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

PHILIPPE LEAL FREIRE DOS SANTOS

TEORIA ESPECTRAL DE GRAFOS APLICADA AO PROBLEMA DE ISOMORFISMO DE GRAFOS

VITÓRIA, 2010

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

TEORIA ESPECTRAL DE GRAFOS APLICADA AO PROBLEMA DE ISOMORFISMO DE GRAFOS

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.

Philippe Leal Freire dos Santos
Orientadora: Profa. Dra. Maria Cristina Rangel
Co-orientadora: Profa. Dra. Maria Claudia Silva Boeres

VITÓRIA, 2010

TEORIA ESPECTRAL DE GRAFOS APLICADA AO PROBLEMA DE ISOMORFISMO DE GRAFOS

Philippe Leal Freire dos Santos

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.

Aprovada em 23 de Agosto de 2010 por:

Profa. Dra. Maria Cristina Rangel - PPGI/UFES

Profa. Dra. Maria Claudia Silva Boeres - PPGI/UFES

Profa. Dra. Lucia Catabriga - PPGI/UFES

Profa. Dra. Nair Maria Maia de Abreu - COPPE/UFRJ

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória, Agosto de 2010

À minha avó Delza (in memorian)

"Sabemos que todas as coisas
cooperam para o bem daqueles
que amam a Deus."
Romanos 8:28

Agradecimentos

Primeiramente a Jesus Cristo por ser o meu Senhor e Salvador, pelas Suas maravilhas constantes em minha vida e por ter me sustentado em tudo para eu chegar até aqui.

Às minhas Professoras orientadoras Maria Cristina Rangel e Maria Claudia Silva Boeres, pelo apoio, dedicação, paciência, amizade e todo o aprendizado que me proporcionaram durante o curso.

Às Professoras Nair Maria Maia de Abreu (PEP/COPPE/UFRJ) e Carla Silva Oliveira (ENCE/IBGE) pelo suporte teórico em Teoria Espectral de Grafos.

À minha esposa Renata pelo amor, carinho, dedicação, oração, paciência e compreensão, fundamentais para a conclusão deste trabalho.

Aos meus pais, por me sustentarem em oração, e à minha avó Delza, que me deu todo o apoio e incentivo para a realização deste curso.

Aos amigos Lucas Augusto Scotta Merlo e Renato Stocco Bonatto, pela amizade e por compartilhar momentos de muito estudo, mas também de descontração, sem os quais o curso não seria o mesmo.

À FAPES pelo apoio financeiro, indispensável para a realização do Mestrado.

Sumário

Lista de Tabelas

Lista de Figuras

Resumo

Resumo da dissertação apresentada ao PPGI/UFES como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência (M.Sc.)

Orientadoras: Maria Cristina Rangel
Maria Claudia Silva Boeres

Departamento: Informática

Neste trabalho investigamos a utilização de conceitos da Teoria Espectral de Grafos (TEG) a fim de auxiliar a construção de algoritmos que solucionem o Problema de Isomorfismo de Grafos (PIG). Três resultados teóricos que consideram informações do espectro e das centralidades de autovetor dos vértices dos grafos foram apresentados. Além disso, foi proposto um algoritmo para detecção de isomorfismo de grafos baseado em dois destes resultados. Por fim, apresentamos os resultados computacionais da comparação deste algoritmo com outros da literatura.

PalavrasChaves: Problema de Isomorfismo de Grafos, Teoria Espectral de Grafos, Centralidades de Autovetor.

Abstract

Abstract of the dissertation presented to PPGI/UFES as a parcial fulfillment of the requirements for the degree of Master in Science (M.Sc.)

Advisors: Maria Cristina Rangel
Maria Claudia Silva Boeres

Department: Informatica

In this work we investigated the use of concepts from Spectral Graph Theory (SGT) to support the construction of algorithms that solve the Graph Isomorphism Problem (GIP). Three theoretical results which consider information from the spectrum of the graphs and from the eigenvector centralities were presented. Furthermore, an algorithm for detection of graph isomorphism based on two of these results was proposed. Finally, we present the computational results comparing this algorithm with others from literature.

Keywords: Graph Isomorphism Problem, Spectral Graph Theory, Eigenvector Centralities.

1 *Introdução*

O Problema de Isomorfismo de Grafos (PIG) tem sido amplamente pesquisado devido a sua aplicabilidade em situações reais, que podem ser modeladas e solucionadas por meio dele. A área de química é um exemplo disto, onde constantemente torna-se necessário determinar se uma molécula possui ou não estrutura similar a uma outra [Fortin, 1996, Oliveira e Greve, 2005]. Também encontramos aplicações no reconhecimento de imagens [Cordella et al., 2000] e na comparação de impressões digitais [Nandi, 2006].

O PIG consiste em estabelecer um mapeamento um a um entre os vértices de dois grafos, que preserve a relação de adjacência entre os vértices. Atualmente não se conhece exatamente a sua complexidade. Sabe-se que ele pertence à classe de problemas NP, no entanto é desconhecido se está em P ou em NP-completo [Jenner et al., 2003]. A suposição comumente aceita é que ele esteja estritamente entre estas duas classes [Arvind e Torán, 2005].

A Teoria Espectral de Grafos (TEG) foi sugerida inicialmente por Hückel em 1931 no seu trabalho na área de química quântica [Hückel, 1931], onde representou por um grafo a estrutura da molécula dos hidrocarbonetos insaturados. Esta teoria vem atraindo um maior interesse dos estudiosos desde a década de 80, em virtude da sua aplicação em diversas áreas, como na química, na matemática, na engenharia e na ciência da computação, conforme [Abreu, 2005].

A TEG é uma parte da matemática discreta que estuda as propriedades de um grafo a partir das informações fornecidas pelo espectro da matriz associada a este grafo, por exemplo, a matriz de adjacência, a Laplaciana e a Laplaciana sem sinal [Hogben, 2009]. Para certas famílias de grafos é possível caracterizar um grafo pelo espectro de uma destas matrizes que o representa, mas em geral isto não é possível.

Alguns problemas de otimização combinatória, como é o caso do PIG, demandam um tempo inviável de processamento pela busca da solução ótima quando tentamos enumerar todas as possíveis soluções para o problema e verificar cada uma delas. Isto se deve ao fato de possuírem um espaço de soluções muito grande, o que requer um alto custo computacional para avaliar cada possível solução. Por isso, algumas estratégias para encontrar uma resposta para o problema sem a necessidade de explorar todo o seu espaço de busca são implementadas.

Alguns algoritmos para o PIG fazem uso desta estratégia, buscando informações estruturais nos grafos de entrada a fim de reduzir o espaço de soluções. Um exemplo é o algoritmo Nauty [McKay, 1981], que realiza uma rotulação canônica nos vértices de ambos os grafos de entrada, restringindo o mapeamento somente entre vértices de mesma rotulação.

Fazendo uso desta estratégia de restringir o espaço de busca do problema, neste trabalho propomos um algoritmo que utiliza propriedades da TEG na tentativa de auxiliar a busca pelo isomorfismo entre dois grafos. Apresentamos também três resultados teóricos (três teoremas) que consideram informações do espectro dos grafos e das centralidades de autovetor. Além disso, comparamos o desempenho do algoritmo proposto com cinco algoritmos exatos da literatura, a citar: os algoritmos propostos por [Ullmann, 1976], [Dharwadker e Tevet, 2009] e [Lee, 2007], e os algoritmos VF2 [Cordella et al., 2001] e Nauty [McKay, 1981].

Este trabalho está assim dividido: o capítulo seguinte trata do PIG, abordando suas definições, alguns exemplos e possíveis aplicações. No Capítulo 3 são descritos três resultados teóricos, bem como alguns conceitos de Álgebra Linear, Teoria dos Grafos e TEG, a fim de auxiliar o entendimento do algoritmo proposto, descrito no Capítulo 4. A especificação das instâncias de teste para a comparação dos algoritmos e os resultados computacionais obtidos são apresentados no Capítulo 5. Por fim, o Capítulo 6 apresenta as conclusões e propostas de trabalhos futuros.

2 O Problema de Isomorfismo de Grafos

Ao longo deste capítulo são apresentados a definição matemática do Problema de Isomorfismo de Grafos (PIG), com exemplos de grafos isomorfos e não isomorfos, o estado da arte deste problema, citando os principais algoritmos exatos para a sua resolução, e possíveis aplicações para o PIG.

2.1 Definição do Problema

O Problema de Isomorfismo de Grafos tem sido amplamente estudado devido a sua grande aplicabilidade em problemas reais, que podem ser modelados e solucionados por meio dele. A área de química é um exemplo disto, onde constantemente torna-se necessário determinar se uma molécula possui ou não estrutura similar a uma outra, para que se possa atribuir-lhe um nome exclusivo. Esta verificação pode ser realizada comparando a molécula a uma base de dados molecular existente. Neste caso, as moléculas seriam representadas por um grafo, sendo os vértices correspondentes aos átomos e as arestas às suas ligações [Fortin, 1996, Oliveira e Greve, 2005].

O processo de comparação supracitado pode ser classificado como um PIG, uma vez que este problema consiste em verificar se dois grafos dados são estruturalmente idênticos, ou seja, se existe um mapeamento um a um entre os vértices do primeiro grafo com os do segundo que preserve a relação de adjacência entre os vértices.

Podemos definir formalmente o PIG como: Dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ são ditos isomorfos se existe uma função bijetora $f : V_1 \rightarrow V_2$ tal que, $\forall a, b \in V_1, (a,b) \in E_1 \Leftrightarrow (f(a), f(b)) \in E_2$, onde V_1 e V_2 são conjuntos de vértices, e E_1 e E_2, conjuntos de arestas, respectivamente.

Um exemplo de dois grafos isomorfos pode ser visto na Figura 2.1, onde é possível encontrar uma função $f : V_1 \rightarrow V_2$, $f = \{(1,1'), (2,5'), (3,3'), (4,4'), (5,2'), (6,6')\}$, que mantém as adjacências existentes entre os vértices dos grafos.

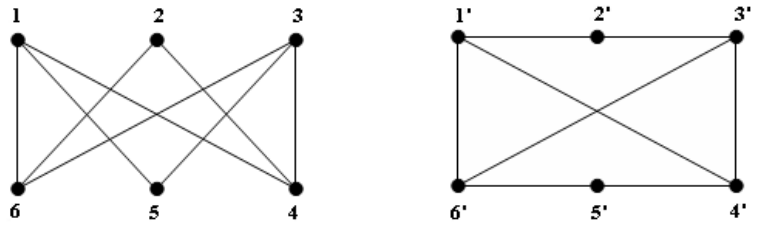

Figura 2.1: Exemplo de grafos isomorfos

Para que dois grafos sejam isomorfos, ao menos as seguintes condições são necessárias [Dalcumune, 2008]:

- Possuir o mesmo número de vértices;
- Possuir o mesmo número de arestas;
- Possuir a mesma sequência de graus dos vértices.

Cada uma destas condições é chamada de **invariante de um grafo**, pois não varia para qualquer grafo da classe de grafos isomorfos a ele. Ainda permanece em aberto o conhecimento de uma lista completa de invariantes de um grafo capaz de o caracterizar. Se tal lista fosse determinada, esta resolveria o Problema de Isomorfismo de Grafos.

Portanto, embora necessárias, estas três condições não são suficientes para que dois grafos sejam isomorfos. Como exemplo temos os grafos da Figura 2.2 que atendem tais condições, porém não são isomorfos, pois não é possível estabelecer uma função bijetora entre os vértices preservando as adjacências.

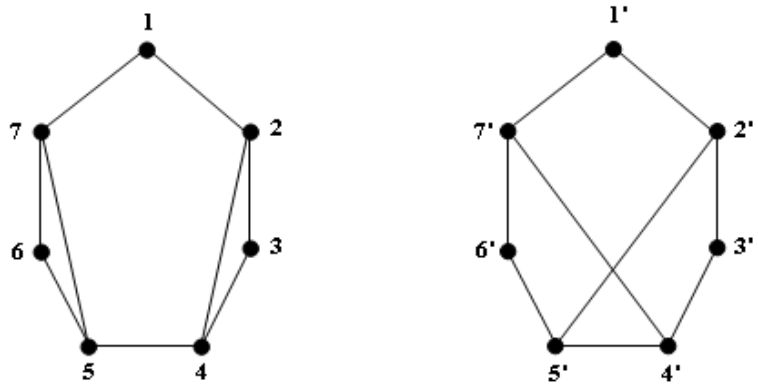

Figura 2.2: Exemplo de grafos não isomorfos com a mesma sequência de graus

2.2 Algoritmos para o PIG

Atualmente não se conhece exatamente a complexidade do PIG. Sabe-se que ele pertence à classe de problemas NP, no entanto, segundo [Jenner et al., 2003], é desconhecido se está em P ou em NP-completo. Conforme [Arvind e Torán, 2005], a suposição comumente aceita é que ele esteja estritamente entre estas duas classes. Apesar disto, foram desenvolvidos algoritmos de tempo polinomial dedicados a muitas classes de grafos. Em [Sorlin e Solnon, 2004], [Uehara et al., 2005] e [Zager, 2005] são citadas algumas destas classes, que incluem grafos planares, grafos de intervalo, grafos limitados por grau, árvores, entre outros.

O algoritmo exato mais eficiente para solucionar o PIG, segundo [Sorlin e Solnon, 2008], foi desenvolvido por McKay [McKay, 1981], tendo sido denominado Nauty. Este algoritmo tem por característica determinar uma representação canônica dos grafos, que é um particionamento ordenado dos vértices, onde todos os vértices de uma partição possuem uma mesma rotulação, distinguindo-os dos demais vértices do grafo. Este particionamento é calculado aplicando-se, de maneira iterativa, um conjunto de invariantes de vértices a uma primeira partição, a qual, inicialmente, agrupa todos os vértices do grafo. Ao final do processo, dois grafos são isomorfos se, e somente se, possuem a mesma representação.

De acordo com [Cordella et al., 2001], o algoritmo de Ullmann [Ullmann, 1976] é um outro algoritmo exato amplamente utilizado devido a sua eficácia em tratar tanto o PIG quanto o Problema de Isomorfismo de Subgrafos. Utilizando o processo de *backtracking* e um método de refinamento, ele obtém consideráveis reduções no espaço de busca do problema. Este refinamento tem por objetivo verificar se uma futura associação entre vértices não conduz a um possível isomorfismo, eliminando assim a possibilidade de mapeamento entre aqueles vértices e, consequentemente, diminuindo o número de associações que devem ser analisadas [Messmer e Bunke, 1995]. Terminada a sua execução, sendo os grafos de entrada isomorfos, o algoritmo é capaz de apresentar como resultado todas as possíveis associações de vértices existentes entre os dois grafos.

A técnica de *backtracking* também é empregada no algoritmo SD [Schmidt e Druffel, 1976] e no algoritmo desenvolvido por Lee [Lee, 2007]. O primeiro utiliza a informação contida na matriz distância que representa o grafo para estabelecer uma partição inicial dos vértices. Esta informação é aplicada no procedimento de *backtracking* para limitar a procura por possíveis mapeamentos de vértices. O segundo realiza um processo de transcrição em cada grafo de entrada, fazendo as arestas originais do grafo receberem valor 1 e as arestas inseridas entre vértices não adjacentes receberem valor 0, gerando assim grafos completos com pesos nas arestas. Ele implementa também a criação de blocos de graus dos vértices, onde vértices de mesmo grau fazem parte do mesmo bloco. Assim, a estratégia de busca em árvore deste algoritmo se baseia em

associar vértices de mesmo bloco e que gerem apenas associações de arestas de mesmo peso, ou seja, uma aresta com valor 1 de um grafo deve ser associada a uma única aresta de mesmo valor do outro grafo, de igual modo as arestas de valor 0. Caso esta associação de arestas não seja atendida, o algoritmo termina a exploração daquele ramo da árvore e realiza *backtracking* para um determinado nó, continuando a exploração a partir deste. Por tratar-se de um algoritmo exato, assim como o algoritmo SD, ele apresenta um isomorfismo ou uma impossibilidade de correspondência entre os grafos de entrada ao final de sua execução.

O algoritmo VF [Cordella et al., 1999] é outro algoritmo utilizado na detecção do isomorfismo que se fundamenta na estratégia de busca em profundidade. Para orientar eficientemente a busca pelo isomorfismo, ele faz uso de um conjunto de regras que impõem condições para o processo de mapeamento. Uma segunda versão deste algoritmo, chamado VF2 [Cordella et al., 2001], tem como principal vantagem em relação ao seu antecessor, a baixa requisição de memória, fazendo-o apropriado para trabalhar com grandes grafos.

Com a finalidade de analisar o desempenho de algoritmos exatos para a resolução do PIG que não impõem restrições na estrutura dos grafos de entrada, ou seja, que não são projetados apenas para uma classe especial de grafos, [Foggia et al., 2001] realizou a comparação entre os algoritmos Nauty, Ullmann, SD, VF e VF2. Foram utilizadas nos testes quatro classes de grafos, num total de 10.000 pares de grafos isomorfos. Analisando os resultados, notou-se que os algoritmos Nauty e VF2 se destacaram em relação aos demais, tendo cada um obtido melhores resultados em determinadas classes de grafos. Também foi observado que somente os algoritmos SD, VF e VF2 foram capazes de encontrar uma solução para o PIG, independentemente do tamanho e do tipo dos grafos de entrada.

Não obstante a incerteza quanto à complexidade do PIG, [Dharwadker e Tevet, 2009] propuseram um algoritmo que executa em tempo polinomial para todas as classes de grafos. Segundo eles, o algoritmo é necessário e suficiente para solucionar o PIG, mostrando que o problema está em P. O algoritmo se baseia em calcular a forma canônica da matriz sinal de cada um dos grafos de entrada, sendo estes isomorfos se, e somente se, suas matrizes sinal na forma canônica forem idênticas.

A seguir, ilustraremos os passos do algoritmo utilizando um exemplo do próprio trabalho. Os conceitos aqui presentes como matriz sinal, grafo par e vetor de frequência de sinal, são apresentados com detalhes no artigo. A Figura 2.3 exibe os grafos de entrada para o exemplo.

O primeiro passo é calcular todos os grafos par para cada par de vértices de G_1 a fim de que, a partir deles, seja possível criar a matriz sinal S_1 de G_1, ilustrada na Figura 2.4. Após esta construção, calcula-se o número de vezes que cada sinal ocorre nas colunas de S_1, obtendo-se os vetores de frequência de sinal VFS_1 para cada coluna de S_1. Estes vetores podem ser vistos

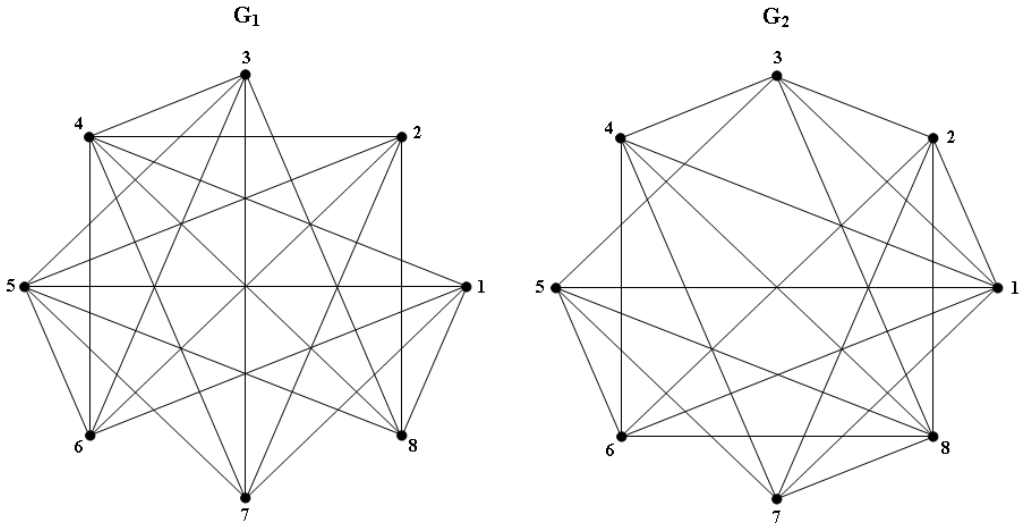

Figura 2.3: Grafos de entrada do exemplo

na Figura 2.5.

S_1	1	2	3	4	5	6	7	8
1	-0.1.0	-2.7.16	-2.7.16	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5
2	-2.7.16	-0.1.0	-2.7.16	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5
3	-2.7.16	-2.7.16	-0.1.0	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5
4	+2.5.7	+2.5.7	+2.5.7	-0.1.0	-2.8.21	+2.5.7	+2.5.7	+2.5.7
5	+2.5.7	+2.5.7	+2.5.7	-2.8.21	-0.1.0	+2.5.7	+2.5.7	+2.5.7
6	+2.4.5	+2.4.5	+2.4.5	+2.5.7	+2.5.7	-0.1.0	-2.7.16	-2.7.16
7	+2.4.5	+2.4.5	+2.4.5	+2.5.7	+2.5.7	-2.7.16	-0.1.0	-2.7.16
8	+2.4.5	+2.4.5	+2.4.5	+2.5.7	+2.5.7	-2.7.16	-2.7.16	-0.1.0

Figura 2.4: Matriz Sinal S_1

VFS_1	1	2	3	4	5	6	7	8
-2.7.16	2	2	2	0	0	2	2	2
-2.8.21	0	0	0	1	1	0	0	0
-0.1.0	1	1	1	1	1	1	1	1
+2.4.5	3	3	3	0	0	3	3	3
+2.5.7	2	2	2	6	6	2	2	2

Figura 2.5: Vetores de frequência de sinal VFS_1

A seguir, reordenam-se as linhas e colunas de S_1 de acordo com a ordem lexicográfica dos vetores de frequência de sinal para se obter a forma canônica da matriz sinal, chamada S_1^*. Na Figura 2.6 são exibidos os vetores de freqüência de sinal VFS_1 em ordem lexicográfica e a S_1^* gerada por meio deles.

Todo o processo anterior é realizado para o grafo G_2, a fim de se determinar os vetores de frequência de sinal VFS_2 em ordem lexicográfica e, consequentemente, a forma canônica da matriz sinal S_2^*, ambos vistos na Figura 2.7.

S_1*	**4**	**5**	**3**	**1**	**2**	**6**	**7**	**8**
4	-0.1.0	-2.8.21	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
5	-2.8.21	-0.1.0	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
3	+2.5.7	+2.5.7	-0.1.0	-2.7.16	-2.7.16	+2.4.5	+2.4.5	+2.4.5
1	+2.5.7	+2.5.7	-2.7.16	-0.1.0	-2.7.16	+2.4.5	+2.4.5	+2.4.5
2	+2.5.7	+2.5.7	-2.7.16	-2.7.16	-0.1.0	+2.4.5	+2.4.5	+2.4.5
6	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-0.1.0	-2.7.16	-2.7.16
7	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-2.7.16	-0.1.0	-2.7.16
8	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-2.7.16	-2.7.16	-0.1.0
VFS_1	**4**	**5**	**3**	**1**	**2**	**6**	**7**	**8**
-2.7.16	0	0	2	2	2	2	2	2
-2.8.21	1	1	0	0	0	0	0	0
-0.1.0	1	1	1	1	1	1	1	1
+2.4.5	0	0	3	3	3	3	3	3
+2.5.7	6	6	2	2	2	2	2	2

Figura 2.6: Forma canônica da matriz sinal S_1^* e os vetores de frequência de sinal VFS_1 em ordem lexicográfica do grafo G_1

S_2*	**1**	**8**	**3**	**4**	**5**	**6**	7	**2**
1	-0.1.0	-2.8.21	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
8	-2.8.21	-0.1.0	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
3	+2.5.7	+2.5.7	-0.1.0	+2.4.5	+2.4.5	-2.7.16	-2.7.16	+2.4.5
4	+2.5.7	+2.5.7	+2.4.5	-0.1.0	-2.7.16	+2.4.5	+2.4.5	-2.7.16
5	+2.5.7	+2.5.7	+2.4.5	-2.7.16	-0.1.0	+2.4.5	+2.4.5	-2.7.16
6	+2.5.7	+2.5.7	-2.7.16	+2.4.5	+2.4.5	-0.1.0	-2.7.16	+2.4.5
7	+2.5.7	+2.5.7	-2.7.16	+2.4.5	+2.4.5	-2.7.16	-0.1.0	+2.4.5
2	+2.5.7	+2.5.7	+2.4.5	-2.7.16	-2.7.16	+2.4.5	+2.4.5	-0.1.0
VFS_2	**1**	**8**	**3**	**4**	**5**	**6**	7	**2**
-2.7.16	0	0	2	2	2	2	2	2
-2.8.21	1	1	0	0	0	0	0	0
-0.1.0	1	1	1	1	1	1	1	1
+2.4.5	0	0	3	3	3	3	3	3
+2.5.7	6	6	2	2	2	2	2	2

Figura 2.7: Forma canônica da matriz sinal S_2^* e os vetores de frequência de sinal VFS_2 em ordem lexicográfica do grafo G_2

O próximo passo é comparar os vetores de frequência de sinal em ordem lexicográfica. Se forem distintos, G_1 e G_2 não são isomorfos e o algoritmo termina sua execução. Se forem iguais, o algoritmo compara as entradas A_{ij} de A com as entradas B_{ij} de B, onde i e j variam de

um até o número de vértices, $A = S_1^*$ e $B = S_2^*$. Se todas as entradas forem iguais, o algoritmo termina e o isomorfismo é dado pela leitura dos rótulos dos vértices de A e B na ordem em que estiverem no cabeçalho das colunas (ou linhas). Caso contrário, o algoritmo tenta encontrar um valor $k > i$, tal que, permutando as linhas (k, j) e as colunas (k, j) de B, garanta que a primeira incompatibilidade ocorra após B_{ij}, ou que nenhuma incompatibilidade aconteça.

Este processo se repete até que k não possa ser encontrado, caracterizando que G_1 e G_2 não são isomorfos, ou todas as entradas correspondentes de A e B sejam iguais, concluindo que existe um isomorfismo entre os grafos. No exemplo, como os vetores de frequência de sinal VFS_1 e VFS_2 são iguais, deve-se comparar as entradas correspondentes de S_1^* com as de S_2^*. Realizando esta comparação, observa-se que o elemento da terceira linha e quarta coluna difere entre as duas matrizes, como pode ser observado na Figura 2.8. Então, o algoritmo verifica que permutando a quarta coluna com a sexta (e a quarta linha com a sexta) de S_2^* resolve-se a primeira incompatibilidade. O resultado destas trocas de linhas e colunas de S_2^* é apresentado na Figura 2.9.

S_1*	**4**	**5**	**3**	**1**	**2**	**6**	**7**	**8**
4	-0.1.0	-2.8.21	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
5	-2.8.21	-0.1.0	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
3	+2.5.7	+2.5.7	-0.1.0	-2.7.16	-2.7.16	+2.4.5	+2.4.5	+2.4.5
1	+2.5.7	+2.5.7	-2.7.16	-0.1.0	-2.7.16	+2.4.5	+2.4.5	+2.4.5
2	+2.5.7	+2.5.7	-2.7.16	-2.7.16	-0.1.0	+2.4.5	+2.4.5	+2.4.5
6	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-0.1.0	-2.7.16	-2.7.16
7	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-2.7.16	-0.1.0	-2.7.16
8	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-2.7.16	-2.7.16	-0.1.0

S_2*	**1**	**8**	**3**	**4**	**5**	**6**	**7**	**2**
1	-0.1.0	-2.8.21	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
8	-2.8.21	-0.1.0	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
3	+2.5.7	+2.5.7	-0.1.0	**+2.4.5**	+2.4.5	-2.7.16	-2.7.16	+2.4.5
4	+2.5.7	+2.5.7	+2.4.5	-0.1.0	-2.7.16	+2.4.5	+2.4.5	-2.7.16
5	+2.5.7	+2.5.7	+2.4.5	-2.7.16	-0.1.0	+2.4.5	+2.4.5	-2.7.16
6	+2.5.7	+2.5.7	-2.7.16	+2.4.5	+2.4.5	-0.1.0	-2.7.16	+2.4.5
7	+2.5.7	+2.5.7	-2.7.16	+2.4.5	+2.4.5	-2.7.16	-0.1.0	+2.4.5
2	+2.5.7	+2.5.7	+2.4.5	-2.7.16	-2.7.16	+2.4.5	+2.4.5	-0.1.0

Figura 2.8: O elemento da terceira linha e quarta coluna de S_2^* difere do seu correspondente em S_1^*

Recomeçando a comparação a partir do início de S_1^* e S_2^*, encontra-se uma segunda in-

S_2*	1	8	3	6	5	4	7	2
1	-0.1.0	-2.8.21	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
8	-2.8.21	-0.1.0	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
3	+2.5.7	+2.5.7	-0.1.0	-2.7.16	**+2.4.5**	+2.4.5	-2.7.16	+2.4.5
6	+2.5.7	+2.5.7	-2.7.16	-0.1.0	+2.4.5	+2.4.5	-2.7.16	+2.4.5
5	+2.5.7	+2.5.7	+2.4.5	+2.4.5	-0.1.0	-2.7.16	+2.4.5	-2.7.16
4	+2.5.7	+2.5.7	+2.4.5	+2.4.5	-2.7.16	-0.1.0	+2.4.5	-2.7.16
7	+2.5.7	+2.5.7	-2.7.16	-2.7.16	+2.4.5	+2.4.5	-0.1.0	+2.4.5
2	+2.5.7	+2.5.7	+2.4.5	+2.4.5	-2.7.16	-2.7.16	+2.4.5	-0.1.0

Figura 2.9: Resultado da troca de linhas e colunas de S_2^* e o surgimento da segunda incompatibilidade entre S_1^* e S_2^*

compatibilidade entre os elementos da terceira linha e quinta coluna das matrizes, como visto também na Figura 2.9. Com isso, o algoritmo verifica que com a troca da quinta coluna com a sétima (e de mesmo modo com as linhas correspondentes) de S_2^*, esta segunda incompatibilidade é resolvida, o que é mostrado na Figura 2.10.

S_2*	1	8	3	6	7	4	5	2
1	-0.1.0	-2.8.21	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
8	-2.8.21	-0.1.0	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7	+2.5.7
3	+2.5.7	+2.5.7	-0.1.0	-2.7.16	-2.7.16	+2.4.5	+2.4.5	+2.4.5
6	+2.5.7	+2.5.7	-2.7.16	-0.1.0	-2.7.16	+2.4.5	+2.4.5	+2.4.5
7	+2.5.7	+2.5.7	-2.7.16	-2.7.16	-0.1.0	+2.4.5	+2.4.5	+2.4.5
4	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-0.1.0	-2.7.16	-2.7.16
5	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-2.7.16	-0.1.0	-2.7.16
2	+2.5.7	+2.5.7	+2.4.5	+2.4.5	+2.4.5	-2.7.16	-2.7.16	-0.1.0

Figura 2.10: A Forma canônica da matriz sinal S_1^* após a resolução da segunda incompatibilidade

Agora, realizando a comparação entre todas as entradas correspondentes de S_1^* e S_2^*, verifica-se que não existe mais qualquer incompatibilidade. Logo, o algoritmo finaliza sua execução informando que G_1 e G_2 são isomorfos e que a reordenação dos vértices de S_2^* para se conseguir $S_1^* = S_2^*$ provê uma função de isomorfismo, como mostrada na Figura 2.11. Por fim, cabe observar que o algoritmo realiza as permutações de linhas e colunas apenas em S_2^* para tentar torná-la idêntica a S_1^* e para mantê-la simétrica.

G_1	4	5	3	1	2	6	7	8
G_2	1	8	3	6	7	4	5	2

Figura 2.11: Mapeamento entre os vértices de G_1 e G_2 fornecido pelo algoritmo proposto por [Dharwadker e Tevet, 2009]

2.3 Aplicações do PIG

Grande parte dos problemas nos quais o PIG pode ser aplicado decorre da área de reconhecimento de padrões [Bunke, 2000]. Uma dessas aplicações pode ser vista em [Nandi, 2006], onde o PIG é aplicado no processo de comparação de impressões digitais. Neste trabalho, as imagens das impressões digitais a serem confrontadas são representadas por grafos, que são gerados a partir das características encontradas em cada imagem, características estas chamadas de minúcias. Cada vértice representa uma minúcia, sendo rotulado com o tipo e a posição geométrica desta na imagem. As arestas representam uma relação de vizinhança entre estas minúcias. A comparação é realizada por meio do cálculo do isomorfismo entre os dois grafos que representam as impressões digitais.

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. Para reduzir o esforço computacional, o algoritmo utiliza regras de viabilidade no mapeamento de arestas. Por exemplo, duas arestas são aceitas como similares somente se possuírem uma diferença de comprimento menor do que 30%. Assim, o algoritmo realiza o cômputo do isomorfismo entre o grafo de entrada, que representa a imagem a ser pesquisada, e o grafo representante de uma região da imagem.

3 *Teoria Espectral de Grafos*

Neste capítulo são apresentadas noções básicas de Álgebra Linear e Teoria dos Grafos, bem como alguns resultados referentes à Teoria Espectral de Grafos, extraídos de [Abreu et al., 2007], com o objetivo de auxiliar no entendimento de assuntos tratados posteriormente. São descritos também três resultados teóricos alcançados a partir dos testes computacionais realizados para este trabalho.

3.1 Conceitos Básicos de Álgebra Linear

Nesta seção apresentamos algumas definições de álgebra linear, como matriz, traço e polinômio, que podem ser encontradas em [Boldrini et al., 1986] e [Santos, 2009].

Uma **matriz** $A_{m\times n}$, é uma tabela de mn números dispostos em m linhas e n colunas, como mostrado na Figura 3.1. Dizemos que a_{ij} é o **elemento** ou **entrada** presente na linha i e coluna j da matriz A. Se $m = n$, A é uma **matriz quadrada** de ordem n e os elementos $a_{11}, a_{22}, \ldots, a_{nn}$ formam a **diagonal principal** de A. Uma **matriz diagonal** é aquela em que os elementos que estão fora da diagonal principal são iguais a zero. Um exemplo desta matriz pode ser visto na Figura 3.2.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Figura 3.1: Matriz A

Uma matriz que só possui uma linha é denominada **matriz linha**, e uma matriz que possui somente uma coluna é denominada **matriz coluna**. Um exemplo de cada uma destas matrizes pode ser visto, respectivamente, na Figura 3.3a e Figura 3.3b. Matrizes linha e matrizes coluna são chamadas de **vetores**.

$$B = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Figura 3.2: Exemplo de uma matriz diagonal

$$C = \begin{bmatrix} 14 & 5 & 11 & 20 \end{bmatrix} \qquad D = \begin{bmatrix} 8 \\ 22 \\ 3 \end{bmatrix}$$

(a) Matriz linha (b) Matriz coluna

Figura 3.3: Exemplo de matriz linha e matriz coluna

A multiplicação de uma matriz $A = (a_{ij})_{m \times n}$ por um escalar (um número) λ é obtida multiplicando-se cada elemento de A pelo escalar λ, ou seja, $\lambda A = \lambda a_{ij}$. Exemplificando, considere a matriz E da Figura 3.4a. Realizando o produto desta matriz pelo escalar 3, temos como resultado a matriz $3E$, apresentada na Figura 3.4b.

$$E = \begin{bmatrix} 8 & 6 & 11 \\ 4 & 1 & 7 \\ 12 & 5 & 1 \end{bmatrix} \qquad 3E = \begin{bmatrix} 24 & 18 & 33 \\ 12 & 3 & 21 \\ 36 & 15 & 3 \end{bmatrix}$$

(a) Matriz E (b) Matriz $3E$

Figura 3.4: Matriz E e o resultado do seu produto pelo escalar 3 (Matriz $3E$)

Chamamos de **traço** de uma matriz quadrada A, denotado por $tr(A)$, a soma dos elementos da sua diagonal principal. Uma matriz quadrada de ordem n é denominada **matriz identidade**, I_n, quando os elementos da sua diagonal principal são iguais a 1 e os demais iguais a 0. O traço de uma matriz identidade é igual a sua ordem, ou seja, $tr(I_n) = n$.

O **determinante** de uma matriz é a associação de um número real, segundo uma determinada lei, a toda matriz quadrada de ordem n. Ou seja, o determinante é uma função

$$det : A_n \rightarrow R$$

do conjunto das matrizes quadradas de ordem n, A_n, no conjunto dos números reais.

Um **polinômio** de grau n é uma função da forma

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0$$

onde os coeficientes $a_0, a_1, \ldots, a_n$ são números reais, denominados **coeficientes** do polinômio.

Em um polinômio, o termo de mais alto grau que possui um coeficiente não nulo é chamado **termo dominante**. O **grau** de um polinômio não nulo é o expoente de seu termo dominante.

Definimos **raiz** (ou **zero**) de um polinômio um valor, tal que, atribuído à variável da função polinomial, faz com que a função resulte em 0. Assim, se r é dito raiz do polinômio $P(x)$, então $P(r) = 0$. Um polinômio de grau n sempre terá n raízes, podendo haver repetição de uma mesma raiz.

3.2 Conceitos Básicos de Teoria dos Grafos

São resumidos nesta seção alguns conceitos básicos da Teoria de Grafos necessários à compreensão do restante deste trabalho, os quais também podem ser encontrados em [Diestel, 2005] e [Bondy e Murty, 1979]. Introduziremos também a notação sobre grafos utilizada nesta dissertação.

Podemos visualizar um grafo utilizando uma representação geométrica, na qual pontos distintos do plano em posições arbitrárias equivalem a seus vértices e cada aresta corresponde a uma linha ligando estes pontos, como faremos com os exemplos aqui apresentados. Computacionalmente, os grafos podem ser representados através de matrizes, como a matriz distância, que veremos nesta seção, a matriz de adjacência, a Laplaciana e a Laplaciana sem sinal, que serão definidas na Seção 3.3.

Assim, um **grafo** é uma estrutura $G = (V,E)$, sendo V um conjunto finito não vazio e E um conjunto de pares não ordenados de V. Os elementos de V são denominados **vértices** e os de E são denominados **arestas** de G. O número de vértices (também conhecido como **ordem** do grafo) e o número de arestas de G são indicados, respectivamente, por $n = |V|$ e $m = |E|$. Cada aresta $e \in E$ será denotada por $e = \{v,w\}$, onde os vértices v e w são os **extremos** de e. Neste caso, como a aresta e é **incidente** a ambos os vértices, eles são chamados de **vértices adjacentes**. O número de arestas que incidem em um vértice v é chamado **grau** de v, denotado por $d(v)$. **Arestas adjacentes** são aquelas que possuem um vértice em comum.

Um grafo pode ter arestas ligando um vértice a ele mesmo e arestas diferentes incidindo em um mesmo par de vértices, o que chamamos de **laços** e **arestas paralelas**, respectivamente. Dizemos que um grafo é **orientado** se $\{v,w\} \neq \{w,v\}$, ou seja, se as arestas possuem uma dada orientação. No entanto, neste trabalho, consideramos apenas grafos simples não orientados, ou seja, sem laços, sem arestas paralelas e sem orientação.

Um exemplo de grafo simples não orientado pode ser observado na Figura 3.5, onde $V = \{1,2,3,4,5,6\}$, $E = \{e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8\}$, $n = 6$ e $m = 8$. Neste exemplo, podemos ob-

servar que os extremos da aresta e_2 são os vértices 2 e 3. Assim, como estes vértices possuem uma aresta incidente a ambos, eles são adjacentes, ao contrário dos vértices 1 e 5, que não são adjacentes. Os graus dos vértices 4 e 6 são iguais a 3 e 4, respectivamente, e as arestas e_5 e e_6 são arestas adjacentes.

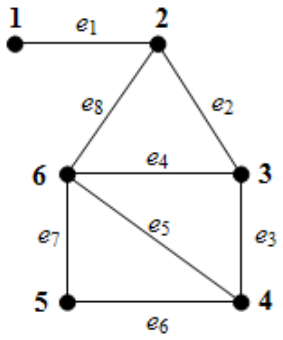

Figura 3.5: Grafos simples não orientado

Dizemos que o grafo H é um **subgrafo** de um grafo G se, e somente se, $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Desta forma, G é um **supergrafo** de H. A Figura 3.6 apresenta um subgrafo do grafo da Figura 3.5.

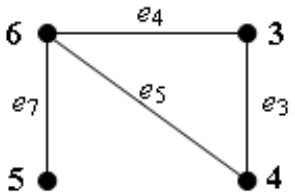

Figura 3.6: Um exemplo de subgrafo

Chamamos um grafo de k**-regular** se todos os seus vértices têm grau k e um grafo é dito **completo** quando há uma aresta entre cada par de seus vértices. Deste modo, todo grafo completo é $(n-1)$-regular. Um grafo completo com n vértices é denotado por K_n. As Figuras 3.7 e 3.8 ilustram, respectivamente, um grafo 3-regular e um grafo completo K_3.

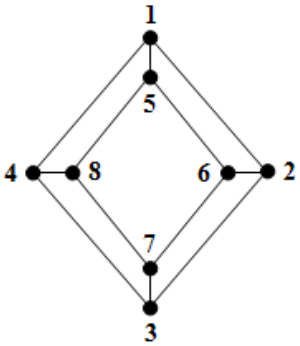

Figura 3.7: Grafo 3-regular

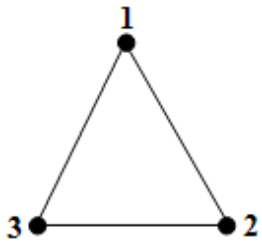

Figura 3.8: Grafo completo K_3

Um grafo k-regular com v vértices é denominado **fortemente regular** se há inteiros ρ e μ, tal que, dois vértices adjacentes têm ρ vizinhos em comum e dois vértices não adjacentes têm μ vizinhos em comum. Assim, um grafo fortemente regular pode ser representado por $srg(v,k,\rho,\mu)$. Como exemplo, temos o grafo $srg(6,3,0,3)$ da Figura 3.9.

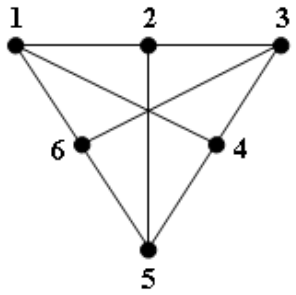

Figura 3.9: Grafo $srg(6,3,0,3)$

Um **percurso** ou **cadeia** é uma sequência de arestas sucessivamente adjacentes. Dizemos que um percurso é **fechado** se a última aresta da sucessão é adjacente a primeira. Caso contrário, o percurso é considerado **aberto**. O **comprimento** de um percurso é o número de arestas por ele utilizado.

Definimos como **caminho** ou **percurso elementar** um percurso em que todos os vértices são distintos. Um caminho fechado é chamado de **ciclo** e um grafo que não possui ciclos é dito **acíclico**. Exemplificando, o grafo da Figura 3.10 possui o ciclo $(2,3,4,2)$. O grafo que contém um caminho ligando quaisquer dois de seus vértices é denominado **conexo**, sendo chamado **desconexo** caso contrário. Denominamos **árvore** um grafo conexo e acíclico. O grafo da Figura 3.11 é uma árvore com $n = 5$ e $m = 4$.

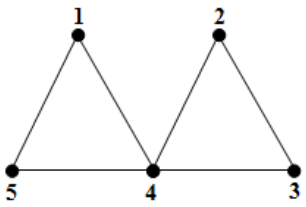

Figura 3.10: Grafo com o ciclo $(2,3,4,2)$

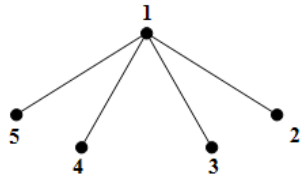

Figura 3.11: Árvore: grafo conexo e acíclico

A **matriz distância** de um grafo simples é uma matriz V x V, onde V representa o conjunto de vértices do grafo, na qual o elemento d_{ij} indica o comprimento do menor caminho entre os vértices v_i e v_j. Se $i = j$, então $d_{ij} = 0$. Caso não exista caminho entre dois vértices, o comprimento é definido como infinito. A Figura 3.12 apresenta um grafo e a sua matriz distância.

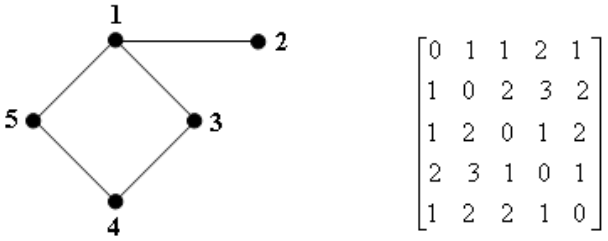

Figura 3.12: Um grafo e sua matriz distância

Um grafo é chamado *k*-**partido** se o seu conjunto de vértices permite uma partição em k subconjuntos não vazios, tendo os extremos de cada aresta em subconjuntos distintos. Um grafo 2-partido, também chamado de bipartido, pode ser visto na Figura 3.13.

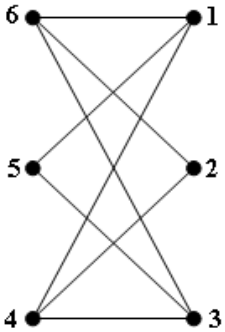

Figura 3.13: Grafo bipartido

Um grafo $\overline{G}$ é dito **complementar** de um grafo G se possuir a mesma ordem de G e se cada uma de suas arestas não pertencer a G. Um grafo **planar** é aquele que pode ser representado no plano de tal forma que suas arestas não se cruzem. A Figura 3.14 exibe um grafo e seu complementar, e a Figura 3.15 um grafo planar.

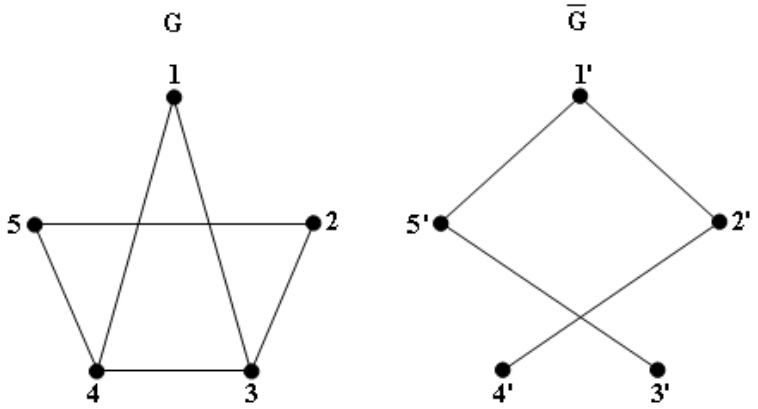

Figura 3.14: Um grafo e seu complementar

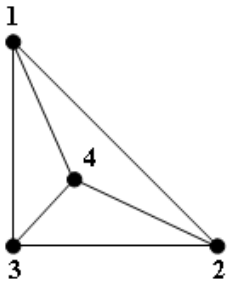

Figura 3.15: Grafo planar

O grafo **linha** de um grafo G, denotado por $l(G)$, é o grafo cujos vértices são as arestas de G, sendo estes conectados se, e somente se, as arestas correspondentes em G forem adjacentes. Na Figura 3.16 podemos observar um grafo e seu grafo linha correspondente.

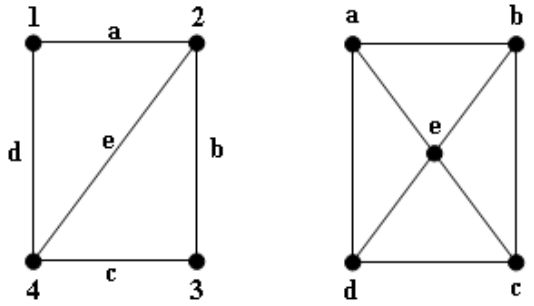

Figura 3.16: Um grafo e seu grafo linha

3.3 Conceitos Básicos de Teoria Espectral de Grafos

A Teoria Espectral de Grafos (TEG) é uma parte da matemática discreta e da Álgebra Linear que estuda as propriedades de um grafo a partir das informações fornecidas pelo espectro da matriz associada a este grafo, por exemplo, a matriz de adjacência e a Laplaciana [Hogben, 2009].

Nesta seção veremos o conceito destas duas matrizes, como também da matriz Laplaciana sem sinal, e alguns resultados associados a elas.

Seja $G = (V,E)$ um grafo simples, não orientado, com n vértices e m arestas. A matriz $n \times n$ cujas entradas são iguais a 1, se u e v são adjacentes, e 0 caso contrário, com u e $v \in V$, é denominada **matriz de adjacência** de G.

O polinômio característico da matriz de adjacência $A(G)$ de um grafo G é chamado **polinômio característico** de G e denotado por $p_G(\lambda)$. Assim, $p_G(\lambda) = det(\lambda I - A(G))$, onde λ é uma raiz deste polinômio e dito ser um **autovalor** de G. Como o grafo tem n vértices, então ele possui n autovalores, sendo o maior deles o **raio espectral** de G, denominado **índice** do grafo.

O espectro de G, indicado por $spect(G)$, é definido como uma matriz $2 \times d$, tendo na primeira linha os d autovalores distintos de G dispostos em ordem decrescente e na segunda linha as suas respectivas multiplicidades algébricas. Como a matriz de adjacência de G é simétrica, todos os seus autovalores são reais.

Como exemplo, observe o grafo G_1 da Figura 3.17.

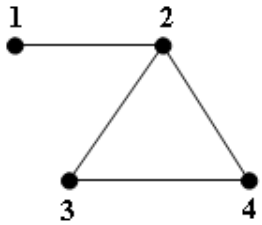

Figura 3.17: Grafo G_1

A matriz de adjacência de G_1 é:

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

O seu polinômio característico é $p_{G_1}(\lambda) = \lambda^4 - 4\lambda^2 - 2\lambda + 1$, tendo como espectro:

$$spect(G_1) = \begin{bmatrix} 2,1701 & 0,3111 & -1 & -1,4812 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Como um exemplo inicial de como propriedades da estrutura de um grafo podem ser descritas a partir da TEG, considere a Proposição 3.1.

Proposição 3.1. *Seja G um grafo com n vértices e m arestas, cujo polinômio característico é o que se segue*

$$p_G(\lambda) = \lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \ldots + a_{n-1}\lambda + a_n$$

Logo, os coeficientes deste polinômio satisfazem:

a) $a_1 = 0$*;*

b) $a_2 = -m$*;*

c) $a_3 = -2t$*, sendo t o número de triângulos contidos no grafo.*

Deste modo, analisando o grafo G_1 da Figura 3.17, verifica-se que os coeficientes do seu polinômio característico atendem as afirmativas anteriores, pois $a_1 = 0$, $a_2 = -4$, sendo 4 o número de arestas de G_1, e $a_3 = -2$, resultando $t = 1$, que é o número de triângulos em G_1.

Agora, vejamos outros resultados onde a TEG determina características estruturais de um grafo.

Considere G um grafo com n vértices, m arestas e autovalores λ_1, λ_2, ..., λ_{n-1}, λ_n. Então:

a) se C_p é o número total de cadeias fechadas de comprimento p em G, então $C_p = tr(A^p)$, onde tr é o traço da matriz. Logo, $C_p = \sum_{i=1}^{n} \lambda_i^p$. Porém, nem sempre ciclos de comprimento p $(p \geq 4)$ são determinados em função de $tr(A^p)$;

b) a soma dos quadrados dos autovalores é igual a duas vezes o número de arestas do grafo, ou seja, $\sum_{i=1}^{n} \lambda_i^2 = tr(A^2) = 2m$;

c) caso G seja um grafo regular de grau k, então $\sum_{i=1}^{n} \lambda_i^2 = kn$, uma vez que $kn = 2m$;

d) a soma dos cubos dos autovalores é igual a seis vezes o número t de triângulos no grafo, ou seja, $\sum_{i=1}^{n} \lambda_i^3 = tr(A^3) = 6t$.

Dada uma matriz A, um vetor não nulo x, tal que $Ax = \lambda x$, é dito **autovetor** associado ao autovalor λ. A **centralidade de autovetor** x_i é definida como a i-ésima componente do autovetor não-negativo x associado ao índice do grafo.

Exemplificando, considere novamente o grafo G_1 da Figura 3.17, que como foi visto tem índice igual a $2,1701$. Um autovetor associado a este índice é: $[0,2818,\ 0,6116,\ 0,5227,\ 0,5227]$, onde cada vértice i do grafo tem uma centralidade de autovetor x_i, $i = 1,\ \ldots, n$, associada a ele.

Proposição 3.2. *Seja G um grafo k-regular. Então:*

a) k é um autovalor de G;

b) G é conexo se, se somente se, a multiplicidade de k for igual a 1*;*

c) $|\lambda_i| \leq k$*, para* $1 \leq i \leq n$

Por exemplo, considere o grafo G_2 da Figura 3.18. O seu espectro é:

$$spect(G_2) = \begin{bmatrix} 2 & 0,6180 & 0,6180 & -1,6180 & -1,6180 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

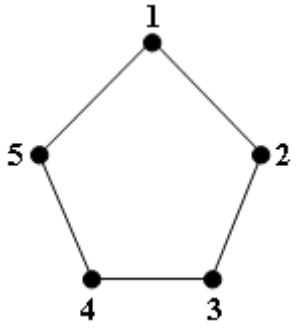

Figura 3.18: Grafo G_2

Assim, como G_2 é 2-regular, ele possui um autovalor igual a 2, que é o índice do grafo. Visto que este autovalor tem multiplicidade igual a 1, então G_2 é conexo. Portanto, todas as assertivas anteriores foram atendidas.

Um grafo G com n vértices possui um único autovalor se, e somente se, ele é totalmente desconexo, ou seja, um grafo sem arestas. Seu polinômio característico é $p_G(\lambda) = \lambda^n$ e seu espectro:

$$spect(G) = \begin{bmatrix} 0 \\ n \end{bmatrix}$$

O grafo completo com n vértices, K_n, é o grafo complementar do grafo totalmente desconexo. O espectro do grafo K_n é:

$$spect(K_n) = \begin{bmatrix} n-1 & -1 \\ 1 & n-1 \end{bmatrix}$$

Assim, observe o grafo completo da Figura 3.19. Seu espectro é:

$$spect(K_4) = \begin{bmatrix} 3 & -1 \\ 1 & 3 \end{bmatrix}$$

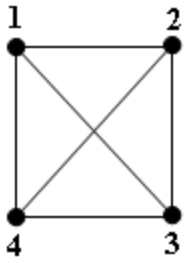

Figura 3.19: Grafo K_4

Dois grafos G_1 e G_2 são chamados **co-espectrais** quando seus autovalores são iguais considerando suas multiplicidades, ou seja, quando $spect(G_1) = spect(G_2)$. Dizemos que um grafo G é **caracterizado pelo seu espectro** se os grafos co-espectrais com G são isomorfos a ele.

Como resultado desta definição, temos que se dois grafos são isomorfos, então eles têm o mesmo espectro. Entretanto, a recíproca desta afirmação não é sempre verdadeira. Para exemplificar, considere os grafos G_3 e G_4 da Figura 3.20. Estes grafos possuem o mesmo polinômio característico:

$$p_{G_3}(\lambda) = p_{G_4}(\lambda) = \lambda^6 - 7\lambda^4 - 4\lambda^3 - 7\lambda^2 + 4\lambda - 1$$

Portanto, são co-espectrais. Contudo, não isomorfos.

A matriz laplaciana $L(G)$ do grafo G é definida como:

$$L(G) = D(G) - A(G)$$

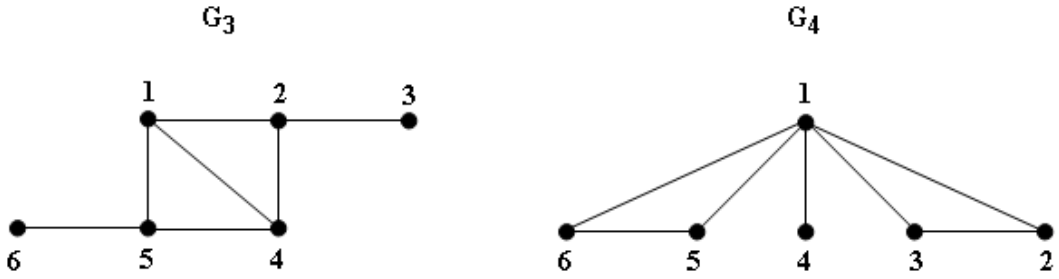

Figura 3.20: Grafos co-espectrais, porém não isomorfos (Extraídos de [Abreu, 2005])

onde $D(G)$ é a matriz diagonal composta pelos graus dos vértices de G e $A(G)$ é a matriz de adjacência de G. Vejamos o grafo G_5 da Figura 3.21.

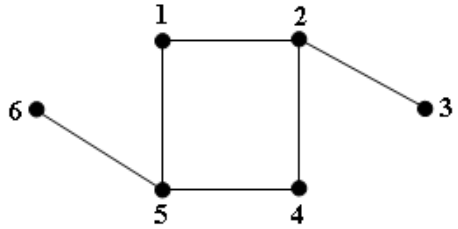

Figura 3.21: Grafo G_5

A matriz diagonal dos graus dos vértices de G_5 e sua matriz de adjacência são, respectivamente:

$$D(G_5) = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

e

$$A(G_5) = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Logo, a matriz laplaciana de G_5 é:

$$L(G_5) = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 2 & -1 & 0 \\ -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

Assim como na matriz de adjacência, é possível encontrar o espectro da matriz laplaciana de um grafo, chamado de **espectro do laplaciano**.

O espectro do laplaciano de G, denotado por $\zeta(G)$, é uma matriz $1 \times n$ na qual as entradas são os autovalores de $L(G)$ ordenados de maneira não-crescente. Então,

$$\zeta(G) = (\mu_1,\ \mu_2,\ \ldots,\ \mu_n)$$

onde $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_{n-1} \geq \mu_n$ são os autovalores de $L(G)$. O maior autovalor, μ_1, é chamado **índice do laplaciano** de G.

Assim, para o grafo G_5 da Figura 3.21, temos como espectro do laplaciano:

$$\zeta(G_5) = (4,7321,\ 3,4142,\ 2,\ 1,2679,\ 0,5858,\ 0)$$

Proposição 3.3. *Sendo $\mu_1 \geq \ldots \geq \mu_{n-1} \geq \mu_n$ os autovalores de $L(G)$, então:*

a) $\mu_n = 0$ e o vetor $(1,\ 1,\ \ldots,\ 1)^T$ é um autovetor associado;

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

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

Para ilustrar, considere o grafo G_2 da Figura 3.18. Seu espectro do laplaciano é:

$$\zeta(G_2) = (3,6180,\ 3,6180,\ 1,3820,\ 1,3820,\ 0)$$

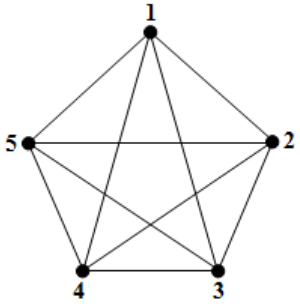

Figura 3.22: Grafo K_5

Portanto, as afirmações anteriores são válidas, pois $\mu_5 = 0$, G_2 é conexo, uma vez que $\mu_4 = 1,3820 > 0$, e como G_2 é 2-regular, seus autovalores seguem a formação da terceira afirmativa.

O espectro do laplaciano de um grafo completo com n vértices é:

$$(K_n) = (n,\ n,\ \ldots,\ 0)$$

O grafo completo K_5 da Figura 3.22 tem o seguinte espectro do laplaciano:

$$\zeta(K_5) = (5,\ 5,\ 5,\ 5,\ 0)$$

Uma **árvore geradora** de um grafo G é um subgrafo que tem os mesmos vértices de G e é uma árvore. O resultado a seguir apresenta a determinação do número de árvores geradoras de um grafo mediante a utilização de sua matriz laplaciana.

O número de árvores geradoras $\tau(G)$ de G com n vértices é dado por:

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

onde J é uma matrix n x n cujos elementos são todos iguais a 1.

Considerando o grafo da Figura 3.23, suas 8 árvores geradoras são apresentadas na Figura 3.24.

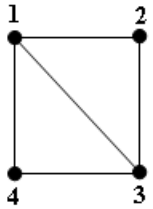

Figura 3.23: Grafo G_6

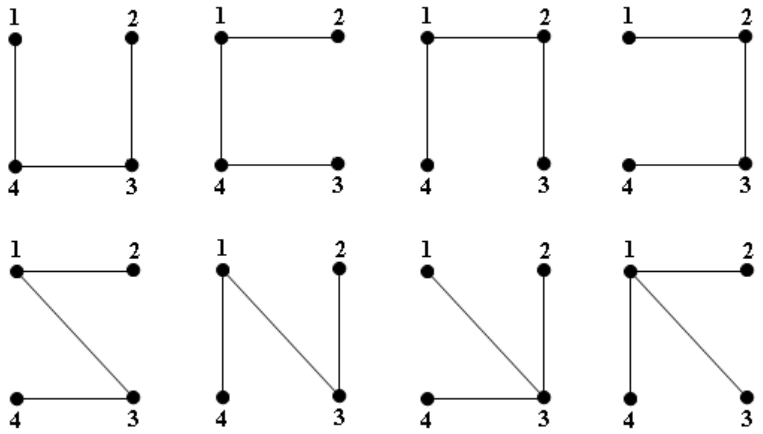

Figura 3.24: As 8 árvores geradoras do grafo G_6

Um dos principais resultados descritos pelo espectro do laplaciano é a conectividade algébrica, que exerce função importante na caracterização das propriedades de um grafo, possuindo várias aplicações em problemas reais, como na vulnerabilidade de redes de computadores. Ela está relacionada com as conectividades de vértices e arestas, como pode ser visto na Proposição 3.4.

A **conectividade algébrica** de um grafo G, denotada por $a(G)$, é o segundo menor autovalor da matriz laplaciana de G. O número mínimo de vértices de um grafo G que ao serem removidos torna-o desconexo é chamado de **conectividade de vértices** de G, denotado por $x(G)$. O número mínimo de arestas de um grafo G que ao serem removidas torna-o desconexo é chamado de **conectividade de arestas** de G, denotado por $u(G)$.

Proposição 3.4. *Sendo G um grafo não completo, então* $a(G) \leq x(G) \leq u(G)$.

Este resultado pode ser visto utilizando o grafo G_7 da Figura 3.25, onde $a(G_7) = 1$, $x(G_7) = 2$ e $u(G_7) = 2$.

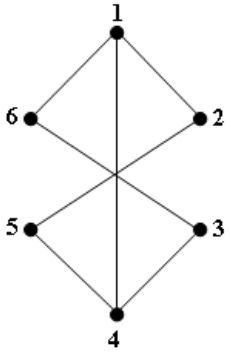

Figura 3.25: Grafo G_7

A matriz laplaciana sem sinal de um grafo G é dada por:

$$Q(G) = D(G) + A(G)$$

onde $D(G)$ é a matriz diagonal composta pelos graus dos vértices de G e $A(G)$ é a matriz de adjacência de G. O **polinômio característico da matriz laplaciana sem sinal** é denotado por $p_Q(\lambda)$ e seus autovalores por $q_1 \geq q_2 \geq \ldots \geq q_{n-1} \geq q_n$, tendo q_1 como o índice de Q.

A seguir veremos alguns resultados sobre a estrutura de grafos a partir de informações espectrais da matriz laplaciana sem sinal.

Seja G um grafo com n vértices e m arestas, e p_1 o coeficiente de λ^{n-1} no polinômio característico de Q. Assim, $m = |p_1|/2$. Vejamos um exemplo com o grafo G_8 da Figura 3.26. O polinômio característico da matriz laplaciana sem sinal de G_8 é $p_{Q(G_8)}(\lambda) = \lambda^4 - 6\lambda^3 + 9\lambda^2 - 4\lambda$. Como $p_1 = -6$, logo $m = |-6|/2 = 3$.

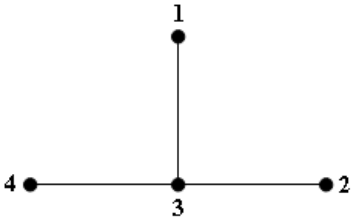

Figura 3.26: Grafo G_8

Considerando grafos bipartidos, o espectro da matriz laplaciana sem sinal é igual ao espectro da matriz laplaciana, ou seja, $q_i = \mu_i$, onde $i = 1, 2, \ldots, n$. Ainda utilizando o grafo G_8 como exemplo, pois é um grafo bipartido, o polinômio característico da sua matriz laplaciana é $p_{L(G_8)}(\lambda) = \lambda^4 - 6\lambda^3 + 9\lambda^2 - 4\lambda$. Ou seja, os polinômios característicos de $Q(G_8)$ e de $L(G_8)$ são iguais. Logo, eles têm o mesmo espectro.

Considere G um grafo com n vértices e m arestas. G é regular se, e somente se, $nq_1 = 4m$, sendo $q_1/2$ os graus dos vértices. Como exemplo, observe o grafo G_9 da Figura 3.27. O espectro da matriz laplaciana sem sinal de G_9 é $spec(Q(G_9)) = (2,\ 2,\ 0,\ 0)$. Então, como $q_1 = 2$, os vértices têm grau igual a 1.

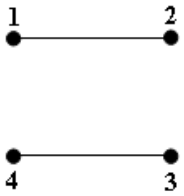

Figura 3.27: Grafo G_9

3.4 Alguns Resultados Teóricos

Nesta seção propomos três resultados teóricos, sendo que dois deles dão suporte ao algoritmo que será apresentado no Capítulo 4 para detecção de isomorfismo de grafos. Todos estes resultados foram obtidos durante a análise dos experimentos computacionais realizados com grafos isomorfos e não isomorfos. Nos testes, observamos que, se existia um isomorfismo entre dois grafos, então as suas centralidades de autovetor eram proporcionais. Este resultado é enunciado no Teorema 1. O segundo resultado é proposto como Teorema 2, que diz respeito a grafos fortemente regulares serem co-espectrais em relação à matriz de adjacência. O terceiro, apresentado no Teorema 3, afirma que grafos com vetores de centralidades proporcionais, porém cada um com suas componentes distintas entre si, são isomorfos.

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

Prova: Sendo dois grafos G_1 e G_2 isomorfos, então o espectro de G_1 é igual ao espectro de G_2. Assim, os seus respectivos índices λ_1^1 e λ_1^2 também são iguais. Além disso, ambos têm multiplicidade igual a 1. Os autoespaços associados ao índice de cada grafo também são iguais, e dado que a multiplicidade deles é 1, os autoespaços têm que ter dimensão 1. Logo, qualquer autovetor dos respectivos autoespaços são múltiplos um dos outros e, portanto, o autovetor de um grafo também será múltiplo escalar do autovetor do outro [Horn e Johnson, 1985]. Assim, se $\vec{x^1}$ é o autovetor de G_1 correspondente ao índice λ_1^1 e $\vec{x^2}$ é o autovetor de G_2 correspondente ao índice λ_1^2, tem-se que $\vec{x^1} = k\vec{x^2}$, para algum real k. Sabendo-se que a i-ésima coordenada x_i^1 do autovetor associado ao índice de G_1 é a centralidade do vértice $i(G_1)$ e que o mesmo acontece para i-ésima coordenada x_i^2 do autovetor associado ao índice de G_2, ou seja, que este é a centralidade do vértice $i(G_2)$, e dado que $\vec{x^1} = k\vec{x^2}$, então $x_i^1 = kx_i^2$. ■

Analisando os resultados de todos os testes computacionais, dos quais sabíamos a priori se os grafos eram ou não isomorfos, observamos que na totalidade dos casos de isomorfismo as centralidades de autovetor de ambos eram proporcionais. Nos casos em que essa característica não se verificava, os grafos não eram isomorfos. Contudo, esta propriedade, apesar de ser necessária, não é suficiente para assegurar a existência de isomorfismo entre dois grafos. Ou seja, existem grafos que possuem as mesmas centralidades de autovetor e não são isomorfos. Assim, vejamos alguns exemplos de pares de grafos nos quais podemos observar o resultado apresentado.

Os grafos da Figura 3.28 são isomorfos e, assim como verificamos em todos os testes entre

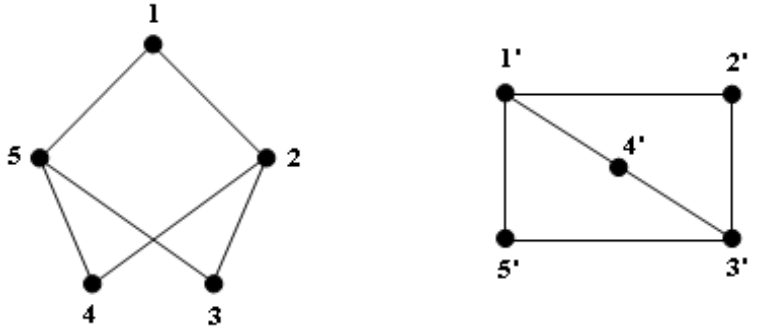

Figura 3.28: Grafos Isomorfos

grafos isomorfos, as suas centralidades de autovetor são proporcionais. Contudo, não podemos afirmar que quaisquer dois grafos com centralidades proporcionais sejam isomorfos, uma vez que existem pares de grafos não isomorfos que satisfazem esta proporcionalidade. Um destes pares, que analisamos em nossos testes, é apresentado na Figura 3.29.

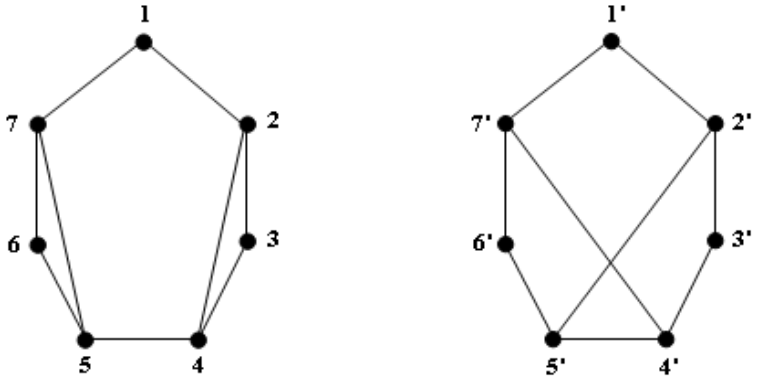

Figura 3.29: Grafos não isomorfos cujas centralidades de autovetor são proporcionais

Verificamos também que em todos os pares de grafos nos quais as centralidades de autovetor não eram proporcionais, não existia um isomorfismo entre eles. Um exemplo é o par de grafos da Figura 3.30.

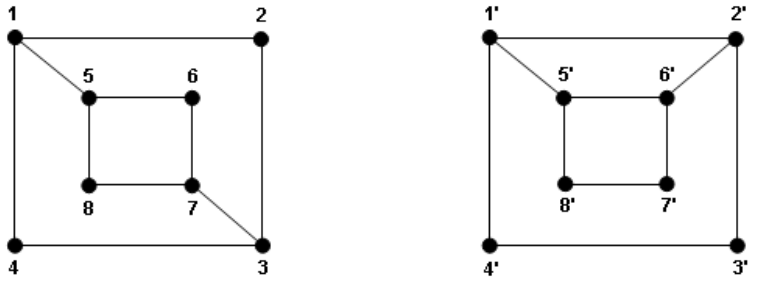

Figura 3.30: Grafos não isomorfos cujas centralidades de autovetor são distintas

Teorema 2. *Grafos fortemente regulares com os mesmos conjuntos de parâmetros são co-espectrais em relação à matriz de adjacência.*

Prova: Seja G um grafo fortemente regular com matriz de adjacência A. De acordo com [Cameron, 2001], A tem somente três autovalores distintos k, λ_2 e λ_3 com multiplicidades 1, f e g, respectivamente, onde f e g são inteiros não negativos. Como G é regular, então $k^2 + \lambda_2^2 + \lambda_3^2 = 2m$ [Abreu et al., 2007]. Considere G e G' grafos fortemente regulares com autovalores k, λ_2, λ_3 e k', λ_2', λ_3', respectivamente. Assim:

$$k^2 + \lambda_2^2 + \lambda_3^2 = 2m \tag{3.1}$$

e

$$k'^2 + \lambda_2'^2 + \lambda_3'^2 = 2m \tag{3.2}$$

Subtraindo (3.2) de (3.1), temos:

$$(\lambda_2^2 + \lambda_3^2) - (\lambda_2'^2 + \lambda_3'^2) = 0 \tag{3.3}$$

Suponha que λ_2 tenha a mesma multiplicidade de λ_2'. Então:

$$\begin{aligned} f\lambda_2^2 + g\lambda_3^2 &= f\lambda_2'^2 + g\lambda_3'^2 \\ f(\lambda_2^2 - \lambda_2'^2) + g(\lambda_3^2 - \lambda_3'^2) &= 0 \end{aligned}$$

Como f e g são não negativos, $\lambda_2 = \lambda_2'$ e $\lambda_3 = \lambda_3'$.

Agora suponha que λ_2 tenha a mesma multiplicidade de λ_3'. Assim:

$$f(\lambda_2^2 - \lambda_3'^2) + g(\lambda_3^2 - \lambda_2'^2) = 0 \tag{3.4}$$

Logo, $\lambda_2 = \lambda_3'$ e $\lambda_3 = \lambda_2'$.

Portanto, G e G' são co-espectrais. ■

Teorema 3. *Sejam G_1 e G_2 dois grafos simples, conexos e com mesmo índice. Se suas centralidades de autovetor forem proporcionais e distintas entre si então os grafos são isomorfos.*

Prova: Sejam $\vec{x^1}$ e $\vec{x^2}$ autovetores positivos associados, respectivamente, aos índices dos grafos G_1 e G_2. Suponha que $|V_1| = |V_2| = n$, $|E_1| = |E_2| = m$ e que $\vec{x^i} = (x_1^i, \ ...,x_n^i)$, tal que $x_j^i \neq x_k^i$, $j,k = 1, \ \ldots, n$, $j \neq k$ e $\ i = 1,2$. Suponha ainda que $\vec{x^1} = \vec{x^2}$ para alguma ordenação dos componentes de $\vec{x^i}$, $i = 1,2$. Pela definição de centralidade de autovetor em [Abreu et al., 2007], $\vec{x^i}$ define as centralidades de autovetor associadas aos vértices dos grafos G_i para alguma ordenação de V_i, $i = 1,2$. Desta forma supomos que G_1 e G_2 possuem as mesmas centralidades de autovetor para alguma ordenação de seus conjuntos de vértices. Seja:

$$f : V_1 \rightarrow V_2 \text{ uma função biunívoca, tal que } f(v) = w \text{ se, e somente se, } c(v) = c(w) \tag{3.5}$$

onde $v \in V_1$, $w \in V_2$ e $c(\cdot)$ representa a centralidade do vértice. Assim:

$$c(v) = \vec{x_j^1} = \vec{x_k^2} = c(w), \text{ para algum } \ j,k \in \{1, \ \ldots, n\} \tag{3.6}$$

Em [Bonacich, 2007], temos que:

$$\lambda x_i = \sum_{j=1}^{n} a_{ij}x_j, \ i = 1,\ldots,n, \text{ onde } a_{ij} \in A(G) \text{ e } \lambda \text{ é o índice de } G. \tag{3.7}$$

Desta forma podemos afirmar que a centralidade de um vértice pode ser obtida a partir do somatório das centralidades de seus vizinhos dividido pelo índice do grafo.

Assim, reescrevendo (3.7) na forma de matriz de G_1, temos:

$$\underbrace{\begin{bmatrix} a_{11}^1 & a_{12}^1 & \cdots & a_{1n}^1 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n1}^1 & a_{n2}^1 & \cdots & a_{nn}^1 \end{bmatrix}}_{A(G_1)} \begin{bmatrix} x_1^1 \\ x_2^1 \\ \vdots \\ x_n^1 \end{bmatrix} = \lambda \begin{bmatrix} x_1^1 \\ x_2^1 \\ \vdots \\ x_n^1 \end{bmatrix} \tag{3.8}$$

onde $A(G_1)$ é a matriz de adjacência de G_1 e, portanto, $a_{ii} = 0, \ i = 1,\ldots,n$.

Considere agora uma reordenação de $\vec{x^2}$ dada pela função f de acordo com (3.5) originando $\vec{x^2}'$. Sendo G_1 e G_2 de mesmo índice, $\lambda_1^1 = \lambda_1^2 = \lambda$. Então, $\lambda\vec{x^1} = \lambda\vec{x^2}$. Daí e de (3.7) temos:

$$\underbrace{\begin{bmatrix} a_{11}^2 & a_{12}^2 & \cdots & a_{1n}^2 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n1}^2 & a_{n2}^2 & \cdots & a_{nn}^2 \end{bmatrix}}_{A'(G_2)} \begin{bmatrix} x_1^{2'} \\ x_2^{2'} \\ \vdots \\ x_n^{2'} \end{bmatrix} = \lambda \begin{bmatrix} x_1^{2'} \\ x_2^{2'} \\ \vdots \\ x_n^{2'} \end{bmatrix} \tag{3.9}$$

Assim, de (3.8) e (3.9) temos que:

$$\begin{bmatrix} a_{11}^1 & a_{12}^1 & \cdots & a_{1n}^1 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n1}^1 & a_{n2}^1 & \cdots & a_{nn}^1 \end{bmatrix} = \begin{bmatrix} a_{11}^2 & a_{12}^2 & \cdots & a_{1n}^2 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n1}^2 & a_{n2}^2 & \cdots & a_{nn}^2 \end{bmatrix} \tag{3.10}$$

Ou seja, $A(G_1) = A^{'}(G_2)$, onde $A^{'}(G_2)$ decorre de uma permutação de linhas e colunas de $A(G_2)$ dada pela função f. Desta forma, todas as arestas de G_1 se preservam em G_2. Assim, $(v,u) \in E_1$ se, e somente se, $(f(v), f(u)) \in E_2$. Então, por definição, a função f define um mapeamento entre V_1 e V_2 que é um isomorfismo. Logo, G_1 e G_2 são isomorfos. ■

4 *Algoritmo para Detecção do Isomorfismo de Grafos*

No decorrer deste capítulo apresentaremos o algoritmo proposto neste trabalho, descrevendo as três fases que o compõe: cálculo dos índices dos grafos e dos autovetores associados, verificação da distinção das centralidades de um mesmo autovetor e a descida na árvore de busca.

4.1 Algoritmo Proposto

Como vimos, embora o PIG seja aplicado a diversos problemas práticos, a sua classe de complexidade ainda permanece uma incógnita. Apesar disto, encontramos na literatura algoritmos exatos para solucioná-lo, entre eles o algoritmo VF2 [Cordella et al., 2001], o Nauty [McKay, 1981], o SD [Schmidt e Druffel, 1976] e o proposto por [Dharwadker e Tevet, 2009]. Algumas metaheurísticas também foram implementadas para o mesmo fim, como o GRASP e o Algoritmo Genético [Boeres e Sarmento, 2005], e o Simulated Annealing [Xiutang e Kai, 2008].

Embora outros algoritmos que solucionam o PIG em seu caso geral já tenham sido propostos, neste trabalho apresentamos um novo algoritmo que o resolve utilizando propriedades da Teoria Espectral de Grafos, em especial os índices dos grafos e as centralidades de autovetor, descritos no Capítulo 3.

A intenção na utilização destas propriedades é contribuir para a redução do espaço de soluções do problema. Este espaço pode ser descrito, para grafos de n vértices, como sendo o conjunto de permutações dos n vértices de um dos grafos de entrada do problema, fixando os vértices do outro grafo, uma vez que o objetivo do PIG é encontrar uma associação biunívoca entre os vértices dos dois grafos de maneira a preservar suas adjacências. Assim, o tamanho do espaço de busca é igual a $n!$.

Deste modo, utilizamos "filtros" com o objetivo de eliminar soluções que não exibem efetivamente um isomorfismo entre os grafos, reduzindo assim o espaço de busca do problema. De

acordo com a literatura, consideramos apenas os grafos que têm o mesmo número de vértices e arestas, e a mesma sequência de graus, pois são estes os candidatos a serem topologicamente equivalentes. Consideramos como "filtro" uma condição que auxilia o algoritmo a tomar decisões com o objetivo de tornar mais simples a busca pela resposta do problema. Além destes filtros, nesta dissertação propomos a análise das componentes do autovetor associado ao índice do grafo. Com essa informação, geramos blocos de centralidades em ambos os grafos de entrada, tendo em um mesmo bloco vértices com centralidades proporcionais. Desta forma, a tentativa de associação fica restrita a vértices de mesmo bloco, sendo respeitados os graus dos vértices, pois segundo [Grassi et al., 2007], vértices com centralidades proporcionais podem ter graus diferentes, o que inviabiliza a associação. Para efeito de comparação das centralidades consideramos que dois autovetores são proporcionais quando um é múltiplo escalar do outro [Santos, 2006].

Esta estratégia de gerar blocos de centralidade é utilizada pelo algoritmo proposto, denominado AEPIG (Algoritmo Espectral para o Problema de Isomorfismo de Grafos), para gerar a árvore de soluções, que será descrita na Seção 4.1.3. Contudo, o pior caso para esta estratégia ocorre quando os grafos de entrada são grafos regulares, uma vez que, para grafos desta classe, as centralidades de autovetor são todas iguais, o que gera apenas um bloco para cada grafo, resultando em um conjunto de $n!$ possíveis soluções para o problema.

Assim, a fim de evitar a exploração da árvore de busca para a detecção do isomorfismo, o AEPIG implementa dois filtros exatos. O primeiro filtro exato fundamenta-se no Teorema 1 (enunciado na Seção 3.4), que apresenta o autovetor associado ao índice do grafo como uma invariante. O segundo filtro exato, baseado no Teorema 3 (enunciado na Seção 3.4), verifica a distinção das centralidades dos autovetores de ambos os grafos de entrada.

O algoritmo proposto é aplicado apenas sobre um par de grafos que possuírem o mesmo número de vértices, de arestas e mesma sequência de graus. Satisfeitas estas condições (invariantes), podemos dividir o algoritmo em três fases, as quais são descritas a seguir. A entrada do algoritmo são as matrizes de adjacência dos grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$.

4.1.1 Fase 1: Cálculo dos índices dos grafos e do autovetor associado

Nesta primeira fase ocorre o cálculo dos índices (λ_1^1 e λ_1^2) de cada grafo G_1 e G_2 de entrada do problema e dos seus respectivos autovetores associados ($\vec{x}^1$ e $\vec{x}^2$). Como foi visto, cada componente x_j^i, $i = 1,\ 2$, do autovetor associado ao índice é a centralidade do vértice v_j do grafo G_i, para alguma ordenação dos rótulos de seus vértices.

Após calcularmos os índices dos grafos e os autovetores associados, ordenamos as centra-

lidades dos vértices de cada grafo de maneira crescente e as comparamos proporcionalmente. Caso o vetor de centralidade x_j^1 do grafo G_1 seja proporcionalmente igual ao vetor de centralidade x_j^2 do grafo G_2, onde $j = 1, \ldots, n$, o algoritmo segue para a Fase 2, uma vez que há a possibilidade dos grafos serem isomorfos. Caso contrário, consideramos estes grafos não isomorfos. Estas decisões são baseadas no Teorema 1, o que faz esta primeira fase corresponder ao primeiro filtro exato do algoritmo.

4.1.2 Fase 2: Verificação da distinção das centralidades de um mesmo autovetor

Alcançando esta fase, os autovetores de ambos os grafos são igualmente proporcionais e estão ordenados de maneira crescente. Como a intenção é somente tentar associar vértices que tenham centralidades proporcionais, o objetivo desta fase é verificar se as centralidades de cada autovetor são distintas entre si. Neste caso, o número de blocos em cada grafo será igual ao número de vértices. Com isso, cada vértice de um grafo só poderá ser associado a um único vértice do outro grafo, respeitando a igualdade dos graus. Se este fato acontecer, de acordo com o Teorema 3, os grafos são considerados isomorfos. Caso contrário, o algoritmo avança para a Fase 3, onde fará a descida na árvore de busca de soluções do PIG. Assim, esta segunda fase corresponde ao segundo filtro exato do algoritmo.

4.1.3 Fase 3: Descida na Árvore de Busca

O algoritmo atinge este ponto quando o segundo filtro exato não pode ser aplicado, ou seja, se existe pelo menos uma repetição de algum valor dentre as centralidades de cada autovetor associados aos índices dos grafos de entrada. Neste caso é construída uma árvore de soluções baseada nos blocos de centralidades dos vértices, sendo explorada utilizando a estratégia de *backtracking*.

Na exploração da árvore o algoritmo tenta associar vértices que estejam em um mesmo bloco, que possuam o mesmo grau e que gerem associações entre arestas de mesmo valor. Esta última condição é para atender ao teorema (aqui denotado por Teorema PIG-PQA) proposto por [Lee, 2007], que será utilizado para avaliar a solução encontrada na descida da árvore. Caso estas três condições não sejam atendidas, o algoritmo termina a exploração daquele ramo da árvore e realiza *backtracking* para um nível acima, continuando a exploração a partir deste ponto.

Como a cada nível da árvore uma nova associação de vértices é inserida na solução, ela possui profundidade igual ao número de vértices dos grafos. Deste modo, quando um nó folha

é atingido, um isomorfismo entre os grafos é identificado, finalizando a execução do algoritmo. Caso contrário, quando a exploração da árvore termina no seu nó raiz, ou seja, quando todas as possibilidades de associação entre os vértices foram examinadas, porém sem sucesso, o algoritmo conclui que os grafos de entrada não são isomorfos. Então, para ratificar a solução encontrada, sendo os grafos isomorfos ou não, utilizamos o Teorema PIG-PQA, que enunciamos a seguir.

Teorema 4 (Teorema PIG-PQA)**.** *Considere G_1 e G_2 dois grafos que definem o PIG(G_1, G_2), onde G_1 e G_2 apresentem o mesmo número de vértices v, número de arestas n e graus dos vértices. Considerando os grafos $G_1^{'}$ e $G_2^{'}$ (grafos resultantes da transcrição dos grafos G_1 e G_2, respectivamente), seja o PQA$(G_1^{'}, G_2^{'})$ adaptado para o PIG(G_1, G_2). O valor da solução ótima do PQA$(G_1^{'}, G_2^{'})$ é igual ao número de arestas n do grafo G_1 (ou G_2) $\Leftrightarrow G_1 \approx G_2$.*

O referido teorema é a consolidação da reformulação do PIG como um Problema Quadrático de Alocação (PQA, [Koopmans e Beckmann, 1957]). Para isso, os grafos de entrada G_1 e G_2 são transcritos em grafos completos valorados $G_1^{'}$ e $G_2^{'}$, respectivamente, cujas arestas recebem valor 1 se existirem nos grafos originais, e valor 0 caso contrário. Com isso, a intenção é permitir apenas a associação entre arestas de $G_1^{'}$ e $G_2^{'}$ que tenham o mesmo valor, ou seja, associar uma aresta de valor 1 em $G_1^{'}$ a uma única aresta de valor 1 em $G_2^{'}$, procedendo da mesma maneira para arestas de valor 0. Com isso, o teorema garante que o valor da solução do PQA relativo aos grafos $G_1^{'}$ e $G_2^{'}$ deve ser igual ao número de arestas de G_1 e G_2 para que estes sejam isomorfos.

O Algoritmo 1 apresenta o pseudocódigo do algoritmo proposto. Para ilustrar os passos do algoritmo na detecção ou não do isomorfismo entre dois grafos, apresentamos um exemplo na seção seguinte.

4.1.4 Exemplo de Execução do Algoritmo Proposto

Como entrada para o algoritmo utilizaremos as matrizes de adjacência dos grafos G_1 e G_2 ilustrados na Figura 4.1. Ambos os grafos possuem seis vértices, nove arestas e mesma sequência de graus $(2,3,3,3,3,4)$. Deste modo, eles obedecem às primeiras invariantes necessárias para que dois grafos sejam isomorfos e o algoritmo execute a sua primeira fase. Nesta fase, os índices dos grafos e seus autovetores associados são calculados (linhas 1 e 2 do Algoritmo 1) para uma posterior comparação. Calculados os autovalores dos grafos, verificamos que eles são co-espectrais:

$$spect(G_1) = spect(G_2) = \begin{bmatrix} 3,08680 & 1,15580 & 0,10963 & -1 & -1,17357 & -2,17865 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Algoritmo 1: Algoritmo Espectral para o Problema de Isomorfismo de Grafos (AEPIG)

Entrada: As matrizes de adjacência dos grafos G_1 e G_2
Saída: Sim (se os grafos são isomorfos) ou Não (caso contrário)
1 Calcular os índices λ_1 e λ_2 de G_1 e G_2, respectivamente *//Fase 1*
2 Calcular os autovetores positivos $\vec{x}^1$ e $\vec{x}^2$ associados, respectivamente, à λ_1 e λ_2
3 Ordenar de maneira crescente os componentes dos autovetores
4 **se** *(*$\vec{x}^1 \neq k\vec{x}^2$, $k \in \mathscr{R}^*$*)* **então**
5 G_1 e G_2 não são isomorfos
6 **senão**
7 **se** *(*$\vec{x}^i = (x_1^i, ..., x_n^i)$, *tal que* $x_j^i \neq x_k^i$, $j,k = 1,\ldots,n$, $j \neq k$ *e* $i = 1,2$*)//Fase 2* **então**
8 G_1 e G_2 são isomorfos
9 **senão**
10 Calcular $G_1^{'}$ e $G_2^{'}$ a partir de G_1 e G_2, respectivamente *//Fase 3*
11 Realizar a descida na árvore de busca
12 **se** *(o Teorema PIG-PQA for satisfeito)* **então**
13 G_1 e G_2 são isomorfos
14 **senão**
15 G_1 e G_2 não são isomorfos
16 **fim se**
17 **fim se**
18 **fim se**

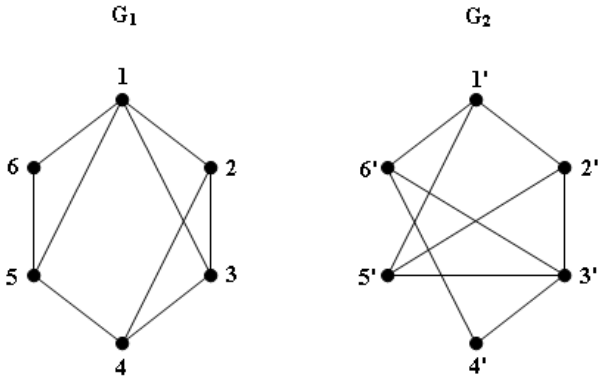

Figura 4.1: Grafos de entrada para o exemplo

Portanto, ambos possuem como índice:

$$\lambda_1^1 = \lambda_1^2 = 3,08680$$

Os autovetores associados aos índices de G_1 e G_2, respectivamente, já ordenados de maneira crescente (linha 3) e com seus vértices correspondentes são apresentados na Figura 4.2. Estando as centralidades em ordem crescente, elas são comparadas para verificar se são ou não proporcionalmente iguais (linha 4 - Primeiro Filtro Exato). Caso não sejam, o algoritmo termina a sua execução e os grafos são considerados não isomorfos. Porém, neste exemplo, as

centralidades são igualmente proporcionais ($k = 1$). Portanto, há a possibilidade de G_1 e G_2 serem isomorfos, o que faz o algoritmo seguir para a Fase 2.

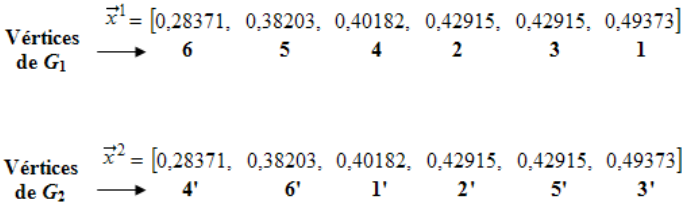

Figura 4.2: Autovetores associados aos índices de G_1 e G_2, respectivamente, já ordenados de maneira crescente e com seus vértices correspondentes

Alcançando a segunda fase, as centralidades são comparadas a fim de verificar se elas são distintas entre si (linha 7 - Segundo Filtro Exato), sendo já considerados isomorfos os grafos que possuírem esta característica. Contudo, podemos perceber que isto não acontece no exemplo (a centralidade $0,42915$ se repete), fazendo o algoritmo avançar para a Fase 3, onde será feita a busca na árvore de soluções.

Nesta última fase, aplica-se a transcrição (descrita na Seção 4.1.3) nos grafos de entrada para se obter os grafos completos valorados $G_1^{'}$ e $G_2^{'}$ (linha 11), que podem ser observados na Figura 4.3, cujas arestas de valor 0 (arestas de cor vermelha) são acrescentadas aos grafos originais para torná-los completos. As arestas originais do grafo (arestas de cor preta) recebem valor 1.

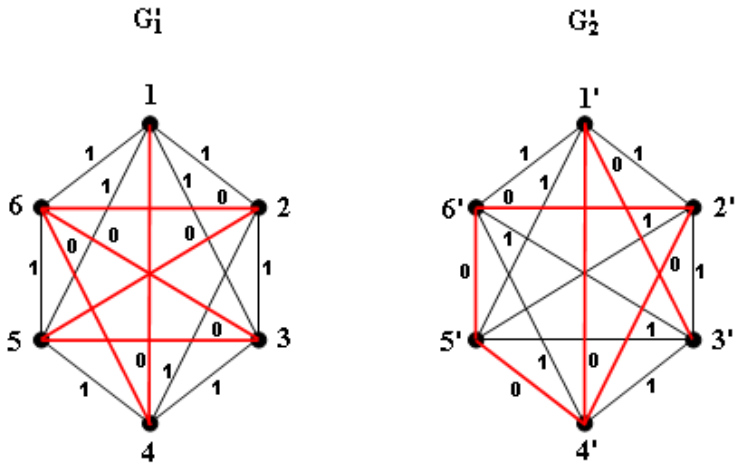

Figura 4.3: Grafos transcritos $G_1^{'}$ e $G_2^{'}$

Os blocos de centralidades são de extrema importância na Fase 3 para reduzir o espaço de solução do PIG, que são utilizados para gerar a árvore de soluções explorada pelo algoritmo na tentativa de encontrar um isomorfismo entre os grafos (linha 11). Os blocos de centralidades para cada grafo e a árvore de soluções podem ser vistos nas Figuras 4.4 e 4.5, respectivamente.

Como explicado anteriormente, vértices de mesma centralidade permanecem em um mesmo bloco. No exemplo, os vértices 2 e 3 de $G_1^{'}$ e os vértices $2'$ e $5'$ de $G_2^{'}$ estão no bloco D. Os demais permanecem um em cada bloco. Com isso foram criados cinco blocos de centralidades, uma vez que temos cinco centralidades de autovetor distintas para ambos os grafos.

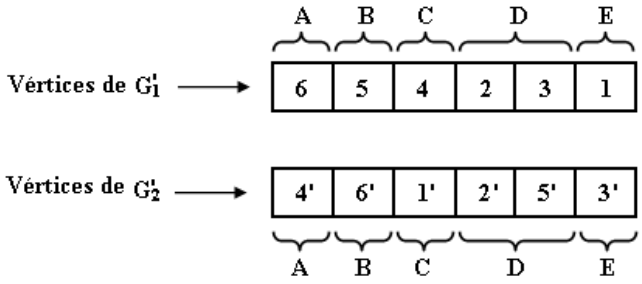

Figura 4.4: Blocos de centralidades

Analisando a árvore de soluções podemos perceber que, a menos dos vértices 2 e 3, cada vértice de $G_1^{'}$ é associado a um único vértice de $G_2^{'}$. Os vértices 2 e 3 podem ser associados aos vértices $2'$ ou $5'$. Com isso, mesmo os grafos possuindo 4 vértices de grau 3, as centralidades auxiliam a reduzir as possibilidades de associação destes vértices se comparado ao uso somente dos graus para guiar esta correspondência.

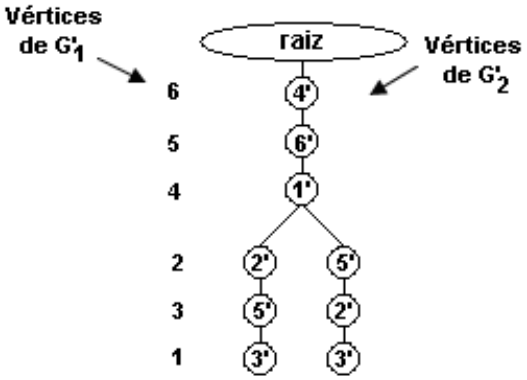

Figura 4.5: Árvore de soluções criada a partir dos blocos de centralidades

Deste modo, o algoritmo inicia a busca em árvore pela raiz e a cada nível da árvore ocorre uma nova tentativa de associação entre os vértices. A nova associação apenas será inserida na solução se ela agregar sobreposições de arestas com valores iguais de $G_1^{'}$ e $G_2^{'}$, e se os vértices forem de mesmo grau. A viabilidade das centralidades dos vértices é garantida pelo fato de associarmos apenas vértices pertencentes ao mesmo bloco.

Na primeira tentativa, o algoritmo tenta fazer a associação dos vértices 6 e $4'$, que é bem sucedida, pois os vértices têm o mesmo grau (grau 2) e estão no mesmo bloco de centralidade (centralidade igual a $0,28371$). Com isso, o vértice $4'$ é inserido na solução, sem a necessidade de analisar as sobreposições de arestas.

A segunda tentativa consiste em associar os vértices 5 e $6'$, que gera uma sobreposição de arestas de mesmo valor. Como esta associação atende as três condições (mesma centralidade, mesmo grau e sobreposições de arestas de valores iguais), ela é válida e o vértice $6'$ é inserido na solução.

O próximo passo do algoritmo é tentar estabelecer uma correspondência entre os vértices 4 e $1'$. Também nesta associação as centralidades e os graus dos vértices são preservados, e ocorre a sobreposição de arestas de valores equivalentes. Logo, esta correspondência é válida, sendo inserido na solução o vértice $1'$.

Para o próximo vértice, de acordo com a árvore, existem duas possibilidades de associação, os vértices $2'$ e $5'$. Como foi utilizada a busca em profundidade, o algoritmo tenta fazer a associação 2 e $2'$. Caso esta não fosse possível, o algoritmo tentaria a associação 2 e $5'$. Não sendo esta também válida, o algoritmo realizaria *backtracking*. Se sucessivos mecanismos de *backtracking* na árvore fizerem a busca da associação retornar à raiz, o algoritmo termina, sendo possível concluir que os grafos não são isomorfos.

No entanto, como a associação dos vértices 2 e $2'$ é possível, o algoritmo insere o vértice $2'$ na solução e segue a descida na árvore tentando uma associação entre os vértices 3 e $5'$ e posteriormente entre os vértices 1 e $3'$. Uma vez que estas duas associações obedecem às três condições, os vértices $5'$ e $3'$ são inseridos na solução.

Portanto, como podemos observar, descendo os seis níveis da árvore conseguimos associar todos os vértices do grafo $G_2^{'}$ aos vértices do grafo $G_1^{'}$, ou seja, encontramos um isomorfismo quando alcançamos um vértice folha da árvore. A Figura 4.6 mostra o vetor com os vértices pertencentes à solução, onde podemos perceber que os vértices visitados foram exatamente os que apresentaram um isomorfismo entre os grafos, não sendo necessário realizar nenhum *backtracking* para encontrar a solução.

Vetor Solução

4'	6'	1'	2'	5'	3'

Figura 4.6: Vetor com os vértices pertencentes à solução

Além disso, a fim de ratificar a resposta encontrada pela exploração da árvore de busca (se os grafos são ou não isomorfos), utilizamos o Teorema PIG-PQA (linha 12) descrito na Seção 4.1.3. No exemplo, o valor do mapeamento das sobreposições de arestas entre os grafos $G_1^{'}$ e $G_2^{'}$ foi igual ao número de arestas de G_1 e G_2, confirmando o resultado de isomorfismo apresentado pela descida na árvore de busca. Caso estes números não fossem iguais, os grafos não seriam isomorfos. Assim, após a avaliação deste teorema, o algoritmo termina a sua execução.

5 *Resultados Computacionais*

A fim de avaliarmos o desempenho do algoritmo AEPIG descrito no Capítulo 4, realizamos a comparação do tempo de processamento deste com os algoritmos propostos por [Lee, 2007], [Dharwadker e Tevet, 2009] e [Ullmann, 1976], e os algoritmos VF2 [Cordella et al., 2001] e Nauty [McKay, 1981]. Neste capítulo apresentamos os resultados desta comparação efetuada sobre parte da base de dados da biblioteca VFLib [Santo et al., 2003] e sobre pares de grafos não isomorfos.

O AEPIG foi implementado utilizando a Linguagem C, fazendo uso da biblioteca time.h para a obtenção dos tempos de execução. Os códigos dos cinco algoritmos citados anteriormente foram obtidos na literatura, com o objetivo de que todos, inclusive o AEPIG, fossem executados sob as mesmas condições de hardware. Assim, os testes foram realizados em um computador com processador Intel® Core^{TM}2 Duo E7500 de 2.93GHz, 3GB de memória RAM, Sistema Operacional Linux Ubuntu 9.10, kernel $2.6.31-20$ e compilador gcc versão 4.4.1. A função utilizada para o cálculo dos autovalores e autovetores dos grafos foi extraída da biblioteca CLAPACK versão 3.2.1 [Anderson et al., 1999].

5.1 Instâncias de Teste

Para a avaliação de desempenho entre os algoritmos, utilizamos 3.000 pares de grafos isomorfos e 11 pares não isomorfos, sendo 4 destes extraídos de [Dharwadker e Tevet, 2009] e os demais gerados aleatoriamente. Os pares isomorfos foram obtidos da biblioteca VFLib, que foi desenvolvida para servir de *benchmark* tanto para o Problema de Isomorfismo de Grafos quanto para o Problema de Isomorfismo de Subgrafos. Esta biblioteca é formada por pares de grafos divididos em categorias, que possui um total de 72.800 pares, sendo 18.200 pares isomorfos e 54.600 pares para os quais existe um isomorfismo de subgrafo.

Neste trabalho utilizamos a categoria dos grafos isomorfos conectados aleatoriamente, que possui 3.000 pares de grafos divididos em três grupos de densidade $\eta = 0,1$, $\eta = 0,05$ e $\eta = 0,01$. Cada grupo possui 100 pares de grafos (que chamamos de instâncias) de tamanhos 20,

40, 60, 80, 100, 200, 400, 600, 800 e 1.000 vértices, totalizando 1.000 pares de grafos por grupo. As arestas dos grafos desta categoria conectam vértices sem qualquer regra determinada, assumindo que a probabilidade de uma aresta conectar dois vértices não depende destes.

Na geração dos grafos desta base foi fixado o valor η de probabilidade de uma aresta estar presente entre dois vértices distintos do grafo, tendo assumida como uniforme a distribuição de probabilidade. Este parâmetro η significa que o número de arestas será igual a $\eta.n.(n-1)$, onde n é o número total de vértices do grafo. Porém, se o grafo obtido com este número for desconexo, mais arestas são devidamente inseridas até a geração de um grafo conexo.

5.2 Algoritmos Exatos VF2, Nauty e o proposto por Ullmann

Para analisar a eficiência do AEPIG, o comparamos com outros cinco algoritmos que também tratam o Problema de Isomorfismo de Grafos (os algoritmos propostos por [Lee, 2007], [Dharwadker e Tevet, 2009] e [Ullmann, 1976], e os algoritmos Nauty [McKay, 1981] e VF2 [Cordella et al., 2001]). Nesta seção descrevemos os algoritmos VF2, Nauty e o proposto por Ullmann, uma vez que os demais foram apresentados na Seção 2.2.

O algoritmo VF2 foi desenvolvido por [Cordella et al., 2001] como uma segunda versão do algoritmo VF, também proposto pelos mesmos autores. A sua principal vantagem em relação ao seu antecessor é a utilização de estruturas de dados adequadas a fim de reduzir a requisição de memória durante a sua execução, possibilitando-o ser aplicado a grafos de ordem grande.

Este algoritmo realiza uma busca enumerativa no espaço de soluções, utilizando o método de busca em profundidade para a exploração deste espaço. Ele trabalha com um conjunto M, inicialmente vazio, de pares ordenados que representam o mapeamento entre os grafos de entrada, ou seja, cada par representa a correspondência de um vértice do primeiro grafo com um vértice do segundo grafo.

A cada passo, o algoritmo tenta incluir um novo par ao conjunto M. Os pares candidatos a inclusão são gerados a partir de regras que consideram as ligações dos vértices que já estão em M com os vértices que ainda não estão. Assim, um destes pares é inserido em M caso uma função de viabilidade seja atendida, o que gera o cálculo de novos pares candidatos. Caso contrário, o algoritmo abandona a exploração daquele caminho, desfazendo-se de algumas associações já realizadas, e segue para o caminho seguinte na árvore de busca. Ao final de sua execução, sendo os grafos isomorfos, o algoritmo apresenta o conjunto M, que possui uma correspondência entre os vértices dos grafos.

O algoritmo Nauty apresentado por [McKay, 1981] tem a particularidade de determinar

uma rotulação canônica nos grafos de entrada, que é um particionamento ordenado dos vértices, onde vértices de mesmo rótulo pertencem à mesma partição. Com isso, o algoritmo constrói uma árvore de busca, onde cada nó corresponde a uma partição dos grafos de entrada.

Caso uma partição contenha dois ou mais vértices, o algoritmo utiliza sucessivos procedimentos de refinamento na tentativa de distinguir estes vértices, buscando assim gerar nós únicos na árvore de busca. Além disso, faz uso da técnica de busca em profundidade para percorrer a árvore a fim de encontrar uma correspondência entre os vértices dos grafos. Todo este processo é realizado em ambos os grafos de entrada, que são considerados isomorfos se, e somente se, possuírem a mesma rotulação canônica.

O algoritmo proposto por Ullmann [Ullmann, 1976] também utiliza a técnica de descida em profundidade para explorar o espaço de soluções. Trabalhando com os dois grafos de entrada ao mesmo tempo, ele define uma matriz de isomorfismo para armazenar os vértices associados durante a busca, além de utilizar as matrizes de adjacências de ambos os grafos.

Esta matriz de isomorfismo possui apenas células com valores 1 e 0, sendo a intenção do algoritmo definir uma célula de valor 1 por linha, de maneira que em cada coluna haja apenas uma célula deste valor ao final da sua execução. Ele explora as adjacências dos vértices para reduzir o espaço de busca. Como cada linha da matriz de isomorfismo representa uma associação de vértices, a cada nível da árvore uma linha é definida e antes de seguir para o nível seguinte, o algoritmo descarta as associações de vértices não adjacentes aos recém-associados com outros adjacentes a eles.

Quando todas as células de alguma linha da matriz de isomorfismo estiverem com valor 0, o algoritmo termina a busca em um ramo da árvore. A altura máxima da árvore de busca será igual ao número de vértices dos grafos. Alcançando uma folha, o algoritmo finaliza sua execução, uma vez que encontrou uma correspondência entre os vértices. Caso sejam esgotadas todas as possibilidades de associações, os grafos são considerados não isomorfos.

5.3 Análise dos Resultados

Para analisar os resultados, a partir da base de dados de grafos isomorfos, nomeamos cada grupo de grafos de acordo com a sua densidade de arestas, sendo $r01$ para grafos com $\eta = 0,1$, $r005$ para os de $\eta = 0,05$ e $r001$ para aqueles de $\eta = 0,01$. Como explicado anteriormente, os grafos destes grupos têm tamanhos 20, 40, 60, 80, 100, 200, 400, 600, 800 e 1.000 vértices, sendo 100 pares de cada tamanho, com um total de 1.000 pares de grafos em cada grupo. Para todos os testes realizados, o tempo máximo de processamento considerado foi de uma hora, sendo cancelada a execução do algoritmo que ultrapassasse este limite. Denotamos por Lee e

DT os algoritmos propostos, respectivamente, por [Lee, 2007] e [Dharwadker e Tevet, 2009].

5.3.1 Grafos Isomorfos

A seguir apresentamos os resultados computacionais sobre a base de grafos isomorfos, sendo exibidos por meio de gráficos. As tabelas com as informações referentes aos gráficos são apresentadas no Apêndice A.

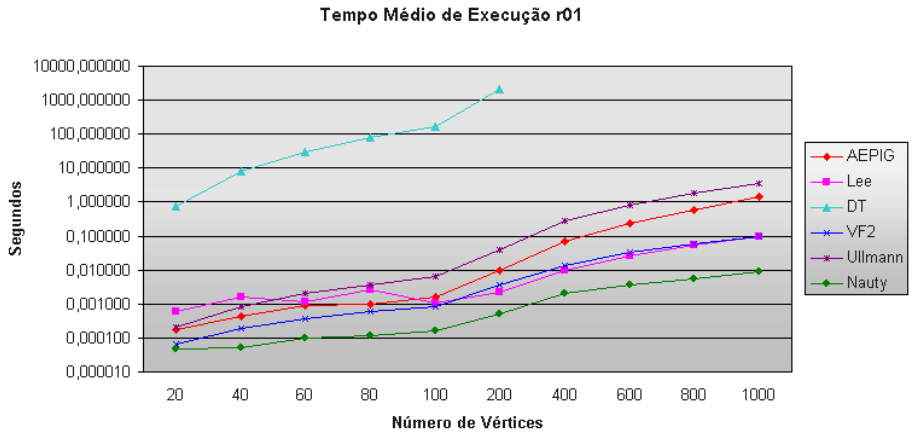

Figura 5.1: Gráfico das médias do tempo de execução dos algoritmos para as instâncias do grupo *r*01

A Figura 5.1 apresenta o gráfico com as médias do tempo de processamento dos algoritmos AEPIG, Lee, DT, VF2, Ullmann e Nauty para os 1.000 pares de grafos do grupo *r*01. Analisando o gráfico, podemos notar que o algoritmo Nauty possuiu menor tempo de execução que os demais para todas as instâncias do grupo. Vemos também que o algoritmo AEPIG apresenta melhor desempenho que o algoritmo Lee para grafos de até 80 vértices e para todos os grafos, se comparado com os algoritmos Ullmann e DT. Este último teve suas execuções canceladas por limite de tempo para grafos de ordem maior que 200 vértices. O algoritmo Lee possui tempo de processamento menor que o algoritmo VF2 para as instâncias acima de 100 vértices, ficando, a partir daí, com tempo maior apenas em relação ao algoritmo Nauty.

As médias do tempo de execução dos algoritmos para cada conjunto de instâncias do grupo *r*005 são apresentadas na Figura 5.2. Nela observamos que, para todos os grafos deste grupo, novamente o algoritmo Nauty foi o que obteve melhor desempenho, seguido pelo algoritmo VF2. O algoritmo AEPIG apresentou menor tempo de processamento que os algoritmos Ullmann e DT para todas as instâncias do grupo e para as de até 400 vértices se comparado com o algoritmo Lee, que obteve desempenho pior que o algoritmo DT nas instâncias de ordem 80.

O algoritmo DT apresentou resultado, dentro do limite de tempo estipulado, somente para grafos de até 200 vértices, sendo este o algoritmo que apresentou pior desempenho nos testes

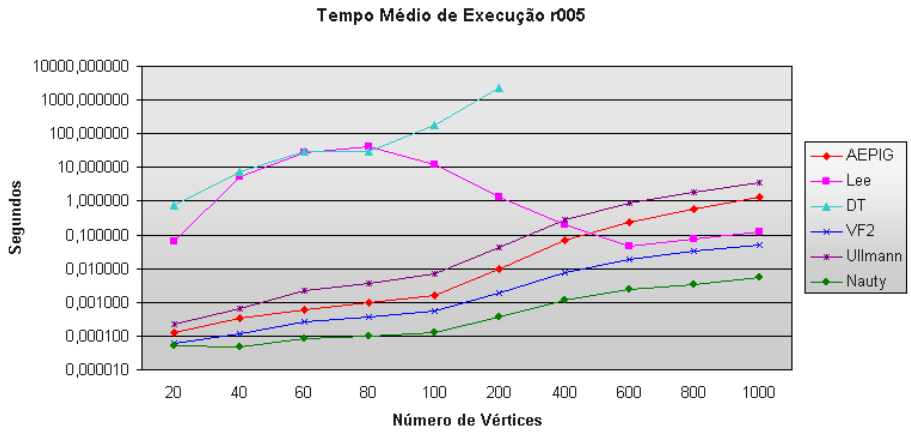

Figura 5.2: Gráfico das médias do tempo de execução dos algoritmos para as instâncias do grupo *r*005

deste grupo.

A Figura 5.3 mostra o gráfico comparativo das médias do tempo de execução dos algoritmos para os grafos do grupo *r*001. Podemos observar que o algoritmo Nauty obteve maior tempo de processamento que o algoritmo VF2 apenas nos grafos de ordem 20, tendo apresentado os melhores resultados para as demais instâncias do grupo. Como ocorrido com testes nos grupos anteriores, o algoritmo AEPIG apresentou tempo de execução inferior aos algoritmos Ullmann e DT para todos os grafos. Este comportamento também foi observado em relação ao algoritmo Lee e para os grafos de 60 vértices se comparado com o algoritmo VF2. Os algoritmos Lee e DT tiveram suas execuções canceladas para os grafos acima de 20 e 100 vértices, respectivamente, por excederem o limite de tempo estipulado. Além disso, para o par de grafos de 20 vértices deste grupo, exibido na Figura 5.4, o algoritmo DT apresentou um resultado incorreto, indicando que o par não era isomorfo, demonstrando assim que ele não é necessário e suficiente para o isomorfismo entre dois grafos, como afirmam os autores do trabalho.

Analisando os resultados apresentados, verificamos que o algoritmo Nauty obteve os melhores resultados para todas as instâncias testadas, com exceção apenas dos grafos de ordem 20 do grupo *r*001, para os quais o algoritmo VF2 apresentou menores tempos de execução. Observamos também que o algoritmo AEPIG tem tempo de processamento menor do que os algoritmos Ullmann e DT em todos os conjuntos de instâncias, para todos os valores de densidade, e que este último apresenta o pior desempenho entre os algoritmos para todos os grupos testados.

Além disso, comparando os algoritmos AEPIG e Lee, vemos que o tempo de execução do primeiro torna-se menor à medida que a densidade de arestas dos grafos diminui. Isso pode ser

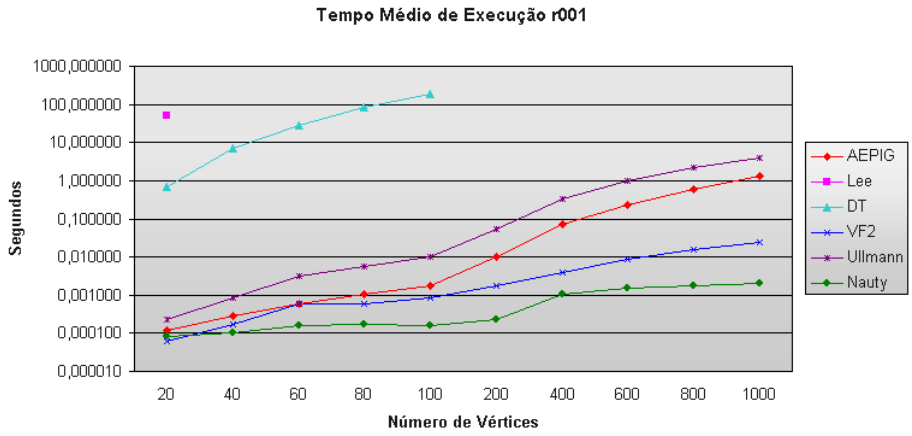

Figura 5.3: Gráfico das médias do tempo de execução dos algoritmos para as instâncias do grupo *r*001

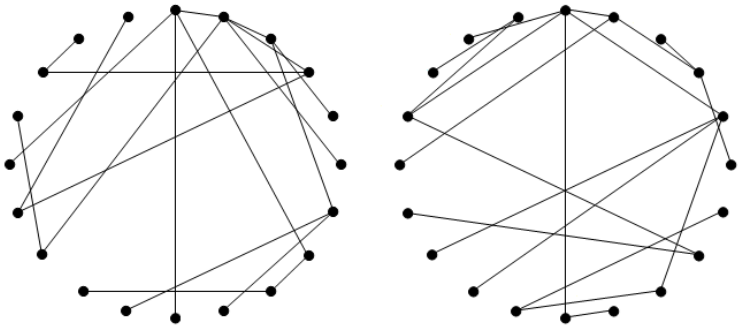

Figura 5.4: Par de grafos do grupo *r*001 para o qual o algoritmo DT apresentou resultado incorreto

explicado pelo fato do AEPIG gerar blocos de vértices de centralidades proporcionais a fim de conduzir eficientemente a descida na árvore de busca (descrita na Seção 4.1.3), reduzindo assim o espaço de soluções do problema. Com a mesma finalidade, o algoritmo desenvolvido por Lee gera blocos de vértices de mesmo grau. Com isso, em ambos os algoritmos, um vértice de um grafo somente pode ser associado a um vértice de outro grafo se eles estiverem em um mesmo bloco (de centralidade e de grau, respectivamente).

Desta forma, o número de blocos gerados influencia a complexidade da busca pela solução, pois quanto maior é este número, menor será o número de vértices por bloco, ocasionando uma diminuição do espaço de soluções viáveis do problema, consequentemente melhorando os tempos de processamento dos algoritmos. Podemos visualizar este comportamento analisando os resultados apresentados na Tabela 5.1. Nela verificamos que o algoritmo AEPIG gera mais blocos de centralidade do que o algoritmo Lee gera blocos de graus, tendo assim a descida na árvore de busca mais eficiente, caso seja necessária a sua utilização. Em média, nos testes para todos os conjuntos de densidades, o número de blocos de centralidades distintas foi maior que o de graus distintos. É possível observar que para o grupo *r*001, o número de blocos de

	*r*01		*r*005		*r*001	
Instâncias	BC	BG	BC	BG	BC	BG
20	20	7	19	5	16	4
40	40	10	40	8	34	6
60	60	14	60	11	55	7
80	80	16	80	13	75	7
100	100	18	100	14	97	8
200	200	27	200	22	200	11
400	400	41	400	33	400	17
600	600	52	600	41	600	21
800	800	62	800	48	800	25
1000	1000	70	1000	55	1000	28

Tabela 5.1: Número médio de blocos de centralidades (BC) e de graus (BG) dos algoritmos AEPIG e Lee, respectivamente

graus é bastante inferior, o que explica o tempo inviável (ou seja, superior ao limite de tempo estipulado) para o algoritmo Lee neste conjunto de instâncias.

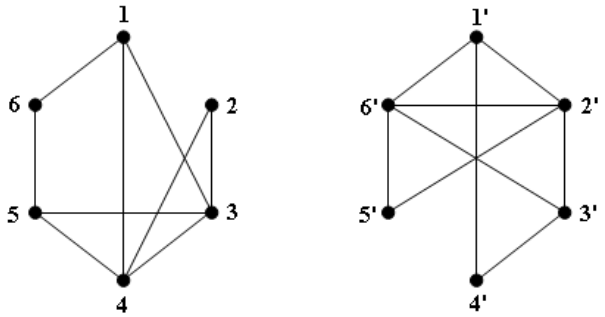

Figura 5.5: Grafos G_1 e G_2

Para exemplificar como o espaço de soluções para o problema está relacionado aos blocos produzidos pelos algoritmos AEPIG e Lee, observe os grafos da Figura 5.5. Os autovetores associados aos índices de G_1 e G_2, respectivamente, são:

$$\vec{x}^1 = [0,39780,\ 0,31955,\ 0,50949,\ 0,50949,\ 0,39780,\ 0,24950]$$

e

$$\vec{x}^2 = [0,39780,\ 0,50949,\ 0,39780,\ 0,24950,\ 0,31955,\ 0,50949]$$

Deste modo, o número de blocos gerados a partir das centralidades dos autovetores e da sequência de graus dos vértices é igual a 4 e 3, respectivamente, para ambos os grafos. Assim, baseado nestes blocos, é possível criar as árvores de soluções com os vértices de G_2, fixando os vértices de G_1 em ordem crescente das centralidades, podendo ser vistas na Figura 5.6(a) e Figura 5.6(b), respectivamente. Portanto, como já mencionado, a busca pela resposta do

problema na árvore de soluções, sendo ela necessária, é altamente influenciada pela quantidade de blocos gerados.

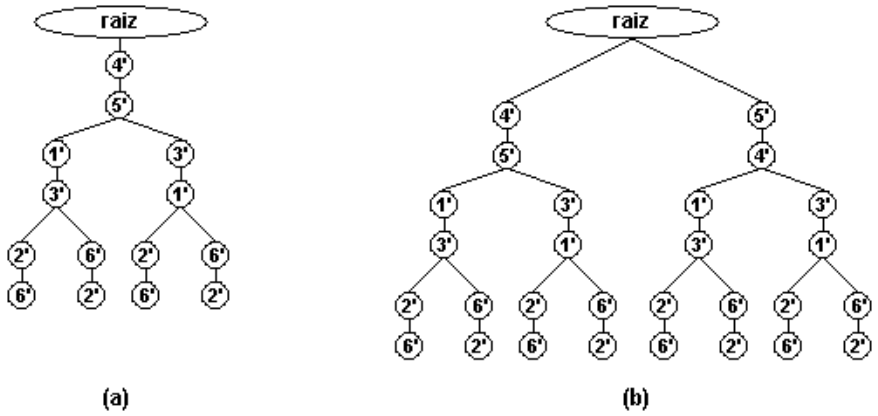

Figura 5.6: Árvores de solução para o isomorfismo entre G_1 e G_2 geradas a partir dos blocos de centralidades (a) e dos blocos de graus (b)

5.3.2 Grafos Não Isomorfos

Finalizada a análise dos testes realizados sobre a base de grafos isomorfos, apresentamos os resultados computacionais da execução dos algoritmos sobre 11 pares de grafos conhecidamente não isomorfos. Os resultados serão demonstrados por meio de gráficos e as tabelas com as informações referentes à eles são apresentadas no Apêndice A.

A Figura 5.7 exibe os resultados obtidos da execução dos algoritmos sobre quatro destes pares, extraídos de [Dharwadker e Tevet, 2009], que possuem, respectivamente, 20, 25, 25 e 40 vértices. Para a instância de ordem 20, o algoritmo que apresentou melhor desempenho foi o Nauty, seguido do VF2 e AEPIG, respectivamente. Com pequena diferença à frente do algoritmo VF2, o Nauty também foi o que obteve menor tempo de processamento para o segundo grafo. Já para o grafo Mathon, o algoritmo Ullmann obteve o melhor comportamento, seguido dos algoritmos VF2 e AEPIG, nesta ordem. Para esta instância, o algoritmo Lee ultrapassou o limite de tempo pré-estabelecido, tendo a sua execução cancelada. Isto pode ser explicado pelo fato desta instância possuir uma certa homogeneidade em relação aos graus dos vértices, tendo 10 vértices de grau 6 e os demais de grau 4. Com isso, o algoritmo gera apenas dois blocos de graus, tornando custosa a busca em árvore. Para o grafo de 40 vértices, novamente o algoritmo Nauty executou em menor tempo que os demais, ficando os algoritmos VF2 e Ullmann, respectivamente, na ordem de desempenho.

Os resultados dos testes realizados sobre os grafos não isomorfos gerados aleatoriamente podem ser vistos na Figura 5.8. As instâncias deste grupo obedecem a seguinte nomenclatura:

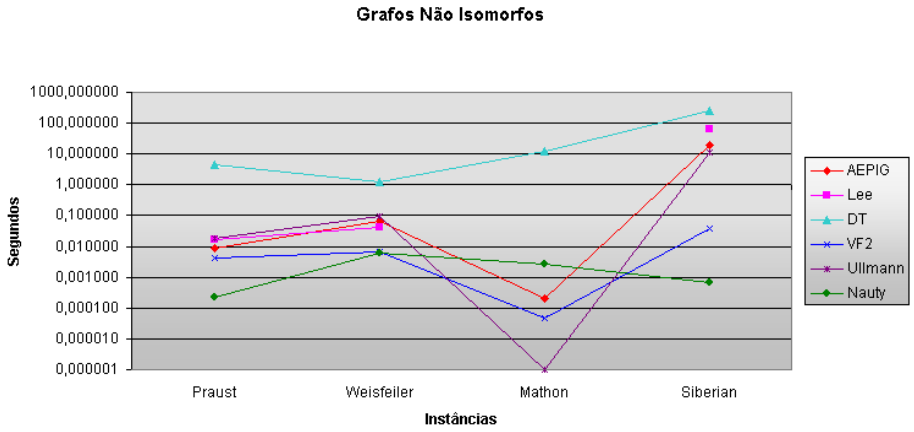

Figura 5.7: Tempo (em segundos) dos algoritmos para instâncias não isomorfas extraídas de [Dharwadker e Tevet, 2009]

gZW, onde Z representa o número de vértices do grafo e W o seu identificador $(A, B, \ldots, H)$ formando os pares $A-B$, $C-D$, $E-F$ e $G-H$.

Para o primeiro par de grafos com 6 vértices, o algoritmo com menor tempo de processamento é o VF2, seguido dos algoritmos Nauty e AEPIG. Para o segundo par de ordem 6, o algoritmo Lee obteve melhor desempenho que os demais, e os algoritmos Nauty e AEPIG possuem pequena diferença nos tempos de execução, sendo que o primeiro obteve tempo menor de processamento. Para os grafos com 7 vértices, o algoritmo VF2 apresentou tempo de execução inferior a todos os outros, contudo, o algoritmo Lee executa muito próximo do seu tempo. Este último conseguiu melhores resultados para grafos de 8 vértices, a exceção do par $C-D$, onde o algoritmo VF2 teve o melhor comportamento. Para as duas últimas instâncias, estes dois algoritmos apresentaram pequena diferença nos tempos de processamento.

Nos testes com as instâncias $A-B$, $E-F$ e $G-H$, todas com 8 vértices, o primeiro filtro exato do algoritmo AEPIG (descrito na Seção 4.1.1) indicou que os grafos tinham autovetores diferentes, portanto eles foram considerados não isomorfos.

Analisando os resultados dos testes sobre os dois grupos de grafos não isomorfos, percebemos que o algoritmo Nauty foi o que obteve melhores resultados para as instâncias do primeiro grupo, porém este comportamento não se manteve para o grupo dos grafos gerados aleatoriamente, onde ele não foi melhor em nenhuma das instâncias testadas. Neste segundo grupo, o algoritmo Lee prevaleceu sobre os demais, tendo o algoritmo VF2 alcançado menores tempos para as instâncias $g6A-g6B$, $g7A-g7B$ e $g8C-g8D$. Para ambos os grupos, o algoritmo DT foi o que apresentou o pior desempenho entre todos os algoritmos.

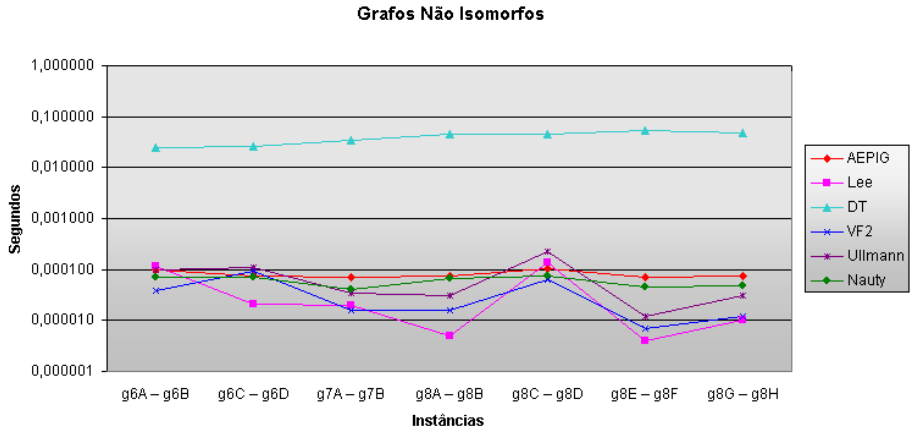

Figura 5.8: Tempo (em segundos) dos algoritmos para instâncias não isomorfas geradas aleatoriamente

6 *Conclusão e Trabalhos Futuros*

Neste trabalho investigamos a utilização de conceitos da Teoria Espectral de Grafos a fim de auxiliar a construção de algoritmos que solucionem o Problema de Isomorfismo de Grafos. Três resultados teóricos que consideram informações do espectro dos grafos e das centralidades de autovetor foram apresentados. Além disso, foi proposto o algoritmo AEPIG para detecção de isomorfismo de grafos baseado em dois destes resultados.

Validamos dois filtros exatos que auxiliam o AEPIG na busca pela solução do problema, sendo o primeiro fundamentado no Teorema 1 (Seção 3.4), o qual afirma que grafos isomorfos possuem as centralidades de autovetor proporcionalmente iguais, e o segundo é baseado no Teorema 3 (Seção 3.4), indicando que grafos com centralidades proporcionais, porém distintas entre si, são isomorfos.

Realizamos uma análise comparativa de desempenho entre o AEPIG e outros cinco algoritmos exatos: os algoritmos propostos por [Ullmann, 1976], [Dharwadker e Tevet, 2009] e [Lee, 2007], e os algoritmos VF2 [Cordella et al., 2001] e Nauty [McKay, 1981]. Aplicamos estes algoritmos sobre parte da base de grafos isomorfos da biblioteca VFLib [Santo et al., 2003] e sobre pares de grafos não isomorfos (alguns extraídos de [Dharwadker e Tevet, 2009] e outros gerados aleatoriamente).

A partir dos resultados obtidos, observamos que o número de blocos de centralidades gerados está relacionado com a densidade de arestas dos grafos, uma vez que há a diminuição deste número com a redução da densidade. Outro fator influenciador na geração destes blocos é a regularidade com relação aos graus dos vértices, pois grafos regulares possuem todas as centralidades de autovetor iguais, resultando na criação de apenas um bloco. Deste modo, a complexidade da busca pela solução é influenciada pelo número de blocos, pois quanto maior é este número, menor será o número de vértices por bloco, ocasionando uma diminuição do espaço de soluções viáveis do problema, consequentemente melhorando os tempos de processamento dos algoritmos.

Comparando o desempenho dos algoritmos nos testes com grafos isomorfos, verificamos que o algoritmo Nauty obteve os melhores resultados para todas as instâncias testadas, exceto

para os grafos de 20 vértices do grupo $r001$, que teve o algoritmo VF2 com os menores tempos de processamento. Observamos também que o algoritmo AEPIG tem tempo de execução inferior aos algoritmos Ullmann e DT em todos os conjuntos de instâncias, para todos os valores de densidade. Este último apresentou o pior desempenho entre os algoritmos, não executando para grafos maiores que 200 vértices em todos os grupos testados, bem como apresentou resultado incorreto para um par de grafos de 20 vértices do grupo $r001$, indicando que o par não era isomorfo, demonstrando assim que ele não é necessário e suficiente para o isomorfismo entre dois grafos.

Analisando os resultados dos testes realizados sobre os dois grupos de grafos não isomorfos, vimos que o algoritmo Nauty apresentou os menores tempos de execução para as instâncias extraídas de [Dharwadker e Tevet, 2009], porém isto não se repetiu para o grupo dos grafos gerados aleatoriamente, onde ele não foi melhor em nenhuma das instâncias. Neste segundo grupo, o algoritmo Lee prevaleceu sobre os demais, tendo o algoritmo VF2 alcançado tempos inferiores para alguns grafos. Para ambos os grupos, novamente o algoritmo DT foi o que apresentou o pior desempenho entre todos os algoritmos.

Além disso, comparando os algoritmos AEPIG e Lee, vemos que o tempo de execução do primeiro torna-se menor à medida que a densidade de arestas dos grafos diminui. Isso pode ser explicado pelo fato de ele gerar mais blocos de centralidade do que o algoritmo Lee gera blocos de graus, tendo assim a descida na árvore de busca mais eficiente, caso seja necessária a sua utilização. Em média, nos testes para todos os conjuntos de densidades, o número de blocos de centralidades foi bastante superior ao de graus distintos.

Ainda de acordo com os testes, verificamos que um gargalo computacional do algoritmo AEPIG é a função para cálculo dos autovalores e autovetores dos grafos. A função utilizada oriunda da biblioteca CLAPACK [Anderson et al., 1999], contribuiu com uma parcela considerável no tempo total de processamento do algoritmo, sendo responsável, em média, por 90% deste tempo, uma vez que na maioria dos testes a Fase 3 não foi executada, ou seja, a árvore de busca de soluções não precisou ser gerada. Mais precisamente, dos 1.000 testes realizados com o AEPIG para o conjunto de instâncias $r01$, em apenas $0,7\%$ delas a árvore de busca foi necessária para encontrar o isomorfismo. Para o conjunto $r005$, apenas $7,1\%$ e, finalmente, no último conjunto ($r001$) foi necessária a geração da árvore para um número maior de instâncias, $52,3\%$.

Portanto, concluímos que a utilização de propriedades da TEG para a resolução do PIG é vantajosa, visto que o Teorema 3 se mostrou poderoso na detecção do isomorfismo para a grande maioria dos testes realizados. Nos outros casos, a exploração da árvore de busca de soluções guiada pelos blocos de centralidades se mostrou bastante eficiente, principalmente

quando tratamos de grafos com uma grande diversidade de graus de vértices.

Como propostas para trabalhos futuros, pretendemos investigar na literatura funções mais eficientes para o cálculo de autovalores e autovetores, na tentativa de melhorar o desempenho do algoritmo proposto, bem como compará-lo com os demais algoritmos sobre uma base mais extensa de grafos não isomorfos.

Referências Bibliográficas

[Abreu, 2005] Abreu, N. M. M. (2005). Teoria espectral dos grafos: um híbrido entre a álgebra linear e a matemática discreta e combinatória com origens na química quântica. *Tendências em Matemática Aplicada e Computacional*, 6(1):1–10.

[Abreu et al., 2007] Abreu, N. M. M., Del-Vecchio, R. R., Vinagre, C. T. M. e Stevanović, D. (2007). In *Introdução à Teoria Espectral de Grafos com Aplicações*, Notas em Matemática Aplicada. Sociedade Brasileira de Matemática Aplicada e Computacional.

[Anderson et al., 1999] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. e Sorensen, D. (1999). *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition.

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

[Boeres e Sarmento, 2005] Boeres, M. C. S. e Sarmento, R. A. (2005). O problema de isomorfismo de grafos e sua resolução como um caso especial do problema de correspondência de grafos através dos algoritmos grasp e genético. In *Anais do XXXVII Simpósio Brasileiro de Pesquisa Operacional*, pages 1385–1396, Gramado - RS. XXXVII SBPO - Simpósio Brasileiro de Pesquisa Operacional.

[Boldrini et al., 1986] Boldrini, J. L., Costa, S. I. R., Figueiredo, V. L. e Wetzler, H. G. (1986). *Álgebra Linear*. Harbra, São Paulo, 3ª edition.

[Bonacich, 2007] Bonacich, P. (2007). Some unique properties of eigenvector centrality. *Social Networks*, 29(4):555–564.

[Bondy e Murty, 1979] Bondy, J. A. e Murty, U. S. R. (1979). *Graph Theory with Applications*. American Elsevier.

[Bunke, 2000] Bunke, H. (2000). Graph matching: Theoretical foundations, algorithms, and applications. *International Conference on Vision Interface*, pages 82–88.

[Cameron, 2001] Cameron, P. J. (2001). Strongly regular graphs. In *Topics in Algebraic Graph Theory*, pages 203–226. Cambridge University Press.

[Cordella et al., 1999] Cordella, L. P., Foggia, P., Sansone, C. e Vento, M. (1999). Performance evaluating of the vf graph matching algorithm. In *Proc. of the 10th International Conference on Image Analysis and Processing*, pages 1172–1177. IEEE Computer Society Press.

[Cordella et al., 2000] Cordella, L. P., Foggia, P., Sansone, C. e Vento, M. (2000). Fast graph matching for detecting cad image components. In *Proc. of the 15th International Conference on Pattern Recognition*, volume 2, pages 1038–1041. IEEE Computer Society Press.

[Cordella et al., 2001] Cordella, L. P., Foggia, P., Sansone, C. e Vento, M. (2001). An improved algorithm for matching large graphs. In *In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159.

[Dalcumune, 2008] Dalcumune, E. (2008). Algoritmos quânticos para o problema do isomorfismo de grafos. Master's thesis, Laboratório Nacional de Computação Científica, Petrópolis.

[Dharwadker e Tevet, 2009] Dharwadker, A. e Tevet, J. (2009). The graph isomorphism algorithm. In *Proceedings of the Structure Semiotics Research Group*. Eurouniversity Tallinn.

[Diestel, 2005] Diestel, R. (2005). *Graph Theory*, volume Electronic Edition of *Graduate Texts in Mathematics*. Springer-Verlag Heidelberg, New York, 3 edition.

[Foggia et al., 2001] Foggia, P., Sansone, C. e Vento, M. (2001). A performance comparison of five algorithms for graph isomorphism. *Proc. of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*, pages 188–199.

[Fortin, 1996] Fortin, S. (1996). The graph isomorphism problem. Technical report, University of Alberta, Edmonton, Alberta, Canada.

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

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

[Horn e Johnson, 1985] Horn, R. A. e Johnson, C. R. (1985). *Matrix Analysis*. Cambridge University Press, Cambridge.

[Hückel, 1931] Hückel, E. (1931). Quantentheoretische beiträge zum benzolproblem. *Z. Phys.*, 70(3–4):204–286.

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

[Koopmans e Beckmann, 1957] Koopmans, T. C. e Beckmann, M. J. (1957). Assignment problems and the location of economics activities. *Econometrica*, 25:53–76.

[Lee, 2007] Lee, L. (2007). Reformulação do problema de isomorfismo de grafos como um problema quadrático de alocação. Master's thesis, Universidade Federal do Espírito Santo - Centro Tecnológico - Departamento de Informática, Vitória.

[McKay, 1981] McKay, B. D. (1981). Practical graph isomorphism. In *Congressus Numerantium*, volume 30, pages 45–87.

[Messmer e Bunke, 1995] Messmer, B. T. e Bunke, H. (1995). Subgraph isomorphism in polynomial time. Technical report, Institute of Computer Science and Applied Mathematics, University of Bern.

[Nandi, 2006] Nandi, R. C. (2006). Isomorfismo de grafos aplicado à comparação de impressões digitais. Master's thesis, Universidade Federal do Paraná.

[Oliveira e Greve, 2005] Oliveira, M. O. e Greve, F. G. (2005). Um novo algoritmo de refinamento para testes de isomorfismo em grafos. *XXV Congresso da Sociedade Brasileira de Computação*.

[Santo et al., 2003] Santo, M. D., Foggia, P., Sansone, C. e Vento, M. (2003). A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recogn. Letters*, 24(8):1067–1079.

[Santos, 2006] Santos, R. J. (2006). *Álgebra Linear e Aplicações*. Imprensa Universitária da UFMG, Belo Horizonte, 1ª edition.

[Santos, 2009] Santos, R. J. (2009). *Introdução à Álgebra Linear*. Imprensa Universitária da UFMG, Belo Horizonte, 1ª edition.

[Schmidt e Druffel, 1976] Schmidt, D. C. e Druffel, L. E. (1976). A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *J. ACM*, 23(3):433–445.

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

[Sorlin e Solnon, 2008] Sorlin, S. e Solnon, C. (2008). A parametric filtering algorithm for the graph isomorphism problem. *Constraints*, 13(4):518–537.

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

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

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

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

APÊNDICE A – Tabelas dos Resultados Computacionais

Instâncias	AEPIG	Lee	DT	VF2	Ullmann	Nauty
20	0,000170	0,000582	0,761822	0,000067	0,000200	0,000046
40	0,000449	0,001569	8,008165	0,000188	0,000858	0,000054
60	0,000896	0,001143	30,953340	0,000371	0,002049	0,000099
80	0,000998	0,002599	81,035536	0,000583	0,003582	0,000116
100	0,001665	0,001051	170,121170	0,000816	0,006553	0,000162
200	0,010055	0,002180	2103,660660	0,003707	0,038574	0,000499
400	0,070377	0,010047	-	0,013507	0,270611	0,002029
600	0,229744	0,027049	-	0,032634	0,830004	0,003522
800	0,591548	0,055887	-	0,058337	1,800550	0,005696
1000	1,402799	0,094803	-	0,096865	3,435734	0,008842

Tabela A.1: Tempo médio (em segundos) da execução dos algoritmos para as instâncias do grupo r01

Instâncias	AEPIG	Lee	DT	VF2	Ullmann	Nauty
20	0,000131	0,064017	0,726549	0,000059	0,000219	0,000052
40	0,000325	5,389129	7,619884	0,000120	0,000664	0,000048
60	0,000590	27,843959	29,599965	0,000259	0,002189	0,000083
80	0,000998	42,007933	29,599965	0,000373	0,003700	0,000097
100	0,001664	12,298832	179,678340	0,000555	0,006742	0,000130
200	0,010090	1,299386	2340,182519	0,001886	0,041314	0,000364
400	0,070791	0,202869	-	0,007855	0,280004	0,001141
600	0,230232	0,044740	-	0,018361	0,895888	0,002344
800	0,594165	0,077939	-	0,032370	1,849139	0,003440
1000	1,380830	0,121431	-	0,050769	3,469621	0,005469

Tabela A.2: Tempo médio (em segundos) da execução dos algoritmos para as instâncias do grupo r005

Instâncias	AEPIG	Lee	DT	VF2	Ullmann	Nauty
20	0,000120	51,878369	0,665118	0,000063	0,000223	0,000081
40	0,000291	-	6,912850	0,000172	0,000826	0,000102
60	0,000579	-	29,173299	0,000600	0,003163	0,000158
80	0,001030	-	83,335449	0,000570	0,005546	0,000170
100	0,001751	-	185,153810	0,000828	0,010253	0,000156
200	0,010102	-	-	0,001811	0,054749	0,000236
400	0,070463	-	-	0,003943	0,342845	0,001072
600	0,231004	-	-	0,008556	0,958771	0,001507
800	0,596862	-	-	0,015172	2,218356	0,001760
1000	1,369227	-	-	0,024450	4,081362	0,001996

Tabela A.3: Tempo médio (em segundos) da execução dos algoritmos para as instâncias do grupo r001

Instâncias	AEPIG	Lee	DT	VF2	Ullmann	Nauty
Siberian	19,609318	63,977461	243,291000	0,038255	10,773584	0,000687
Weisfeiler	0,064125	0,040066	1,191310	0,006480	0,091121	0,005882
Mathon	0,000198	-	12,508300	0,000045	0,000001	0,002632
Praust	0,009030	0,016708	4,433540	0,004170	0,018658	0,000216

Tabela A.4: Tempo (em segundos) dos algoritmos para instâncias não isomorfas extraídas de [Dharwadker e Tevet, 2009]

Instâncias	AEPIG	Lee	DT	VF2	Ullmann	Nauty
g6C - g6D	0,000096	0,000115	0,024244	0,000038	0,000097	0,000071
g6E - g6F	0,000075	0,000021	0,026451	0,000095	0,000110	0,000070
g7A - g7B	0,000072	0,000020	0,034142	0,000016	0,000034	0,000041
g8A - g8B	0,000075	0,000005	0,046095	0,000016	0,000031	0,000067
g8C - g8D	0,000104	0,000136	0,044584	0,000063	0,000225	0,000075
g8E - g8F	0,000072	0,000004	0,053096	0,000007	0,000012	0,000045
g8G - g8H	0,000075	0,000010	0,047372	0,000012	0,000031	0,000049

Tabela A.5: Tempo (em segundos) dos algoritmos para instâncias não isomorfas geradas aleatoriamente