Diego Barcelos Rodrigues

Teoria Espectral e o Problema de Isomorfismo de Grafos Regulares

Vitória (ES), Brasil

Agosto de 2011

Diego Barcelos Rodrigues

Teoria Espectral e o Problema de Isomorfismo de Grafos Regulares

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

Orientadora:
Profª. Maria Cristina Rangel

Co-orientadora:
Profª. Maria Claudia Silva Boeres

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória (ES), Brasil

Agosto de 2011

Teoria Espectral e o Problema de Isomorfismo de Grafos Regulares

DIEGO BARCELOS RODRIGUES

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

Profª. Maria Cristina Rangel
Orientadora
PPGI/UFES

Profª. Maria Claudia Silva Boeres
Co-orientadora
PPGI/UFES

Profª. Nair Maria Maia de Abreu
COPPE/UFRJ

Prof. Arlindo Gomes de Alvarenga
PPGI/UFES

Agradecimentos

Primeiramente a Deus, por ter estado comigo e me dado a força necessária para superar os obstáculos que algumas vezes me fizeram fraquejar durante esta caminhada.

À minha família, que me apóia em todos os momentos. Em especial agradeço à minha Mãe, que batalhou muito para que seus filhos tivessem o estudo que ela não teve a oportunidade de ter. Obrigado Mãe!

Às minhas Professoras Orientadoras Maria Cristina Rangel e Maria Claudia Silva Boeres, pela paciência, disponibilidade, amizade, oportunidade oferecida, pelos conhecimentos partilhados e orientação neste trabalho.

À minha namorada Luciana, pelo apoio, carinho, companheirismo, paciência e compreensão imprescindíveis durante os períodos mais difíceis.

À minha amiga Kamila e aos meus amigos Leonardo e Wesley pelo companheirismo, pelas ideias trocadas, pelas dificuldades compartilhadas durante esse período.

À FAPES pelo importante apoio financeiro, indispensável para a realização deste trabalho.

Resumo

A Teoria Espectral de Grafos (TEG) busca analisar propriedades dos grafos através de matrizes representativas de grafos e seus espectros. De uma propriedade proveniente da TEG, a autocentralidade, surge um importante invariante para o Problema de Isomorfismo de Grafos: se dois grafos são isomorfos então eles possuem autocentralidades proporcionais. Porém, esta propriedade não pode ser usada diretamente para resolução do Problema de Isomorfismo de Grafos Regulares (PIGR), pois todo grafo regular possui autocentralidades iguais. Este trabalho apresenta uma estratégia para resolver o PIGR através do uso das autocentralidades para podar a árvore de busca e restringir as possibilidades de mapeamento.

PALAVRAS CHAVE. Isomorfismo. Espectro. Autocentralidade.

Abstract

Spectral Graph Theory (SGT) studies graph properties by graph representation matrix and its spectrum. A property from SGT, the eigencentrality, provides an important invariant to Graph Isomorphism Problem: if two graphs are isomorphic, they have proportional eigencentralities. However, this property can not be directly used for solving the Regular Graph Isomorphism Problem (RGIP), as every regular graph has the same eigencentralities. This work presents a strategy for solving the RGIP through the use of eigencentralities to prune the search tree and restricting the possibilities for mapping.

KEYWORDS. Isomorphism. Spectro. Eigencentrality.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 *Introdução*

O Problema de Isomorfismo de Grafos (PIG) é um problema de grande interesse entre os pesquisadores, pois é sabido que o PIG pertence à classe NP (PRESA; ANTA, 2009), porém, sua classificação é considerada indefinida com respeito às classes de complexidade P e NP-completo (PORUMBEL, 2009; PRESA; ANTA, 2009). A existência de algoritmos polinomiais para determinadas classes de grafos, por exemplo, grafos de valência limitada (LUKS, 1982), levam alguns pesquisadores a acreditarem no desenvolvimento de um algoritmo polinomial que comprove que o PIG está em P. Contudo, outras classes de grafos exigem um comportamento exponencial de algoritmos eficientes (MIYAZAKI, 1996), induzindo outros pesquisadores a acreditarem que o PIG não está em P. Por outro lado, Karp (1972) e Oliveira e Greve (2005) creem que o PIG não pertença à classe P e nem à classe NP-completo.

Todavia, vários algoritmos eficientes foram desenvolvidos para o PIG como, por exemplo, o algoritmo denominado VF (CORDELLA *et al.*, 1999) que consiste em uma busca em profundidade com técnicas de poda, a sua versão melhorada denominada VF2 (CORDELLA *et al.*, 2001) e o filtro paramétrico IDL(d) (SORLIN; SOLNON, 2008), que faz uso de um forte invariante, distâncias entre os vértices, para rerrotular grafos e restringir as possibilidades de mapeamento. Além desses, existem algoritmos que buscam identificar o isomorfismo entre os grafos através de rotulação canônica e automorfismos, caso dos algoritmos *Bliss* (JUNTTILA; KASKI, 2007), *Conauto* (PRESA; ANTA, 2009) e *Nauty* (MCKAY, 1981), que é precursor dos dois primeiros e um dos algoritmos mais referenciados na literatura por sua eficiência.

Outro fator que atrai os pesquisadores são as aplicações do PIG, usado para modelar problemas em diversas áreas, por exemplo, na identificação de compostos químicos em Química (CORNIEL; GOTLIEB, 1970; READ; CORNEIL, 1977), reconhecimento de moléculas de proteína em Biologia (ABDULRAHIM; MISRA, 1998), reconhecimento biométrico para identificação de pessoas (NANDI, 2006), entre outros.

Dois grafos são isomorfos se existir uma função bijetora que mapeie os conjuntos de vértices preservando as suas relações de adjacências. O PIG consiste em verificar se dois grafos são

isomorfos ou não, apresentando a bijeção mencionada, no caso afirmativo.

A principal estratégia utilizada na resolução do PIG é a identificação de vértices similares para mapeamento, isto é, vértices que possuam as mesmas características extraídas de propriedades estruturais dos grafos como sequências de graus, distâncias entre os vértices, centralidades de autovetor, entre outras.

A Teoria Espectral de Grafos (TEG) busca analisar propriedades dos grafos através de suas matrizes representativas (Adjacência, Laplaciana, Laplaciana Sem Sinal, entre outras) e seus espectros. He, Zhang e Li (2005), Rajasekaran e Kundeti (2009), Santos, Rangel e Boeres (2010), entre outros, utilizam TEG como ferramenta para identificar características comuns entre os vértices do par de grafos comparados. Através dessas características é possível confirmar ou refutar a presença de isomorfismo, ou simplesmente reduzir o espaço de busca no qual são feitas tentativas de mapeamento.

Neste trabalho são considerados apenas grafos regulares não orientados, simples e conexos. Grafos regulares constituem a classe de grafos de maior desafio, principalmente os fortemente regulares, pois são grafos que possuem vértices difíceis de serem distinguidos uns dos outros, o que torna mais difícil identificar se grafos desse tipo são isomorfos, motivando a busca por um algoritmo eficiente para resolução do Problema de Isomorfismo de Grafos Regulares (PIGR). Além disso, não foi encontrada uma abordagem que utilize autocentralidades como base de um procedimento de resolução deste problema. A principal razão para isto é que as informações advindas do vetor de autocentralidades não possibilitam distinção alguma dos vértices de grafos regulares, pois o vetor de autocentralidades desses tipos de grafos é proporcional ao vetor unitário (GRASSI; STEFANI; TORRIERO, 2007). O *Problema de Isomorfismo de Grafos Orientados* é polinomialmente redutível ao *Problema de Isomorfismo de Grafos Não Orientados* como demonstra Miller (1979), o que justifica a restrição a grafos não direcionados. Grafos não direcionados com arestas paralelas podem ser transformados em grafos simples através da substituição das arestas paralelas por uma única que as represente, assim como grafos com laços são facilmente transformados em grafos simples através da retirada dos laços, não interferindo na propriedade de isomorfismo. Por fim, a resolução do *Problema de Isomorfismo de Grafos Desconexos*, implica em resolvê-lo para cada componente conexa.

Este trabalho apresenta uma abordagem de resolução do PIGR que utiliza conceitos de TEG, as autocentralidades, para buscar um mapeamento que atenda às condições de isomorfismo. A matriz de representação de grafos adotada para a implementação da abordagem proposta é a matriz de adjacência. Além disso, as notações G_1 e G_2, $V(G_1)$ e $V(G_2)$ e $E(G_1)$ e $E(G_2)$, referem-se respectivamente a dois grafos distintos, seus respectivos conjuntos de vérti-

ces e de arestas.

O próximo capítulo apresenta conceitos básicos de Teoria de Grafos, baseados em Diestel (2006) e Abreu *et al.* (2007), bem como notações e terminologias utilizadas nesta dissertação que são necessárias à compreensão deste trabalho. O Capítulo 3 é baseado em Abreu *et al.* (2007) e apresenta conceitos básicos e resultados teóricos de Teoria Espectral de Grafos. O Capítulo 4 trata do Problema de Isomorfismo de Grafos apresentando algoritmos para sua resolução, aplicações e ilustrações de grafos isomorfos e não isomorfos. O algoritmo proposto neste trabalho e variações dele são apresentados no Capítulo 5 e seu desempenho é avaliado no Capítulo 6. Por fim, o Capítulo 7 apresenta as conclusões e as perspectivas de trabalhos futuros.

2 *Algumas definições de Teoria dos Grafos necessárias ao trabalho*

Um **grafo** G é uma estrutura (V,E), onde V é um conjunto finito não vazio e E é um conjunto composto por pares não ordenados formados por elementos de V. Os elementos de V são denominados **vértices**, enquanto que os elementos de E são denominados **arestas**. O número de vértices de V, conhecido como a **ordem** de G, é normalmente denotado por n e o número de arestas, o **tamanho** de G, por m.

Cada aresta $e \in E$ é definida e normalmente referenciada por seus **extremos** u e v, ou seja, $e = \{u,v\} \in E \subseteq V \times V$, e a aresta e é dita **incidente** aos seus extremos. O número de arestas incidentes em um vértice v designa o **grau** ou **valência** de v e é denotado por $d(v)$. Dois vértices u e v são chamados de **adjacentes** se $\{u,v\} \in E$, caso contrário, são chamados de **independentes**. Também existe a relação de adjacência entre arestas, que são assim denominadas quando possuem um extremo em comum. Quando isto não ocorre as arestas são chamadas de independentes. Uma aresta com um único extremo é denominada **laço**. Arestas distintas com o mesmo par de vértices extremos são chamadas de **arestas paralelas**. Grafos que não possuam laços nem arestas paralelas são denominados **grafos simples**.

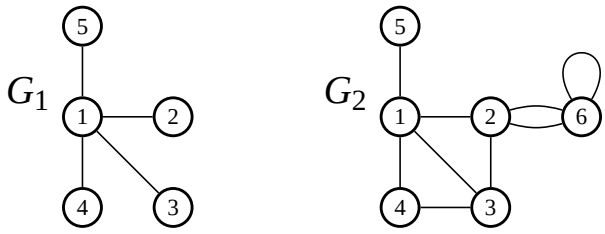
Percurso ou **cadeia** é uma sequência de arestas sucessivamente adjacentes. O percurso cuja última aresta da sucessão é adjacente a primeira é chamado de **percurso fechado**. Quando isto não acontece o percurso é denominado **aberto**. O comprimento de um percurso é a quantidade de arestas nele contidas. Se um percurso não contém arestas repetidas, então ele é denominado **percurso simples**. Um percurso simples que não repete vértices é chamado de **caminho** ou **percurso elementar**. Um percurso simples e fechado é denominado **ciclo** e um grafo que não possui ciclos é chamado de **acíclico**. O comprimento do menor caminho entre dois vértices u e v representa a **distância** entre eles e é denotado por $\delta(u,v)$ ou, simplesmente, δ_{uv}. Um grafo pode ser visualizado através de uma representação geométrica, onde os vértices correspondem a pontos distintos em um plano e as arestas são representadas por linhas arbitrárias ligando seus extremos. Na Figura 2.1 são apresentados exemplos de representações geométricas de grafos.

No grafo G_2, Figura 2.1(b), as arestas $\{1,2\},\{2,3\}$ e $\{3,4\}$ formam um percurso aberto e as arestas $\{1,2\},\{2,3\},\{3,4\}$ e $\{1,4\}$ formam um percurso fechado. Este exemplo de percurso fechado também corresponde a um ciclo. Por sua vez, o grafo G_1 (Figura 2.1-a) é exemplo de um grafo acíclico.

Dois grafos G_1 e G_2 são isomorfos se, e somente se, existe uma função bijetora $\phi : V(G_1) \rightarrow V(G_2)$ tal que, $\forall u,v \in V(G_1)$, $\{u,v\} \in E(G_1) \Leftrightarrow \{\phi(u),\phi(v)\} \in E(G_2)$ (DIESTEL, 2006). Tal função bijetora é chamada de *isomor fismo*. Se $G_1 = G_2$, então ϕ é denominada **automorfismo**. Um isomorfismo pode ser visto como uma permutação π, que aplicada ao conjunto de vértices de G_2 (ou G_1) resulta em $G_1 = G_2^{\pi}$ (ou $G_1^{\pi} = G_2$). $G_1 \simeq G_2$ denota que G_1 e G_2 são grafos isomorfos. Uma característica que não varia em grafos isomorfos é chamada de **invariante**. O número de vértices, o número arestas e os graus dos vértices de um grafo são exemplos de invariantes.

Existem várias classificações para grafos. Podem ser simples, orientados, não orientados, conexos, desconexos, regulares, fortemente regulares, etc.. Este trabalho resume algumas dessas denominações de grafos.

O grafo que pode ser representado geometricamente em um plano de forma que as suas arestas não se cruzem é denominado **grafo planar**. A Figura 2.1 apresenta exemplos de representações geométricas de dois grafos planares, sendo que um deles é um exemplo de grafo simples enquanto que o outro contêm arestas paralelas e laços.



(a) (b)

Figura 2.1: Grafo planar simples (G_1) e um grafo planar com laços e arestas paralelas (G_2).

Grafo direcionado ou simplesmente **digrafo** é um grafo formado por um conjunto de vértices e um conjunto de pares ordenados de vértices (u,v), geralmente chamados de **arcos** ou **arestas direcionadas**, onde *u* é chamado de **vértice inicial** e *v*, de **vértice terminal**. O grafo da Figura 2.2 ilustra um digrafo.

Um **subgrafo** é um grafo formado por vértices e arestas contidos nos conjuntos de vértices e arestas de outro grafo. Mais formalmente, considere dois grafos G_1 e G_2, G_1 é subgrafo

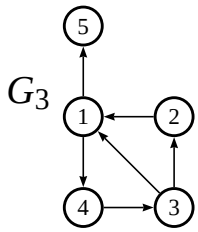


Figura 2.2: Exemplo de um grafo direcionado.

de G_2 ($G_1 \subseteq G_2$) se somente se $V(G_1) \subseteq V(G_2)$ e $E(G_1) \subseteq E(G_2)$. Por outro lado, G_2 é um **supergrafo** de G_1. Se $G_1 \subset G_2$ então G_1 é chamado **subgrafo próprio** de G_2. G_1 é chamado de **subgrafo induzido** de G_2, se $G_1 \subseteq G_2$ e contém todas as arestas $\{u,v\} \in E(G_2)$ tais que $u,v \in V(G_1)$. A Figura 2.3 serve de exemplo. Observe que G_4 é subgrafo próprio e induzido de G_2, grafo da Figura 2.1(b). O subgrafo que contém todos os vértices de seu supergrafo é denominado **subgrafo gerador**.

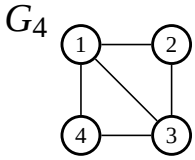


Figura 2.3: Ilustração de um subgrafo próprio e induzido do grafo G_2 da Figura 2.1(b).

Grafo conexo é aquele que contém ao menos um caminho ligando qualquer par de vértices. Um grafo não conexo é denominado **grafo desconexo**. Todo grafo conexo e acíclico é chamado de **árvore**. Uma **árvore geradora** é um subgrafo gerador acíclico. A Figura 2.4 ilustra uma árvore geradora do grafo da Figura 2.1(b).

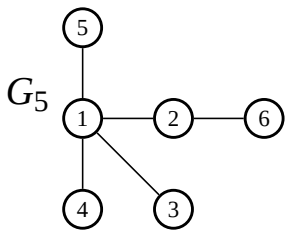


Figura 2.4: Árvore geradora do grafo G_2, Figura 2.1(b).

Seja S um conjunto e $S_1 \subseteq S$. Diz-se que S_1 é um conjunto **maximal** em relação a uma propriedade P, se S_1 satisfaz a propriedade P e não existe $S_2 \subseteq S$ que contenha S_1 e também satisfaça P. Ou seja, S_1 não está propriamente contido em nenhum subconjunto de S que satisfaça P. Se não existir $S_2 \subset S_1$ que satisfaça P, então S_1 é **minimal**.

Denominam-se **componentes conexas** de um grafo G aos subgrafos maximais de G que sejam conexos. A propriedade P, neste caso, é equivalente a ser conexo. Então, um grafo é conexo somente se ele próprio for uma componente conexa, caso contrário, ele é desconexo. O grafo G_1 da Figura 2.1(a) é um exemplo de grafo conexo e o grafo da Figura 2.5, um exemplo de grafo desconexo com duas componentes conexas.

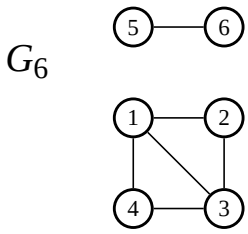


Figura 2.5: Exemplo de um grafo desconexo.

O grafo que possui todos os vértices com mesmo grau é chamado **grafo regular**. O grafo regular que contém vértices de grau k é normalmente chamado de **grafo *k*-regular**. Se G é um grafo regular tal que $k = n$-1, então G é chamado de **grafo completo** e denotado por K_n. Um grafo k-regular com n vértices tais que, cada par de vértices adjacentes possua λ vizinhos em comum e cada par de vértices não adjacentes possua μ vizinhos em comum é conhecido como **grafo fortemente regular** e denotado por $srg(n,k,\lambda,\mu)$. Um exemplo de grafo regular, que também é fortemente regular de parâmetros $(5,2,0,1)$, pode ser visto na Figura 2.6.

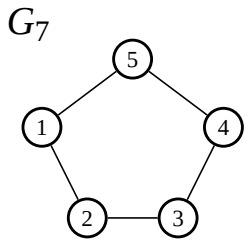


Figura 2.6: Exemplo de um grafo regular que também é fortemente regular.

Um grafo G é **assimétrico** se admitir apenas o mapeamento identidade em se tratando de

automorfismo. Ou seja, se G é um grafo assimétrico com conjunto de vértices $\{1,2,\ldots,n\}$ e $\phi : G \to G$ é um isomorfismo, então ϕ é tal que $\phi(1) = 1$, $\phi(2) = 2$, ..., $\phi(n) = n$. Figura 2.7, apresenta um exemplo de grafo assimétrico.

Grafos podem ser representados por matrizes de Adjacência, Distância, Laplaciana, Laplaciana sem Sinal, entre outras. As matrizes apresentadas a seguir são todas indexadas por vértices, ou seja, a linha ou coluna i contém informações do vértice i.

Seja G um grafo de ordem n com $V = \{1,2,\ldots,n\}$ e i, j vértices de V, onde $1 \leq i, j \leq n$. A **matriz de adjacência** de G, geralmente denotada por $A(G)$, é uma matriz quadrada de ordem n com entradas a_{ij} iguais a 1, se $\{i, j\}$ é aresta de G, e iguais a zero em caso contrário. A Figura 2.7 é exemplo de um grafo não orientado, simples e conexo do qual foram obtidas as matrizes $A(G)$, $\Theta(G)$, $L(G)$ e $Q(G)$ presentes neste capítulo.

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

A **matriz distância** de G, denotada por $\Theta(G)$, é a matriz quadrada de ordem n cujas entradas θ_{ij} são iguais a $\delta(i, j)$.

$$\Theta(G) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 2 & 1 \\ 1 & 1 & 0 & 1 & 2 & 2 \\ 1 & 2 & 1 & 0 & 2 & 3 \\ 1 & 2 & 2 & 2 & 0 & 3 \\ 2 & 1 & 2 & 3 & 3 & 0 \end{bmatrix}.$$

Outras duas matrizes de representação de grafos bem conhecidas são as matrizes **laplaciana** e **laplaciana sem sinal**. A primeira, também chamada por **laplaciano** de G e denotada por $L(G)$, é dada pela diferença $D(G) - A(G)$, onde $D(G)$ é a matriz diagonal composta pelos graus dos vértices de G ($D_{ii} = d(i), i \in V$). A segunda é dada pela soma $D(G) + A(G)$ e denotada por $Q(G)$.

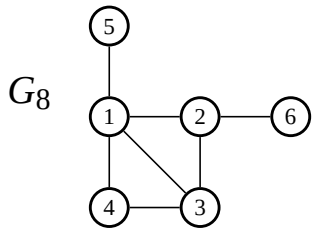


Figura 2.7: Grafo ao qual se referem as matrizes $A(G)$, $\Theta(G)$, $L(G)$ e $Q(G)$ presentes neste capítulo.

$$L(G) = \begin{bmatrix} 4 & -1 & -1 & -1 & -1 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ -1 & 0 & -1 & 2 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$Q(G) = \begin{bmatrix} 4 & 1 & 1 & 1 & 1 & 0 \\ 1 & 3 & 1 & 0 & 0 & 1 \\ 1 & 1 & 3 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Densidade de um grafo G de ordem n e m arestas é a razão $\frac{m}{n}$ ou a razão entre m e a quantidade de arestas de K_n. Um grafo é denominado **esparso** se possuir poucas arestas ao ponto de ser vantajoso explorar esta característica. Esta definição é análoga a definição de matriz esparsa presente em Saad (2003).

3 *Teoria Espectral de Grafos*

O espectro de um grafo, definido a partir do seu conjunto de autovalores, depende da matriz de representação utilizada. A Teoria Espectral de Grafos busca relacionar propriedades dos grafos através de suas matrizes de representação e seus espectros. Sendo assim, é possível utilizar teoria espectral de grafos para identificar características que devem ser comuns a grafos isomorfos, no intuito de eliminar a possibilidade de isomorfismo, filtrar possibilidades de mapeamento entre os conjuntos de vértices dos grafos ou até mesmo apresentar uma bijeção entre os vértices desses conjuntos.

Neste capítulo são apresentados conceitos básicos de Teoria Espectral de Grafos, bem como resultados teóricos importantes que relacionam o espectro de um grafo e informações provenientes dele com propriedades estruturais de grafos.

Seja G um grafo com n vértices e m arestas. O polinômio característico da matriz de adjacência $A(G)$ do grafo G é dado por $det(\lambda I - A(G))$ e denominado **polinômio característico de** G, denotado por $p_G(\lambda)$.

Considere λ uma raiz do polinômio característico de G, então λ é denominado um **autovalor de** G quando é uma raiz de $p_G(\lambda)$. Se $A(G)$ possui s autovalores distintos $\lambda_1 > \ldots > \lambda_s$ com multiplicidades algébrica a(λ_1), ... , a(λ_s), respectivamente, o **espectro de** G, denotado *spect(G)*, é definido como a matriz $2 \times$ s, onde na primeira linha se encontra os s autovalores distintos de G e na segunda linha suas respectivas multiplicidades. Ou seja:

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

O maior autovalor de G é denominado **índice de** G e denotado por λ_1.

Exemplo 1 *Para ilustrar, considere o grafo* G_8*, Figura 2.7, e seu polinômio característico* $p_G(\lambda) = \lambda^6 - 7\lambda^4 - 4\lambda^3 + 6\lambda^2 + 2\lambda - 1$*, então o espectro de G é:*

$$spect(G) = \begin{bmatrix} 2.75371 & 0.77267 & 0.30636 & -0.60928 & -1.32926 & -1.89420 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

A partir do polinômio característico de um grafo é possível obter a sua quantidade de triângulos. Seja G um grafo de ordem n, m arestas e polinômio característico $p_G(\lambda) = \lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \ldots + a_{n-1}\lambda + a_n$. Então $a_1 = 0$, $a_2 = -m$ e $a_3 = -2t$, onde t é o número de triângulos contidos em G.

Os autovalores de um grafo também fornecem a quantidade de triângulos que ele contém. Seja G um grafo de ordem n, m arestas e autovalores $\lambda_1, \ldots, \lambda_n$. Para obter o total de triângulos de G basta dividir a soma dos cubos de seus autovalores por seis. Este resultado é parte do Corolário 2.1 presente em Abreu *et al.* (2007).

Os resultados que relacionam $p_G(\lambda)$ e os autovalores com o total de triângulos que um grafo G contém podem ser verificados no Exemplo 1. Neste exemplo pode ser visto que $a_3 = -4$ e os autovalores são $2.7537, 0.7727, 0.3064, -0.6093, -1.3293$ e -1.8942, sendo t o número de triângulos do grafo, então, pelo polinômio característico de G:

$$t = \frac{a_3}{-2} = 2$$

E pelos autovalores de G:

$$t = \frac{(2.7537)^3 + (0.7727)^3 + (0.3064)^3 + (\text{-}0.6093)^3 + (\text{-}1.3293)^3 + (\text{-}1.8942)^3}{6}$$

$$t = 1.936$$

Portanto, a quantidade de triângulos presentes no grafo é igual a 2, o que pode ser facilmente verificado na Figura 2.7.

Grafos **coespectrais** ou **isoespectrais** são grafos que possuem o mesmo espectro. Sabe-se que o espectro é um invariante quando se trata de isomorfismo de grafos, portanto, se dois grafos são isomorfos eles são coespectrais (ABREU *et al.*, 2007). Porém, a recíproca desta afirmação não é verdadeira. Um exemplo que comprova esta afirmação pode ser visto na Figura 3.1 (retirada de Santos (2010)).

Um **autovetor** de um grafo G é um vetor não nulo $\boldsymbol{v}$, tal que $M(G)\boldsymbol{v} = \lambda\boldsymbol{v}$, onde $M(G)$ é uma matriz que representa G e λ é um autovalor dessa matriz. O vetor $\boldsymbol{v}$ é dito autovetor

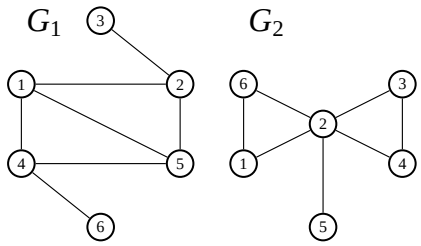


Figura 3.1: Grafos não isomorfos de mesmo espectro.

associado ao autovalor λ. O autovetor não negativo associado ao índice de um grafo é também conhecido como **autovetor principal** ou **vetor de autocentralidades**.

Seja $\boldsymbol{u}$ o autovetor principal de um grafo G simples e conexo. A **centralidade de autovetor** (autocentralidade) u_i de um vértice de índice i em $V(G)$ é definida como a i-ésima coordenada do autovetor principal $\boldsymbol{u}$ e denotada neste trabalho por $ac(i)$. Esta definição é base para dois resultados relacionados ao PIG. Se dois grafos são isomorfos eles possuem autocentralidades proporcionais (SANTOS, 2010; HE; ZHANG; LI, 2005). Além disso, as autocentralidades fornecem um resultado que é suficiente quanto ao isomorfismo. Se G_1 e G_2 são grafos simples, conexos, com mesmo índice, de autocentralidades proporcionais e distintas entre si em cada grafo, então os grafos são isomorfos (SANTOS; RANGEL; BOERES, 2010). O grafo G_8, Figura 2.7, é um exemplo de grafo que possui autocentralidades distintas: $ac(1) = 0.564072$, $ac(2) = 0.449088$, $ac(3) = 0.509502$, $ac(4) = 0.389864$, $ac(5) = 0.204841$ e $ac(6) = 0.163084$.

Outro resultado obtido com TEG, presente em He, Zhang e Li (2005), diz que dois grafos G_1 e G_2 que tenham todos os autovalores com multiplicidade 1 são isomorfos se, e somente se, possuírem o mesmo espectro e para cada autovetor u de G_1 existir autovetor v em G_2 tal que $\boldsymbol{u} = \alpha \boldsymbol{v}$, onde α é uma constante real. Um resultado similar também presente em He, Zhang e Li (2005), porém, não necessário e suficiente como o anterior, diz que se dois grafos com um ou mais autovalores de multiplicidade 1 são isomorfos, eles possuem o mesmo espectro e os autovetores associados a esses autovalores de multiplicidade 1 são proporcionais.

Considerando a matriz laplaciana de G, $L(G)$, o **espectro do laplaciano** de G, denotado por $\zeta(G)$, é a matriz $1 \times n$ composta pelos autovalores de $L(G)$ organizados em ordem não crescente, os quais correspondem as raízes do polinômio característico de $L(G)$, denotado por $\sigma_G(\mu) = det(\mu I - L(G))$. Portanto, se $\mu_1 \geq \ldots \geq \mu_n$ denotam os autovalores de $L(G)$, então:

$$\zeta(G) = \begin{bmatrix} \mu_1 & \ldots & \mu_n \end{bmatrix}.$$

A seguir é apresentado um exemplo, retirado de Abreu *et al.* (2007), que ilustra os espectros do laplaciano de dois grafos G_1 e G_2, o primeiro conexo e o segundo desconexo.

Exemplo 2 *Sejam G_1 e G_2 os grafos da Figura 3.2. Então $\zeta(G_1) = [4,3,1,0]$ e $\zeta(G_2) = [4,3,2,1,0,0,0]$.*

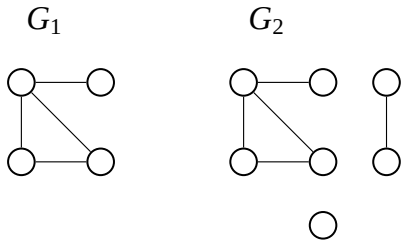


Figura 3.2: Grafos referentes ao Exemplo 2.

A matriz laplaciana fornece algumas informações interessantes, três delas são relatadas a seguir. Sejam $\mu_1 \geq \ldots \geq \mu_n$ os autovalores do laplaciano de um grafo G, então:

a) $\mu_n = 0$ e o vetor unitário $\vec{\mathbf{1}}$ é um autovetor associado;

b) G é conexo se, e somente se, $\mu_{n\text{-}1} > 0$;

c) Se G é regular de grau k, então $\mu_i = k$ - $\lambda_{n\text{-}i}$, onde λ_i são os autovalores de $A(G)$ e $0 \leq i < n$.

A partir da matriz laplaciana é possível determinar o número de árvores geradoras de um grafo. Seja $\tau(G)$ o número de árvores geradoras de um grafo G de ordem n, então

$$\tau(G) = n^{\text{-}2} det(J + L(G)) \tag{3.1}$$

onde J é uma matriz de ordem n com todos os seus elementos iguais a 1.

A quantidade de árvores geradoras $\tau(G)$ também pode ser obtida diretamente dos autovalores não nulos de $L(G)$. Sejam $\mu_1, \ldots, \mu_{n\text{-}1}$ os autovalores não nulos de $L(G)$, então:

$$\tau(G) = \frac{\mu_1 \mu_2 \cdots \mu_{n\text{-}1}}{n}. \tag{3.2}$$

Além disso, se G é k-regular é possível utilizar a relação existente entre os autovalores de $L(G)$ e $A(G)$ para obter

$$\tau(G) = n^{-1} p_G{}'(k), \tag{3.3}$$

onde $p_G{}'(k)$ denota a derivada do polinômio característico de $A(G)$.

O polinômio característico $p_Q(\lambda)$ da matriz laplaciana sem sinal $Q(G)$ e suas raízes $q_1 \geq q_2 \geq \cdots \geq q_n$, que são autovalores desta matriz, também fornecem informações interessantes. Seja G um grafo de ordem n, m arestas e $-p_1$ o coeficiente de λ^{n-1} em $p_Q(\lambda)$, então $m = \frac{-p_1}{2}$.O grafo G é k-regular se, e somente se, $nq_1 = 4m$ e, neste caso, $k = \frac{q_1}{2}$ e o número de componentes conexas é igual a multiplicidade de q_1. Além disso, se G é k-regular, então $p_G(\lambda) = p_Q(\lambda + r)$ e $\sigma_G = (-1)^n p_Q(2k - \lambda + k)$.

Por fim, vale ressaltar que, de acordo com estudos realizados por alguns pesquisadores (ABREU *et al.*, 2007), quando se trata de refutar o isomorfismo entre pares de grafos através do espectro, a matriz laplaciana sem sinal apresenta melhores resultados que as demais matrizes abordadas nesta seção.

4 Problema de Isomorfismo de Grafos

A definição de grafos isomorfos, presente no Capítulo 2, afirma que dois grafos são isomorfos se existir um isomorfismo entre eles, isto é, uma função bijetora que mapeia seus conjuntos de vértices preservando adjacências. O Problema de Isomorfismo de Grafos consiste em averiguar a existência de um isomorfismo entre dois grafos.

Exemplos de pares de grafos isomorfos e não isomorfos podem ser vistos na Figura 4.1. É fácil ver que os grafos G_1 e G_2 são isomorfos, apresentando a função bijetora $\phi : V(G_1) \rightarrow V(G_2)$ com a qual se obtém o mapeamento $\{(1,6),(2,3),(3,2),(4,1),(5,4),(6,5)\}$. Porém, não é possível apresentar uma função bijetora $\psi : V(G_1) \rightarrow V(G_3)$ (ou $\psi : V(G_2) \rightarrow V(G_3)$) com a qual seja obtido um mapeamento que atenda à definição de isomorfismo, pois esses grafos possuem claramente diferenças estruturais: G_1 e G_2 são planares enquanto G_3 não é.

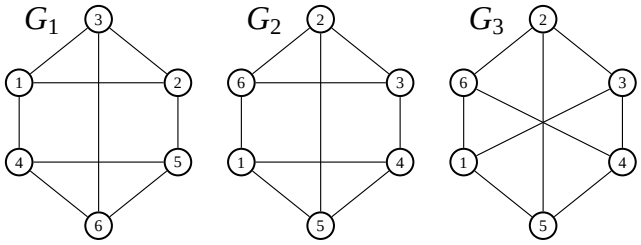


Figura 4.1: G_1 e G_2 são isomorfos enquanto G_1 e G_3, e G_2 e G_3, não o são.

A próxima seção apresenta vários algoritmos presentes na literatura, especialmente algoritmos espectrais, pois fazem uso de TEG tal como este trabalho.

4.1 Algoritmos para Resolução do PIG

Existem vários algoritmos desenvolvidos para resolver o PIG, muitos fazendo uso de condições necessárias ao isomorfismo, os invariantes, para reduzir o tempo de resolução do problema (OLIVEIRA; GREVE, 2005; RAJASEKARAN; KUNDETI, 2009; SANTOS; RANGEL; BO-

ERES, 2010). Esses invariantes servem para refutar o isomorfismo entre um par de grafos ou reduzir as possibilidades de mapeamento entre os vértices subdividindo-os em grupos e formando partições, utilizadas como ponto de partida para algoritmos de busca em árvore. Existem ainda situações nas quais é possível fazer uso de métodos de refinamento para subdividir ainda mais as partições. Quando isto não for possível, pode-se recorrer a uma individualização forçada, por exemplo, atribuindo um rótulo distinto de qualquer outro, para então refinar a partição de acordo com essa nova característica no grafo.

Segundo Presa e Anta (2009) são duas as principais abordagens utilizadas por algoritmos de resolusão do PIG. Uma faz uso do que o autor chama de abordagem direta, enquanto a outra faz uso de *rotulação canônica*. Em uma abordagem direta busca-se um isomorfismo entre os grafos comparados, enquanto que em abordagens que fazem uso de rotulação canônica, calcula-se uma função $C(G_1)$, de um grafo G_1, de tal forma que todo grafo G_2 isomorfo a G_1 tenha rotulação canônica $C(G_2) = C(G_1)$.

O *Nauty* (MCKAY, 1981) é um algoritmo de rotulação canônica que estabelece um procedimento complexo de resolução do PIG, como o próprio autor afirma em McKay (2009). O algoritmo recebe um grafo com vértices rotulados, cria uma partição de acordo com esses rótulos, ordena lexicográficamente esta partição e aplica um algoritmo de refinamento que gera uma nova partição ordenada. Se esta partição for discreta, isto é, possuir subconjuntos de vértices (denominados células) unitários, o algoritmo apresenta a nova rotulação baseada na ordenação lexicografica das células. Caso contrário, o algoritmo força a subdivisão de uma célula através da individualização de um vértice, gerando uma nova partição, e realiza um novo refinamento. Esse procedimento é repetido até que ao menos uma partição discreta seja encontrada. O processo de subdivisão e refinamento representa uma busca na árvore referente as possibilidades de subdivisões forçadas. O *Nauty* não explora todas as possibilidades de descida na árvore. Ele evita descidas desnecessárias através dos automorfismos encontrados durante o processo de busca das partições discretas. Uma vez que um automorfismo entre nós de um mesmo nível da árvore é detectado, aquele recém alcançado não sofre subdivisões e o algoritmo realiza *backtracking*. Para mais detalhes consulte o trabalho que deu origem ao *Nauty* (MCKAY, 1981) ou o trabalho de Hartke e Radcliffe (2009).

Os algoritmos *Bliss* (JUNTTILA; KASKI, 2007) e *Conauto* (PRESA; ANTA, 2009), apresentam melhorias na abordagem de McKay (1981). Assim como o *Nauty*, o *Bliss* foi projetado para obter rotulação canônica de grafos, enquanto que o *Conauto* é um algoritmo dedicado a resolver o PIG através da obtenção de um isomorfismo. Segundo os autores do *Bliss*, este algoritmo utiliza melhores estruturas de dados e melhores formas de podar a árvore de busca,

enquanto que Presa e Anta (2009) afirmam que o *Conauto* evita a construção do grupo de automorfismo por completo. Em ambos os trabalhos os autores relatam resultados que demonstram a praticidade de seus algoritmos, que de maneira geral apresentam melhor comportamento que o *Nauty*, o algoritmo precursor.

Babai, Grigoryev e Mount (1982) utilizam TEG em um complexo algoritmo polinomial para o PIG de grafos não direcionados com as multiplicidades de seus autovalores menores ou iguais a m, isto é, grafos com multiplicidade de autovalor limitada. Dado um grafo G não direcionado e sua matriz de adjacência $A(G)$, o autores relacionam os autoespaços desta matriz com o grupo de automorfismo do grafo G. Os autoespaços são utilizados para a obtenção de um conjunto de permutações das quais é possível gerar o grupo de automorfismo de G em $O(n^{2m+c})$, onde m é a multiplicidade do autovalor de maior multiplicidade do grafo e c é uma constante não negativa.

Segundo Babai, Grigoryev e Mount (1982), foi provado por Leighton e Miller (1979) um caso especial de detecção de isomorfismo de grafos com autovalores todos distintos na ordem de $O(n^3)$. Miller (2011) disponibiliza um manuscrito original digitalizado deste trabalho. Neste manuscrito fica clara a ideia de utilização dos autovetores para resolução do PIG, porém o procedimento apresentado não tem por base as matrizes de adjacências de dois grafos, mas um par de matrizes que sejam não singulares, simétricas e coespectrais. Dessa forma a ideia é desenvolvida sobre o isomorfismo de matrizes. Duas matrizes A_1 e A_2 são isomorfas se, e somente se, existir uma matriz de permutação P tal que $A_1 = PA_2P^{-1}$. Leighton e Miller (1979) apresentam um resultado que é a base do procedimento de resolução: Sejam A_1 e A_2, matrizes não singulares, simétricas e coespectrais, U e V matrizes cujas colunas correspondem aos autovetores de A_1 e A_2, P uma matriz de permutação e E uma *matriz bloco diagonal*[1] tal que cada bloco E_i é ortogonal e possui dimensão igual a multiplicidade do autovalor λ_i (quando ordenados de forma não decrescente) de A_1. Então $A_1 = PA_2P^{-1}$ se, e somente se, $UE = PV$. Se A_1 e A_2 são matrizes de autovalores todos distintos, então E é uma matriz diagonal com $|e_{ii}| = 1$.

O procedimento descrito por Leighton e Miller (1979) usa as coordenadas dos autovetores para particionar U e V em blocos de linhas mapeadas por P. Se $UE = PV$, então $e_j u_j = Pv_j$ para todo $1 \leq j \leq n$. Uma vez que $|e_j| = 1$, P permuta as linhas de V de forma que, para todo $1 \leq i, j \leq n$, $|u_{ij}| = |v^p_{ij}|$, onde v^p_{ij} denota as entradas de V permutadas segundo P. Consequentemente, U e V podem ser particionados em blocos correspondentes de vetores linha. Os autores apresentam um algoritmo para realização deste particionamento, no entanto, o procedimento de

[1] Uma matriz M é bloco diagonal se seus elementos não nulos se encontram agrupados em submatrizes quadradas, que possuam suas diagonais principais formadas pelos elementos da diagonal principal de M.

resolução do PIG apresentado a seguir é baseado na reprodução recente desse trabalho feita por Rajasekaran e Kundeti (2009).

Considere G_1 e G_2 grafos coespectrais com respectivos autovalores ordenados de forma não crescente $\lambda_1, \ldots, \lambda_n$ e $\beta_1, \ldots, \beta_n$, todos com multiplicidade 1, obtidos de suas matrizes de adjacência. Sejam $v_1, \ldots, v_n$ os respectivos autovetores associados a $\lambda_1, \ldots, \lambda_n$ e $u_1, \ldots, u_n$ os respectivos autovetores associados a $\beta_1, \ldots, \beta_n$, onde $v_i = (v_i^1, v_i^2, \ldots, v_i^n)$ e $u_i = (u_i^1, u_i^2, \ldots, u_i^n)$. Então, G_1 e G_2 são isomorfos se, e somente se, $u_i = c(Pv_i)$, $1 \leq i \leq n$, c uma constante real e P uma matriz de permutação (HE; ZHANG; LI, 2005; RAJASEKARAN; KUNDETI, 2009). Considerando esta afirmação, é possível resolver o PIG verificando se cada par $\{v_i, u_i\}$ é composto por autovetores proporcionais. Isto pode ser feito através da normalização dos autovetores, ordenação e posterior comparação de suas coordenadas. Se os autovetores forem proporcionais então os grafos são isomorfos, caso contrário, não isomorfos. Se os grafos são isomorfos é possível obter um isomorfismo como segue. Se as coordenadas de v_i são todas distintas, então o vértice 1 de $V(G_1)$ pode ser unicamente identificado por v_i^1, o vértice 2 de $V(G_1)$ por v_i^2, e assim sucessivamente. De maneira análoga, cada vértice $i \in V(G_2)$ pode ser unicamente identificado por u_i^i. Após esse processo de rotulação, os vértices de mesmo rótulo são mapeados e, assim, o isomorfismo é obtido. Caso contrário, partições dos conjuntos de vértices dos grafos são obtidas com o mesmo processo de rotulação descrito anteriormente utilizando-se v_1 e u_1, seguido de uma ordenação por rótulos para agrupar vértices com rótulos iguais. Depois, as coordenadas de v_2 e u_2 são utilizadas para refinar essas partições, agregando ao rótulo de cada $i \in V(G_1)$ a coordenada v_2^i e ao rótulo de cada $i \in V(G_2)$ a coordenada u_2^i, gerando novos rótulos que consistem em tuplas (v_1^i, v_2^i) para os vértices de $V(G_1)$ e (u_1^i, u_2^i) para os vértices de $V(G_2)$. O refinamento cessa com a reordenação dos vértices segundo os novos rótulos. Se após este primeiro refinamento as partições forem discretas, o isomorfismo é obtido pelo mapeamento dos vértices de mesmo rótulo. Caso contrário, um novo processo de refinamento é realizado com as coordenadas de v_3 e u_3. Obviamente os refinamentos se repetem até que seja obtida uma partição discreta ou todos os autovetores tenham sido utilizados para refinamento.

He, Zhang e Li (2005) apresentam um procedimento que usa a mesma ideia de Leighton e Miller (1979), mas sem a limitação de que os grafos tenham autovalores todos distintos. Eles particionam os conjuntos de vértices utilizando os autovetores que forem associados a autovalores de multiplicidades 1. Se após todos os refinamentos as partições geradas forem discretas, o PIG está resolvido. No entanto, caso isto não ocorra, os autores ressaltam a necessidade de um algoritmo para tentar encontrar um isomorfismo baseando-se nessas partições, eles mesmos propuseram um procedimento que pode ser visto em He *et al.* (apud HE; ZHANG; LI, 2005, p. 663)

Santos, Rangel e Boeres (2010) apresentam um algoritmo de resolução do PIG semelhante aos algoritmos de Leighton e Miller (1979) e He, Zhang e Li (2005). A diferença para seus precursores é a utilização de um único autovetor para particionar os conjuntos de vértices, o autovetor principal. Em caso dessa partição não ser discreta os autores utilizam um algoritmo de busca em árvore para tentar encontrar um isomorfismo, como He, Zhang e Li (2005).

Além de apresentarem uma reprodução do trabalho de Leighton e Miller (1979), Rajasekaran e Kundeti (2009) apresentam mais três algoritmos espectrais para resolução do PIG, o primeiro $O(n^4)$, o segundo $O(n^6)$ e o terceiro $O(n^5)$, todos com base em uma ideia de modificação que adiciona laços aos vértices dos grafos.

O primeiro algoritmo proposto por Rajasekaran e Kundeti (2009), o $GraphIsomorphism2$, calcula o que eles chamam de família de espectros de cada grafo. A **família de espectro** de um grafo G de ordem n consiste em um conjunto de espectros $S^1, S^2, \ldots, S^n$, onde cada S^i, $1 \leq i \leq n$, corresponde ao espectro de G^i, grafo com um único laço que é resultante da adição da aresta $\{i,i\}$. De posse das famílias de espectros de cada grafo, o algoritmo as compara e informa que os grafos são isomorfos se elas forem iguais, caso contrário, informa que os grafos não são isomorfos.

O segundo e o terceiro algoritmos propostos por Rajasekaran e Kundeti (2009), respectivamente denominados $FindPermutation$ e $RandFindPerm$, utilizam o conceito de *vértices simétricos* definido pelos autores como a seguir. Considere dois grafos G_1 e G_2 com suas respectivas matrizes de adjacência $A(G_1)$ e $A(G_2)$. Sejam $A^i(G_1)$ e $A^j(G_2)$ matrizes resultantes das inserções das arestas $\{i,i\}$ a $E(G_1)$, $i \in V(G_1)$, e $\{j,j\}$ a $E(G_2)$, $j \in V(G_2)$, respectivamente. Os vértices i e j são **simétricos** se o espectro de $A^i(G_1)$ for igual ao espectro de $A^j(G_2)$.

O algoritmo $FindPermutation$ procura mapear cada vértice $i \in G_1$ a um vértice simétrico $j \in G_2$, separando este processo em duas fases. Na primeira, apenas o vértice $i \in G_1$ que possui um único vértice simétrico em G_2 são mapeados. Na segunda fase o algoritmo busca mapear cada vértice de G_1 não mapeado na primeira fase com qualquer vértice simétrico em G_2. Se para algum $i \in G_1$ não houver simétrico em G_2, o algoritmo informa que os grafos comparados não são isomorfos. O $RandFindPerm$ também busca mapear vértices simétricos em G_1 e G_2, mas este algoritmo mapeia cada $i \in G_1$ com o primeiro simétrico encontrado em G_2. Caso o $RandFindPerm$ não encontre em G_2 um vértice simétrico para $i \in G_1$, o algoritmo desfaz todos os mapeamentos, realiza uma permutação aleatória em $A(G_1)$ e reinicia o processo de mapeamento desde o começo. Este algoritmo encerra ao encontrar um mapeamento ou após um número arbitrário de permutações aleatórias realizadas em $A(G_1)$. Obviamente, se um mapeamento for encontrado os grafos são considerados isomorfos, caso contrário, não isomorfos.

Rajasekaran e Kundeti (2009) relatam que o *GraphIsomorphism*2 pode não funcionar com certos tipos de grafos fortemente regulares, em contrapartida, o *FindPermutation* e o *RandFindPerm* parecem não apresentar o mesmo problema.

O trabalho de Spielman (1996) usa informações do espectro para realizar uma análise de tempo de procedimentos de individualização e refinamento voltados ao Problema de Isomorfismo de Grafos Fortemente Regulares (PIGFR). De acordo com certas características, Spielman subdivide esse conjunto de grafos em classes para determinar a complexidade de tempo de resolução do PIGFR classe por classe. O autor descarta todos os grafos fortemente regulares formados por uniões disjuntas de subgrafos completos e seus complementos, afirmando que tratam-se de grafos com resolução trivial, e considera apenas aqueles que possuem $k \geq \sqrt{n\text{-}1}$. Ao final de sua análise, Spielman apresenta um resultado afirmando que o PIGFR pode ser resolvido em $n^{O(n^{1/3} log\ n)}$.

4.2 Aplicações

As aplicações do PIG normalmente são voltadas ao reconhecimento de *objetos*[2], pois estes podem ser transcritos em um grafo. O PIG auxilia tal identificação no momento em que uma ou mais comparações entre grafos são realizadas para a reconhecimento de imagens. Desde o momento em que o sucesso do reconhecimento está ligado à qualidade da transcrição dos objetos pode-se notar a importância dos esforços dispendidos para transcrevê-los de forma fiel.

Nandi (2006) e Ferreira, Pereira e Carreira (2001) são exemplos de trabalhos que relatam processos de transcrição e utilizam o PIG para reconhecimento biométrico baseado em impressões digitais. Devido a peculiaridades presentes em impressões digitais denominadas minúcias (terminações de linhas, bifurcações, lagos, entre outras) é possível gerar um grafo para representá-las. Para isto basta fazer de cada minúcia um vértice rotulado com o seu tipo e sua posição geométrica na imagem, bem como fazer de cada segmento de reta delimitado por minúcias, uma aresta valorada pela distância entre elas.

Outra aplicação do problema é apresentada em Cordella *et al.* (2000), que trata da utilização do PIG na detecção de componentes em uma imagem. Nesta abordagem, considerando a codificação adotada para as imagens a serem analisadas, as arestas representam as linhas dos componentes e os vértices a junção ou pontos terminais destas linhas. Estes vértices possuem rótulos de posição e formato, e as arestas rótulos de orientação e comprimento.

O reconhecimento de componentes químicos é um tipo de aplicação cuja obtenção de um

[2] A palavra objeto é utilizada para referenciar tudo aquilo que pode ser representado por um grafo.

grafo representativo é trivial. Basta fazer de cada átomo um vértice rotulado e de suas ligações, as arestas. A identificação de componentes químicos é importante em Química, como relata Read e Corneil (1977).

5 *Algoritmos para Detecção de Isomorfismo de Grafos Regulares*

Neste capítulo são propostos quatro algoritmos para resolução do Problema de Isomorfismo de Grafos Regulares (PIGR). Esses algoritmos modificam os grafos de entrada como fazem Rajasekaran e Kundeti (2009), mas através da inclusão de vértices e arestas de maneira a evidenciar assimetrias nos mesmos, gerando grafos simples e conexos, o que não ocorre após cada modificação feita pelos algoritmos de Rajasekaran e Kundeti (2009). As assimetrias geradas facilitam a identificação dos vértices dos grafos de entrada a serem mapeados.

Considerando dois grafos regulares, simples e conexos G_1 e G_2, os algoritmos são baseados na ideia de preservação de isomorfismo entre G_1 e G_2 após serem realizadas *modificações equivalentes*[1] em ambos os grafos. Ou seja, a realização de *modificações equivalentes* em um par de grafos isomorfos deve resultar em outro par de grafos isomorfos.

Um exemplo de uma modificação equivalente pode ser visto na Figura 5.1 onde, considerando os grafos da Figura 4.1, G'_1 é o grafo resultante das adições do vértice 7 e da aresta $\{2,7\}$ ao grafo G_1 e G'_2 é o grafo resultante de uma modificação equivalente caracterizada pelas adições do vértice 7 e da aresta $\{3,7\}$ a G_2.

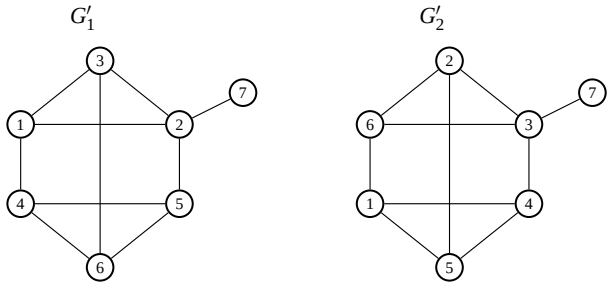


Figura 5.1: *Exemplo de uma modificação equivalente.*

Todos os algoritmos consistem em procedimentos de busca na árvore de mapeamentos pos-

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

síveis que, como no trabalho de Santos, Rangel e Boeres (2010), utilizam o resultado que afirma que grafos isomorfos possuem as mesmas autocentralidades para restringir possibilidades de mapeamento e evitar descidas por caminhos que não levam a uma solução válida do problema. Esta é mais uma diferença para os algoritmos propostos por Rajasekaran e Kundeti (2009), os quais possuem como base o resultado que afirma que grafos isomorfos são coespectrais.

Os algoritmos propostos neste trabalho diferem de acordo com o método de modificação utilizado (Seção 5.3) e a quantidade de modificações equivalentes realizada.

5.1 Autocentralidades para o Isomorfismo de Grafos Regulares (AIGR)

Dados dois grafos k-regulares G_1 e G_2 simples e conexos de ordem n, o principal algoritmo proposto neste trabalho, denominado AIGR (Autocentralidades para o Isomorfismo de Grafos Regulares), busca evidenciar assimetrias nesses grafos através da inclusão de vértices e arestas, de forma a facilitar a busca por um isomorfismo (SANTOS; RANGEL; BOERES, 2010; RAJASEKARAN; KUNDETI, 2009). O AIGR realiza n modificações equivalentes em G_1 e G_2 de maneira que os respectivos grafos resultantes, G_1^n e G_2^n, sejam simples e conexos. Em cada modificação realizada, cada vértice adicionado a um dos grafos é ligado somente a um **único** vértice do mesmo por uma nova aresta. Para cada $v \in V(G_1)$ que tem seu grau e autocentralidade modificados, o algoritmo busca $u \in V(G_2)$ que possua os mesmos valores de grau e autocentralidade após sofrer modificação equivalente ao vértice v. Dessa forma é esperado em caso de isomorfismo entre G_1 e G_2 que, para cada sequência de adições de vértices e arestas realizadas em G_1, exista uma sequência de modificações equivalentes a serem realizadas em G_2 que permita a identificação de um mapeamento atendendo às condições de isomorfismo.

O AIGR é apresentado no Algoritmo 1. É fácil perceber que o algoritmo realiza uma busca em profundidade com poda na árvore relativa a todos os mapeamentos que podem ser realizados entre $V(G_1)$ e $V(G_2)$, na tentativa de encontrar um que caracterize o isomorfismo entre os grafos comparados. A poda da árvore de busca é realizada através da utilização das autocentralidades dos vértices, antes e depois de cada modificação equivalente, como pode ser visto na ilustração dos passos do AIGR feita no Exemplo 3. Além disso, o algoritmo faz uso da técnica do *backtracking*, caso não seja possível encontrar $u \in V(G_2)$ que após sofrer modificação equivalente a $v \in V(G_1)$, fique com o mesmo valor de autocentralidade e os grafos resultantes com vetores principais proporcionais.

As funções $ac(\cdot)$ e $avp(\cdot)$ representam, respectivamente, a autocentralidade de um vértice

e o autovetor principal de um grafo. O algoritmo utiliza os seguintes critérios:

- **C1 - Seleção do vértice a ser modificado**

  Estabelece que seja selecionado um vértice de grau k que não seja vizinho de vértices adicionados ao grafo, ou seja, $v \in V(G_1) \subseteq V(G_1^i)$ tal que, $\forall\, w \in V(G_1^i) \backslash V(G_1)$, $\{v,w\} \notin E(G_1^i) \backslash E(G_1)$, $1 \leq i \leq n$, onde G_1^i é o grafo resultante de i modificações realizadas em G_1. Este critério é aplicado somente a vértices de G_1.

- **C2 - Seleção de um candidato a mapeamento**

  Este critério é utilizado para reduzir as tentativas de mapeamento. Ele estabelece que seja selecionado para tentativa de mapeamento com $v \in V(G_1) \subset V(G_1^i)$, um vértice $u \in V(G_2) \subseteq V(G_2^{i\text{-}1})$, $1 \leq i \leq n$, tal que d(u) = k, a autocentralidade de u seja igual a x, onde x é a autocentralidade de $v \in V(G_1) \subseteq V(G_1^{i\text{-}1})$, e u ainda não tenha sido mapeado com v.

- **C3 - Critério de mapeamento**

  Determina que os vértices selecionados em **C1** e **C2** sejam mapeados somente se possuírem as mesmas autocentralidades e os grafos resultantes G_1^i e G_2^i, $1 \leq i \leq n$, possuírem autovetores principais proporcionais. Além disso, este critério desempenha papel fundamental na poda da árvore de busca, pois caso nenhuma das tentativas de mapeamento atenda a este critério, o AIGR realiza *backtracking*.

Uma vez selecionado de acordo com o critério **C1**, o vértice $v \in V(G_1) \subset V(G_1^{i\text{-}1})$ (*Passo 2*), $0 < i \leq n$, i novos vértices são ligados a ele e o autovetor principal do grafo resultante G_1^i é calculado (*Passo 3*). Em seguida o algoritmo procura um vértice $u \in V(G_2) \subset V(G_2^{i\text{-}1})$ que atenda ao critério **C2** (*Passo 4* - linhas 12 e 13). Se não encontrar tal vértice u e não for possível realizar *backtracking* (*Passo 5*), o algoritmo identifica os grafos como não isomorfos (*Passo 7*). Se não encontrar u e $i \neq 0$, o algoritmo realiza *backtracking* (*Passo 5*). Se u for encontrado, i novos vértices são ligados a u e o autovetor principal do grafo resultante G_2^i é calculado (*Passo 4* - linhas 14 e 15). Então, se o critério **C3** for atendido, um mapeamento parcial entre v e u é realizado (*Passo 4* - linha 17) e o algoritmo desce ao próximo nível da árvore (*Passo 6*). Caso contrário, o algoritmo remove os novos vértices ligados a u (*Passo 4* - linha 18) e busca em $V(G_2) \subset V(G_2^{i\text{-}1})$ outro candidato a mapeamento com v. Por fim, caso um vértice folha seja alcançado, isto é, caso G_1^n e G_2^n sejam gerados, o algoritmo verifica o mapeamento encontrado e o retorna caso seja válido, senão, realiza *backtracking*.

Exemplo 3 Para ilustrar os passos do algoritmo são utilizados como grafos iniciais, G_1^0 e G_2^0, os grafos 3-regulares isomorfos G_1 e G_2 da Figura 4.1, respectivamente. A obtenção de G_1^0 e

Algoritmo 1: *Algoritmo AIGR.*

Dados: [*Dois grafos G_1 e G_2 k-regulares simples de mesma ordem n*]

Passo *1* :
  $G_1^0 \leftarrow G_1$; $G_2^0 \leftarrow G_2$; $i \leftarrow 1$.
  Calcule os autovetores principais de G_1^0 e G_2^0;

Passo *2* :
  Se $(\nexists\, \boldsymbol{v} \in V(G_1) \subseteq V(G_1^{i\text{-}1})$ *de grau* $\boldsymbol{k})$ **então** *Vá ao Passo* 7;
  Escolha $\mathbf{v} \in V(G_1) \subseteq V(G_1^{i\text{-}1})$ *de grau* $\mathbf{k}$.

Passo *3* :
  $\mathbf{x} \leftarrow$ ac(**v**);
  Obtenha G_1^i ligando i novos vértices a **v**;
  Calcule o autovetor principal de G_1^i;

Passo *4* :
  Para cada $\boldsymbol{u} \in V(G_2) \subseteq V(G_2^{i\text{-}1})$ **faça**
    Se $((d(\boldsymbol{u}) = \boldsymbol{k})$ & $(ac(\boldsymbol{u}) = \boldsymbol{x}))$, **então**
      Obtenha G_2^i ligando i novos vértices a **u**;
      Calcule o autovetor principal de G_2^i;
      Se $(avp(G_2^i)$ *é proporcional a* $avp(G_1^i)$ & $ac(\boldsymbol{u}) = ac(\boldsymbol{v}))$, **então**
        Mapeie **u** *e* **v** *e vá ao Passo* 6;
      Remova os novos vértices que foram ligados a **u**;

Passo *5* $(Backtracking)$:
  $i \leftarrow i$-1;
  Se $(i < 1)$, **então** *Vá ao Passo* 7;
  Remova os novos vértices que foram ligados a **v**;
  Recupere $\mathbf{w} \in V(G_1^i)$ *que foi selecionado na modificação de número i*;
  Remova os novos vértices que foram ligados ao vértice **s** *mapeado com* **w**;
  Desfaça o mapeamento entre **s** *e* **w**; $\mathbf{v} \leftarrow \mathbf{w}$;
  Calcule o autovetor principal de G_1^i;
  $\mathbf{x} \leftarrow ac(\mathbf{v})$;
  Vá ao Passo 4.

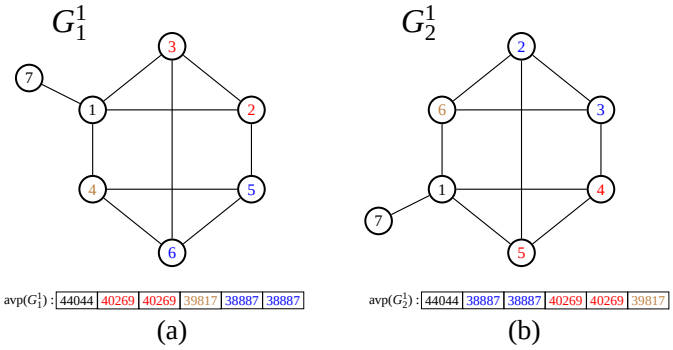
Passo *6* :
  $i \leftarrow i + 1$;
  Se $(i \leq n)$, **então** *Vá ao Passo* 2.

Passo *7* :
  Se $(i = n + 1)$, **então**
    Se *o mapeamento encontrado for válido,* **então**
    "*Apresentar o mapeamento* $V(G_1) \rightarrow V(G_2)$ *encontrado.*"
    Senão *Vá ao Passo 5;*
  Senão "*Os grafos G_1 e G_2 não são isomorfos*!"

G_2^0 corresponde ao primeiro passo do AIGR. Note que não é necessário calcular as autocentralidades desses grafos, pois é sabido que todo grafo regular possui autocentralidades iguais

(GRASSI; STEFANI; TORRIERO, 2007). As modificações realizadas ao longo do algoritmo são representadas pelos grafos resultantes G_1^i e G_2^i, $1 \leq i \leq n$.

No *Passo 2* o algoritmo escolhe um vértice de grau 3 em $V(G_1)$, por exemplo o vértice 1. Então, no *Passo 3* G_1^1 (Figura 5.2) é obtido com as adições do vértice 7 a $V(G_1)$ e da aresta $\{1,7\}$ a $E(G_1)$, e suas autocentralidades (autovetor principal) são calculadas. Veja o grafo G_1^1 presente na Figura 5.2(a).



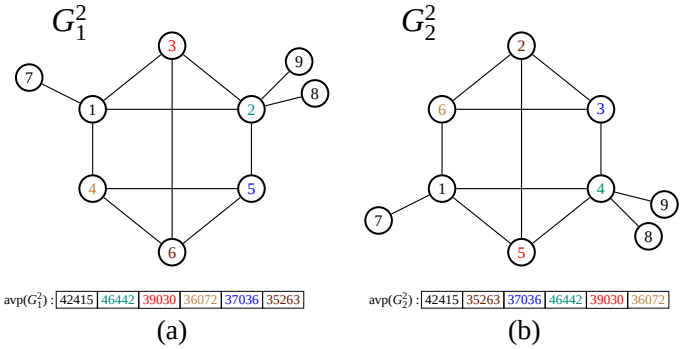
(a) (b)

Figura 5.2: *Grafos* G_1^1 *e* G_2^1 *e as autocentralidades do vértice* 1 *ao vértice* 6.

No *Passo 4* busca-se um vértice u em $V(G_2)$ que atenda ao critério **C2**, isto é, que tenha grau *3* e mesma autocentralidade que $1 \in V(G_1)$. É sabido que os autovetores principais de grafos *k*-regulares de mesma ordem são proporcionais ao vetor unitário (GRASSI; STEFANI; TORRIERO, 2007), logo, qualquer vértice em $V(G_2)$ possui mesma autocentralidade que $1 \in V(G_1)$. Assim, o vértice $1 \in V(G_2)$ é um candidato a mapeamento com $1 \in V(G_1)$. Então, G_2^1 (Figura 5.2-b) é obtido com as adições do vértice 7 a $V(G_2)$ e da aresta $\{1,7\}$ a $E(G_2)$, e suas autocentralidades são calculadas. Pode ser visto nos autovetores principais de cada grafo resultante (Figura 5.2-a e Figura 5.2-b) que para cada autocentralidade em G_1^1 existe uma autocentralidade proporcional em G_2^1 e as autocentralidades dos vértices $1 \in V(G_1) \subset V(G_1^1)$ e $1 \in V(G_2) \subset V(G_2^1)$ são proporcionais, ou seja, o critério de mapeamento **C3** pode ser verificado, por conseguinte esses vértices são mapeados. Então, o algoritmo segue para o *Passo 6* e, $1 = i \leq 6$, retorna ao *Passo 2*.

Novamente no *Passo 2* o algoritmo escolhe um vértice de grau *3* em $V(G_1^1)$, por exemplo o vértice 2. Então, no *Passo 3*, G_1^2 (Figura 5.3-a) é obtido com as adições dos vértices 8 e 9 a $V(G_1^1)$ e das arestas $\{2,8\}$ e $\{2,9\}$ a $E(G_1^1)$, e suas autocentralidades são calculadas.

No *Passo 4* busca-se u em $V(G_2^1)$ que atenda ao critério de seleção de candidato a mapeamento **C2**. Pela Figura 5.2(b) pode ser visto que os vértices 4 e 5 em $V(G_2) \subset V(G_2^1)$ atendem a este critério. Selecionando pelo menor índice, G_2^2 (Figura 5.3-b) é obtido com as adições dos vértices 8 e 9 a $V(G_2^1)$ e das arestas $\{4,8\}$ e$\{4,9\}$ a $E(G_2^1)$, e suas autocentralidades são cal-



(a) (b)

Figura 5.3: *Grafos* G_1^2 *e* G_2^2 *e as autocentralidades do vértice* 1 *ao vértice* 6.

culadas. Como após as modificações o critério **C3** pode ser verificado (Figura 5.3), os vértices $2 \in V(G_1) \subset V(G_1^2)$ e $4 \in V(G_2) \subset V(G_2^2)$ são mapeados. O algoritmo vai ao *Passo 6* e, como $2 = i \leq 6$, retorna ao *Passo 2*.

Seguindo os passos do AIGR como exemplificado com as duas primeiras modificações obtém-se G_1^6 e G_2^6. As Figuras 5.4, 5.5, 5.6 e 5.7 ilustram os grafos gerados da terceira modificação em diante.

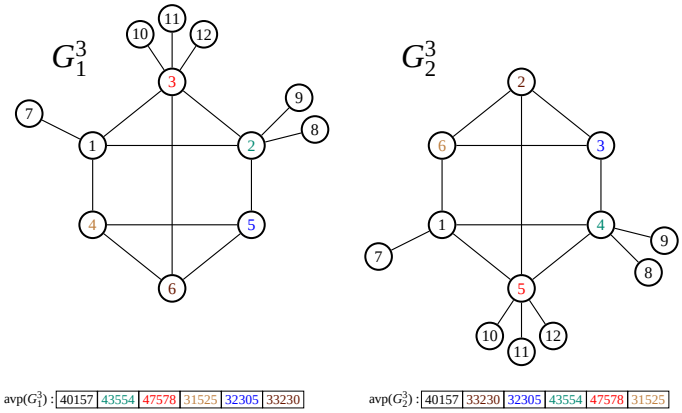


Figura 5.4: *Grafos* G_1^3 *e* G_2^3 *e as autocentralidades do vértice* 1 *ao vértice* 6.

Após todas as modificações, o algoritmo vai ao *Passo 7*, verifica o mapeamento ϕ: $\{1 \rightarrow 1,\ 2 \rightarrow 4,\ 3 \rightarrow 5,\ 4 \rightarrow 6,\ 5 \rightarrow 3, 6 \rightarrow 2\}$ encontrado e o apresenta, pois trata-se de um mapeamento que atende às condições de isomorfismo.

É importante ressaltar que as autocentralidades de grafos regulares são todas iguais e que não é necessário calculá-las efetivamente, basta passar vetores de entradas iguais no *Passo1*. Além disso, note que o algoritmo pode ser interrompido no momento em que o critério **C3** for atendido e todos os vértices de grau *k* estiverem com autocentralidades distintas, pois neste ponto cada vértice de $G_1 \in G_1^i$ possui apenas um candidato a mapeamento em $G_2 \in G_2^i$ (veja os

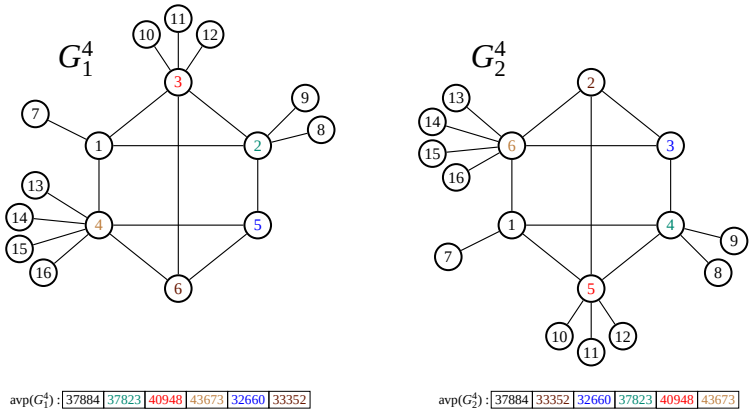


Figura 5.5: *Grafos* G_1^4 *e* G_2^4 *e as autocentralidades do vértice* 1 *ao vértice* 6.

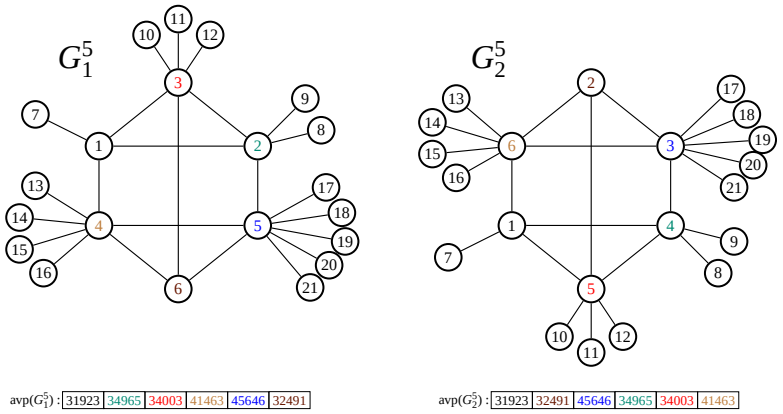


Figura 5.6: *Grafos* G_1^5 *e* G_2^5 *e as autocentralidades do vértice* 1 *ao vértice* 6.

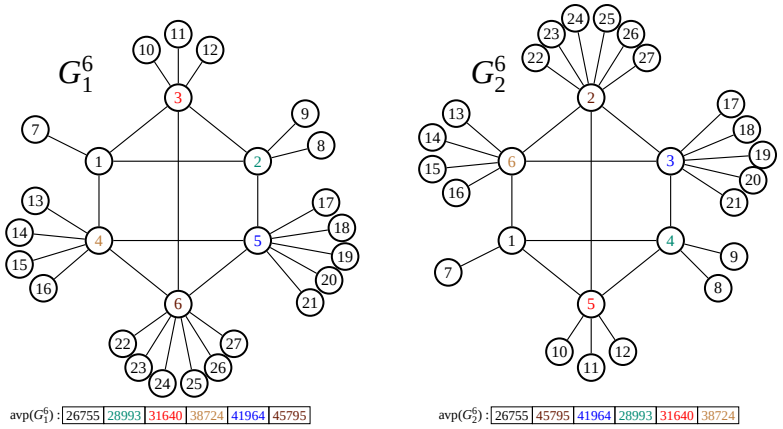


Figura 5.7: *Grafos* G_1^6 *e* G_2^6 *e as autocentralidades do vértice* 1 *ao vértice* 6.

autovetores da Figura 5.3).

No Exemplo 3 não houve realização de *backtracking*, pois os critérios **C2** e **C3** foram atendidos em todas as iterações do algoritmo. Para ilustrar o *backtracking*, suponha que no Exemplo 3 o critério **C3** não foi satisfeito na tentativa de mapeamento do vértice 2 em $V(G_1^1)$ com o vértice 4 de $V(G_2^1)$. Suponha, também, que o critério **C3** não foi atendido em uma tentativa de mapeamento do vértice 2 em $V(G_1^1)$ com o vértice 5 de $V(G_2^1)$. Como em G_2^1 são apenas dois os candidatos a mapeamento com o vértice 2 de $V(G_1^1)$ (veja a Figura 5.2), o algoritmo vai ao *Passo 5*. Neste passo a variável de controle é decrementada e verifica-se a possibilidade de execução de *backtracking* (linhas 20 e 21 do Algoritmo 1). Como o algoritmo estava na segunda tentativa de realização de modificação equivalente, o *backtracking* pode ser executado. Isto é feito como descrito a seguir. Primeiro os vértices e arestas adicionados na segunda modificação realizada sobre G_1 para obtenção de G_1^2 são removidos, voltando-se ao grafo G_1^1 (Figura 5.2-a). Depois, o vértice e a aresta adicionados para obtenção de G_2^1 são removidos e volta-se a G_2. Neste ponto o algoritmo retorna ao *Passo 4* e tenta mapear o vértice 1 de $V(G_1)$ com outro vértice de $V(G_2)$.

5.1.1 Resultados Teóricos

É sabido que se dois grafos são isomorfos, então eles possuem autovetores principais proporcionais (SANTOS, 2010; HE; ZHANG; LI, 2005) e para cada vértice de grau g em um grafo, existe vértice de mesmo grau no outro. Se isto ocorre, então o mapeamento dos vértices deve ser realizado somente entre vértices de mesma autocentralidade e mesmo grau. Isso valida o uso das autocentralidades na poda da árvore de busca, mas não garante o sucesso do AIGR na busca por um isomorfismo. A seguir são apresentados resultados teóricos para validação do AIGR.

Lema 5.1.1 *Considere G um grafo regular de ordem n e m arestas e $G^1, G^2, \ldots, G^n$ os grafos resultantes de n modificações realizadas de acordo com o AIGR. Então, G^n é tal que*

$$|V(G^n)| = n + \frac{n(n+1)}{2} \quad e \quad |E(G^n)| = m + \frac{n(n+1)}{2} \tag{5.1}$$

Prova:

Em cada modificação de número i, $1 \leq i \leq n$, i vértices são adicionados. Então, $|V(G^n)| = n + \sum_{1 \leq i \leq n} i$. Mas o somatório nesta igualdade corresponde a $\frac{n(n+1)}{2}$, isto é, a soma de uma progressão aritmética de razão 1. Analogamente para as arestas de G^n.

■

Lema 5.1.2 *Considere G^n um grafo obtido de acordo com o Lema 5.1.1. Em se tratando de automorfismo, os vértices de $G \subset G^n$ admitem apenas o mapeamento identidade.*

Prova:

Trivial uma vez que em G^n os vértices de G possuem graus distintos que variam de $k+1$ a $k+n$.

■

Com base nos grafos resultantes de modificações equivalentes realizadas pelo AIGR, o Teorema 5.1.1 fornece uma condição suficiente para o isomorfismo de grafos regulares.

Teorema 5.1.1 *Considere G_1 e G_2 grafos k-regulares de ordem n e G_1^q e $G_2^q, 0 < q \leq n$, grafos resultantes de q modificações equivalentes realizadas pelo AIGR. Se G_1^q e G_2^q são isomorfos, então G_1 e G_2 são isomorfos.*

Prova:

Sendo G_1^q e G_2^q grafos isomorfos, existe $\phi : V(G_1^q) \rightarrow V(G_2^q)$ que atende às condições de isomorfismo. Pelos critérios **C1** e **C2** do AIGR é garantido que, para cada $v \in V(G_1)$ de grau k ao qual são ligados i vértices na i-ésima modificação, $1 \leq i \leq q$, existe $u \in V(G_2)$ de grau k que sofreu modificação equivalente tal que $grau(v) = grau(u)$. Então, existem q vértices em ambos os grafos modificados cuja sequência de graus é $(k+1, k+2, \ldots, k+q)$, cuja ϕ obrigatoriamente mapeia-os.

Os demais vértices podem ser divididos em dois grupos distintos: *n-q* vértices de grau k e os vértices de grau 1 inseridos durante a execução do AIGR. Os vértices de grau k de G_1^q só podem ser associados aos vértices de grau k de G_2^q. Por fim, cada vértice de grau 1 em G_1^q com vizinho de grau $k+j$, $1 \leq j \leq q$, é mapeado com um vértice de grau 1 de G_2^q que tenha vizinho de grau $k+j$.

Portanto, ϕ estabelece um mapeamento entre os vértices de $G_1 \subseteq G_1^q$ e $G_2 \subseteq G_2^q$ preservando as adjacências, dado que G_1^q e G_2^q são isomorfos.

■

O Lema 5.1.3 comprova que os grafos resultantes da realização de modificações equivalentes sobre vértices mapeáveis por um isomorfismo são isomorfos.

Lema 5.1.3 *Sejam G_1 e G_2 grafos k-regulares isomorfos de ordem n, $u \in G_1$ e $v \in G_2$, $1 \leq u,v \leq n$, vértices mapeados pelo isomorfismo ϕ: $G_1 \to G_2$. Se G_1^q é o grafo resultante das adições dos q vértices $n+1,\ldots,n+q$ e arestas $\{u,n+1\},\ldots,\{u,n+q\}$ ao grafo G_1, e G_2^q é o grafo resultante das adições de outros q vértices de mesmos rótulos $n+1,\ldots,n+q$ e arestas $\{v,n+1\},\ldots,\{v,n+q\}$ ao grafo G_2, então G_1^q e G_2^q são isomorfos.*

Prova:

De fato, sendo G_1 e G_2 isomorfos, existe um isomorfismo ϕ entre eles. Seja

$$\theta: \begin{cases} V(G_1^q)\backslash V(G_1) & \to \quad V(G_2^q)\backslash V(G_2) \\ \omega & \to \quad r \end{cases}$$

onde r é um vértice que possui mesmo rótulo que ω. Com ϕ e θ é possível apresentar um isomorfismo ψ entre G_1^q e G_2^q como a seguir

$$\psi(s) = \begin{cases} \phi(s), & 1 \leq s \leq n \\ \theta(s), & s > n \end{cases}$$

Logo, G_1^q e G_2^q são isomorfos.

■

O Lema 5.1.3 demonstra que uma modificação equivalente feita pelo AIGR pode levar grafos isomorfos G_1 e G_2 a grafos resultantes G_1^q e G_2^q, também isomorfos. Portanto, se dois grafos são isomorfos, existe um isomorfismo ϕ entre eles, tal que, se o AIGR realizar as modificações equivalentes sobre os vértices mapeados por ϕ, os grafos resultantes dessas modificações serão isomorfos. É exatamente isto que faz o AIGR com o auxílio das autocentralidades. No entanto, esta é uma afirmação baseada no fato do AIGR ter encontrado um isomorfismo para todos os pares de grafos isomorfos testados. Portanto, é necessário provar que

Conjectura 1 *Se G_1 e G_2 são grafos k-regulares isomorfos de ordem n, então o AIGR encontra um isomorfismo entre eles.*

5.2 β-AIGR

O AIGR adiciona um vértice na primeira modificação realizada nos grafos, dois vértices na segunda, três na terceira e esse processo é repetido até que *n* modificações sejam realizadas. Em

se tratando de métodos de obtenção do autovetor principal que utilizam a matriz de adjacências no cálculo deste autovetor, é fácil ver que são $n(n+1)/2$ vértices adicionados em cada grafo de entrada, o que resulta em uma complexidade de espaço no pior caso de $O(t[n+(n(n+1)/2)]^2)$, onde t é o tamanho em *bytes* do tipo de dado utilizado e n é a ordem do grafo. Dependendo da ferramenta utilizada para calcular as autocentralidades é necessário alocar uma matriz ponto flutuante. Então, para ilustrar, considere uma matriz de adjacências de dados ponto flutuante de precisão simples e de ordem 200. Como o tipo de dado em questão normalmente ocupa 4 *bytes*, o espaço em memória exigido para alocação dessa matriz é aproximadamente 1.54 GB. Logo, é preciso encontrar uma estratégia para ao menos atenuar o problema. Desta forma, estratégias para reduzir o uso excessivo de memória foram investigadas.

Quatro maneiras distintas para tratar o problema de memória foram propostas. A primeira é utilizar um método de obtenção do autovetor principal que admita o uso de uma estrutura de representação de grafos menos custosa em termos de memória. A segunda é a implementação de uma condição de parada que faça com que o algoritmo seja interrompido se todos os vértices de grau k possuírem autocentralidades distintas antes da realização de todas as n modificações. A terceira é a implementação de uma versão paramétrica do AIGR, ou seja, uma versão que tenha como parâmetro um valor β, $0 < \beta < n$, com o objetivo de reduzir a quantidade de modificações equivalentes a serem realizadas nos grafos de entrada e, consequentemente, a necessidade de memória. Esta versão do AIGR seria utilizada para facilitar a resolução do PIG por um algoritmo exato, que verificaria se os grafos resultantes obtidos com o AIGR são isomorfos ou não isomorfos. A quarta e última maneira é utilizar uma outra forma de modificação dos grafos que não seja custosa em termos de espaço e tempo de execução. Esta solução pode ser vista em mais detalhes na Seção 5.3.

O *β-AIGR* é a terceira maneira de tratar o problema da necessidade de muito espaço em memória exigido pelo AIGR. Este algoritmo recebe dois grafos regulares simples e conexos e um parâmetro β, $0 < \beta < n$, que define a quantidade de modificações a serem realizadas pelo *β-AIGR*. Após essas modificações o processo de resolução do problema passa a uma segunda fase (Algoritmo 4 - Apêndice A), na qual os grafos resultantes são submetidos a um algoritmo de busca em árvore que restringe as possibilidades de mapeamento através das autocentralidades e verifica as adjacências para cada mapeamento realizado, com o objetivo de encontrar uma bijeção entre os vértices que atenda às condições de isomorfismo, no caso de grafos isomorfos. Nota-se, portanto, que o processo de resolução é dividido em duas fases, a primeira consiste na execução do *β-AIGR* e a segunda, na execução de um algoritmo exato que toma como entrada os grafos resultantes da primeira fase. Essa solução muda a complexidade de espaço para $O(t(n+[\beta(\beta+1)/2)]^2)$, o que pode viabilizar a comparação de grafos de maior ordem, dependendo do

valor de β.

A diferença entre os algoritmos é que o AIGR realiza um mapeamento ao passo que evidencia assimetrias, informando ao término de sua execução, se os grafos são isomorfos ou não isomorfos. Por sua vez, o β-*AIGR* apenas realiza um mapeamento parcial, sendo possível a ele apenas informar que dois grafos não são isomorfos caso não consiga realizar β modificações equivalentes em ambos os grafos. Portanto, o β-*AIGR* é um algoritmo heurístico.

Além do β-*AIGR* não ser capaz de identificar se dois grafos G_1 e G_2 são isomorfos, os grafos resultantes deste algoritmo possuem autovetores principais que podem não ser válidos para uso em uma verificação de isomorfismo. Como o β-*AIGR* é interrompido ao atingir β modificações, não é possível assegurar que não seja necessário realizar *backtracking* para o nível β-*1* devido a uma eventual impossibilidade de descida para um nível seguinte. Logo, ao final de um algoritmo exato (segunda fase) que tenha como entrada G_1^β e G_2^β, ou é apresentado um isomorfismo, o que comprovaria que G_1 e G_2 são isomorfos (Teorema 5.1.1), ou os grafos comparados não são isomorfos. Na segunda resposta, não é possível afirmar se G_1 e G_2 são isomorfos ou não isomorfos.

Exemplo 4 Para ilustrar o uso do β-*AIGR* na resolução do PIGR, considere $\beta = 2$ e os grafos da Figura 4.1 como grafos de entrada. Dessa forma, a Figura 5.3 apresenta exatamente os grafos resultantes das β modificações equivalentes realizadas pelo β-*AIGR* (primeira fase). Na segunda fase é possível utilizar qualquer método de resolução do PIG que mapeie vértices segundo suas autocentralidades para obter o isomorfismo $1 \to 1$, $2 \to 4$, $3 \to 5$, $4 \to 6$, $5 \to 3$ e $6 \to 2$ dado pelas autocentralidades dos vértices. Isto porque todos os véritces de grau 3 em G_1^2 e G_2^2 possuem autocentralidades distintas.

O Exemplo 4, também serve para ilustrar a versão do AIGR que implementa a segunda proposta para redução de memória. Observe que todos os vértices dos grafos da Figura 5.3 que possuem grau 3 são todos de autocentralidades distintas. Então para cada vértice de autocentralidade a em G_1^2 existe apenas uma opção de mapeamento em G_2^2 dada por este valor de autocentralidade.

5.3 Formas Alternativas de Modificação dos Grafos

Um procedimento de adição de vértices e arestas que não seja tão custoso quanto o utilizado pelos algoritmos AIGR e β-*AIGR* pode ser obtido com o aproveitamento de vértices já adicionados aos grafos. A seguir são descritas duas propostas.

Para ilustrar a primeira proposta diferente da apresentada na Seção 5.1, considere o grafo G da Figura 5.8.

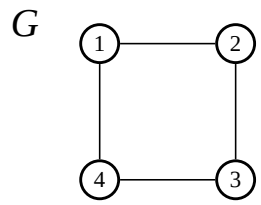


Figura 5.8: *Grafo 2-regular simples e conexo.*

Seguindo o método de modificação apresentado na Seção 5.1, que torna cada vértice de grau k em um vértice de grau $k+i$ na i-ésima modificação, a cada modificação i um vértice de $V(G) \in V(G^{i\text{-}1})$, $0 < i \leq 4$, passa a ter grau $2+i$. Na primeira modificação um vértice é adicionado. Na segunda modificação dois vértices são adicionados. Para cada nova modificação a partir da terceira, apenas dois novos vértices são adicionados e i-2 vértices de $V(G^{i\text{-}1}) \backslash V(G)$ são utilizados para que o vértice modificado tenha grau $2+i$. Os grafos G^1, G^2, G^3 e G^4 resultantes de cada modificação são apresentados na Figura 5.9. Como são n modificações, é fácil perceber que a complexidade de espaço é $O(t[n+2n\text{ - }1]^2)$, onde t é o tamanho do tipo de dado e n é a ordem do grafo. Logo, no caso de métodos de cálculo das autocentralidades que necessitem de uma matriz ponto flutuante, aproximadamente 1,4 MB de memória seria necessário para alocar uma matriz deste tipo que seja de precisão simples e tenha ordem 200, bem menos que o exigido com o uso do método de modificação descrito na Seção 5.1, que exige 1.54 GB. Além da menor necessidade de memória, esse método proporciona a obtenção do autovetor principal em menor tempo, pois as matrizes de adjacência possuem dimensões menores. Para facilitar, este método é referenciado no restante deste trabalho como método de modificação mod_{2n}, enquanto que o método de modificação originalmente proposto é referenciado por mod_{nn}.

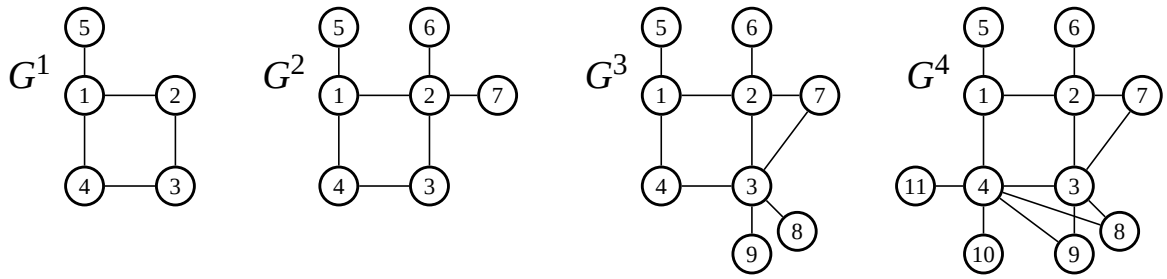


Figura 5.9: *Grafos resultantes obtidos com o método de modificação que adiciona* $2n-1$ *vértices.*

A segunda proposta diferente da apresentada na Seção 5.1 está ilustrada na Figura 5.10. A

diferença para os demais métodos de modificação pode ser notada a partir da segunda modificação, quando o vértice 2 passa a ter grau 4. As modificações são feitas de forma que apenas um vértice seja adicionado para cada nova modificação e aproveitando-se todos os vértices adicionados nas modificações anteriores, proporcionando um menor gasto de memória. Então, para n modificações são necessários n novos vértices, o que resulta em uma complexidade de $O(t[2n]^2)$. Este método é referenciado no restante deste trabalho como método de modificação mod_n.

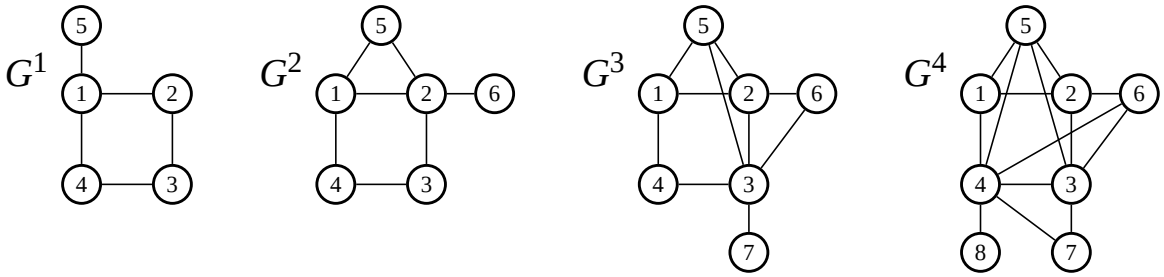


Figura 5.10: *Grafos resultantes obtidos com o método de modificação que adiciona n vértices.*

Devido a grande vantagem que esses procedimentos fornecem, seis versões do AIGR foram implementadas para testes computacionais: o $AIGR_{nn}$ que implementa o método mod_{nn}, o $pAIGR_{nn}$ (parcial AIGR$_{nn}$) que implementa o método mod_{nn} e a segunda solução apresentada nesta seção para o problema da grande necessidade memória do AIGR e a sua versão $pAIGR_n$ que implementa o método mod_n, o β-$AIGR_{nn}$, o $AIGR_n$ que usa o método de modificação mod_n e a sua versão paramétrica, o β-$AIGR_n$.

Versões do AIGR que implementam o método de modificação mod_{2n} não foram consideradas no Capítulo 6, capítulo de resultados computacionais, pois o método que realiza n adições consiste, principalmente em termos de uso de memória, na melhor opção dentre todos os métodos descritos neste trabalho.

6 Resultados Computacionais

Nos testes computacionais foram utilizados grafos regulares gerados aleatoriamente com um programa de geração aleatória de grafos regulares implementado por Luchi e Rangel (2010). Trata-se da implementação de um algoritmo de tentativa e erro que gera grafos k-regulares de ordem n a partir do grafo 2-regular de mesma ordem. Esse algoritmo realiza n^2 tentativas de inserções aleatórias de arestas sem que o grau de qualquer dos vértices torne-se maior que k. Ao final dessas tentativas é feita uma busca por pares de vértices que tenham graus inferiores a k para, então, continuar com o processo de inserção de arestas até que não existam mais pares de vértices {u,v} tais que os graus de u e v sejam inferiores a k. Neste ponto se o grafo gerado for regular, ele é armazenado em um arquivo, caso contrário, o processo de inserção aleatória de arestas é retomado a partir do grafo 2-regular.

A partir desse programa, foi gerado um grupo de grafos regulares não isomorfos de ordens 100, 150, 200, 250, 300, 350, 400, 450 e 500, cada grafo com *densidade*[1] percentual de 4%, denotado por *grd*04. Para cada valor de n foram gerados 10 grafos com essas características. Grafos regulares não isomorfos de ordens 50, 100, 150, 200, 250 e 300 vértices e densidades percentuais nos intervalos [10,20), [20,30), [30,40) e [40,50) também foram gerados com o mesmo programa para verificar o comportamento do algoritmo proposto com grafos não esparsos. Neste caso, para cada par $\{n,d\}$, onde n é o número de vértices e d a densidade, foram gerados 10 grafos. Esse segundo grupo de grafos foi denotado por *grd*10-49.

Outros três grupos de grafos regulares não isomorfos foram obtidos da literatura, sendo dois deles compostos por grafos fortemente regulares (veja a definição no Capítulo 2). Dos grupos formados por grafos fortemente regulares, um deles consiste em subgrupos compostos pelos 10 primeiros grafos de mesmos parâmetros n,k,λ,μ, de ordens 25, 26, 29, 35, 36, 40, 45 e 64, e pelos 4 grafos de ordem 28 e mesmos parâmetros, disponíveis em Spence (2011). Este grupo foi denotado por *esp*. O outro grupo consiste nos grafos das famílias *sts-sw*, *latin-sw* e *had-sw* presentes em Junttila e Kaski (2011), que foram escolhidos por possuírem vários subgrupos

[1] Neste trabalho é considerado como densidade de um grafo de ordem n e m arestas a relação $\frac{m}{t}$, onde t representa o máximo de arestas que um grafo de ordem n pode conter.

formados por 11 grafos fortemente regulares não isomorfos com os mesmos parâmetros. Por serem não esparsos, os grafos escolhidos destas famílias são de ordens inferiores a 200. O terceiro grupo de grafos retirados da literatura é composto pelos grafos regulares esparsos de ordens 1000, 2000, 3000, 4000 e 5000 da família *rnd*-3-*reg*, também presente em Junttila e Kaski (2011). Nesta família de grafos, para cada ordem n existem 11 grafos regulares não isomorfos. Para detalhes sobre o primeiro grupo de grafos fortemente regulares consulte Spence (2011), para o segundo consulte Junttila e Kaski (2007).

De cada um dos subgrupos de grafos determinados pelo par $\{n,d\}$ e contidos nos grupos *grd*04 e *grd*10-49, foi escolhido um grafo para geração de 10 grafos isomorfos. Desta forma, foram criados os grupos de grafos isomorfos *igrd*04 e *igrd*10-49. Para isto, foi utilizado um algoritmo de rerrotulação de vértices implementado durante o desenvolvimento desse trabalho (Algoritmo 5 - Apêndice A). Logo, os grupos de grafos isomorfos *igrd*04 e *igrd*10-49 possuem as mesmas ordens e densidades dos grupos *grd*04 e *grd*10-49. O mesmo foi feito com grafos dos grupos retirados da literatura, o que resultou nos grupos de grafos isomorfos denominados *iesp*, *ists-sw*, *ilatin-sw*, *ihad-sw* e *irnd*-3-*reg*.

As instâncias do problema foram formadas por pares de grafos distintos de mesma ordem, densidade e pertencentes aos mesmos grupos. Os testes foram realizados em um computador com processador Intel® Core™ 2 Duo T6600 de 2.2GHz, 2GB de memória RAM e Sistema Operacional Ubuntu 10.04 32 bits. Os tempos médios de execução, em segundos, foram obtidos através da função *gettimeofday* da biblioteca *sys/time.h*. Essas médias foram calculadas com os tempos de execução obtidos nos testes realizados com instâncias com os mesmos valores de ordem n e densidade d. De forma análoga foram aferidos, também, os tempos médios de execução referentes à segunda fase do processo de resolução que utiliza as versões paramétricas e as médias das quantidades de *backtrackings* realizados.

Foi estabelecido o tempo limite de uma hora para resolução do PIGR. Em todos os casos em que a resolução do problema com alguma instância ultrapassou o tempo limite, instâncias de mesmas características ou dimensões maiores não foram executadas.

6.1 Detalhes de Implementação

A linguagem de programação escolhida para a implementação das versões do AIGR foi a linguagem C. Na implementação dessas versões foram utilizadas várias estruturas de dados para armazenar, basicamente, inteiros sem sinal. A seguir são descritas as principais estruturas utilizadas.

Para representar os grafos foi utilizada uma matriz de n linhas e k colunas para cada grafo, onde n é a ordem dos grafos de entrada e k o grau dos vértices. Considerando os vértices indexados por inteiros de 0 a n-1, cada linha i dessa matriz armazena os índices dos vértices adjacentes ao vértice i. Foram alocados dois vetores para armazenar as autocentralidades de cada grafo. Para auxiliar no *backtracking* e reduzir o número de chamadas a função de cálculo das autocentralidades, outras duas matrizes foram utilizadas para armazenar todos os autovetores principais calculados em cada par de modificações, uma em cada grafo, que resultou em mapeamento. Além dessas estruturas, duas filas foram construídas para controle de vértices disponíveis e uma pilha foi criada para auxiliar no *backtracking*.

Outra informação relevante está na forma de implementação do critério de seleção de vértices **C1**. Para auxiliar na poda da árvore de busca, o critério **C1** foi implementado de maneira tal que o primeiro vértice selecionado sempre é aquele que possuir índice zero. A partir da segunda seleção os vértices escolhidos são todos adjacentes a vértices selecionados em iterações anteriores, com a restrição de que todos os vértices adjacentes ao primeiro selecionado devem ser escolhidos antes de qualquer outro. Depois todos os vértices adjacentes ao segundo selecionado devem ser escolhidos antes de qualquer outro, e assim sucessivamente. Obviamente o critério de seleção **C2** foi implementado seguindo a mesma ideia.

6.1.1 Métodos de obtenção das autocentralidades

Foram testadas duas formas de obtenção dos autovetores principais. Uma delas consiste na função de precisão dupla *_dssyevr* presente no pacote CLAPACK versão 3.2.1, que é um pacote obtido do LAPACK (*Linear Algebra PACKage*) através da ferramenta de conversão *f2c* presente no próprio CLAPACK, que converte as rotinas de cálculo de sistemas lineares do LAPACK implementadas em Linguagem Fortran para a Linguagem C. Essa função é uma das mais eficientes no cálculo de autovetores de matrizes simétricas do tipo ponto flutuante presente no CLAPACK. Ela permite calcular apenas uma faixa de autovalores e seus autovetores associados. Neste trabalho, essa função foi utilizada de maneira a retornar apenas o índice do grafo e seu autovetor associado (o autovetor principal).

O outro método de obtenção das autocentralidades utilizado é bem semelhante a uma adaptação do *Power Method* (SAAD, 2003) feita por Baroni, Boeres e Rangel (2011). Considere A uma matriz de adjacência do grafo G e v_0 um vetor inicial não nulo. Sendo $i \geq 1$ e α_i a maior coordenada em valor absoluto do vetor $Av_{i\text{-}1}$, o *Power Method* realiza sucessivos produtos $\frac{1}{\alpha_i}Av_{i\text{-}1}$ até que o vetor resultante v_i seja correspondente ao vetor de autocentralidades.

A adaptação proposta por Baroni, Boeres e Rangel (2011) utiliza uma lista de adjacências

no lugar da matriz, não realiza a divisão por α_i e estabelece como critério de parada a obtenção de v_i que não forneça uma partição de $V(G)$ maior que a obtida com $v_{i\text{-}1}$ ou tenha tamanho igual a n. Quando o critério de parada é atingido, $v_{i\text{-}1}$ é retornado. Considerando L a lista de adjacências de um grafo G, L_z a lista de adjacência do vértice $z \in G$ e $\pi(v_i)$ a partição obtida com as coordenadas do vetor v_i, o *Power Method Adaptado* pode ser escrito como o Algoritmo 2.

Algoritmo 2: *Algoritmo Power Method Adaptado.*

Dados: $[\, L, v_0 \,]$

1 $i \leftarrow 1$;
2 **Enquanto** $|\pi(v_i)| > |\pi(v_{i\text{-}1})|$ *e* $|\pi(v_i)| < n$, **faça**
3 $\quad \vec{v}_i^z = \sum_j \vec{v}_{i\text{-}1}^j$, onde $j \in L_z$;
4 $\quad i \leftarrow i+1$;

O algoritmo proposto por Baroni, Boeres e Rangel (2011) foi adaptado para este trabalho, para fins de ganho de desempenho. Como os vértices adicionais são ligados apenas a vértices dos grafos de entrada seguindo um padrão bem definido, é possível identificar facilmente os vértices aos quais eles são vizinhos. Logo, não é preciso modificar as matrizes que representam os grafos (que neste trabalho são exatamente como listas de adjacências). Para cada grafo G, basta utilizar um vetor que represente $V(G)$ e os vértices adicionais e calcular os produtos $Av_{i\text{-}1}$ em partes. Esta adaptação é referenciada neste texto apenas por *Power*.

A versão do *Power* para o método de modificação mod_{nn}, que pode ser vista no Algoritmo 3, divide o processo de cálculo de cada v_i em três partes. Na primeira parte são calculadas as coordenadas referentes aos vértices do grafo k-regular G (grafo representado pela lista L no Algoritmo 3) diretamente envolvidos em modificações (*Passo 1*). Este cálculo é subdividido em duas partes: uma é referente aos vértices que possuem grau k e são vizinhos a z (Linha 3) — vértice cuja autocentralidade é calculada — e a outra é referente aos vértices vizinhos a z que foram adicionados nas modificações (Linha 4). Na segunda parte do *Power* são calculadas as coordenadas referentes aos vértices de G não envolvidos diretamente nas modificações (*Passo 2*). Por fim, na terceira são calculadas as coordenadas referentes aos vértices adicionados em cada modificação. Observe que com esta divisão do processo de cálculo das coordenadas de v_i, troca-se um laço de $grau(z)$ - $k+1$ comparações (no caso de condição pré-testada) pela execução de um único produto (Linha 4). Isto evita a realização de $(r+1)(r+2)/2$ comparações em cada iteração do *Power*, onde r é a quantidade de modificações realizadas sobre G.

O *Power* ainda permite trocar as $(r+1)(r+2)/2$ comparações do laço (de condição pré-testada) a ser utilizado na implementação do *Passo 3* por r comparações. Basta observar que é

Algoritmo 3: *Algoritmo de Power adaptado ao método de modificação $mod_n n$.*

Dados: $[\, L, v_0 \,]$

Enquanto $|\pi(v_1)| > |\pi(v_0)|$ *e* $|\pi(v_1)| < n$, **faça**

  Passo *1*:

    $\vec{v}_1^z = \sum_j \vec{v}_0^j$, $1 \leq j \neq z \leq n$, $grau(z) > k$ e j adjacente a z;

    $\vec{v}_1^z += (grau(z) - k) * \vec{v}_0^q$, $grau(q) = 1$ e q adjacente a z;

  Passo *2*:

    $\vec{v}_1^z = \sum_j \vec{v}_0^j$, $1 \leq j \neq z \leq n$, $grau(z) = k$ e j adjacente a z;

  Passo *3*:

    $\vec{v}_1^z = \vec{v}_0^j$, $z > n$ e j adjacente a z;

  $\vec{v}_0 \leftarrow \vec{v}_1$;

possível utilizar apenas um único vértice para representar cada modificação. Para isto, substituiu-se a diferença entre o grau de z e k pelo número da modificação na qual z foi diretamente envolvido. Além de reduzir a complexidade de tempo do *Power*, esta otimização resulta em um menor uso de memória, pois com ela não é preciso alocar vetores de tamanho $n + r(r+1)/2$, em vez disso bastam vetores de tamanho $n+r$. No entanto, esta otimização não foi utilizada nos testes computacionais, cujos resultados são apresentados no Capítulo 6. Uma versão do *Power* voltada ao método de modificação mod_n pode ser vista no Algoritmo 6 do Apêndice A.

Selecionado o melhor método de obtenção de autocentralidades

Objetivando selecionar um desses métodos de obtenção das autocentralidades para testes com todas as instâncias geradas neste trabalho e selecionadas da literatura, nesta subseção são apresentados os resultados obtidos nos testes realizados com as versões $AIGR_{nn}$ e $AIGR_n$ implementadas com o algoritmo *Power* e com a função *_dssyevr*, tendo como entrada instâncias formadas por grafos dos grupos *grd*04, *igrd*04, *esp* e *iesp*.

O gráfico da Figura 6.1 apresenta os resultados do $AIGR_{nn}$ com o algoritmo *Power* e com a função *_dssyevr*, enquanto que os gráficos das Figuras 6.2 e 6.3 apresentam os resultados do $AIGR_n$ com esses métodos. Todos esses gráficos são referentes aos testes realizados com instâncias formadas por grafos dos grupos *grd*04 e *igrd*04. Os gráficos dos resultados do $AIGR_{nn}$ com esses dois métodos de cálculo de autocentralidades para o grupo *igrd*04 não são apresentados, pois a implementação deste algoritmo com a função *_dssyevr* não resolve as instâncias de ordem 100 desse grupo em menos de uma hora, enquanto que a versão desse algoritmo que utiliza o *Power* não teve dificuldades em resolvê-las, como pode ser visto na Figura 6.20.

Os gráficos das Figuras 6.4 e 6.5 apresentam os resultados do $AIGR_{nn}$ com a adaptação do

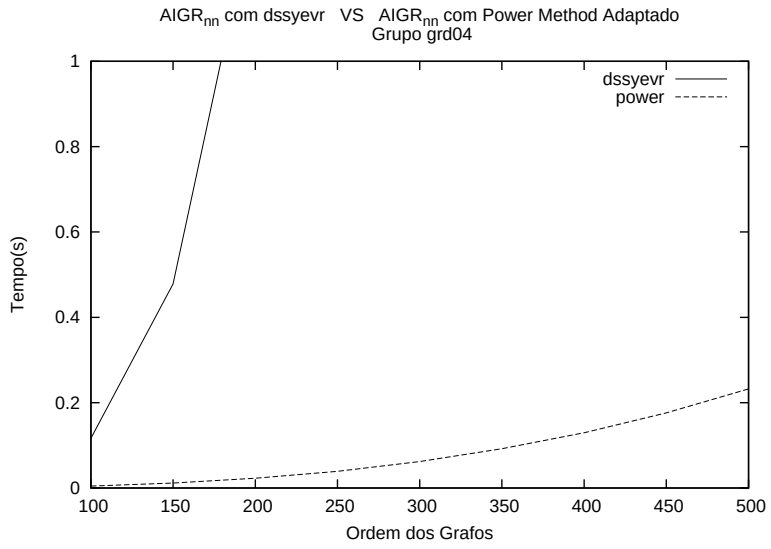


Figura 6.1: *AIGR$_{nn}$* com a adaptação do *Power* e com a função *_dssyevr* para o grupo *grd*04.

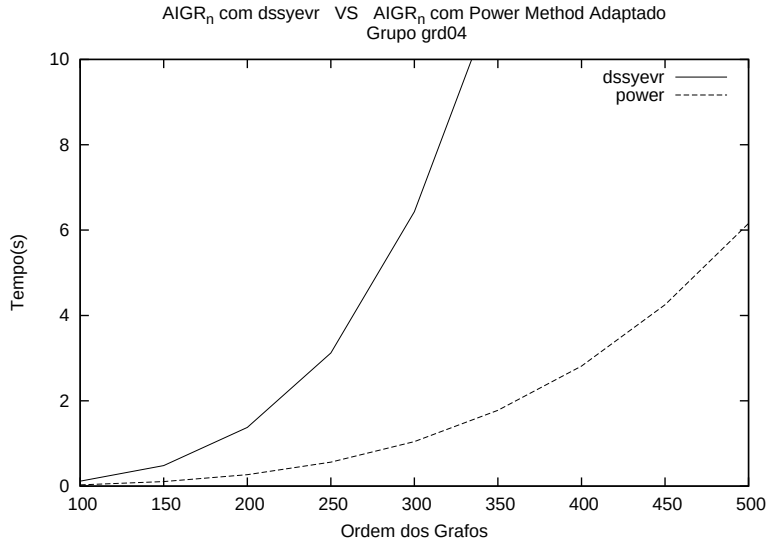


Figura 6.2: *AIGR$_n$* com a adaptação do *Power* e com a função *_dssyevr* para o grupo *grd*04.

Power e com a função *_dssyevr*, enquanto que os gráficos das Figuras 6.6 e 6.7 apresentam os resultados do *AIGR$_n$* com esses métodos. Esses gráficos são referentes a testes com instâncias formadas por grafos dos grupos *esp* e *iesp*.

O *Power* utiliza para o cálculo das autocentralidades, as matrizes originalmente alocadas para armazenamento dos grafos de entrada. Já a função *_dssyevr* necessita de uma nova matriz de ordem *n* a cada chamada da função. Enquanto esta função exige a matriz de adjacência como parâmetro e modifica as entradas dessa matriz, impossibilitando um melhor uso de memória, o *Power* permite tirar vantagem das densidades dos grafos, pois é possível utilizar uma lista de adjacência para representá-los. A consequência disso é um melhor uso da memória e produtos

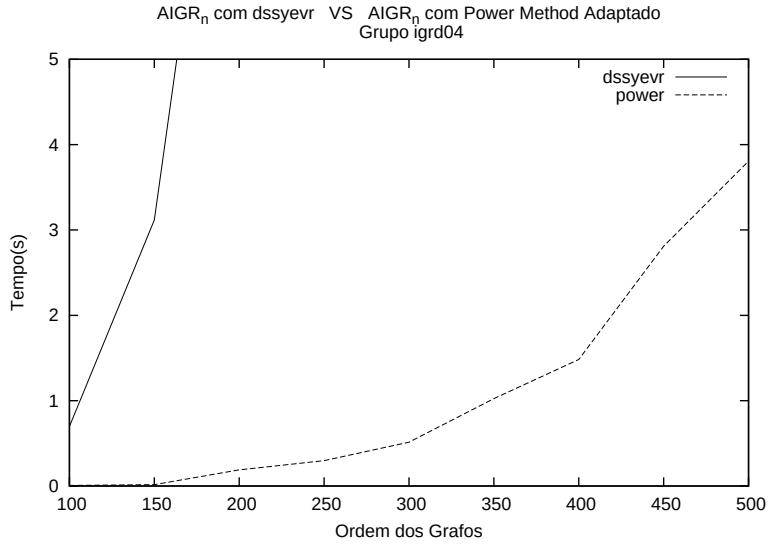


Figura 6.3: $AIGR_n$ com a adaptação do *Power* e com a função *_dssyevr* para o grupo *igrd*04.

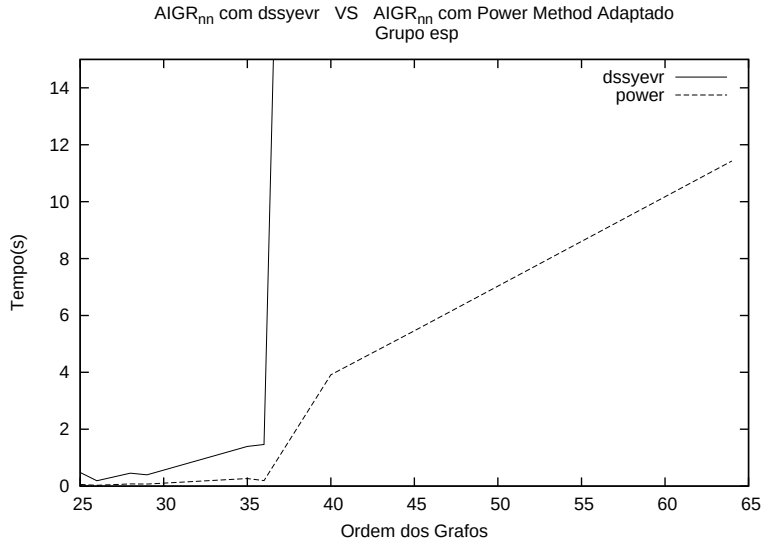


Figura 6.4: $AIGR_{nn}$ com a adaptação do *Power* e com a função *_dssyevr* para o grupo *esp*.

$A(G)v_i$ mais eficientes. Para o PIG em geral, essas características do *Power* tornam-se ainda mais interessantes ao lembrar que, uma instância formada por grafos G_1 e G_2 de densidades superiores a 50%, pode ser resolvida pela instância formada pelo grafo complementar de G_1 e o grafo complementar de G_2.

Observando os gráficos percebe-se o ganho acentuado de tempo de execução das versões do AIGR implementadas com o *Power*.

Diante dessas vantagens que o *Power* fornece e do ganho acentuado de tempo de execução observado nos resultados apresentados nos gráficos, nas próximas seções todos os resultados apresentados são referentes às versões do AIGR definidas ao final da Seção 5.3 e implementadas

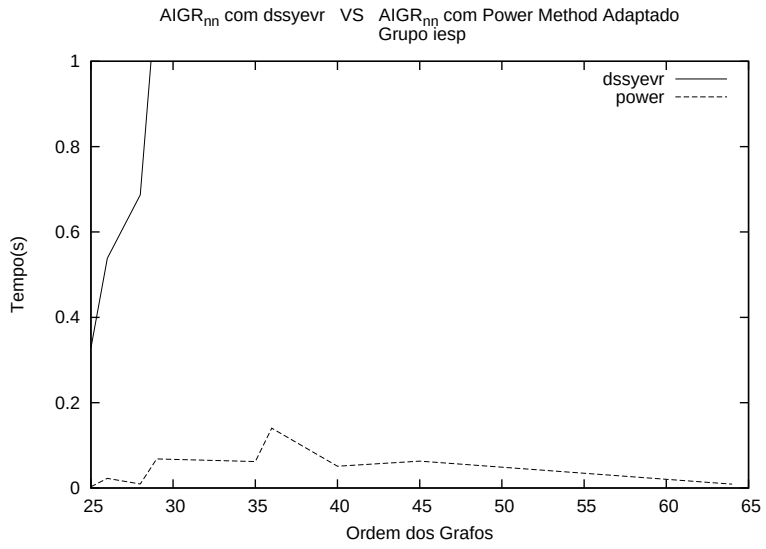


Figura 6.5: $AIGR_{nn}$ com a adaptação do *Power* e com a função *_dssyevr* para o grupo *iesp*.

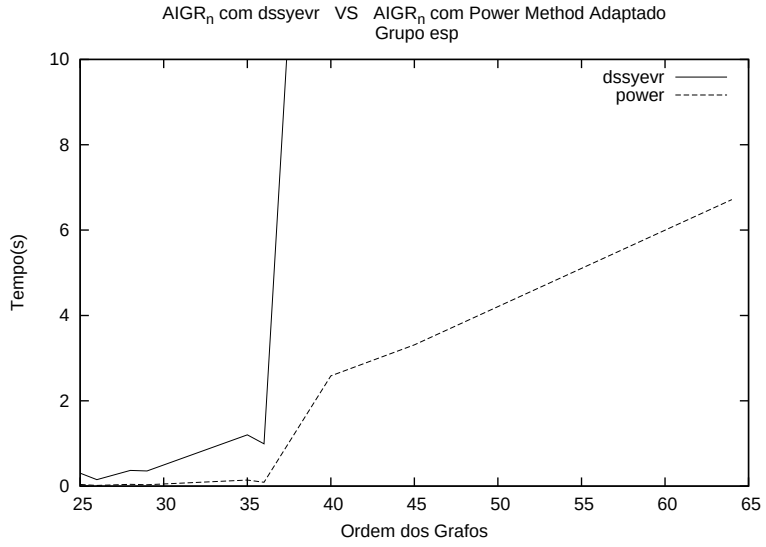


Figura 6.6: $AIGR_n$ com a adaptação do *Power* e com a função *_dssyevr* para o grupo *esp*.

com o algoritmo *Power*.

6.1.2 Definindo o parâmetro das versões paramétricas

O parâmetro β consiste em um valor percentual relativo á ordem dos grafos. Para determinar esse valor são utilizadas as versões $pAIGR_{nn}$ e $pAIGR_n$. O ideal seria utilizar valores de β que fossem inferiores as quantidades de modificações necessárias às versões $pAIGR_{nn}$ e $pAIGR_n$, pois de outra forma os algoritmos paramétricos teriam, muito provavelmente, desempenho inferior a essas versões. No entanto, esses valores podem variar entre instâncias do PIGR de mesma ordem. Além disso, como dito na Seção 5.2, não é possível garantir quando a inter-

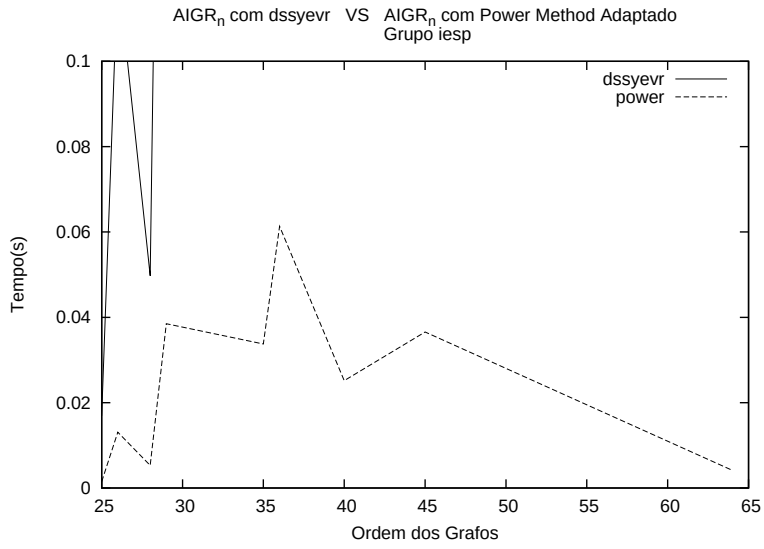


Figura 6.7: $AIGR_n$ com a adaptação do *Power* e com a função *_dssyevr* para o grupo *iesp*.

rupção do AIGR em qualquer nível da árvore de busca é segura. Logo, quão maior for o valor de β, maiores são as chances dos mapeamentos parciais realizados pelas versões β-$AIGR_{nn}$ e β-$AIGR_n$ serem válidos quanto ao isomorfismo.

Nesta subseção as quantidades de modificações realizadas pelas versões $pAIGR_{nn}$ e $pAIGR_n$ são analisadas. Com base nesta análise, o valor de β é definido. Para aferir as quantidades de modificações que as versões $pAIGR_{nn}$ e $pAIGR_n$ necessitam para distinguir todos os vértices de um grafo, foram utilizados os grupos de grafos isomorfos *igrd*04 e *iesp*.

A Tabela 6.1 apresenta as médias das quantidades de modificações realizadas pelas versões $pAIGR_{nn}$ e $pAIGR_n$ nos testes realizados com o grupo *igrd*04, bem como o percentual relativo ao número de vértices que cada média representa, enquanto que a Tabela 6.2 apresenta essas informações obtidas nos testes dessas mesmas versões do AIGR com o grupo *iesp*.

Ordem	$pAIGR_{nn}$	$pAIGR_{nn}$**(%)**	$pAIGR_n$	$pAIGR_n$**(%)**
100	2,0	2,0	99,0	99,0
150	2,0	1,3	149,0	99,3
200	2,0	1,0	199,0	99,5
250	2,0	0,8	249,0	99,6
300	2,0	0,7	299,0	99,7
350	2,0	0,6	349,0	99,7
400	2,0	0,5	399,0	99,8
450	2,0	0,4	449,0	99,8
500	2,0	0,4	499,0	99,8

Tabela 6.1: Quantidades de modificações realizadas pelas versões $pAIGR_{nn}$ e $pAIGR_n$ nos testes com o grupo *igrd*04.

Ordem	$pAIGR_{nn}$	$pAIGR_{nn}$**(%)**	$pAIGR_{n}$	$pAIGR_{n}$**(%)**
25	6,6	26,5	24	96,0
26	7,4	28,3	25	96,2
28	8,5	30,6	27	96,4
29	7,9	27,0	28	96,6
35	8,4	24,1	34	97,1
36	9,3	25,9	35	97,2
40	9,4	23,1	39	97,5
45	16,4	36,4	44	97,8
64	14,7	23,0	63	98,4

Tabela 6.2: Quantidades de modificações realizadas pelas versões $pAIGR_{nn}$ e $pAIGR_{n}$ nos testes com o grupo *iesp*.

Pelos resultados apresentados nas Tabelas 6.1 e 6.2 observa-se que a versão $pAIGR_{n}$ realiza exatamente n-1 modificações durante o processo de resolução do PIGR com as instâncias formadas com os grafos da amostra selecionada. Logo, apenas a versão $pAIGR_{nn}$ é levada em consideração na obtenção do parâmetro β.

Na Tabela 6.1 nota-se que a versão $pAIGR_{nn}$ necessita realizar apenas duas modificações nos grafos do grupo *igrd*04 para que todos os vértices sejam distintos. Pela Tabela 6.2 observam-se médias não inferiores a 23%, logo, os resultados dessa Tabela 6.1 não servem de parâmetro para a obtenção de β.

Na 6.2 observa-se que a maior parte das médias estão entre 20% e 30%. No entanto, pelas médias percentuais máximas que podem ser visualizadas na Tabela 6.3, nota-se que existem grafos que necessitam de $0.4n$ modificações para que seus vértices sejam todos distintos. Além disso, observa-se que a maior parte dos percentuais máximos estão entre 30% e 40%. Logo, escolher um valor que esteja na casa de 20% da quantidade de vértices de um grafo fortemente regular pode levar as versões paramétricas a realizarem mapeamentos parciais que não sejam parte de um isomorfismo, aumentando a probabilidade do algoritmo que continua a descida na árvore informar o não isomorfismo de instâncias formadas por grafos isomorfos. Por isso, o valor escolhido para β é equivalente a 30% do número de vértices de cada grafo.

6.2 Comparando as versões do AIGR

Objetivando selecionar as melhores versões para comparação com uma das melhores ferramentas de resolução do PIG presentes na literatura, o *Nauty*, esta seção apresenta uma comparação entre as versões do AIGR que utilizam os métodos de modificação mod_{nn} e mod_{n}. Para tanto, os grupos selecionados para a realização desses testes foram: *grd*04 e *igrd*04; *grd*10-49

Ordem	Média(%)	Máximo(%)
25	26,5	32,0
26	28,3	34,6
28	30,6	39,3
29	27,0	37,9
35	24,1	28,6
36	25,9	30,6
40	23,1	25,0
45	36,4	40,0
64	23,0	26,6

Tabela 6.3: Médias e máximos percentuais das quantidades de modificações realizadas pelo $pAIGR_{nn}$ nos testes com o grupo *iesp*.

e *igrd*10-49; e *esp* e *iesp*. Primeiramente são apresentados os gráficos relativos aos tempos de execução obtidos nos testes realizados com os grupos de grafos não isomorfos. Os gráficos correspondentes às versões de modificação mod_{nn} e mod_n são apresentados lado a lado. Em seguida são apresentados os resultados dos testes realizados com os grupos de grafos isomorfos. Os gráficos dos grupos *grd*10-49 e *igrd*10-49, são subdivididos em grupos de grafos com densidades nos intervalos [10,20), [20,30), [30,40) e [40,50), com cada intervalo sendo referenciado, respectivamente, por *grd*10, *grd*20, *grd*30, e *grd*40.

6.2.1 Grafos não isomorfos

Nesta subseção são apresentados os gráficos construídos a partir dos tempos de execução obtidos nos testes realizados com as versões mod_{nn} e mod_n do AIGR e os grupos de grafos não isomorfos *grd*04, *grd*10-49 e *esp*.

Grupo *grd*04

As Figuras 6.8 e 6.9 apresentam os gráficos dos tempos de execução obtidos com os testes realizados com o grupo *grd*04 para as versões mod_{nn} e mod_n do AIGR.

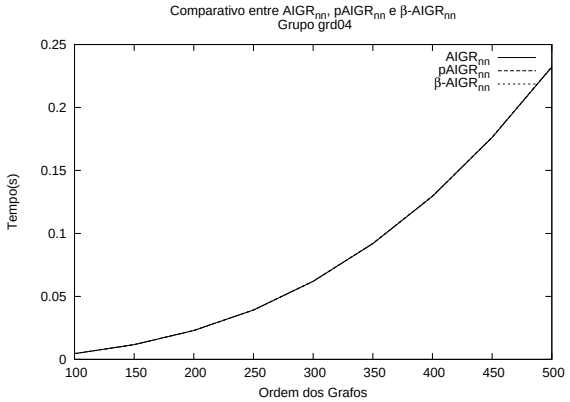


Figura 6.8: Resultados dos testes com o grupo *grd*04 e as versões mod_{nn}.

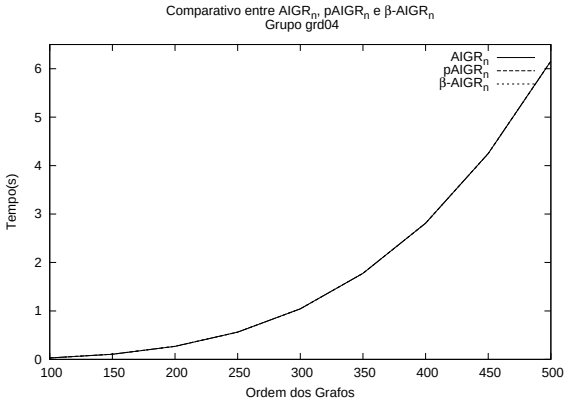


Figura 6.9: Resultados dos testes com o grupo *grd*04 e as versões e mod_n.

Grupo *grd*10-49

Do par formado pelas Figuras 6.10 e 6.11, ao par formado pelas Figuras 6.16 e 6.17, são apresentados os gráficos dos tempos de execução obtidos nos testes realizados com os subgrupos *grd*10, *grd*20, *grd*30 e *grd*40, respectivamente, e as versões mod_{nn} e mod_n do AIGR.

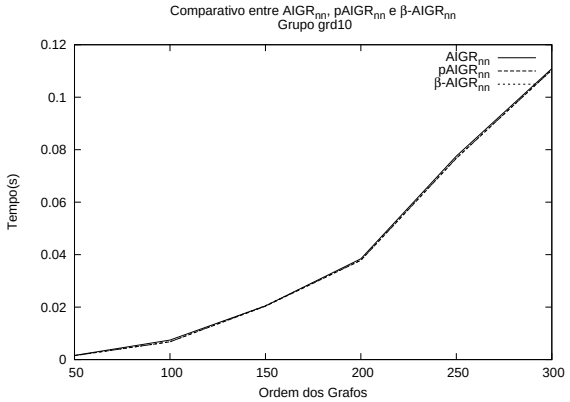


Figura 6.10: Resultados dos testes com o grupo *grd*10 e as versões mod_{nn}.

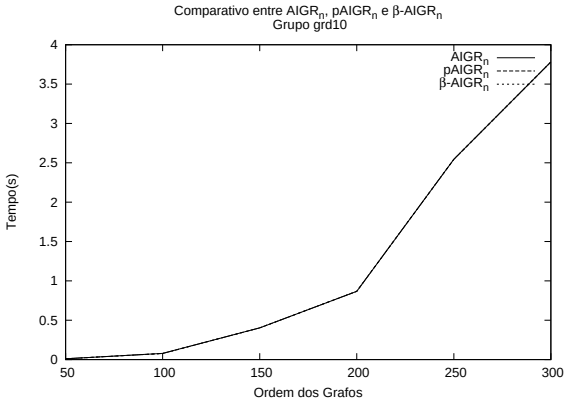


Figura 6.11: Resultados dos testes com o grupo *grd*10 e as versões mod_n.

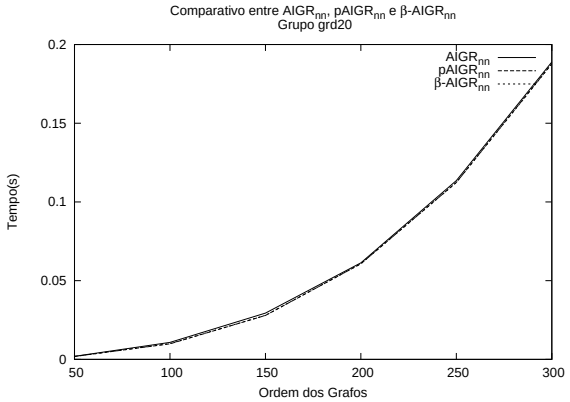


Figura 6.12: Resultados dos testes com o grupo *grd*20 e as versões mod_{nn}.

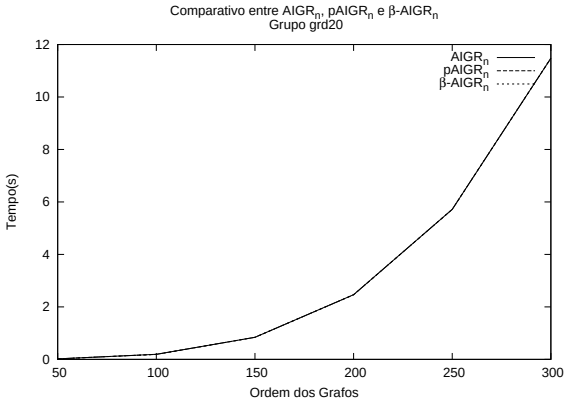


Figura 6.13: Resultados dos testes com o grupo *grd*20 e as versões mod_{n}.

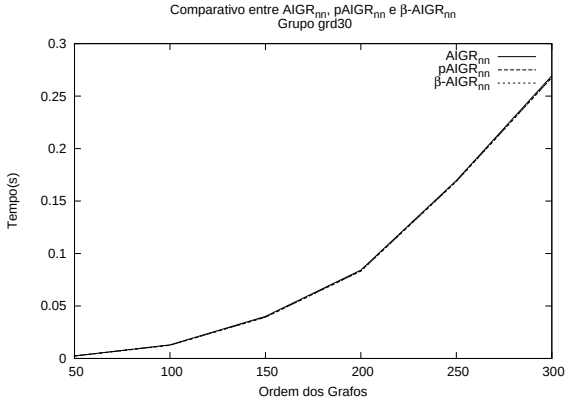


Figura 6.14: Resultados dos testes com o grupo *grd*30 e as versões mod_{nn}.

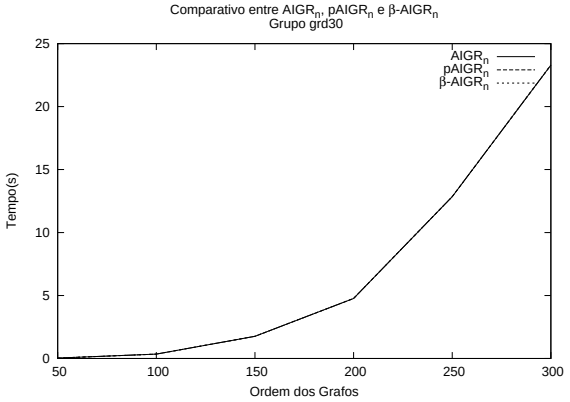


Figura 6.15: Resultados dos testes com o grupo *grd*30 e as versões mod_{n}.

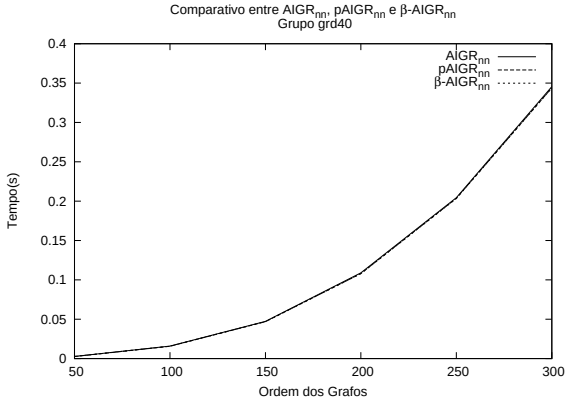


Figura 6.16: Resultados dos testes com o grupo *grd*40 e as versões mod_{nn}.

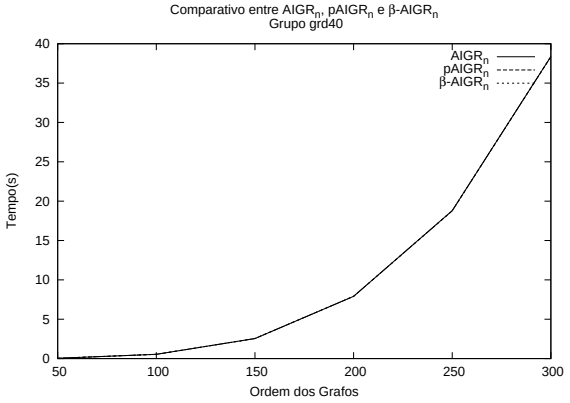


Figura 6.17: Resultados dos testes com o grupo *grd*40 e as versões mod_{n}.

Grupo *esp*

As Figuras 6.18 e 6.19 apresentam os gráficos dos tempos de execução obtidos nos testes computacionais realizados com o grupo *esp* e para as versões mod_{nn} e mod_n do AIGR .

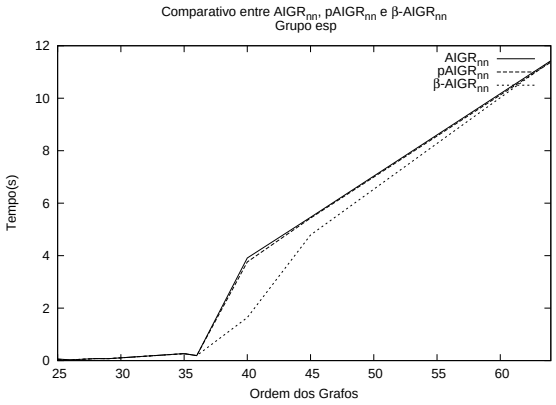


Figura 6.18: Resultados dos testes com o grupo *esp* e as versões mod_{nn}.

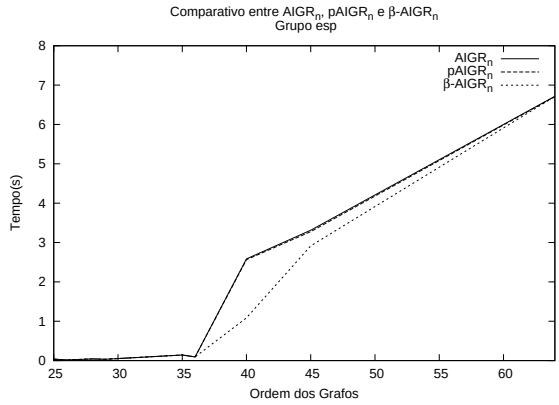


Figura 6.19: Resultados dos testes com o grupo *esp* e as versões mod_n.

Análise dos resultados

Dos gráficos apresentados nesta subseção, de imediato nota-se que quase todas as curvas estão sobrepostas, a exceção do par de gráficos gerados com os tempos de execução obtidos nos testes com os grafos do grupo *esp*. Isto indica que as versões do AIGR de mesmo método de modificação, possuem o mesmo comportamento ao resolver o PIGR de grafos não isomorfos gerados aleatoriamente. Contudo, para grafos fortemente regulares, observa-se que as versões paramétricas apresentaram melhor desempenho com instâncias formadas por grafos de 40 e 45 vértices. Isto é devido as quantidades de *backtrackings* realizadas por estas versões, como mostram as Tabelas 6.4 e 6.5.

A Tabela 6.4 mostra que a versão β-$AIGR_{nn}$ realizou uma quantidade de *backtrackings* um pouco inferior para as instâncias de ordens 25, 26 e 29, e significativamente inferior para as instâncias de ordens 40 e 45. O mesmo pode ser observado na Tabela 6.5, com uma diferença um pouco mais significativa entre as quantidades de *backtrackings* na linha referente as instâncias de ordem 25.

Outro fato a ser observado é a grande diferença entre os tempos de execução das versões mod_{nn} e das versões mod_n em relação aos grupos *grd*04 e *grd*10-49. Isto também se justifica pelas quantidades de *backtrackings*, as quais foram exatamente iguais as ordens dos gra-

Ordem	**AIGRnn**	**pAIGRnn**	**β-AIGRnn**
25	1109,31	818,00	721,42
26	623,40	622,89	572,73
28	653,00	653,00	653,00
29	458,56	458,53	448,04
35	979,11	979,00	979,11
36	2726,38	2726,38	2726,38
40	52693,98	50345,53	20351,07
45	62999,87	62951,33	55278,44
64	43657,53	43657,53	43657,53

Tabela 6.4: Médias das quantidades de *backtrackings* realizadas pelas versões mod_{nn} nos testes com o grupo *esp*.

Ordem	**AIGRn**	**pAIGRn**	**β-AIGRn**
25	1109,31	1109,31	721,42
26	623,40	623,40	572,73
28	653,00	653,00	653,00
29	458,56	458,56	448,04
35	979,11	979,11	979,11
36	2726,38	2726,38	2726,38
40	52693,98	52693,98	20351,07
45	62999,87	62999,87	55278,44
64	43657,53	43657,53	43657,53

Tabela 6.5: Médias das quantidades de *backtrackings* realizadas pelas versões mod_n nos testes com o grupo *esp*.

fos para as versões mod_n, enquanto que foram iguais a zero para as versões mod_{nn}. O grupo *esp* novamente foi a exceção. Para este grupo todas as versões apresentaram quantidades de *backtrakings* diferindo pouco quando não iguais. Porém, as versões mod_{nn} são mais lentas que suas correspondentes mod_n. A disparidade nos tempos de execução, neste caso, é consequência da diferença entre as implementações do algoritmo *Power* que, para as versões mod_{nn}, realizam mais operações de cálculo de autocentralidades, pois o número de vértices adicionados cresce de forma quadrática à medida que as modificações são realizadas, enquanto que para as versões mod_n o número de vértices cresce de forma linear.

Por fim, foi observado nesses testes que nenhuma das versões paramétricas realizaram as $0.3n$ modificações equivalentes, pois os tempos de execução do Algoritmo 4, referentes à segunda fase da descida na árvore de busca, são todos iguais a zero.

6.2.2 Grafos isomorfos

Nesta subseção são apresentados os gráficos construídos a partir dos tempos de execução obtidos nos testes realizados com as versões mod_{nn} e mod_n do AIGR e os grupos de grafos isomorfos *igrd*04, *igrd*10-49 e *iesp*.

Grupo *igrd*04

As Figuras 6.20 e 6.21 apresentam os gráficos gerados com os tempos de execução dos testes realizados com o grupo *igrd*04 e para as versões mod_{nn} e mod_n do AIGR.

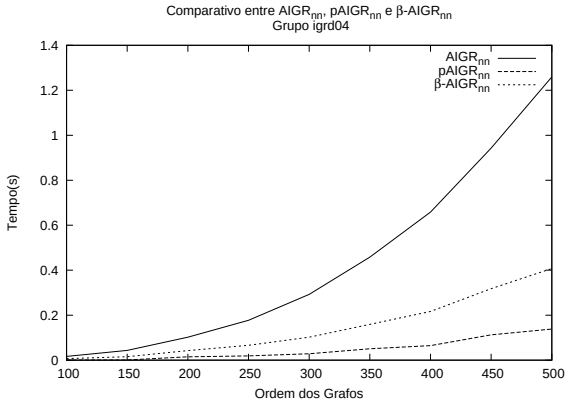


Figura 6.20: Resultados dos testes com o grupo *igrd*04 e as versões mod_{nn}.

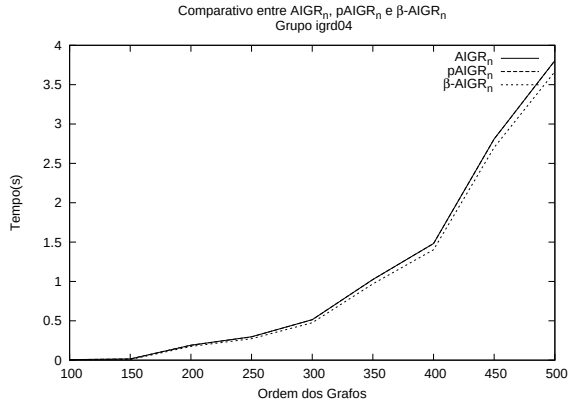


Figura 6.21: Resultados dos testes com o grupo *igrd*04 e as versões e mod_n.

Grupo *igrd*10-49

Do par formado pelas Figuras 6.22 e 6.23, ao par formado pelas Figuras 6.28 e 6.29, são apresentados os gráficos dos tempos de execução obtidos nos testes realizados com os subgrupos *igrd*10, *igrd*20, *igrd*30 e *igrd*40, respectivamente, e as versões mod_{nn} e mod_n do AIGR .

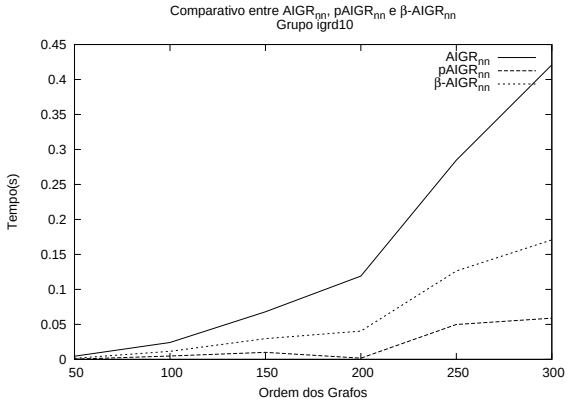


Figura 6.22: Resultados dos testes com o grupo *igrd*10 e as versões mod_{nn}.

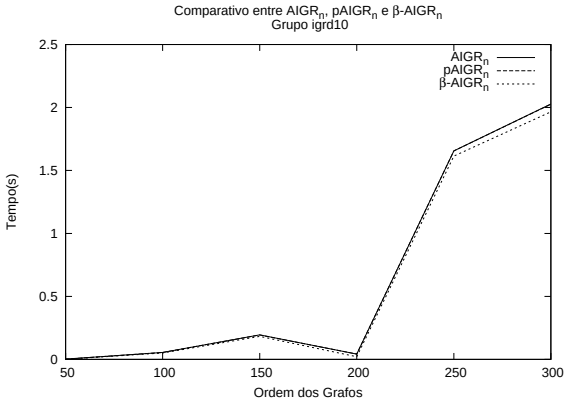


Figura 6.23: Resultados dos testes com o grupo *igrd*10 e as versões mod_{n}.

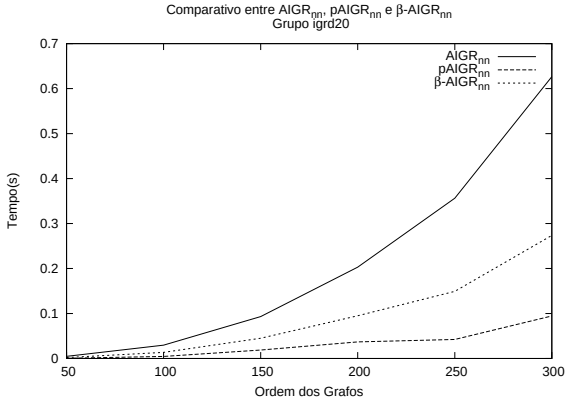


Figura 6.24: Resultados dos testes com o grupo *igrd*20 e as versões mod_{nn}.

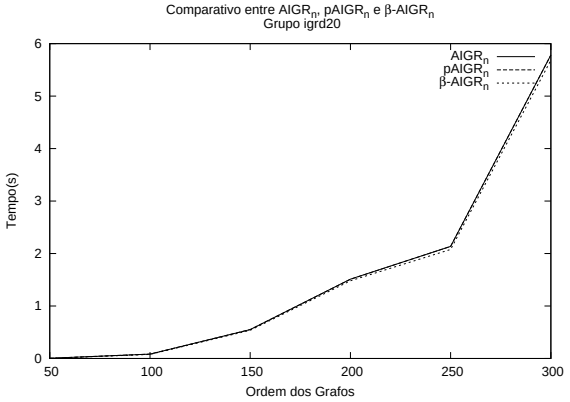


Figura 6.25: Resultados dos testes com o grupo *igrd*20 e as versões mod_{n}.

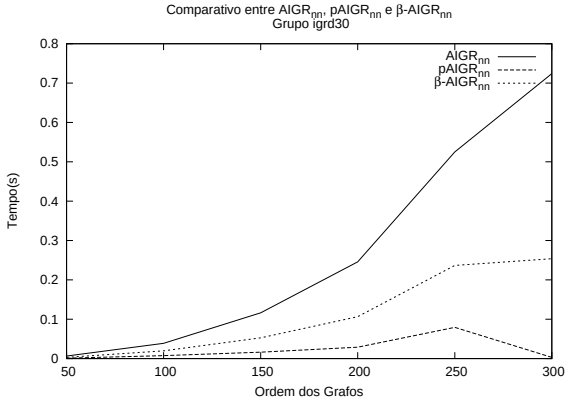


Figura 6.26: Resultados dos testes com o grupo *igrd*30 e as versões mod_{nn}.

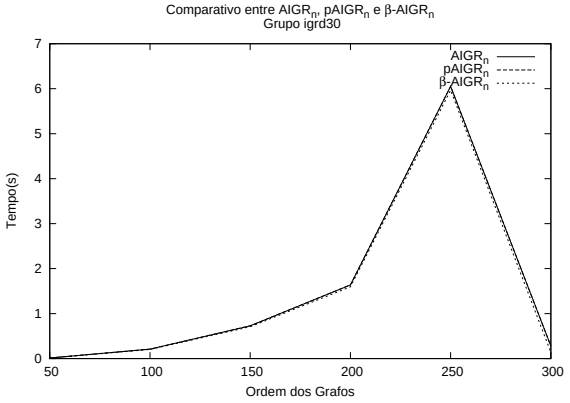


Figura 6.27: Resultados dos testes com o grupo *igrd*30 e as versões mod_{n}.

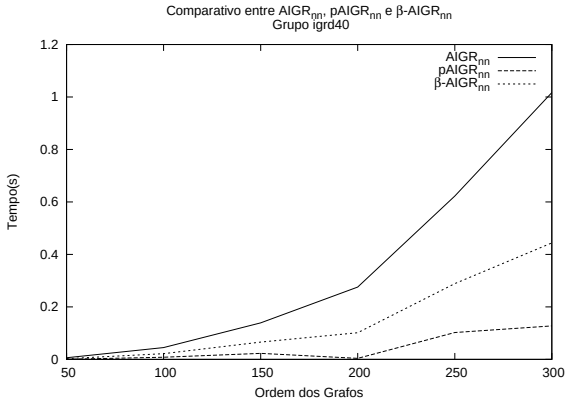


Figura 6.28: Resultados dos testes com o grupo *igrd*40 e as versões mod_{nn}.

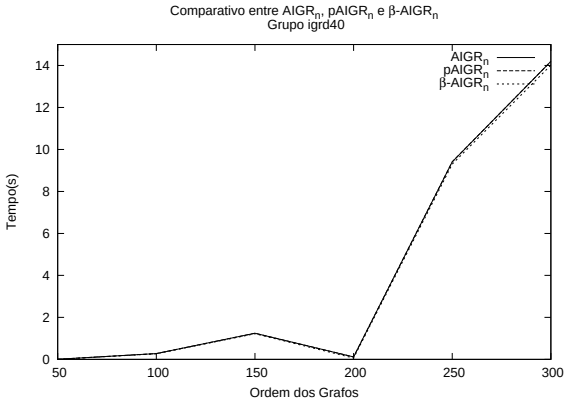


Figura 6.29: Resultados dos testes com o grupo *igrd*40 e as versões mod_{n}.

Grupo *iesp*

As Figuras 6.30 e 6.31 apresentam os gráficos dos tempos de execução obtidos nos testes computacionais realizados com o grupo *iesp* e para as versões mod_{nn} e mod_{n} do AIGR .

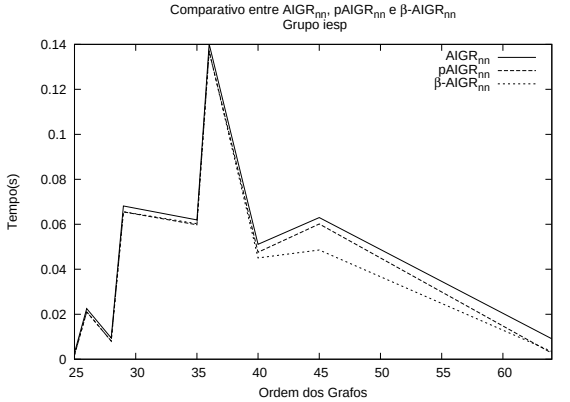


Figura 6.30: Resultados dos testes com o grupo *iesp* e as versões mod_{nn}.

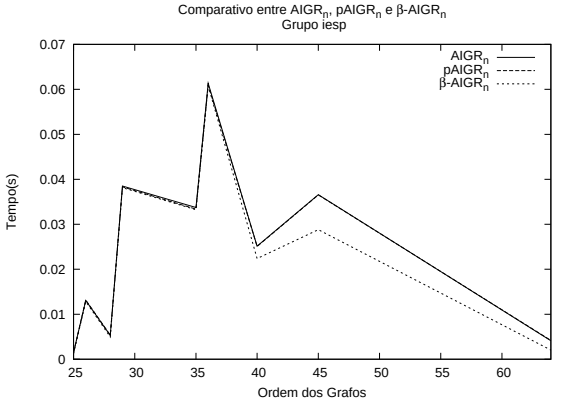


Figura 6.31: Resultados dos testes com o grupo *iesp* e as versões mod_{n}.

Análise dos resultados

Assim como nos gráficos construídos com os tempos dos testes com instâncias não isomorfas, os gráficos desta subseção referentes às versões mod_{n} apresentam curvas praticamente sobrepostas para os grupos *igrd*04 e *igrd*10-49, o que indica um comportamento muito similar entre essas versões. Por sua vez, os resultados das versões mod_{nn} apresentam comportamentos diferenciados com esses dois grupos. Está claro nos gráficos que a versão $pAIGR_{nn}$ teve o me-

lhor desempenho com instâncias formadas por grafos isomorfos gerados aleatoriamente, com a versão β-$AIGR_{nn}$ em segundo. Isto ocorreu mesmo com todas as versões mod_{nn} apresentando as mesmas quantidades de *backtrackings*, todas iguais a zero. Portanto, a única justificativa para essas diferenças está nas quantidades de modificações equivalentes realizadas por cada versão mod_{nn}. Enquanto que a versão $AIGR_{nn}$ realizou n modificações em cada grafo, a paramétrica realizou sempre $0.3n$ e a versão $pAIGR_{nn}$ apenas 1 ou 2.

Em relação aos gráficos gerados com os resultados dos testes com o grupo *iesp*, nota-se desempenho um pouco melhor das versões paramétricas. No entanto, essas versões indicaram como não isomorfos 26 pares de grafos de ordem 40 e 10 de ordem 45, exatamente as quantidades de vértices que fazem com que as curvas dessas versões fiquem um pouco mais abaixo das outras duas.

Ao comparar os resultados dos testes com os grupos *igrd*04 e *igrd*10-49 com os resultados dos testes com *grd*04 e *grd*10-49, notam-se tempos médios maiores para as versões mod_{nn} e tempos médios menores para as versões mod_n, enquanto que para os grafos fortemente regulares, todas as versões apresentaram tempos menores. Esses comportamentos podem ser explicados observando-se as quantidades de *backtrackings*. As versões mod_{nn} não realizaram *backtrackings* nos testes com os grafos gerados aleatoriamente, logo, os maiores tempos nos testes com os grupos *igrd*04 e *igrd*10-49 são devidos ao maior número de modificações equivalentes realizadas durante o processo de busca por um isomorfismo. Já as versões mod_n realizaram exatamente *n backtrackings* nos testes com as instâncias formadas por grafos dos grupos *grd*04 e *grd*10-49, enquanto que as quantidades de *backtrackings* nos testes com grafos dos grupos *igrd*04 e *igrd*10-49 foram muito inferiores as ordens dos grafos, como mostram as Tabelas 6.6 e 6.7.

Ordem	**AIGRn**	**β-AIGRn**	**pAIGRn**
100	0,00	0,00	0,00
150	0,00	0,00	0,00
200	118,69	118,69	118,69
250	109,84	109,84	109,80
300	126,24	126,24	126,24
350	180,56	180,56	180,56
400	189,11	189,11	189,11
450	276,38	276,38	276,38
500	287,53	287,53	287,53

Tabela 6.6: Médias das quantidades de *backtrackings* das versões mod_n obtidas nos testes com o grupo *igrd*04.

Em relação aos menores tempos médios de execução obtidos nos testes realizados com os

Ordem	Densidade(%)	AIGRn	β-AIGRn	pAIGRn
50	10	0,00	0,00	0,00
	20	3,31	3,31	3,13
	30	16,96	16,96	16,91
	40	0,16	0,16	0,11
100	10	60,04	60,04	59,33
	20	37,91	37,91	38,11
	30	56,80	56,80	55,49
	40	47,48	47,48	48,24
150	10	63,76	63,76	63,31
	20	93,27	93,27	92,47
	30	58,29	58,29	57,36
	40	70,07	70,07	72,24
200	10	0,11	0,11	0,36
	20	117,38	117,38	113,64
	30	65,02	65,02	67,93
	40	0,22	0,22	0,80
250	10	154,98	154,98	152,60
	20	88,22	88,22	90,53
	30	114,33	114,33	108,96
	40	122,29	122,29	123,80
300	10	151,20	151,20	151,82
	20	146,09	146,09	144,33
	30	0,13	0,13	4,80
	40	108,00	108,00	106,76

Tabela 6.7: Médias das quantidades de *backtrackings* das versões mod_n obtidas nos testes com o grupo *igrd*1049.

grafos dos grupos *iesp*, quando comparados os resultados dos testes com o grupo *esp*, tanto para as versões mod_{nn} quanto para as versões mod_n, notam-se menores quantidades de *backtrackings* realizadas por todas as verões.

Ordem	AIGRnn		pAIGRnn		β-AIGRnn	
	esp	*iesp*	*esp*	*iesp*	*esp*	*iesp*
25	1109,31	19,42	818,00	19,42	721,42	19,42
26	623,40	389,58	622,89	389,36	572,73	388,73
28	653,00	129,22	653,00	129,22	653,00	129,22
29	458,56	698,07	458,53	698,07	448,04	698,07
35	979,11	353,60	979,00	353,60	979,11	353,60
36	2726,38	901,71	2726,38	901,44	2726,38	901,71
40	52693,98	446,04	50345,53	432,24	20351,07	397,04
45	62999,87	474,29	62951,33	473,84	55278,44	381,13
64	43657,53	0,00	43657,53	0,00	43657,53	0,00

Tabela 6.8: Médias das quantidades de *backtrackings* das versões mod_{nn} obtidas nos testes com os grupos *esp* e *iesp*.

Ordem	AIGRn		pAIGRn		β-AIGRn	
	esp	*iesp*	*esp*	*iesp*	*esp*	*iesp*
25	1109,31	19,56	1109,31	19,56	721,42	19,42
26	623,40	389,78	623,40	389,78	572,73	388,73
28	653,00	129,51	653,00	129,51	653,00	129,22
29	458,56	698,27	458,56	698,27	448,04	698,07
35	979,11	353,80	979,11	353,80	979,11	353,60
36	2726,38	901,91	2726,38	901,91	2726,38	901,71
40	52693,98	446,13	52693,98	446,13	20351,07	397,04
45	62999,87	474,73	62999,87	474,73	55278,44	381,13
64	43657,53	0,24	43657,53	0,24	43657,53	0,00

Tabela 6.9: Médias das quantidades de *backtrackings* das versões mod_n obtidas nos testes com os grupos *esp* e *iesp*.

6.3 Comparação com o NAUTY

Para realizar essa comparação foram escolhidas as versões $pAIGR_{nn}$ e β-$AIGR_n$. Na seção anterior, a primeira ou foi melhor ou teve o mesmo desempenho que as demais versões mod_{nn} nos testes com quase todos os grupos de grafos, as exceções foram os resultados com os grupos *esp* e *iesp*, que tiveram a versão β-$AIGR_{nn}$ com melhor desempenho devido aos tempos apresentados com instâncias de ordens 40 e 45, sendo que com o grupo *iesp* isto aconteceu devido essas versões terem retornado não isomorfismo para várias instâncias. Por sua vez, a versão β-$AIGR_n$ teve o mesmo desempenho que as demais versões mod_n, exceto nos testes com o grupo *esp*, nos quais foi melhor. Isto indica uma pequena vantagem desse algoritmo na resolução do PIGFR, ainda que considerados os erros de detecção de isomorfismo apresentados nos testes com o grupo *iesp*, pois com instâncias desse grupos que possuem ordem 64, esse algoritmo se mostrou melhor que as demais versões mod_n.

Todos os grupos de grafos selecionados da literatura e gerados aleatoriamente foram utilizados na comparação com o *Nauty*. No entanto, os gráficos comparativos referentes aos tempos obtidos nos testes com os grupos *latim-sw* e *had-sw* não são apresentados pois nenhuma das versões do AIGR foram capazes de resolver as instâncias do PIGR formadas por grafos desses grupos.

A apresentação dos gráficos foi organizada como descrito a seguir. Primeiramente são apresentados os gráficos dos grupos formados por grafos regulares e depois os gráficos dos grupos de grafos fortemente regulares. Os gráficos gerados com instâncias isomorfas foram postos lado a lado com os gráficos gerados com instâncias não isomorfas. Da mesma forma que na Seção 6.2, o grupo *grd*10-49 é subdividido em *grd*10, *grd*20, *grd*30 e *grd*40. Isto também é feito com o grupo *igrd*10-49, que é subdividido em *igrd*10, *igrd*20, *igrd*30 e *igrd*40. Ao final

da seção é apresentada uma análise dos resultados.

Grupos *grd*04 **e** *igrd*04

As Figuras 6.32 e 6.33 apresentam os gráficos gerados com os tempos de execução dos testes realizados com os grupos *grd*04 e *igrd*04 para as versões $pAIGR_{nn}$ e β-$AIGR_n$ e o *Nauty*.

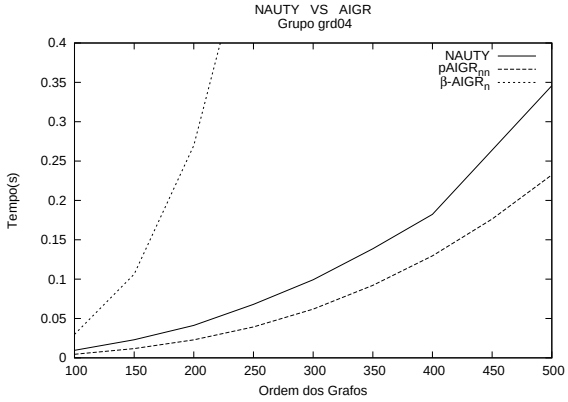


Figura 6.32: Comparativo com o *Nauty* para o grupo *grd*04.

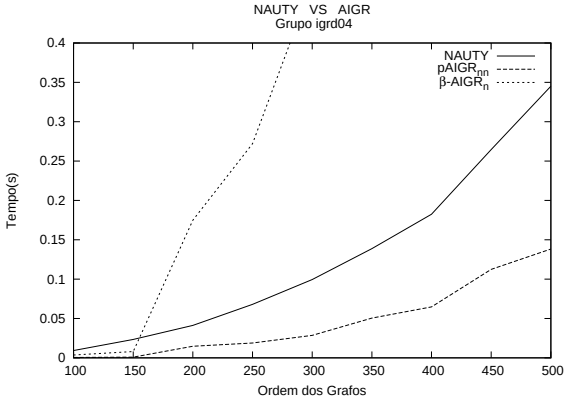


Figura 6.33: Comparativo com o *Nauty* para o grupo *igrd*04.

Grupos *grd*10-49 **e** *igrd*10-49

Do par formado pelas Figuras 6.34 e 6.35, ao par formado pelas Figuras 6.40 e 6.41, são apresentados os gráficos gerados com os tempos de execução dos testes realizados com os respectivos pares de grupos *grd*10 e *igrd*10, *grd*20 e *igrd*20, *grd*30 e *igrd*30, *grd*40 e *igrd*40, e para as versões $pAIGR_{nn}$ e β-$AIGR_n$ e o *Nauty*.

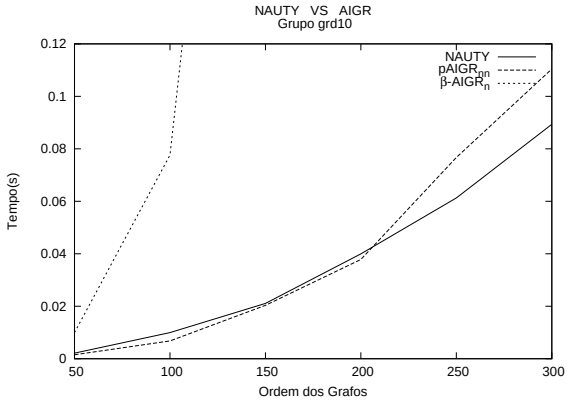


Figura 6.34: Comparativo com o *Nauty* para o grupo *grd*10.

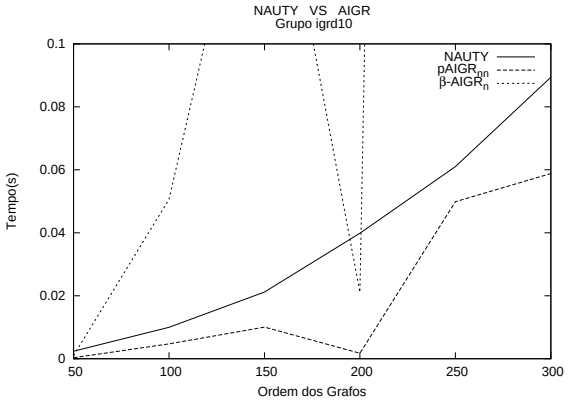


Figura 6.35: Comparativo com o *Nauty* para o grupo *igrd*10.

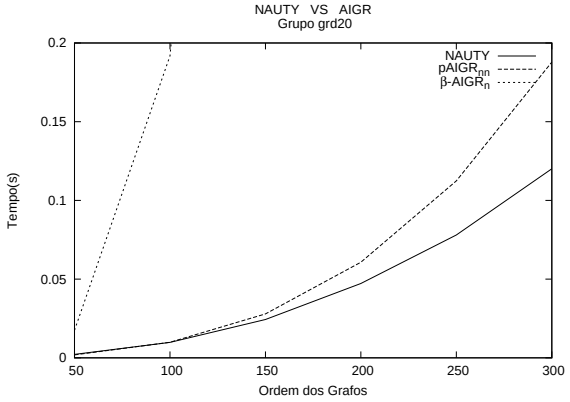


Figura 6.36: Comparativo com o *Nauty* para o grupo *grd*20.

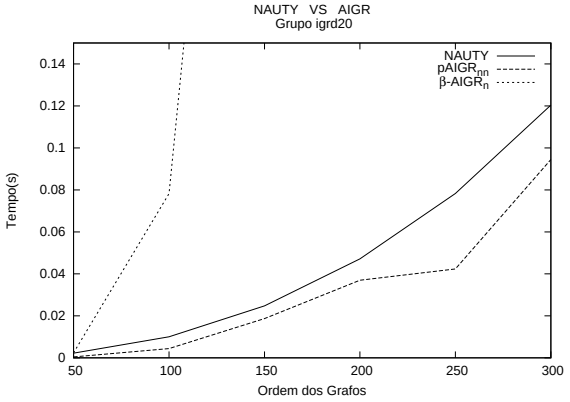


Figura 6.37: Comparativo com o *Nauty* para o grupo *igrd*20.

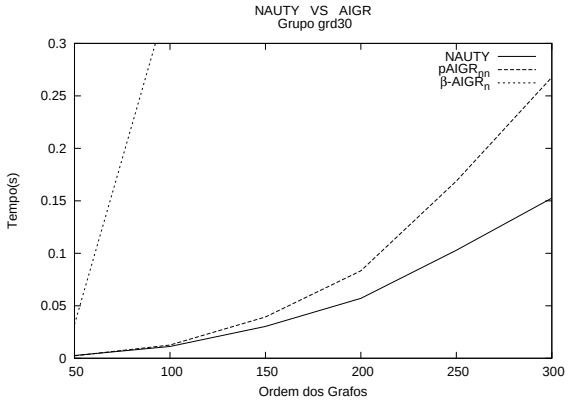


Figura 6.38: Comparativo com o *Nauty* para o grupo *grd*30.

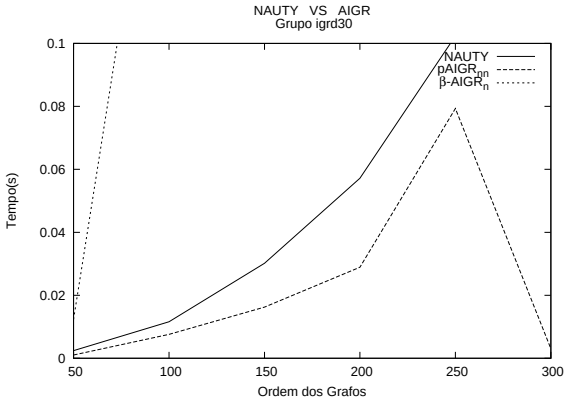


Figura 6.39: Comparativo com o *Nauty* para o grupo *igrd*30.

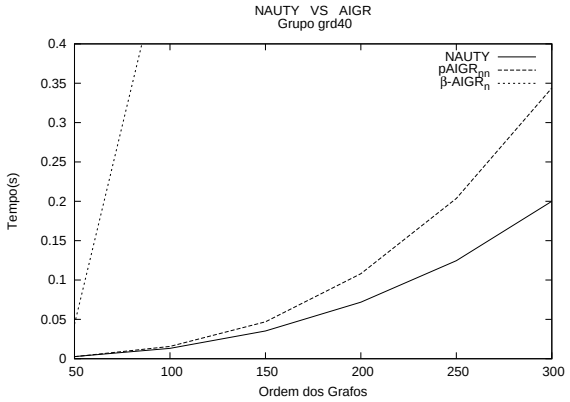


Figura 6.40: Comparativo com o *Nauty* para o grupo *grd*40.

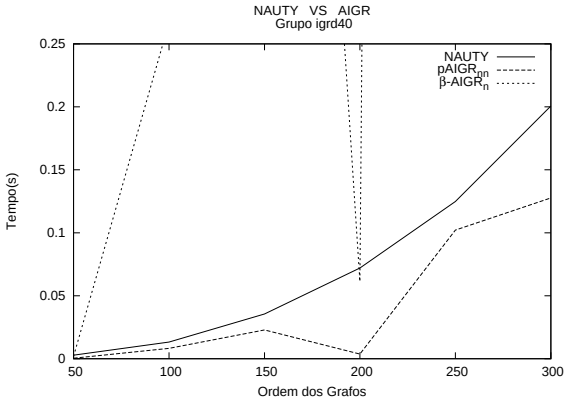


Figura 6.41: Comparativo com o *Nauty* para o grupo *igrd*40.

Grupos *rnd*-3-*reg* e *irnd*-3-*reg*

As Figuras 6.42 e 6.43 apresentam os gráficos gerados com os tempos de execução dos testes realizados com os grupos *rnd*-3-*reg* e *irnd*-3-*reg*, as versões $AIGR_{nn}$ e β-$pAIGR_n$ e o *Nauty*.

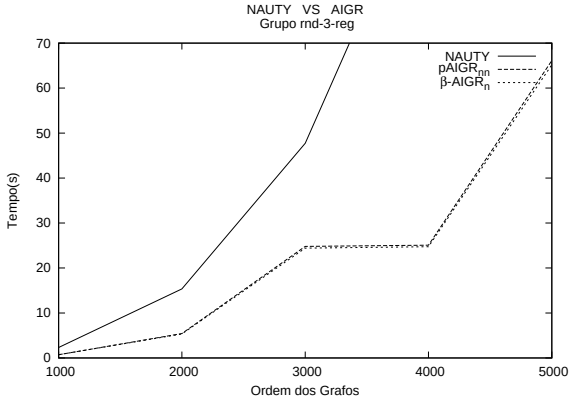


Figura 6.42: Comparativo com o *Nauty* para o grupo *rnd*3.

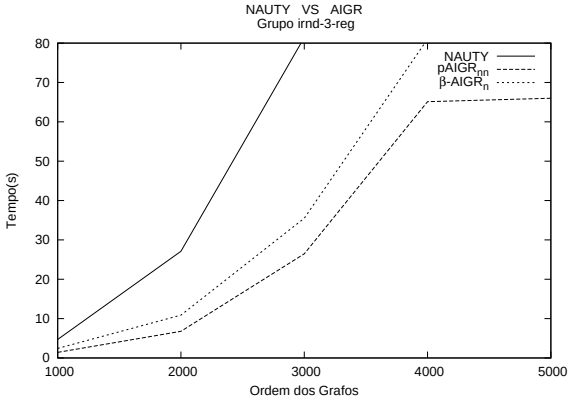


Figura 6.43: Comparativo com o *Nauty* para o grupo *irnd*3.

Grupos *esp* e *iesp*

As Figuras 6.44 e 6.45 apresentam os gráficos gerados com os tempos de execução dos testes realizados com os grupos *esp* e *iesp* para as versões $pAIGR_{nn}$ e β-$pAIGR_n$ e o *Nauty*.

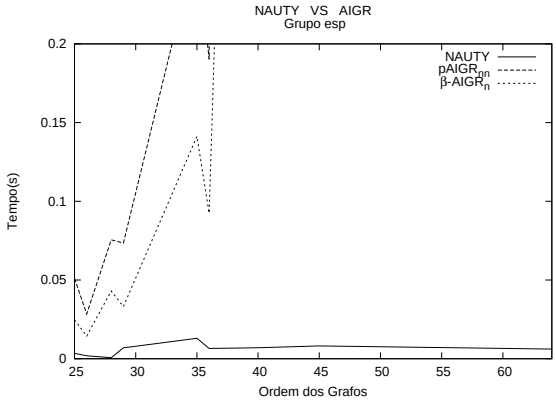


Figura 6.44: Comparativo com o *Nauty* para o grupo *esp*.

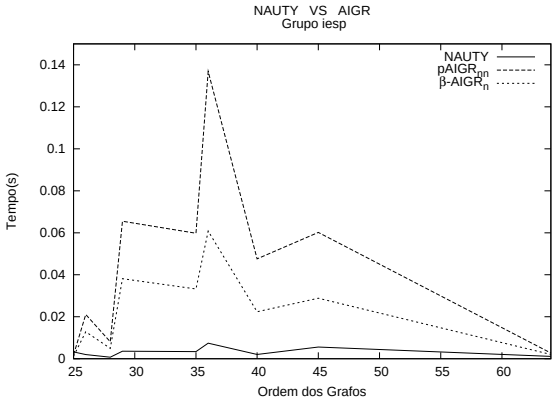


Figura 6.45: Comparativo com o *Nauty* para o grupo *iesp*.

Grupos *sts-sw* e *ists-sw*

As Figuras 6.46 e 6.47 apresentam os gráficos gerados com os tempos de execução dos testes realizados com os grupos *sts-sw* e *ists-sw* para as versões $pAIGR_{nn}$ e β-$AIGR_n$ e o *Nauty*.

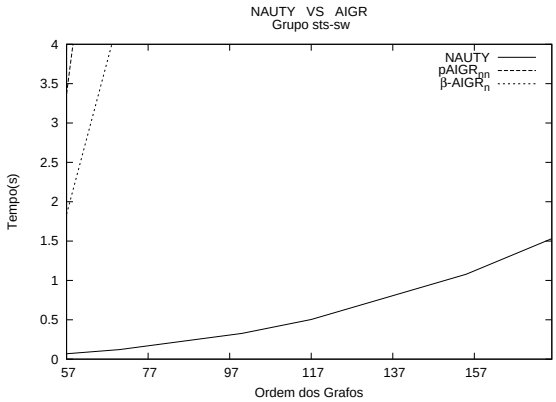


Figura 6.46: Comparativo com o *Nauty* para o grupo *stssw*.

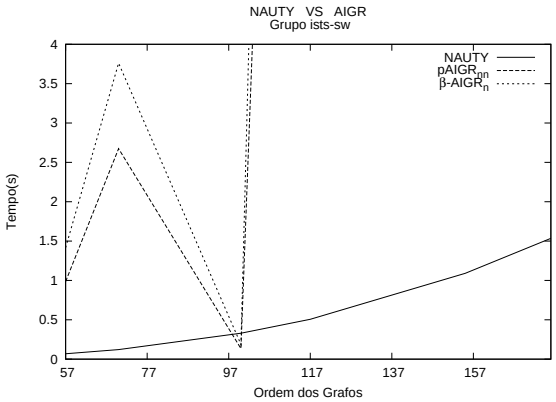


Figura 6.47: Comparativo com o *Nauty* para o grupo *istssw*.

Análise dos resultados

Para analisar os resultados, os grupos de grafos regulares usados nos testes são classificados em três conjuntos: grafos regulares não esparsos, grafos regulares esparsos e grafos fortemente regulares não esparsos. O primeiro conjunto é composto pelos grupos *grd*04, *igrd*04, *grd*10-49 e *igrd*10-49. O segundo conjunto é composto pelos grupos *rnd*-3-*reg* e *irnd*-3-*reg*. Por sua vez, o terceiro conjunto é composto pelos grupos *esp*, *iesp*, *sts-sw* e *ists-sw*.

Com o grupo de grafos regulares não esparsos e não isomorfos o *Nauty* se apresentou melhor em quase todos os gráficos, sendo o $pAIGR_{nn}$ o segundo melhor. A exceção é o gráfico da Figura 6.32, que apresenta o $pAIGR_{nn}$ com melhor desempenho e depois o *Nauty*. Com grafos não esparsos isomorfos, o $pAIGR_{nn}$ foi o melhor em todos os gráficos, sendo o *Nauty* o segundo.

Para os grupos de grafos regulares esparsos, as duas versões do AIGR apresentaram melhor desempenho, sendo que os resultados dos testes da versão $pAIGR_{nn}$, com instâncias formadas pelos grafos do grupo *irnd*-3-*reg*, foram melhores que os resultados do β-$AIGR_n$. Nos testes com instâncias formadas pelos grafos do grupo *rnd*-3-*reg*, as versões do AIGR apresentaram o mesmo desempenho.

Nas curvas referentes aos testes com grafos fortemente regulares, observa-se o *Nauty* muito melhor que as versões do AIGR que, dentre elas, aquela que apresentou melhor desempenho foi a versão β-$AIGR_n$. A exceção a este comportamento das versões do AIGR foi com o grupo *ists-sw*, com o qual é possível perceber que o $pAIGR_{nn}$ teve melhor desempenho que o β-$AIGR_n$ com instâncias de ordens inferiores a 97.

Outro fator observado está relacionado aos valores de desvios padrões dos tempos médios de execução. O *Nauty* apresenta baixos valores de desvios padrões, indicando que trata-se de um algoritmo consistente. Por sua vez, as versões do AIGR comparadas ao *Nauty*, apresentam valores altos de desvios padrões, principalmente em relação aos tempos médios de execução obtidos com os grupos *rnd*-3-*reg*, *irnd*-3-*reg* e os grupos de grafos fortemente regulares, cujas tabelas com os desvios padrões podem ser vistas no Apêndice B.

Por fim, foi observado que para todas as instâncias formadas por grafos fortemente regulares do grupo *ists-sw* o processo de resolução com o β-$AIGR_n$ retornou um isomorfismo. Além disso, esta versão do AIGR não realizou as $0.3n$ modificações equivalentes nos testes com os grupos *rnd*-3-*reg*, *irnd*-3-*reg*, *sts-sw* e *ists-sw*.

7 Conclusão

Neste trabalho foi proposto um algoritmo de abordagem direta para a resolução do Problema de Isomorfismo de Grafos Regulares (PIGR), denominado AIGR. Ao comparar dois grafos regulares, este algoritmo realiza modificações equivalentes em ambos os grafos, fazendo uso de um conceito proveniente de Teoria Espectral de Grafos (TEG), a autocentralidade, para guiar essas modificações e restringir possibilidades de mapeamento.

Durante o desenvolvimento deste trabalho foram propostas algumas variações do AIGR que, de acordo com os métodos de modificação de grafos propostos, o mod_{nn} e o mod_n, resultaram em seis algoritmos: $AIGR_{nn}$, $AIGR_n$, $pAIGR_{nn}$, $pAIGR_n$, β-$AIGR_{nn}$ e β-$AIGR_n$. Com as versões $AIGR_{nn}$ e $AIGR_n$, foram realizados testes para definir quais dos métodos de obtenção de autocentralidades seria utilizado para avaliar todos os algoritmos, a função *_dssyevr* ou uma adaptação do método proposto por Baroni, Boeres e Rangel (2011), que neste trabalho foi denominada *Power*. Selecionado o método de obtenção de autocentralidades, esses algoritmos foram testados com instâncias formadas por grafos regulares de diferentes ordens e densidades. Após esses testes, dois desses algoritmos foram selecionados para comparação com uma das melhores ferramentas de resolução do Problema de Isomorfismo de Grafos no caso geral, o *Nauty*.

Nos resultados dos testes com os métodos de obtenção de autocentralidades, as versões $AIGR_{nn}$ e $AIGR_n$ implementadas com o *Power*, obtiveram desempenho muito melhor que as implementações dessas versões com a função *_dssyevr*. Nas comparações entre as versões do AIGR implementadas com o *Power*, o algoritmo $pAIGR_{nn}$ foi o que obteve os melhores resultados dentre os três que usam o método de modificação mod_{nn}, enquanto que as versões que usam o método de modificação mod_n apresentaram resultados praticamente iguais.

Na comparação realizada com o *Nauty*, em relação às médias dos tempos de execução, foi observado que este algoritmo apresentou melhor desempenho nos testes com os grafos regulares não isomorfos dos grupos *grd*04 e *grd*10-49, enquanto que o $pAIGR_{nn}$ foi melhor nos testes com as instâncias formadas por grafos isomorfos dos grupos *igrd*04 e *igrd*10-49, sendo o β-*AIGR*

aquele que teve o pior desempenho para esses grupos de grafos. Com os grupos *rnd*-3-*reg* e *irnd*-3-*reg*, as versões do *AIGR* apresentaram resultados melhores que o *Nauty*, já com os grupos de grafos fortemente regulares o *Nauty* apresentou resultados muito melhores que as versões do AIGR. Em relação a consistência dos algoritmos, o *Nauty* apresentou tempos de execução com baixas variações nos testes com instâncias de mesmas características, o que não aconteceu com as versões do AIGR, que apresentaram desvios padrões altos para os grupos de grafos fortemente regulares e os grupos *rnd*-3-*reg* e *irnd*-3-*reg*.

Nos resultados dos testes com instâncias formadas por grafos do grupo *grd*10-49 (Seção 6.2), é possível perceber que os tempos de execução apresentados pelas versões do AIGR aumentam com a densidade dos grafos. Isto deixa claro que o AIGR tem seu desempenho influenciado pela densidade das instâncias formadas por grafos não isomorfos.

Por todos os resultados observados, pode-se concluir que o AIGR é um algoritmo eficiente para a resolução do PIGR com instâncias formadas por grafos regulares não esparsos gerados aleatoriamente. Porém, demonstrou que precisa de estratégias mais eficientes para lidar com instâncias formadas por grafos fortemente regulares e para apresentar resultados mais consistentes.

Para trabalhos futuros, serão estudadas melhores estratégias de obtenção das autocentralidades. Durante os testes foi observado que o critério de parada do método proposto por Baroni, Boeres e Rangel (2011) não é o mais adequado. Existem situações em que a execução de algumas iterações após o critério de parada ser atingido, resultam em valores mais adequados de autocentralidades, isto é, valores que representam melhor o quão central é um vértice de acordo com esse conceito. Outra proposta de trabalho futuro é a incorporação de um método de refinamento ao AIGR, objetivando a utilização de informações da vizinhança dos vértices para melhor distingui-los. Outro trabalho previsto está relacionado a uma evolução dos algoritmos paramétricos. Quanto mais essas versões do AIGR diferenciarem os conjuntos de vértices dos grafos, mais eficiente será o algoritmo executado na segunda fase de descida na árvore de busca. No entanto, o ideal é dividir o trabalho de detecção de isomorfismo com o algoritmo da segunda fase de busca, de forma que cada algoritmo contribua com 50% dos tempos totais de execução. Então, em vez de usar apenas o parâmetro β para determinar o momento de interromper esses algoritmos, fazer uso também dos tamanhos das partiçoes geradas com os vetores de autocentralidades.

Bibliografia

ABDULRAHIM, M. A.; MISRA, M. A graph isomorphism algorithm for object recognition. *Pattern Analysis and Applications,*, Springer London, v. 1, p. 189–201, 1998. ISSN 1433-7541.

ABREU, N. M. M.; DEL-VECCHIO, R. R.; VINAGRE, C. T. M.; STEVANOVIć, D. In: *Introdução à Teoria Espectral de Grafos com Aplicações*. [S.l.]: Sociedade Brasileira de Matemática Aplicada e Computacional, 2007, (Notas em Matemática Aplicada).

BABAI, L.; GRIGORYEV, D. Y.; MOUNT, D. M. Isomorphism of graphs with bounded eigenvalue multiplicity. In: *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1982. p. 310–324. ISBN 0-89791-070-2.

BARONI, M. V.; BOERES, M. C.; RANGEL, M. C. An adapted power method for eigenvector computing applied to a graph isomorphism algorithm. In: *aceito para XLIII Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2011.

CORDELLA, L. P.; FOGGIA, P.; SANSONE, C.; VENTO, M. Performance evaluating of the vf graph matching algorithm. In: *Proc. of the 10th International Conference on Image Analysis and Processing*. [S.l.]: IEEE Computer Society Press, 1999. p. 1172–1177.

CORDELLA, L. P.; FOGGIA, P.; SANSONE, C.; VENTO, M. Fast graph matching for detecting cad image components. In: *Proc. of the 15th International Conference on Pattern Recognition*. [S.l.]: IEEE Computer Society Press, 2000. v. 2, p. 1038–1041.

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*. [S.l.: s.n.], 2001. p. 149–159.

CORNIEL, D. G.; GOTLIEB, C. C. An efficient algorithm for graph isomorphism. *J. ACM,*, ACM, New York, NY, USA, v. 17, p. 51–64, January 1970. ISSN 0004-5411.

DIESTEL, R. *Graph Theory (3º Ed.)*. [S.l.]: Springer, 2006. I-XVI, 1-344 p. ISBN 978-3-540-21391-8.

FERREIRA, A.; PEREIRA, A. S.; CARREIRA, T. G. M. *Proposta de Implementação de Sistema de Segurança Utilizando Sistemas Biométricos*. 2001.

GRASSI, R.; STEFANI, S.; TORRIERO, A. Some new results on the eigenvector centrality. *The Journal of Mathematical Sociology,*, v. 31, p. 237–248, 2007.

HARTKE, S. G.; RADCLIFFE, A. J. Mckay's canonical graph labeling algorithm. In: CHOW, T. Y.; ISAKSEN, D. C. (Ed.). *Communicating Mathematics: In Honor of Joseph A. Gallian?s 65th Birthday*. [S.l.]: AMS Contemporary Mathematics book series, 2009. p. 99–111.

HE, P. R.; ZHANG, W. J.; LI, Q. Some further development on the eigensystem approach for graph isomorphism detection. *Journal of the Franklin Institute,*, v. 342, n. 6, p. 657–673, 2005. ISSN 0016-0032. The 2004 Franklin Institute Awards.

HE, P. R.; ZHANG, W. J.; LI, Q.; WU, F. X. A new method for detection of graph isomorphism based on the quadratic form. *Journal of Mechanical Design*, ASME, v. 125, n. 3, p. 640–642, 2003. Disponível em: <http://link.aip.org/link/?JMD/125/640/1>.

JUNTTILA, T.; KASKI, P. Engineering an efficient canonical labeling tool for large and sparse graphs. In: APPLEGATE, D.; BRODAL, G. S.; PANARIO, D.; SEDGEWICK, R. (Ed.). *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*. [S.l.]: SIAM, 2007. p. 135–149.

JUNTTILA, T. A.; KASKI, P. *Benchmark graphs for evaluating graph automorphism and canonical labeling algorithms*. 2011. Last visited: 13/06/2011. Disponível em: <http://www.maths.gla.ac.uk/~es/srgraphs.html>.

KARP, R. M. Reducibility among combinatorial problems. In: *Complexity of computer computations*. New York, USA: Plenum Press, 1972. p. 85–103.

LEIGHTON, F. T.; MILLER, G. Certificates for graphs with distinct eigenvalues. Documento Digital Manuscrito Original. 1979.

LUCHI, D.; RANGEL, M. C. *Geração Aleatória de Grafos Regulares*. Vitória, ES, Brasil, 2010.

LUKS, E. M. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Science,*, v. 25, n. 1, p. 42–65, 1982.

MCKAY, B. D. Practical graph isomorphism. In: *Congressus Numerantium*. [S.l.: s.n.], 1981. v. 30, p. 45–87.

MCKAY, B. D. *nauty User's Guide (Version 2.4)*. [S.l.], 2009.

MILLER, G. Graph isomorphism, general remarks. *Journal or Computer and Systems Sciences*, v. 18, n. 2, p. 128–142, 1979.

MILLER, G. *Gary L. Miller's Publications*. 2011.

MIYAZAKI, T. The complexity of mckay's canonical labeling algorithm. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. [S.l.: s.n.], 1996.

NANDI, R. C. *Isomorfismo de Grafos Aplicado à Comparação de Impressões Digitais*. Dissertação (Programa de Pós-Graduação em Informática) — Universidade Federal do Paraná, Curitiba, PR, Brasil, 2006.

OLIVEIRA, M. O.; GREVE, F. G. P. Um novo algoritmo de refinamento para testes de isomorfismo em grafos. *Revista Eletrônica de Iniciação Científica (REIC),*, v. 5, n. 3, p. 140–148, junho 2005.

PORUMBEL, D. C. A polynomial graph extension procedure for improving graph isomorphism algorithms. *Computing Research Repository,*, abs/0903.0136, 2009.

PRESA, J. L. L.; ANTA, A. F. Fast algorithm for graph isomorphism testing. In: *Proceedings of the 8th International Symposium on Experimental Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2009. (SEA '09), p. 221–232. ISBN 978-3-642-02010-0.

RAJASEKARAN, S.; KUNDETI, V. Spectrum based techniques for graph isomorphism. *International Journal of Foundations of Computer Science,*, v. 20, n. 3, p. 479–499, 2009.

READ, R. C.; CORNEIL, D. G. The graph isomorphism disease. *Journal of Graph Theory*, Wiley Subscription Services, Inc., A Wiley Company, v. 1, n. 4, p. 339–363, 1977. ISSN 1097-0118. Disponível em: <http://dx.doi.org/10.1002/jgt.3190010410>.

SAAD, Y. *Iterative Methods for Sparse Linear Systems*. 2nd. ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003. ISBN 0898715342.

SANTOS, P. L. F. *Teoria Espectral de Grafos Aplicada ao Problema de Isomorfismo de Grafos*. Dissertação (Programa de Pós-Graduação em Informática) — Universidade Federal do Espírito Santo, Vitória, ES, Brasil, 2010.

SANTOS, P. L. F.; RANGEL, M. C.; BOERES, M. C. S. Teoria espectral de grafos aplicada ao problema de isomorfismo de grafos. In: *XLII Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2010. v. 1, p. 1–12.

SORLIN, S.; SOLNON, C. A parametric filtering algorithm for the graph isomorphism problem. *Constraints,*, Springer Netherlands, v. 13, p. 518–537, 2008. ISSN 1383-7133.

SPENCE, E. *Strongly Regular Graphs on at most 64 vertices*. 2011. Last visited: 13/06/2001. Disponível em: <http://www.maths.gla.ac.uk/~es/srgraphs.html>.

SPIELMAN, D. A. Faster isomorphism testing of strongly regular graphs. In: *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1996. p. 576–584.

APÊNDICE A – Algoritmos

Neste apêndice são apresentados o algoritmo referente a fase 2 do processo de resolução do PIGR descrito na Seção 5.2, o algoritmo de rerrotulação de vértices usado para gerar os grupos de grafos isomorfos, e a adaptação do método de obtenção das autocentralidades proposto por Baroni, Boeres e Rangel (2011).

Considere G_1^β e G_2^β, $q = 0.3n$, grafos resultantes das β modificações equivalentes realizadas por uma das versões paramétricas do AIGR sobre G_1 e G_2, grafos comparados. Considere $ac(v)$ uma função que retorna a autocentralidade do vértice v. O Algoritmo 4 apresenta o pseudocódigo de um algoritmo de busca em profundidade com *backtracking*, que tenta mapear $V(G_1)$ em $V(G_2)$, partindo do mapeamento parcial feito pela versão paramétrica do AIGR, sem que as condições de isomorfismo sejam violadas.

Algoritmo 4: *Algoritmo Fase2.*

Dados: [*Grafos* G_1, G_2, G_1^β *e* G_2^β *e o mapeamento parcial feito pelo* β*-AIGR*]

```
1  Passo 1 :
2      Se (∃v ∈ G1 ⊂ G1^n ainda não mapeado) então
3          Selecione v;
4      Senão Retorne o mapeamento encontrado.;
5  Passo 2 :
6      Para cada (u ∈ G2 ⊂ G2^n ainda não mapeado com v para o mapeamento corrente)
       faça
7          Se (ac(u) = ac(v)), então
8              Se (mapear u e v não invalidar mapeamentos anteriores) então
9                  Mapeie u e v e vá ao Passo 1;
10 Passo 3 (Backtracking) :
11     Se (for possível realizar backtracking) então
12         Backtracking( );
13     Senão Os grafos não são isomorfos;
```

O Algoritmo 5 tem como parâmetro a matriz de adjacência $A(G)$ de um grafo G e sua ordem. Ele realiza $2n$ trocas de linhas e colunas alterando a matriz recebida como parâmetro.

Algoritmo 5: *Algoritmo de Rerrotulação.*

Dados: $[Matriz\ A(G)\ sua\ ordem\ n]$

$z \leftarrow 0$; **Enquanto** $(z < 2 * n)$ **faça**
  $num \leftarrow NumeroAleatorio(\ )$;
  $i \leftarrow Resto(num, n)$;
  $num \leftarrow NumeroAleatorio(\ )$;
  Enquanto $[(j \leftarrow Resto(num, n)) = i]$ *]* **faça**
    $num \leftarrow NumeroAleatorio(\ )$;
    $j \leftarrow Resto(num, n)$;
  Troque as linhas i e j de A(G);
  Troque as colunas i e j de A(G);
  $z \leftarrow z + 1$;

A seguir é apresentado o pseudocódigo da adaptação do algoritmo proposto por Baroni, Boeres e Rangel (2011) para as versões do AIGR que utilizam o método de modificação mod_n. Seja L a lista de adjacência de um grafo G k-regilar de ordem n, v_0 um vetor não nulo e μ a quantidade de modificações realizadas pelo AIGR sobre G. O Algoritmo 6 calcula as autocentralidades de G^{μ}.

Algoritmo 6: *Algoritmo de Power adaptado ao método de modificação mod_n.*

Dados: [L, v_0, n, μ]

Enquanto $|\pi(v_1)| > |\pi(v_0)|$ *e* $|\pi(v_1)| < n$, **faça**
  Passo *1*:
    $aux \leftarrow l \leftarrow 0$;
    $\vec{v}_1^{z} = \sum_j \vec{v}_0^{j}$, $1 \leq j \neq z \leq n$, $grau(z) > k$ e j adjacente a z;
    $aux \leftarrow aux + v_0^{\omega}$, ω é o vértice adicionado na modificação do grau de z;
    $\vec{v}_1^{z} += aux$;
    $l \leftarrow l + 1$;
  Passo *2*:
    $\vec{v}_1^{z} = \sum_j \vec{v}_0^{j}$, $1 \leq j \neq z \leq n$, $grau(z) = k$ e j adjacente a z;
  Passo *3*:
    $aux \leftarrow 0$; **Para** $l = \mu 1$, **faça**
      $\vec{v}_1^{l+n} \leftarrow v_0^{\omega} + aux$, ω é o l-ésimo vértice modificado;
      $aux \leftarrow v_0^{l+n}$;
      $l \leftarrow l - 1$;
  $\vec{v}_0 \leftarrow \vec{v}_1$;

APÊNDICE B – Tabelas dos Resultados Computacionais - Seção da 6.3

Neste capítulo são apresentadas as tabelas com as médias dos tempos de execução e desvios padrões referentes aos testes realizados com os grupos *rnd*-3-*reg*, *irnd*-3-*reg*, *esp*, *iesp*, *sts*-*sw* e *ists*-*sw*, para as versões $pAIGR_{nn}$ e β-$AIGR_n$ e o *Nauty*. Em cada tabela, a coluna σ apresenta os valores de desvios padrões.

Grupos *rnd*-3-*reg* e *irnd*-3-*reg*

As Tabelas B.1 e B.2 apresentam os resultados obtidos nos testes realizados com os grupos *rnd*-3-*reg* e *irnd*-3-*reg* para as versões $pAIGR_{nn}$ e β-$AIGR_n$ e o *Nauty*..

	pAIGRnn		**β-AIGRn**		**Nauty**	
Ordem	**Média**	**σ**	**Média**	**σ**	**Média**	**σ**
1000	0,731668	0,423961	0,721407	0,414555	2,317999	0,018418
2000	5,418519	4,743493	5,334830	4,656415	15,344949	0,062568
3000	24,798643	15,759178	24,419095	15,486477	47,727888	0,09505
4000	25,063763	30,181311	24,724867	29,682601	109,820805	0,641958
5000	66,103348	59,770003	65,173537	58,828266	208,910138	0,786041

Tabela B.1: Médias dos tempos de execução e desvios padrões obtidos nos testes com o grupo *rnd*-3-*reg*.

	pAIGRnn		**β-AIGRn**		**Nauty**	
Ordem	**Média**	**σ**	**Média**	**σ**	**Média**	**σ**
1000	0,624628	0,322221	1,272246	0,316885	2,330706	0,017228
2000	3,718996	2,890333	6,469542	2,841718	15,364188	0,063478
3000	11,02987	11,662763	17,446522	11,479385	47,645993	0,075724
4000	40,353547	24,708608	51,639554	24,337249	109,592702	0,691746
5000	73,524328	51,118863	91,210775	50,288367	208,845792	0,64319

Tabela B.2: Médias dos tempos de execução e desvios padrões obtidos nos testes com o grupo *irnd*-3-*reg*.

Grupos *esp* e *iesp*

As Tabelas B.3 e B.4 apresentam os resultados obtidos nos testes realizados com os grupos *esp* e *iesp* para as versões $pAIGR_{nn}$ e β-$AIGR_n$ e o *Nauty*.

	pAIGRnn		β-AIGRn		Nauty	
Ordem	**Média**	σ	**Média**	σ	**Média**	σ
25	0,051117	0,005825	0,024765	0,019502	0,003404	0,001262
26	0,028435	0,009077	0,014396	0,006131	0,001880	0,000840
28	0,075515	0,035044	0,043030	0,023284	0,000623	0,000224
29	0,073485	0,000777	0,033164	0,005075	0,006942	0,001373
35	0,264346	0,074468	0,141271	0,040930	0,012999	0,003350
36	0,190224	0,015452	0,092418	0,009192	0,006548	0,001616
40	3,758341	3,852912	1,085705	2,320956	0,006998	0,003219
45	5,428484	12,549249	2,913840	7,826347	0,008126	0,003566
64	11,372736	18,406308	6,709360	11,357784	0,006148	0,005696

Tabela B.3: Médias dos tempos de execução e desvios padrões obtidos nos testes com o grupo *esp*.

	pAIGRnn		β-AIGRn		Nauty	
Ordem	**Média**	σ	**Média**	σ	**Média**	σ
25	0,001841	0,000967	0,001224	0,000612	0,003255	0,000639
26	0,021063	0,022513	0,012846	0,013549	0,001968	0,000185
28	0,008100	0,007189	0,004951	0,004424	0,000742	0,000110
29	0,065521	0,050518	0,038146	0,029754	0,003578	0,000306
35	0,059780	0,062792	0,033237	0,034657	0,003401	0,000955
36	0,136875	0,082799	0,060794	0,035814	0,007411	0,001959
40	0,047554	0,042540	0,022383	0,020967	0,002032	0,000453
45	0,060153	0,049928	0,028810	0,028005	0,005572	0,003078
64	0,002696	0,000271	0,002031	0,000636	0,001111	0,000278

Tabela B.4: Médias dos tempos de execução e desvios padrões obtidos nos testes com o grupo *iesp*.

Grupos *sts-sw* e *ists-sw*

As Tabelas B.5 e B.6 apresentam os resultados obtidos nos testes realizados com os grupos *sts-sw* e *ists-sw* para as versões $pAIGR_{nn}$ e β-$AIGR_n$ e o *Nauty*.

	pAIGRnn		**β-AIGRn**		**Nauty**	
Ordem	**Média**	σ	**Média**	σ	**Média**	σ
57	3,369805	0,858213	1,840930	0,490628	0,069105	0,002961
70	8,614213	2,339985	4,370470	1,215319	0,121835	0,003699
100	58,462888	13,757434	28,734743	6,925653	0,326504	0,008996
117	129,063228	25,128355	61,594605	11,993355	0,504115	0,014722
155	639,341281	270,023589	271,267468	131,289645	1,078737	0,022272
176	1048,803231	366,222342	467,086242	163,297129	1,531307	0,370083

Tabela B.5: Médias dos tempos de execução e desvios padrões obtidos nos testes com o grupo *sts-sw*.

	pAIGRnn		**β-AIGRn**		**Nauty**	
Ordem	**Média**	σ	**Média**	σ	**Média**	σ
57	0,985807	0,659180	1,415824	0,954709	0,069082	0,002851
70	2,674836	1,868566	3,761282	2,635553	0,121427	0,004138
100	0,133964	0,108794	0,187761	0,156748	0,329238	0,009409
117	23,614189	29,423962	33,593028	41,847502	0,506460	0,013412
155	443,415192	376,489673	212,967179	181,048204	1,090191	0,024744
176	859,382020	691,887647	426,532212	374,898257	1,535441	0,034268

Tabela B.6: Médias dos tempos de execução e desvios padrões obtidos nos testes com o grupo *ists-sw*.