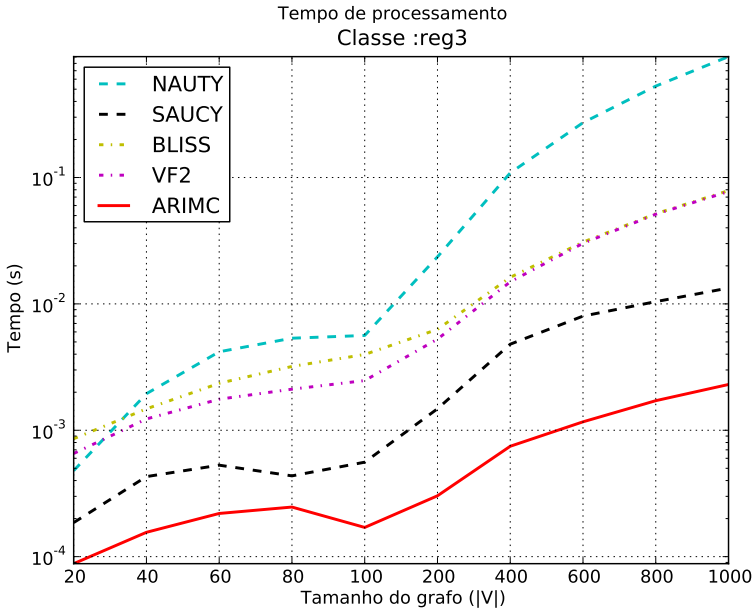


Figura 6.9: Tempo de processamento para pares de grafos isomorfos da classe *reg*3.

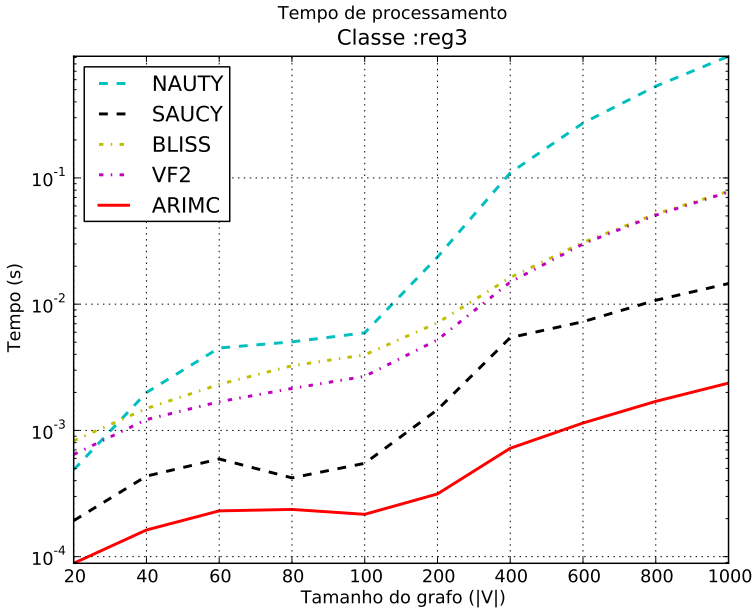


Figura 6.10: Tempo de processamento para pares de grafos não isomorfos da classe *reg*3.

Segundo os gráficos das Figuras 6.9 e 6.10 os tempos de processamento para os grafos da classe *reg*3 variaram de forma copiosa entre os algoritmos. O Nauty obteve o maior dos tempos, em seguida o Bliss juntamente com o VF2. O Saucy obteve o segundo melhor tempo em todas as intâncias. O ARIMC obteve os melhores em todas os tamanhos de grafos, com uma diferença considerável entre os outros algoritmos.

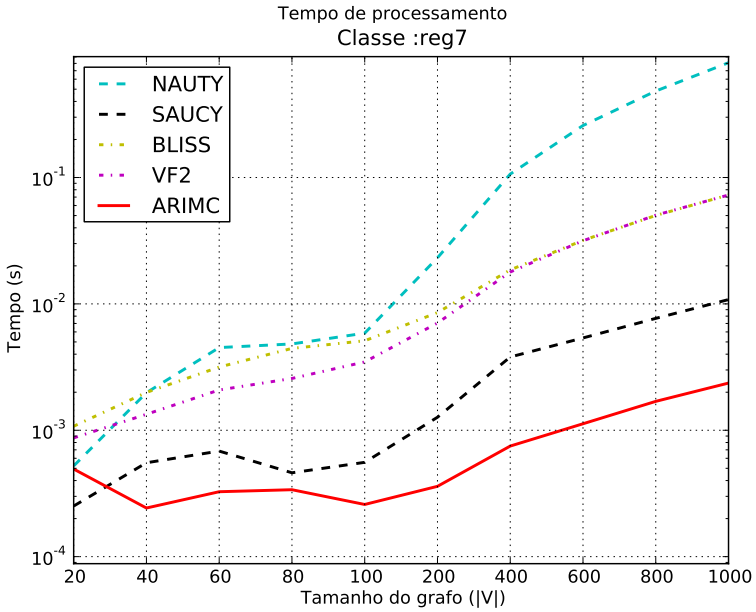


Figura 6.11: Tempo de processamento para pares de grafos isomorfos da classe *reg*7.

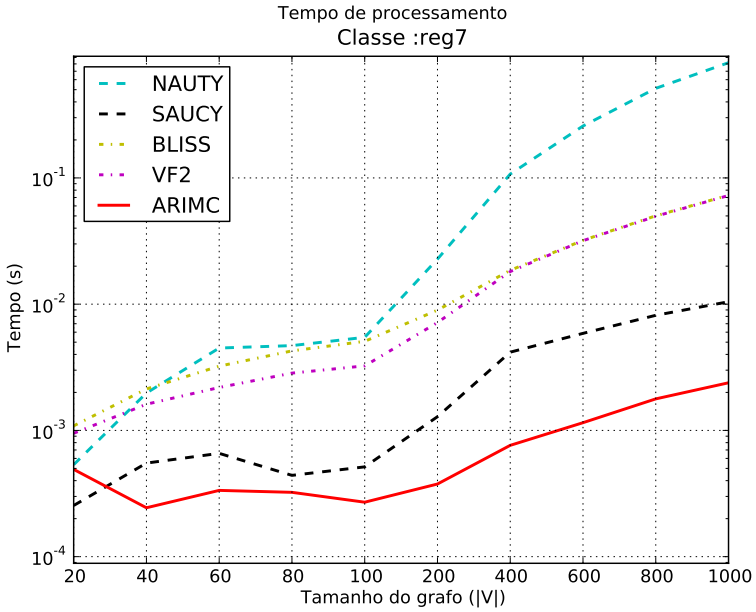


Figura 6.12: Tempo de processamento para pares de grafos não isomorfos da classe *reg*7.

Os gráficos das Figuras 6.11 e 6.12 possuem comportamento semelhante aos das Figuras 6.9 e 6.10. As diferenças entre os tempos dos algoritmos porém aumentaram para maiores número de vértices.

Na Seção 5.1 foi mencionada a esperança de se computar uma propriedade invariante com maior poder discriminativo em um menor número de iterações quando utilizado o método de rotulação iterativa ao invés do método das potências adaptado. Tendo esta hipótese como motivação são apresentados a seguir os gráficos com o número médio de iterações executadas pelo método das potências adaptado (AEPIG2) e o método de rotulação iterativa (ARIMC) ambos respeitando o mesmo critério de parada.

Vale ressaltar que para todos os pares de grafos não regulares, tanto o método das potências adaptado quanto o método de rotulação iterativa obtiveram uma rotulação capaz de distribuir os vértices em grupos distintos de associações, exceto para os grafos de tamanho 20 a 100 da classe *r*001 e de tamanho 20 para a classe *r*005. Para os grafos regulares o método de rotulação iterativa obteve tal rotulação em todos os casos.

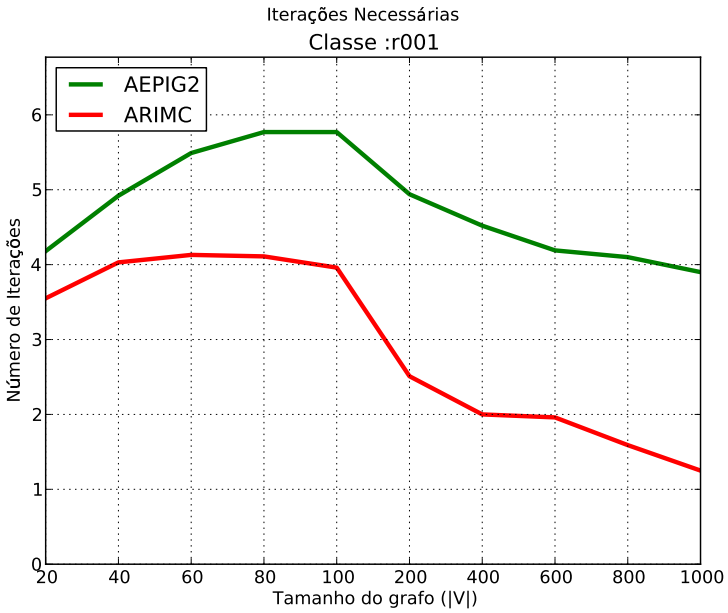


Figura 6.13: Número médio de iterações executadas para classe *r*001.

Observando o gráfico da Figura 6.13 é possível concluir que a aplicação do método de rotulação iterativa reduziu bastante o número de iterações necessárias para se alcançar o critério de parada para grafos da classe *r*001, principalmente para grafos com mais de 100 vértices. Para grafos com 1000 foram necessárias, em média, 4 iterações do método das potências adaptado, enquanto que para o método de rotulação iterativa foi necessário, em média, aproximadamente 1 iteração apenas.

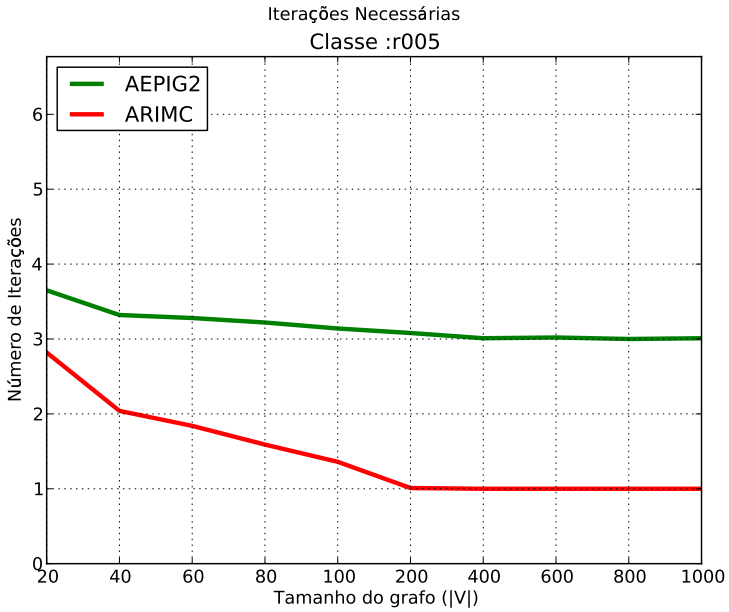


Figura 6.14: Número médio de iterações executadas para classe *r*005.

Observando o gráfico da Figura 6.14 observa-se que para a classe *r*005 não houve grande diferença para o número de iterações entre os dois métodos para os grafos com 20 vértices, porém esta diferença começa a crescer a partir de 40 vértices. A partir de 200 vértices o método de rotulação iterativa requer apenas 1 iteração para alcançar o critério de parada enquanto que o método das potências requer 3 iterações.

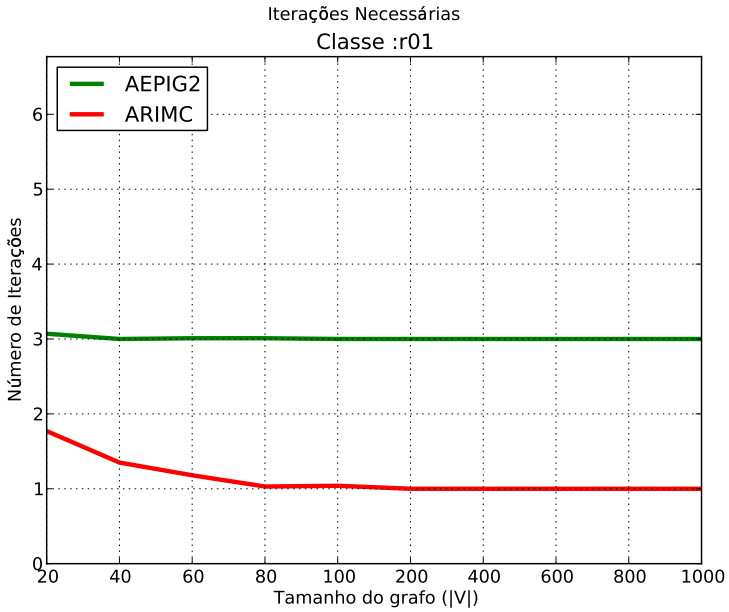


Figura 6.15: Número médio de iterações executadas para classe $r01$.

O gráfico da Figura 6.15 possui comportamento semelhante ao da Figura 6.14 porém a diferença entre o número de iterações realizadas pelos métodos começa a crescer a partir de grafos bem pequenos. Para grafos maiores que 60 o método de rotulação iterativa requereu apenas 1 iteração, enquanto que o método das potências requereu 3 iterações.

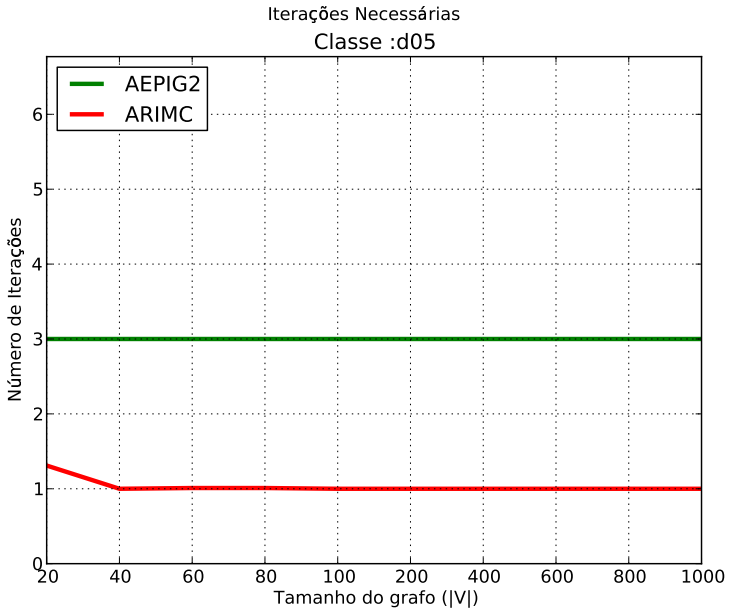


Figura 6.16: Número médio de iterações executadas para classe $d05$.

Observando a Figura 6.16 é possível concluir que para a classe de grafos $d05$ o número de iterações realizadas para se atingir o critério de parada foi constante em ambos os métodos para grafos com mais de 20 vértices sendo necessárias 3 iterações para o método das potências adaptado e 1 iteração para o método de rotulação iterativa.

A seguir são apresentados os gráficos mostrando o número médio de iterações executadas pelo método de rotulação iterativa para as classes de grafos regulares.

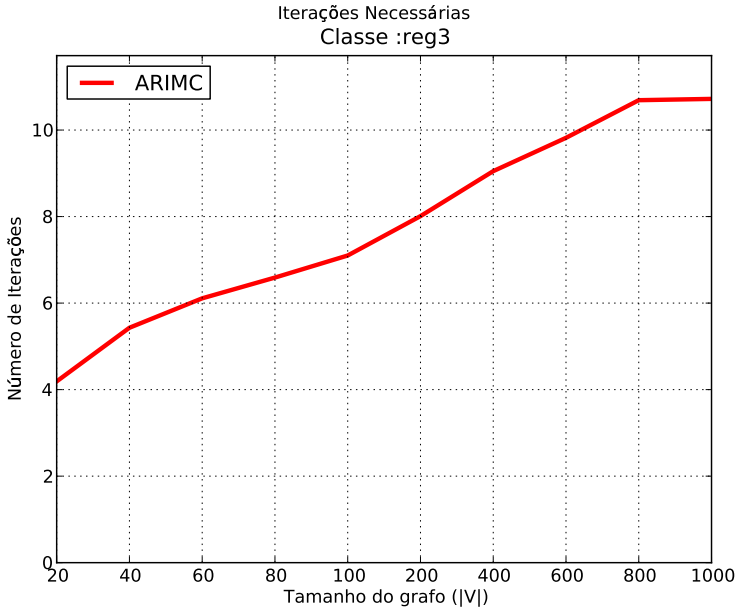


Figura 6.17: Número médio de iterações executadas para classe *reg*3.

No caso da classe *reg*3 o número de iterações necessárias para se atingir o critério de parada aumentou quase linearmente até grafos de tamanho 800. Este crescimento no número de iterações pode ter ocorrido pelo fato de ser uma instância esparsa, ou seja, com menores vizinhanças $\Gamma_2(v)$. Como estas vizinhanças são menores torna-se mais difícil observar variações no tamanho das mesmas, exigindo do método mais iterações para se construir um bom filtro.

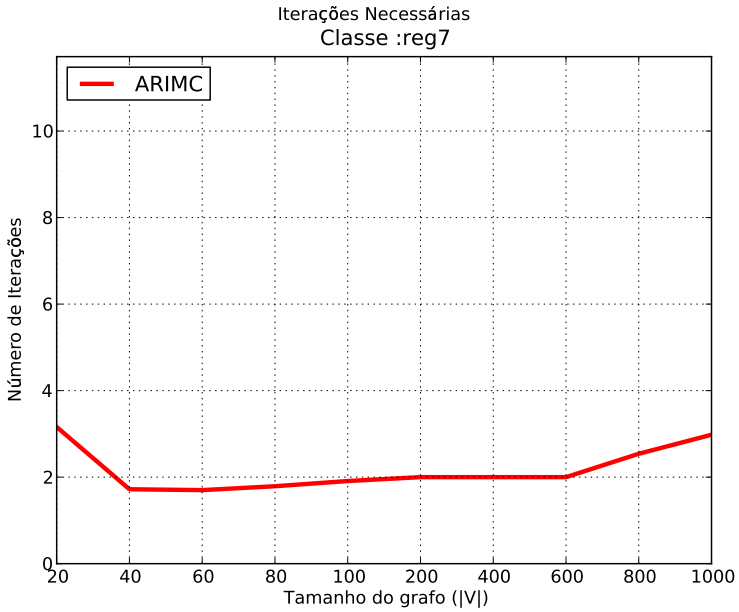


Figura 6.18: Número médio de iterações executadas para classe *reg*7.

Observando o gráfico da Figura 6.18 é possível concluir que foi mais fácil construir uma rotulação discriminativa

De modo geral o algoritmo ARIMC obteve os menores tempos de execução acompanhado pelo algoritmo Saucy. Em todos os casos o ARIMC foi executado mais rapidamente que o AEPIG2. Pode-se obervar também que o AEPIG mantém o seu tempo constante em relação a densidade dos grafos. Isto se deve à função utilizada para o cálculo das autocentralidades que, por ser uma função que calcula autovetores de uma matriz no caso geral, desconsidera a esparsidade ou qualquer outra peculiaridade da matriz.

7 Conclusões e Trabalhos Futuros

Este trabalho teve por objetivo (1) investigar a eficiência da aplicação da autocentralidade no Problema de Isomorfismo de Grafos, (2) propor uma forma mais eficiente de cálculo das autocentralidades além de (3) propor o aperfeiçoamento de sua aplicação.

A eficiência da aplicação da autocentralidade, bem como a facilidade em agrupar os vértices em grupos distintos de autocentralidade, encontrada especialmente em grafos densos e também observada em [SANTOS 2010], a meta (1) pôde ser compreendida através do método das potências aliado ao Teorema 4. Vértices de grafos mais densos tendem a possuir vizinhanças mais variadas resultando em autocentralidades também mais variadas.

Como forma mais eficiente de cálculo da autocentralidade (meta 2) foi proposto um método das potências adaptado ao PIG, dando origem ao AEPIG2 que executou em $97,3\%$ menos tempo que o AEPIG.

A variação na vizinhança dos grafos (meta 3) pôde ser melhor explorada através da proposta do método de rotulação iterativa, o qual obtém grupos de associações de vértices em um menor número de iterações que o método das potências, especialmente para grafos grandes e densos, para os quais foi necessária apenas uma 1 iteração para que os vértices fossem agrupados em grupos de associações distintos. O método de rotulação iterativa também possibilitou a resolução do PIG no caso de grafos regulares para os quais apresentou tempo de execução inferior a dos algoritmos abordados nos testes computacionais para grafos com mais de 20 vértices.

A comparação dos resultados computacionais constataram que o ARIMC soluciona o PIG com eficiência, sendo superior ao AEPIG2 em todos os casos. A eficiência do método de rotulação iterativa se destaca ainda mais nos casos de grafos regulares, com os quais apresentou tempo de processamento bem inferior aos demais algoritmos, nos levando a concluir que as vizinhanças $\Gamma_2(v)$ e $\Gamma_3(v)$ dão origem a excelentes invariantes discriminativas para grafos regulares.

Os trabalhos futuros são:

- investigar se as propriedades invariantes utilizadas pelo ARIMC são propriedades invariantes completas. Caso contrário, investigar como torná-las completas, evoluindo assim o ARIMC para um algoritmo de rotulação canônica;
- testar o desempenho dos algoritmos propostos aplicados a grafos regulares de maior densidade e a grafos fortemente regulares;
- encontrar uma melhor forma de rotular multiconjuntos de inteiros. Por ser inversa de si própria, a operação XOR é ineficaz se aplicada duas vezes sobre um mesmo valor, fazendo com que alguns multiconjuntos diferentes, como por exemplo $\{\!\{1,3,3\}\!\}$ e $\{\!\{1\}\!\}$, obtenham o mesmo rótulo.

Referências Bibliográficas

[ANDERSON ET AL. 1999] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., , Sorensen, D. LAPACK User's Guide. http://www.netlib.org/lapack/lug/, 1999.

[ARVIND , TORÁN 2005] Arvind, V. , Torán, J. Isomorphism Testing: Perspective and Open Problems. *Bulletin European Association of Theoretical Computer Science*, volume 86:66–84, 2005.

[CONTE ET AL. 2004] Conte, D., Foggia, P., Sansone, C. , Vento, M. Thirty Years Of Graph Matching In Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 2004.

[COOK , HOLDER 2007] Cook, D. , Holder, L. Mining graph data. Wiley-Interscience, 2007. ISBN 9780471731900.

[CORDELLA ET AL. 2001] Cordella, L. P., Foggia, P., Sansone, C. , Vento, M. An Improved Algorithm for Matching Large Graphs. In 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, pp. 149–159. 2001.

[CSÁRDI , NEPUSZ 2010] Csárdi, G. , Nepusz, T. The igraph library for complex network research. 2010. URL http://igraph.sourceforge.net/.

[DALCUMUNE 2008] Dalcumune, E. Algoritmos Quânticos para o Problema do Isomorfismo de Grafos. dissertação de mestrado, Laboratório Nacional de Computação Científica, Petrópolis, 2008.

[DARGA ET AL. 2008a] Darga, P. T., Katebi, H., Liffiton, M., Markov, I. , Sakallah, K. Saucy 2.0. http://vlsicad.eecs.umich.edu/BK/SAUCY/, 2008a.

[DARGA ET AL. 2008b] Darga, P. T., Sakallah, K. A. , Markov, I. L. Faster symmetry discovery using sparsity of symmetries. In Proceedings of the 45th annual Design Automation Conference, DAC '08, pp. 149–154. ACM, New York, NY, USA, 2008b. ISBN 978-1-60558-115-6.

[DE ABREU ET AL. 2007] de Abreu, N. M. M., Del-Vecchio, R. R., Vinagre, C. T. M. , Stevanović, D. Introdução à Teoria Espectral de Grafos com Aplicações, SBMAC. 2007.

[DE CÁSSIA NANDI 2006] de Cássia Nandi, R. Isomorfismo de Grafos Aplicado à Comparação de Impressões Digitais. dissertação de mestrado, Universidade Federal do Paraná, 2006.

[DE FREITAS 2010] de Freitas, L. Q. Medidas de Centralidades em Grafos. dissertação de mestrado, Universidade Federal do Rio de Janeiro, 2010.

[DIESTEL 2005] Diestel, R. Graph Theory (Graduate Texts in Mathematics). Springer, 2005. ISBN 3540261826.

[FAROUK 2011] Farouk, R. M. Iris recognition based on elastic graph matching and Gabor wavelets. *Computer Vision and Image Understanding*, volume 115(8):1239 – 1244, 2011. ISSN 1077-3142. doi:10.1016/j.cviu.2011.04.002.

[FORTIN 1996] Fortin, S. The Graph Isomorphism Problem. Technical report, University of Alberta, Edmonton, Alberta, Canada, 1996.

[HOGBEN 2009] Hogben, L. Spectral Graph Theory and The Inverse Eigenvalue Problem of a Graph. *Chamchuri Journal of Mathematics*, volume 1(1):51–72, 2009.

[HORN , JOHNSON 1990] Horn, R. A. , Johnson, C. R. Matrix Analysis. Cambridge University Press, 1990. ISBN 0521386322.

[JENNER ET AL. 2003] Jenner, B., K obler, J., McKenzie, P. , Torán, J. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, volume 66(3):549–566, 2003.

[JUNTTILA , KASKI 2007] Junttila, T. , Kaski, P. Engineering an efficient canonical labeling tool for large and sparse graphs. In D. Applegate, G. S. Brodat, D. Panario , R. Sedgewick, editors, Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics. SIAM, 2007.

[KNUTH 1998] Knuth, D. The art of computer programming: Sorting and searching. The Art of Computer Programming. Addison-Wesley, 1998. ISBN 9780201896855.

[MARTINS ET AL. 2011] Martins, C., Cesar, R., Jorge, L. , Freitas, A. Segmentation of Similar Images Using Graph Matching and Community Detection, *Graph-Based Representations in Pattern Recognition*, volume 6658. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-20843-0.

[MCKAY 1981] McKay, B. D. Practical Graph Isomorphism. In Congressus Numerantium, volume 30, pp. 45–87. 1981.

[MCKAY 1984] McKay, B. D. The nauty page. 1984. URL `http://cs.anu.edu.au/~bdm/nauty/`.

[OLIVEIRA , GREVE 2005] Oliveira, M. O. , Greve, F. G. Um Novo Algoritmo de Refinamento para Testes de Isomorfismo em Grafos. *XXV Congresso da Sociedade Brasileira de Computação*, 2005.

[PEDARSANI , GROSSGLAUSER 2011] Pedarsani, P. , Grossglauser, M. On the Privacy of Anonymized Networks. In Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD-2011). 2011.

[RODRIGUES ET AL. 2011] Rodrigues, D. B., Rangel, M. C. , Boeres, M. C. S. O Uso de Autocentralidades na Resolução do Problema de Isomorfismo de Grafos Regulares. *Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional*, 2011.

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

[SANTOS 2010] Santos, P. L. F. Teoria Espectral de Grafos Aplicada ao Problema de Isomorfismo de Grafos. dissertação de mestrado, Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória, ES, 2010.

[SIVALAB 2001] SIVALab. The Graph Database CD. `http://amalfi.dis.unina.it/graph/doc/graphdb.html`, 2001. A huge collection of graphs realized by the *Intelligent Systems and Artificial Vision Lab. (SIVALab) of the University of Naples "Federico II"*.

APÊNDICE A – Tabelas dos Resultados Computacionais

*r*001	Nauty	Saucy	Bliss	VF2	ARIMC	AEPIG	AEPIG2
20	0.00011	**0.00006**	0.00048	0.00035	**0.00006**	0.00040	**0.00006**
40	0.00021	**0.00009**	0.00059	0.00048	**0.00013**	0.00096	0.00016
60	0.00023	**0.00010**	0.00068	0.00054	**0.00017**	0.00160	0.00020
80	0.00015	**0.00007**	0.00082	0.00062	**0.00014**	0.00180	0.00028
100	0.00013	**0.00007**	0.00084	0.00060	**0.00012**	0.00277	0.00022
200	**0.00018**	**0.00011**	0.00097	0.00091	**0.00018**	0.01771	0.00029
400	0.00037	**0.00020**	0.00200	0.00186	**0.00025**	0.13127	0.00049
600	0.00059	**0.00031**	0.00330	0.00330	**0.00039**	0.43512	0.00075
800	0.00084	**0.00045**	0.00572	0.00573	**0.00053**	1.09406	0.00117
1000	0.00110	**0.00061**	0.00894	0.00893	**0.00062**	2.22899	0.00148

Tabela A.1: Tempo médio (em segundos) de processamento para a classe *r*001.

*r*005	Nauty	Saucy	Bliss	VF2	ARIMC	AEPIG	AEPIG2
20	0.00008	**0.00005**	0.00035	0.00031	**0.00006**	0.00037	0.00007
40	0.00011	**0.00007**	0.00043	0.00034	**0.00007**	0.00098	0.00009
60	0.00013	**0.00009**	0.00062	0.00052	**0.00009**	0.00152	0.00012
80	0.00011	**0.00007**	0.00079	0.00055	**0.00008**	0.00189	0.00014
100	0.00010	**0.00007**	0.00109	0.00074	**0.00005**	0.00272	0.00010
200	0.00022	**0.00014**	0.00194	0.00182	**0.00012**	0.01754	0.00017
400	0.00053	**0.00032**	0.00655	0.00657	**0.00027**	0.14001	0.00043
600	0.00103	**0.00061**	0.01563	0.01553	**0.00053**	0.43754	0.00073
800	0.00165	**0.00100**	0.03048	0.03049	**0.00085**	1.09086	0.00130
1000	0.00243	**0.00154**	0.05151	0.05088	**0.00126**	2.22977	0.00166

Tabela A.2: Tempo médio (em segundos) de processamento para a classe *r*005.

*r*01	Nauty	Saucy	Bliss	VF2	ARIMC	AEPIG	AEPIG2
20	0.00007	**0.00004**	0.00029	0.00025	**0.00004**	0.00037	0.00005
40	0.00011	**0.00007**	0.00054	0.00039	**0.00007**	0.00094	0.00007
60	0.00014	**0.00009**	0.00087	0.00055	**0.00008**	0.00142	0.00012
80	0.00011	**0.00008**	0.00114	0.00072	**0.00007**	0.00182	0.00013
100	0.00011	**0.00007**	0.00122	0.00092	**0.00005**	0.00274	0.00010
200	0.00028	**0.00017**	0.00319	0.00321	**0.00014**	0.01743	0.00020
400	0.00080	**0.00049**	0.01382	0.01399	**0.00040**	0.13369	0.00056
600	0.00170	**0.00098**	0.03712	0.03659	**0.00085**	0.43717	0.00109
800	0.00290	**0.00165**	0.07420	0.07364	**0.00142**	1.08346	0.00176
1000	0.00415	**0.00258**	0.12997	0.12916	**0.00216**	2.23015	0.00275

Tabela A.3: Tempo médio (em segundos) de processamento para a classe *r*01.

*d*05	Nauty	Saucy	Bliss	VF2	ARIMC	AEPIG	AEPIG2
20	0.00008	**0.00004**	0.00037	0.00032	**0.00004**	0.00038	**0.00004**
40	0.00013	**0.00008**	0.00084	0.00059	**0.00007**	0.00088	**0.00008**
60	0.00018	**0.00012**	0.00142	0.00088	**0.00010**	0.00144	0.00013
80	0.00015	**0.00010**	0.00172	0.00136	**0.00008**	0.00181	0.00015
100	0.00017	**0.00011**	0.00219	0.00210	**0.00009**	0.00269	0.00014
200	0.00051	**0.00030**	0.00940	0.00934	**0.00026**	0.01758	0.00031
400	0.00167	**0.00104**	0.05212	0.05281	**0.00087**	0.13040	0.00106
600	0.00407	**0.00256**	0.14729	0.14706	**0.00215**	0.43686	0.00239
800	0.00677	**0.00430**	0.31524	0.31467	**0.00331**	1.08371	0.00385
1000	0.01040	**0.00650**	0.57351	0.57298	**0.00514**	2.22806	0.00592

Tabela A.4: Tempo médio (em segundos) de processamento para a classe *d*05.

reg3	Nauty	Saucy	Bliss	VF2	ARIMC
20	0.00048	**0.00019**	0.00086	0.00065	**0.00009**
40	0.00195	**0.00043**	0.00148	0.00123	**0.00016**
60	0.00418	**0.00053**	0.00235	0.00176	**0.00022**
80	0.00535	**0.00044**	0.00320	0.00211	**0.00025**
100	0.00563	**0.00056**	0.00398	0.00247	**0.00017**
200	0.02356	**0.00148**	0.00628	0.00526	**0.00030**
400	0.10876	**0.00480**	0.01622	0.01495	**0.00075**
600	0.27159	**0.00800**	0.03083	0.03014	**0.00117**
800	0.52884	**0.01042**	0.05170	0.05123	**0.00171**
1000	0.90469	**0.01335**	0.07876	0.07743	**0.00230**

Tabela A.5: Tempo médio (em segundos) de processamento para a classe *reg*3.

reg7	Nauty	Saucy	Bliss	VF2	ARIMC
20	0.00052	**0.00025**	0.00108	0.00087	**0.00049**
40	0.00198	**0.00055**	0.00199	0.00134	**0.00024**
60	0.00451	**0.00068**	0.00316	0.00208	**0.00033**
80	0.00482	**0.00046**	0.00444	0.00257	**0.00034**
100	0.00587	**0.00056**	0.00511	0.00347	**0.00026**
200	0.02317	**0.00127**	0.00854	0.00705	**0.00036**
400	0.10644	**0.00380**	0.01859	0.01788	**0.00075**
600	0.25783	**0.00538**	0.03183	0.03159	**0.00113**
800	0.48327	**0.00768**	0.05008	0.05035	**0.00169**
1000	0.80982	**0.01082**	0.07274	0.07270	**0.00236**

Tabela A.6: Tempo médio (em segundos) de processamento para a classe *reg*7.

	*r*001		*r*005		*r*01		*r*05	
	AEPIG2	ARIMC	AEPIG2	ARIMC	AEPIG2	ARIMC	AEPIG2	ARIMC
20	4.1800	3.5500	3.6500	2.8200	3.0700	1.7700	3.0000	1.0000
40	4.9200	4.0300	3.3200	2.0400	3.0000	1.3500	3.0000	1.0000
60	5.4900	4.1300	3.2800	1.8400	3.0100	1.1800	3.0000	1.0000
80	5.7700	4.1100	3.2200	1.5900	3.0100	1.0300	3.0000	1.0000
100	5.7700	3.9600	3.1400	1.3600	3.0000	1.0000	3.0000	1.0000
200	4.9400	2.5100	3.0800	1.0100	3.0000	1.0000	3.0000	1.0000
400	4.5200	2.0000	3.0100	1.0000	3.0000	1.0000	3.0000	1.0000
600	4.1900	1.9600	3.0200	1.0000	3.0000	1.0000	3.0000	1.0000
800	4.1000	1.5900	3.0000	1.0000	3.0000	1.0000	3.0000	1.0000
1000	3.9000	1.2500	3.0100	1.0000	3.0000	1.0000	3.0000	1.0000

Tabela A.7: Número médio de iterações executadas para as classes de grafos não regulares.

	reg3	reg7
	ARIMC	ARIMC
20	4.1900	3.1600
40	5.4300	1.7200
60	6.1100	1.7000
80	6.5900	1.7900
100	7.1000	1.9100
200	8.0100	2.0000
400	9.0500	2.0000
600	9.8200	2.0000
800	10.6900	2.5400
1000	10.7200	2.9800

Tabela A.8: Número médio de iterações executadas para as classes de grafos regulares.