Luciana Lee

Reformulação do Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação

Vitória, ES – Brasil
10 de Setembro de 2007

Luciana Lee

Reformulação do Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Informática para obtenção do título de Mestre em Informática pela Universidade Federal do Espírito Santo

Orientador:
Maria Cristina Rangel

Co-orientador:
Maria Claudia Silva Boeres

Departamento de Informática
Centro Tecnológico
Universidade Federal do Espírito Santo

Vitória, ES – Brasil
10 de Setembro de 2007

Dissertação de Mestrado sob o título *"Reformulação do Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação"*, defendida por Luciana Leee aprovada em 10 de Setembro de 2007, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída pelos professores:

Profa. D.Sc. Maria Cristina Rangel
Orientadora

Profa. D.Sc. Maria Claudia Silva Boeres
Co-orientadora

Profa. D.Sc. Nair Maria Maia de Abreu

Prof. D.Sc. Arlindo Gomes de Alvarenga

Aos meus pais Fong e Meng Tsu.

Agradecimentos

Gostaria de agradecer a todos que, direta ou indiretamente, contribuiram para a realização deste trabalho. Em especial, agradeço:

Às professoras Maria Cristina Rangel e Maria Claudia Silva Boeres, as orientadoras, que me acompanharam por todo o curso e compartilharam não só o conhecimento para a realização deste trabalho, mas também para a vida. Obrigada pela amizade, apoio e dedicação tão fundamentais para meu amadurecimento durante o curso.

À Petrobrás pelo apoio financeiro através da bolsa de mestrado, indispensável para a realização deste trabalho.

Aos amigos que conheci no mestrado. Destacando Livia Lopes de Azevedo, José Jerônimo Camata e Idílio Drago, que compartilharam comigo ótimos momentos, sem os quais o mestrado não teria a mesma graça.

Ao Rodrigo Sarmento por me fornecer a adaptação do algoritmo de Ullmann, que contribuiu para o enriquecimento do estudo realizado.

Aos amigos Saulo Bortolon, Mariella Berger, Eduardo Zambon, Adriano José Abreu Moreno e João Olavo Baião de Vasconcelos, por todo o apoio e amizade durante o curso.

À minha família pelo apoio incondicional, sem os quais não encontraria forças para trilhar meu caminho.

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

Reformulação do Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação

Luciana Lee

Setembro/2007

Orientador: Maria Cristina Rangel
Maria Claudia Silva Boeres
Departamento: Informática

Neste trabalho apresentamos uma reformulação do Problema de Isomorfismo de Grafos (PIG) como um Problema Quadrático de Alocação (PQA). Analisamos as vantagens da aplicação das características do PIG para resolver o problema reformulado. Validamos a reformulação através de um teorema e propomos a adaptação de três algoritmos, sendo dois baseados em meta-heurísticas e um algoritmo de *backtracking*. Finalmente, apresentamos os resultados computacionais da aplicação dos três algoritmos ao problema reformulado.

PALAVRAS CHAVES: Meta-heurísticas, Isomorfismo de Grafos, Problema Quadrático de Alocação

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

Reformulation of the Graph Isomorphism Problem as a Quadratic Assignment Problem

Luciana Lee

September/2007

Advisor: Maria Cristina Rangel
Maria Claudia Silva Boeres
Department: Informatica

In this work we present a reformulation of the Graph Isomorphism Problem (GIP) as a Quadratic Assignment Problem (QAP). We analyse the advantages of the application of the GIP's characteristics to solve the reformulated problem. We validate the reformulation through a theorem and propose three algorithms, being two based in metaheuristics and one backtracking algorithm. Finally, we present the computational results of the application of the three algorithms to the reformulated problem.

KEYWORDS: Metaheuristics, Graph Isomorphism, Quadratic Assignment Problem

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O Problema de Isomorfismo de Grafos (PIG) pode ser aplicado a diversos problemas reais, como no processo de sombreamento 3D para animação 2D, [Varela et al., 2006], em problemas de reconhecimento de imagens [Cross et al., 1997, Rohe, 2004] aplicados a sistemas de segurança, como o reconhecimento de impressões digitais e da íris ocular [Ferreira et al., 2001], entre outros. O PIG consiste em encontrar uma correspondência um a um dos vértices de dois grafos dados, obedecendo as adjacências existentes entre os vértices [Berge, 1985].

Atualmente a sua complexidade ainda permanece como uma incógnita. Este é um dos poucos problemas que pertence à classe dos problemas NP, do qual não se sabe se ele está na classe P ou NP-completo, entretanto é conhecido que ele não se trata de um problema co-NP [Fortin, 1996].

Temos na literatura alguns problemas de difícil solução que foram pouco explorados, muitas vezes, devido ao fato de sua formulação possuir muitas restrições aumentando a complexidade da busca pela solução ótima. Nesses casos, há a possibilidade de reformulá-los para problemas mais conhecidos que possuem uma definição mais simples. Por exemplo, o problema de alocação de horários reformulado para o Problema Quadrático de Alocação [Schaerf, 1999].

O Problema Quadrático de Alocação (PQA) foi sugerido inicialmente por Koopmans e Beckmann em 1957 [Koopmans and Beckmann, 1957] para tratar o problema de alocar de forma eficiente atividades a localidades, de acordo com os fluxos entre as atividades e as distâncias entre as localidades.

Este problema pertence à classe de problemas NP-árduos [Burkard et al., 1984]. Ele foi reconhecido como um modelo para diversas situações reais, dentre elas o planejamento

de um campus universitário [Dickey and Hopkins, 1972] e a distribuição de departamentos em hospitais [Elshafei, 1977].

Muitas vezes, tais problemas demandam um tempo computacional inviável para a busca pela solução ótima quando utilizamos algoritmos exatos. Isso se deve ao fato de possuírem um espaço de busca muito grande ou por demandarem um alto custo computacional na avaliação de uma solução candidata. Então buscamos estratégias para resolvê-los de modo que não seja explorado todo o espaço de busca do problema. Uma dessas estratégias corresponde às meta-heurísticas, que buscam pela solução ótima (geralmente um ótimo local) em um tempo computacional aceitável, porém não garantem que a solução encontrada seja um ótimo global [Reeves, 1993].

Neste trabalho é investigada a reformulação do Problema de Isomorfismo de Grafos como o Problema Quadrático de Alocação, sugerida por [Abreu et al., 2006] explorando algumas particularidades do PIG na resolução do problema reformulado, principalmente no que diz respeito aos graus dos vértices dos grafos. Para analisar a influência da exploração das características do PIG na resolução do problema reformulado, utilizamos duas meta-heurísticas (GRASP e Busca Tabu) e um algoritmo exato, baseado na técnica de descida em árvore. Além disso, propomos um teorema que valida a reformulação apresentada.

No próximo capítulo, os problemas citados acima são descritos com mais detalhes. No Capítulo 3, a reformulação é apresentada e sua validade é demonstrada através de apresentação de um teorema. No Capítulo 4, algumas particularidades do PIG são apresentadas e os algoritmos GRASP, Busca Tabu, e o algoritmo exato utilizados na resolução do PIG reformulado, são descritos no Capítulo 5. O Capítulo 6 apresenta as características da base de dados da qual exemplos foram extraídos para execução dos algoritmos e os resultados computacionais obtidos. Neste mesmo capítulo realizamos a comparação de desempenho entre os algoritmos apresentados. O Capítulo 7 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Problemas de Isomorfismo de Grafos e Quadrático de Alocação

Neste trabalho estudamos os Problemas de Isomorfismo de Grafos e Quadrático de Alocação. A seguir, apresentamos os referidos problemas, analisando as características de cada um e algumas formulações existentes do PQA.

2.1 Problema de Isomorfismo de Grafos (PIG)

O Problema de Isomorfismo de Grafos (PIG) tem atraído o interesse de vários estudiosos por ter a sua aplicação em vários problemas práticos. Seu estudo tem sido realizado na área da matemática, na química e na ciência da computação. Os grafos são utilizados tanto para dar suporte a decisões como para representar computacionalmente as informações trabalhadas [Fortin, 1996, Foggia et al., 2001].

2.1.1 Definição do problema

Para definir o PIG, considere os grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$. Dizemos que os grafos G_1 e G_2 são isomorfos se existir uma função bijetora $f : V_1 \rightarrow V_2$, onde as seguintes condições são satisfeitas:

(i) Para cada aresta (a,b) de E_1, temos uma aresta (f(a),f(b)) em E_2;

(ii) Toda aresta de E_2 tem a forma (f(a),f(b)) para alguma aresta (a,b) de E_1.

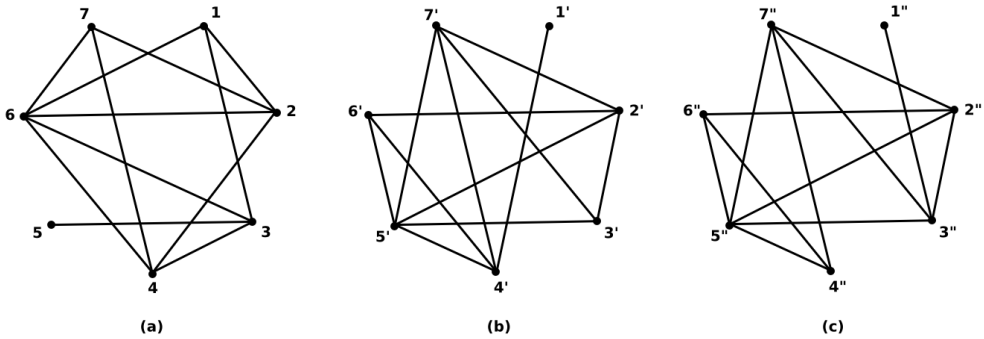


Figura 2.1: Exemplo de grafos isomorfos (a e b) e não isomorfos (a e c)

Na Figura 2.1 são apresentados três grafos não completos. Os grafos da Figura 2.1(a) e 2.1(b) são isomorfos, pois encontramos uma função bijetora f entre os vértices, $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6' & 2' & 4' & 7' & 1' & 5' & 3' \end{pmatrix}$, que satisfaz as condições (i) e (ii), isto é, mantém as características dos vértices em relação ao grau e a conectividade entre eles. Os grafos da Figura 2.1(a) e 2.1(c) não são isomorfos, pois não encontramos $f : V_1 \rightarrow V_2$ que mantenha as características dos vértices descritos anteriormente. O mesmo ocorre entre os grafos da Figura 2.1(b) e 2.1(c).

2.1.2 Estado da arte

Em [Fortin, 1996] é feito uma análise dos estudos realizados sobre o problema quanto à sua complexidade, aos algoritmos propostos e aos tipos de grafos de entrada. Ainda hoje, a sua complexidade continua uma incógnita. Sabe-se que o problema pertence à classe NP, mas não se sabe se está na classe P ou NP-completo. Foram propostos vários algoritmos exatos de tempo polinomial para a resolução do PIG aplicado a classes restritas de grafos. Mas não há um algoritmo que resolva o problema aplicado a todos os tipos de grafos em tempo polinomial provado no seu pior caso. Nesse artigo, também são apresentados as classes de grafos que possuem algoritmos de tempo polinomial para a resolução do PIG aplicado a eles. Dentre essa classes são citados grafos que são limitados por grau, grafos conceituais, grafos de arcos circulares, árvores, grafos cordais, k-árvores parciais, entre outros.

Um dos trabalhos pioneiros nesta área é [Corneil and Gotlieb, 1970]. Nele é apresentado um algoritmo de complexidade de tempo polinomial para resolução do PIG. Esse algoritmo consiste em etapas de transformações nos grafos de entrada até chegar ao o que Corneil chama de *grafo representativo* (G_R). Cada grafo de entrada é tratado independentemente, gerando um G_R para cada entrada. O algoritmo se baseia na suposição

de uma conjectura, que se verdadeira e os G_R's gerados forem idênticos, então pode-se concluir que os grafos de entrada são isomorfos. Mas de acordo com [Foggia et al., 2001], foi mostrado em [Mathon, 1978] que a conjectura nem sempre é verdadeira.

O algoritmo exato mais comumente utilizado para tratar o PIG foi prosposto por [Ullmann, 1976]. Trata-se de um algoritmo, baseado na técnica de *backtracking*, que tem por características reduzir consideravelmente o espaço de busca do problema e trabalhar com os grafos de entrada de forma conjunta, ou seja, ele utiliza uma matriz para armazenar as associações de vértices entre os grafos. Antes desse algoritmo, os outros existentes realizavam o processamento dos grafos de forma independente, um exemplo é o algoritmo apresentado por [Corneil and Gotlieb, 1970]. O algoritmo foi elaborado para tratar tanto o Problema de Isomorfismo de Grafos, como o Problema de Isomorfismo de Sub-Grafos. Ele explora as adjacências dos vértices na busca pelo isomorfismo, de forma que ao agregar uma associação de vértices à solução, o algoritmo é capaz de descartar associações de vértices não adjacentes aos recém-associados com outros adjacentes a eles.

Em 2001 foi disponibilizada na internet a versão 2.0 da biblioteca VFLib, que consiste de uma base de dados de grafos para os problemas de isomorfismo de grafos e isomorfismo de sub-grafos. Essa biblioteca foi desenvolvida no Laboratório de Sistemas Inteligentes e Visão Artificial (*Intelligent Systems and Artificial Vision Lab.* (SIVALab)) da Universidade de Nápoles "Federico II", com a finalidade de testar e comparar o algoritmo chamado VF [Cordella et al., 1999] com outros algoritmos como o algoritmo de Ullmann [Ullmann, 1976] e o algoritmo de Schmidt e Druffel [Schmidt and Druffel, 1976].

Em [Foggia et al., 2001] é feito uma comparação de desempenho entre cinco algoritmos para a resolução do PIG. São eles: o algoritmo de Ullmann, o algoritmo de Schmidt e Druffel, VF, VF2 (uma versão de VF que utiliza estruturas de dados mais efetivas, com o objetivo de otimizar o processo de busca) e o algoritmo de Nauty. Para a realização dos testes foi utilizada parte dos dados da biblioteca VFLib composta de 10.000 pares de grafos isomorfos. Dos resultados obtidos não foi observado a predominância de um único algoritmo em todos os casos, mas que os algoritmos Nauty e VF2 obtiveram melhores resultados que os demais.

2.1.3 Aplicações do PIG

Como foi dito anteriormente, o PIG pode ser aplicado a vários problemas práticos, muitos deles são derivados das áreas de reconhecimento de padrões [Foggia et al., 2001]. Uma dessas aplicações é apresentada em [Ferreira et al., 2001]. Trata-se de uma proposta

de implantação de um sistema de identificação de impressões digitais no aeroporto de Belém do Pará, com o objetivo de aumentar a segurança no embarque de passageiros. A impressão digital é recolhida em dois momentos: no momento do *check-in* a impressão digital do passageiro é recolhida através de um *scanner*, processada e transcrita para um grafo e armazenada no sistema. O processo de transcrição passa por etapas em que a digital é refinada e os pontos terminais e bifurcações das linhas que compõem a digital são identificados e um grafo é gerado, cujos vértices representam os pontos e as arestas indicam as direções das linhas. No momento do embarque, a digital do passageiro é novamente escaneada, processada e transcrita para um grafo. Então o sistema busca algum grafo isomorfo ao recentemente gerado que esteja gravado no sistema.

Outra aplicação do problema é apresentada em [Bezerra et al., 2005], onde é descrito um processo de produção de animações e outro, para automatização da colorização de cada quadro é proposto. Este último tem por objetivo diminuir o esforço do animador no processo de colorização 3D para animações 2D.

No trabalho são descritos os passos para o processamento de imagens 2D para imagens 3D, e então é apresentada a colorização automática dos quadros da animação dado um quadro previamente colorido, onde o PIG é aplicado.

Primeiramente as cenas ou imagens (coloridas em 2D) são divididas em regiões de acordo com a cor de cada região. Em seguida é construído um grafo a partir de uma cena inicial e este é associado aos grafos gerados nas cenas seguintes utilizando o conceito de isomorfismo de grafos. Os grafos serão construídos de forma que os vértices corresponderão a regiões da imagem e as arestas interliguem regiões vizinhas. Cada vértice conterá as informações sobre o volume correspondente à região ao qual ele está representando. Como é assumido que as diferenças com relação ao volume e forma da imagem entre uma cena e outra não sofre modificações bruscas, as informações contidas nos vértices podem ser passadas de uma cena a outra (o que caracteriza a automação no processo de colorização das imagens da animação).

O PIG também possui aplicações na área de reconhecimento de imagens [Rohe, 2004], quando a identificação de algumas regiões em imagens são realizadas por computadores de forma automática, ou seja, sem ajuda de uma pessoa experiente no assunto. De acordo com [Rohe, 2004], uma forma de armazenamento das informações do problema é através de grafos, e os algoritmos geralmente trabalham com o problema de isomorfismo entre dois grafos com tolerâncias, pois na maioria dos casos, as imagens e seus grafos correspondentes não possuem nós com domínio específico, mas são distinguidos através de suas cores, formas e tamanhos. Existem exemplos também de reconhecimento

estrutural de imagens. Nestes casos abstrai-se de detalhes de descrição da imagem que podem eventualmente sobrecarregar o processo de reconhecimento. Exemplos de tratamento destes casos podem ser encontrados em [El-Sonbaty and Ismail, 1998, Wong, 1992, Messmer and Bunke, 1999].

2.2 Problema Quadrático de Alocação (PQA)

O Problema Quadrático de Alocação (PQA) foi inicialmente sugerido por Koopmans e Beckmann em 1957 [Koopmans and Beckmann, 1957], com o objetivo de encontrar uma associação de setores de atividades econômicas a localidades de forma a maximizar o rendimento líquido total desse conglomerado. Cada setor é alocado a uma única localidade, e da operação desse setor nessa localidade se obtém um rendimento parcial. Este é independente da alocação dos outros setores às outras localidades e é dado pelo rendimento bruto da produção do setor menos os gastos primários sem contar com os custos de transporte. Dado uma alocação dos setores de atividades às localidades, o rendimento líquido total é obtido através da soma dos rendimentos parciais dos setores subtraído dos custos com o transporte de mercadorias entre os setores.

Também pode-se visualizar o PQA como um problema de minimização dos custos totais gastos em um conglomerado, dado uma alocação dos setores de atividades aos locais, considerando o custo da produção de um setor quando alocado a uma determinada localidade, os custos de transporte entre as localidades e a demanda de mercadoria entre os setores.

De acordo com [BURKARD et al., 1998], o PQA é muito difícil no ponto de vista teórico. O problema não pode ser resolvido de forma eficiente como também não pode ser aproximado eficientemente com alguma constante de aproximação. Sahni e Gonzalez [Sahni and Gonzalez, 1976] mostram que o PQA é um problema NP-árduo.

A seguir veremos algumas formulações do problema a partir da ilustração de uma aplicação do problema baseado no trabalho de [Dickey and Hopkins, 1972] e apresentaremos outras formulações para o problema. Na Seção 2.2.2 e 2.2.3 apresentamos alguns algoritmos encontrados na literatura para resolução do PQA e algumas aplicações práticas do problema, respectivamente.

2.2.1 Formulações do PQA

Na literatura encontramos várias formulações para este problema, estas são apresentadas em [Loiola et al., 2004]. Para explicar algumas dessas formulações vamos considerar o problema de distribuição de centros acadêmicos e administrativos em um campus universitário, baseado no trabalho de [Dickey and Hopkins, 1972].

Uma universidade é composta de centros acadêmicos e administrativos. Vamos supor, que a universidade possui quatro centros acadêmicos (Centro Biomédico, Centro de Ciências Jurídicas e Econômicas, Centro de Ciências Exatas e o Centro Tecnológico), a administração central, a biblioteca e o restaurante universitário, totalizando sete centros. Vamos representar esses centros pelo conjunto $A = \{a_1, a_2, \ldots, a_7\}$. Cada centro acadêmico abrigará os cursos referentes a sua área, como por exemplo, o centro biomédico abrigará os cursos de medicina, enfermagem, odontologia, etc. A universidade possui um terreno no qual sete áreas foram identificadas como locais possíveis para as construções dos centros descritos. Vamos representar esses locais pelo conjunto $B = \{b_1, b_2, \ldots, b_7\}$. A Figura 2.2 mostra a planta da universidade, ilustrando os locais possíveis para construção dos centros.

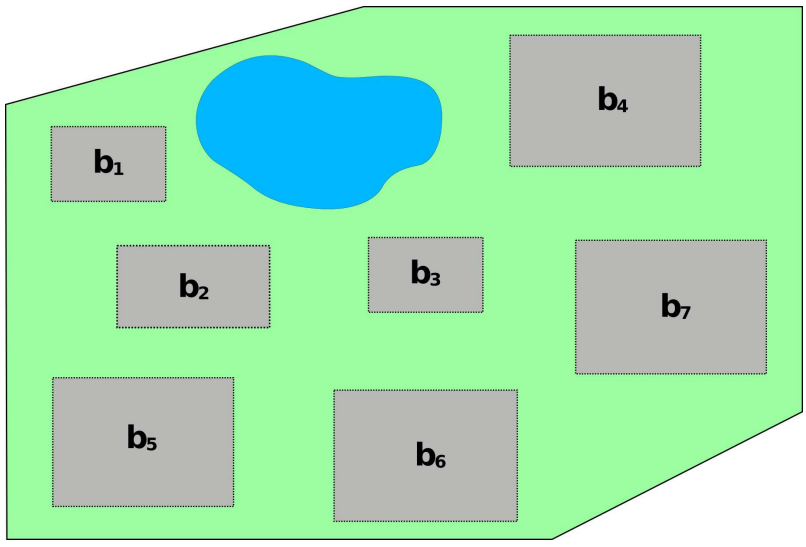


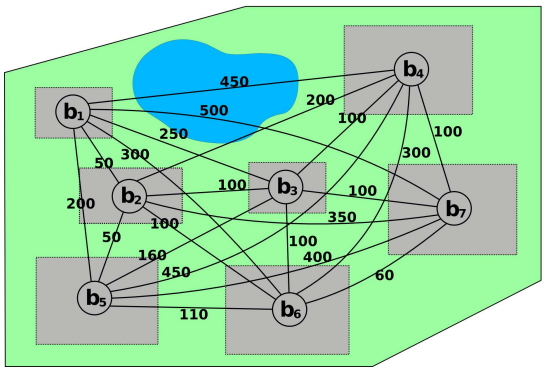
Figura 2.2: Mapeamento do campus da universidade

Consideraremos c_{ij} o custo da construção do centro a_i no local b_j e representaremos por π uma solução para o problema como sendo uma permutação das localidades fixados os centros em uma ordem pré-definida. Se quisermos alocar os centros às localidades de forma a minimizar o custo da construção e supondo n=7, então podemos descrever o problema como segue:

$$min \quad \sum_{i=1}^{n} c_{i\pi(i)}.$$

A utilização de uma permutação (π) para representar a solução nos garante que cada centro será construído em um local distinto.

Vamos supor ainda, que o custo da construção também leva em conta as distâncias percorridas pelos alunos e funcionários na universidade. Ou seja, temos que levar em consideração as informações referentes às distâncias entre as localidades na planta do terreno e ao fluxo de pessoas entre os centros. Assim, vamos denotar por $d(b_i, b_j)$ a distância entre os locais b_i e b_j, e por $f(a_k, a_l)$ o fluxo de pessoas entre os centros a_k e a_l. Armazenaremos essas informações nas matrizes $F_{n\times n} = [f_{ij}] = [f(a_i, a_j)]$, $D_{n\times n} = [d_{kl}] = [d(b_k, b_l)]$ e $C_{n\times n} = [c_{ik}] = [c(a_i, b_k)]$. A Figura 2.3 mostra a planta do terreno com as distâncias entre as localidades e uma tabela contendo as informações sobre o fluxo de pessoas entre os centros.



Fluxos de pessoas entre os centros							
	ADM	**Biblioteca**	**CB**	**CCE**	**CCJE**	**CT**	**RU**
ADM	0	3	6	7	6	8	15
Biblioteca	3	0	250	300	350	400	200
CB	6	250	0	70	2	2	300
CCE	7	300	70	0	50	200	400
CCJE	6	350	2	50	0	70	250
CT	8	400	2	200	70	0	400
RU	15	200	300	400	250	400	0

Legenda:

ADM	Centro administrativo
Biblioteca	Biblioteca central
CB	Centro biomédico
CCE	Centro de ciências exatas
CCJE	Centro de ciências jurídicas e econômicas
CT	Centro tecnológico
RU	Restaurante universitário

Figura 2.3: Informações para a resolução do problema de alocação dos centros

Se na solução temos o centro a_i associado à localidade $\pi(i)$ e o centro a_j associado à localidade $\pi(j)$, então o deslocamento total de pessoas entre os centros i e j é dado por: $f_{ij}d_{\pi(i)\pi(j)}$. O deslocamento total realizado pelas pessoas no campus pode ser dado pela soma dos deslocamentos realizado entre os centros. Assim, se o nosso objetivo é minimizar o custo da alocação dos centros nas localidades, e este leva em consideração o deslocamento total das pessoas e o custo da construção de um centro a uma localidade, então o problema pode ser escrito como:

$$min \left\{ \sum_{i,j=1}^{n} f_{ij}d_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)} \right\}. \tag{2.1}$$

Se o problema não levasse em conta o custo da construção de um centro em uma localidade, então o termo linear da equação (2.1) poderia ser descartado e o problema se resumiria à equação (2.2).

$$min \quad \sum_{i,j=1}^{n} f_{ij}d_{\pi(i)\pi(j)}. \tag{2.2}$$

A formulação apresentada é chamada de formulação por permutação. Podemos também formular o problema através de grafos. Neste caso, trabalharemos com dois grafos completos, $G_1 = (V_1, E_1)$, representando os centros (vértices) e o fluxo de pessoas entre eles (arestas), e $G_2 = (V_2, E_2)$, representando as localidades (vértices) e as distâncias entre elas (arestas). A solução consiste na sobreposição dos vértices de um dos grafos nos vértices do outro. O custo total é o resultado da soma dos produtos das arestas que se superpõem. A fórmula matemática para representar esta formulação é dado pela equação (2.2), onde fixamos os vértices de um dos grafos e permutamos os vértices do outro grafo e cada elemento da permutação representa uma sobreposição de vértices. Podemos observar que esta formulação é apenas uma representação gráfica para a formulação por permutação. A Figura 2.4 ilustra esta formulação aplicada ao exemplo descrito.

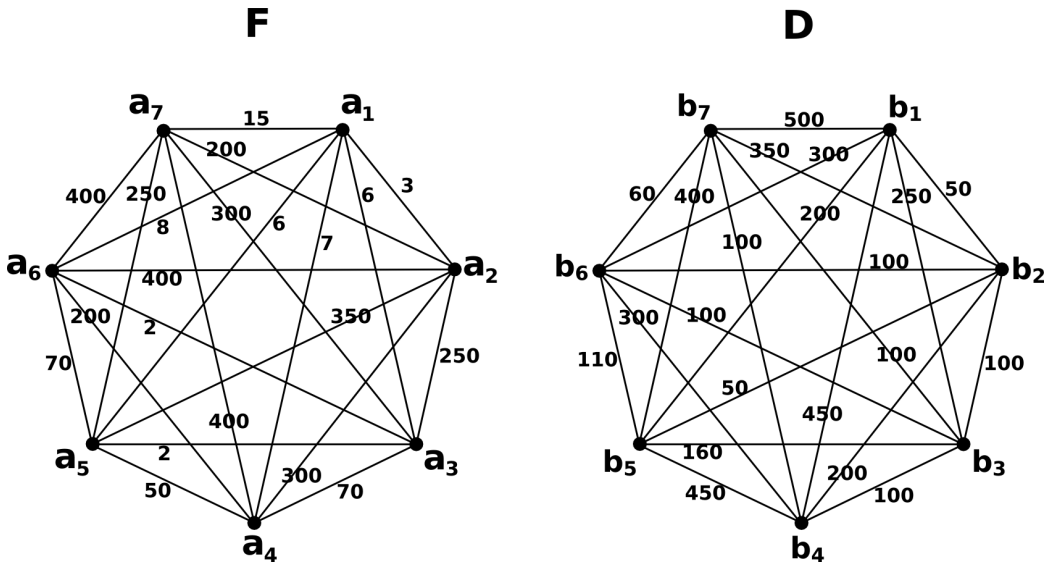


Figura 2.4: Grafos com as informações de fluxos e distâncias do problema

Outra formulação existente é a formulação por programação inteira. Nela trabalhamos com variáveis booleanas para indicar a existência de associações entre os centros e os locais. Assim, sejam $F_{n\times n} = [f_{ij}]$ a matriz com os valores dos fluxos entre os centros, $D_{n\times n} = [d_{kl}]$ a matriz com os valores das distâncias entre as localidades, e $X_{n\times n} = [x_{ik}]$ a matriz de variáveis de decisão $x_{ik} \in \{0, 1\}$. Então o custo de alocação de n centros a n locais é dado por:

$$min \quad \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} f_{ij} d_{kp} x_{ik} x_{jp} \tag{2.3}$$

$$s.a \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad 1 \leq j \leq n, \tag{2.4}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad 1 \leq i \leq n, \tag{2.5}$$

$$x_{ij} \in \{0,1\} \qquad 1 \leq i,j \leq n. \tag{2.6}$$

Observe que os valores possíveis para x_{ik} são 1 quando os vértices i e k são associados na solução e 0 caso contrário. Essa formulação, também chamada de formulação booleana, foi inicialmente proposta por [Koopmans and Beckmann, 1957], e posteriormente utilizada em vários outros trabalhos citados em [Loiola et al., 2004].

Se considerarmos o custo da construção de um centro a uma determinada localidade, então, dados as matrizes F, D, C de tamanho $n \times n$, o problema teria a seguinte fórmula:

$$min \left\{ \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} f_{ij} d_{kp} x_{ik} x_{jp} + \sum_{i,k=1}^{n} c_{ik} x_{ik} \right\} \tag{2.7}$$

s.a
(2.4) (2.5) (2.6).

Uma versão mais geral é apresentada por [Lawler, 1963]. Esta versão envolve coeficientes de custos c_{ijkp} que não são necessariamente produtos entre fluxos f_{ij} e distâncias d_{kp}. Considerando o exemplo da alocação de centros acadêmicos no campus universitário, o coeficiente c_{ijkp} se refere ao deslocamento total das pessoas entre os centros i e j, quando estes são alocados aos locais k e p, respectivamente. Ou seja, se considerarmos o custo de alocação de um centro i a uma localidade k (b_{ik}), então o coeficiente c_{ijkp} pode ser calculado da seguinte forma:

$$c_{ijkp} = f_{ij} d_{kp} + b_{ik} + b_{jp}. \tag{2.8}$$

A formulação é dada a seguir:

$$min \quad \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} c_{ijkp} x_{ik} x_{jp} \tag{2.9}$$

s.a
(2.4) (2.5) (2.6).

Outras formulações para o problema podem ser encontradas em [Loiola et al., 2004]. Apresentamos aquelas que consideramos importantes no contexto deste trabalho. Dentre as formulações apresentadas, a formulação por grafos será utilizada para aplicar a reformulação do Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação, tratado no próximo capítulo.

2.2.2 Algoritmos aplicados ao problema

O estudo dos métodos de resolução do problema seguem duas vertentes:

- O estudo de métodos exatos;
- O estudo de heurísticas.

Na literatura temos vários algoritmos exatos propostos para a resolução do PQA, e um dos principais é o *Branch-and-Bound* pois, de acordo com [BURKARD et al., 1998], este é um dos mais eficientes algoritmos exatos aplicado ao problema. Um dos ingredientes principais para seu bom desempenho é uma boa definição do limite inferior aplicado. Para este, tem-se destacado o limite de Gilmore-Lawler (*Gilmore Lawler bound* (GLB)). Existem outras funções de limite inferior que apresentam melhores valores que o GLB, mas exigem um maior tempo computacional de execução. Nos últimos anos a aplicação da técnica de *Branch-and-Bound* com implementação paralela vem sendo muito estudada e tem obtido os melhores resultados para o problema. Esse bom desempenho está diretamente relacionado ao avanço tecnológico dos equipamentos utilizados para sua execução.

Entre outros algoritmos exatos [Loiola et al., 2004] cita os métodos de plano de corte que, apesar de não apresentarem resultados satisfatórios, contribuíram na formulação de algumas heurísticas baseadas na decomposição de Bender e formulações da programação linear inteira mista (PLIM) .

Como já foi mencionado, o PQA é um problema de difícil resolução e a aplicação de algoritmos exatos apenas resolvem instâncias de pequeno porte. Para buscar boas soluções para instâncias maiores, utilizam-se técnicas de busca que não exploram todo o espaço de soluções do problema, chamadas de heurísticas. Estas não garantem encontrar a solução ótima para o problema, mas geralmente encontram boas soluções. Dentre as técnicas utilizadas nas heurísticas descritas na literatura, temos algumas que vale a pena destacar:

- **métodos construtivos:** têm como principal característica a inserção de uma associação de um objeto a um local a cada passo do algoritmo;
- **métodos de enumeração limitada:** se caracterizam pela aplicação de um método enumerativo com algum limite imposto no processo, por exemplo, o tempo de processamento ou um número máximo de iterações que o algoritmo pode realizar.
- **métodos de melhoramento:** iniciam a busca a partir de uma solução inicial viável, explorando a vizinhança da solução corrente, escolhendo o melhor vizinho;
- **meta-heurísticas:** vieram com o objetivo de desenvolver algoritmos que sejam mais gerais, dado que as heurísticas são geralmente desenvolvidas visando diretamente os problemas aos quais elas eram propostas.

Em [Loiola et al., 2004] são citadas algumas meta-heurísticas abordadas na literatura e aplicadas ao PQA. A seguir apresentamos em linhas gerais algumas delas.

- **Busca Tabu:** introduzido por [Glover, 1989, Glover, 1990], esta meta-heurística possui como principais componentes uma lista Tabu para indicar quais os movimentos mais recentes, proibindo que os muito recentes sejam repetidos; critérios de aspiração que permitem que movimentos proibidos sejam efetuados, intensificando a busca; e mecanismos de diversificação da busca na vizinhança. Em [Taillard, 1991] é apresentado a aplicação do método ao problema estudado.
- **GRASP:** apresentado por [Li et al., 1994] para resolver o PQA. Essa meta-heurística busca explorar o espaço de soluções do problema de forma a alcançar vários ótimos locais, realizando um número determinado de iterações, onde cada iteração é composta de 2 fases: a primeira é a construção randômica gulosa de uma solução inicial e a segunda, é uma busca local aplicada à solução inicial gerada.
- **Colônia de formigas:** apresentado em [Gambardella et al., 1999] para tratar o PQA, essa meta-heurística tomou como base a análise do trabalho cooperativo de formigas reais na busca de um caminho do formigueiro até o alimento e vice-versa. A interação entre esses agentes simples levam a uma boa solução.

Além das meta-heurísticas citadas acima, [Loiola et al., 2004] ainda cita o algoritmo Genético [Goldberg, 1989], o Scatter Search [Glover, 1977] e o algoritmo de Busca em Vizinhança Variável (VNS) [Mladenovic and Hansen, 1997].

2.2.3 Aplicações do PQA

Este problema tem atraído a atenção de muitos pesquisadores, não só pela sua complexidade como também por ser aplicável a vários problemas práticos. Dentre eles temos o problema de fiação de Steinberg [Brixius and Anstreicher, 2001], o objetivo é alocar componentes computacionais em uma placa, de forma a minimizar o total de fios necessários para conectá-los. Neste problema são alocados n componentes à n locais geograficamente pré-definidos em uma placa, onde há as informações do número de fios que conectam os componentes dois-a-dois e as distâncias entre os locais na placa.

Outra aplicação é apresentada em [Carvalho and Rahmann, 2006], que propõe um novo modelo de avaliação de arranjos físicos de microarranjos de DNA, e mostra que o problema de alocação de fragmentos de DNA no microarranjo é uma instância do PQA.

Um microarranjo de DNA, também chamado de *chip de DNA*, consiste em um arranjo pré-definido de fragmentos de DNA quimicamente ligados a uma lâmina de vidro ou plástico. O trabalho utilizou para seu estudo os *chips* produzidos pela empresa *Affymetrix*. Estes são são compostos de pontos (*spots*) que comportam muitos milhões de cópias de segmentos de DNA. O processo de produção de um *chip* consiste na sintetização em paralelo dos nucleotídeos que compõem os segmentos de DNA, em uma série de passos repetitivos, em regiões selecionadas do *chip*. Essas seleções são feitas a partir da exposição da lânima à luz com a ajuda de máscaras fotolitográficas que permitem ou bloqueiam a passagem da luz. Devido a difração da luz ou reflexões internas, pontos não-alvos podem ser acidentalmente ativados em certos passos de mascaramento, produzindo segmentos de DNA imprevistos que podem comprometer os resultados de um experimento.

O objetivo do trabalho é encontrar um arranjo dos segmentos de DNA na lâmina de forma a minimizar as chances de obter iluminação indesejada os passos de exposição das máscaras.

Capítulo 3

Reformulação

Como vimos no Capítulo 2, o PQA possui um vasto campo de estudos na literatura. Além de inúmeras aplicações práticas citadas em [Loiola et al., 2004], temos o estudo de várias heurísticas para resolução do problema, tais como Busca Tabu [Taillard, 1991], Colônia de Formigas [Gambardella and Taillard, 1997] e GRASP [Li et al., 1994]. Temos também uma biblioteca QAPLib de instâncias para o problema, disponível em [BURKARD et al., 1991], onde, além das instâncias, são disponibilizadas implementações de algoritmos heurísticos e exatos para o problema.

Em face de todos os estudos realizados e trabalhos disponíveis sobre o PQA, utilizamos a reformulação do Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação proposta por [Abreu et al., 2006] com o objetivo de aproveitar esses estudos e analisar a eficiência da reformulação aplicando algoritmos heurísticos e exatos para resolvê-lo. Essa reformulação pode ser descrita como a seguir.

Sejam $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ grafos não necessariamente completos, e seja $f : V_1 \rightarrow V_2$ uma bijeção entre vértices. Considerando que os dois grafos possuem o mesmo número de vértices, o mesmo número de arestas e a mesma quantidade de vértices com graus iguais, podemos transcrever os grafos dados para grafos completos $G'_1 = (V'_1, E'_1)$ e $G'_2 = (V'_2, E'_2)$, valorados nas arestas respectivamente, onde uma aresta do grafo transcrito possuirá valor 1 se ela existir no grafo original, e 0 caso contrário. Para efeito de ilustração, podemos considerar o grafo G'_1 como sendo o grafo de distâncias e G'_2 como sendo o grafo de fluxos. Assim, transcritos os dois grafos, podemos aplicar um algoritmo para a resolução do PQA com o objetivo adequado ao PIG, isto é, encontrar uma alocação de custo máximo entre eles.

Considerando S_n o espaço de soluções do PQA e que a solução do PQA seja represen-

tada por uma permutação $\pi \in S_n$ dos vértices do grafo G'_2 fixando os vértices do grafo G'_1. Sejam (i,j) uma aresta do grafo G'_1, $(\pi(i), \pi(j))$ uma aresta do grafo G'_2, c_{ij} o custo da aresta (i,j), $c_{\pi(i)\pi(j)}$ o custo da aresta $(\pi(i), \pi(j))$. Além disso, seja π^* a solução ótima para o problema e $F(\pi^*)$, o custo total da alocação π^*. Temos por objetivo calcular o valor de $F(\pi^*)$ que é dado por:

$$\begin{aligned} & F(\pi^*) = \max_{\pi \in S_n} \sum_{i<j=1}^{n} c_{ij} c_{\pi(i)\pi(j)} \\ & \textit{sujeito a} \\ & c_{ij} \in \{0,1\} \qquad\qquad 1 \leq i < j \leq n. \end{aligned} \tag{3.1}$$

Dessa forma, o $PQA(G_1', G_2')$ é o PQA adaptado ao $PIG(G_1, G_2)$. Como vimos na definição de isomorfismo de grafos e considerando a transcrição de grafos descrita anteriormente, podemos enunciar o teorema abaixo que garante a validade da reformulação apresentada.

Teorema 3.1 *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 N_V, número de arestas N_E 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_E do grafo G_1 (ou G_2) $\Leftrightarrow$ $G_1 \approx G_2$.*

Prova:

($\Rightarrow$) Seja π^* a solução ótima para o $PQA(G'_1, G'_2)$ e c^1_{ij},c^2_{ij} os custos de uma aresta de G'_1 e G'_2, respectivamente. Pela formulação proposta:

$$\begin{aligned} F(\pi^*) \quad = \quad & c^1_{12} c^2_{\pi^*(1)\pi^*(2)} + c^1_{13} c^2_{\pi^*(1)\pi^*(3)} + \cdots + c^1_{1n} c^2_{\pi^*(1)\pi^*(n)} \\ & c^1_{23} c^2_{\pi^*(2)\pi^*(3)} + c^1_{24} c^2_{\pi^*(2)\pi^*(4)} + \cdots + c^1_{2n} c^2_{\pi^*(2)\pi^*(n)} \\ & \vdots \\ & c^1_{(n-2)(n-1)} c^2_{\pi^*(n-2)\pi^*(n-1)} + c^1_{(n-2)n} c^2_{\pi^*(n-2)\pi^*(n)} \\ & c^1_{(n-1)n} c^2_{\pi^*(n-1)\pi^*(n)}. \end{aligned} \tag{3.2}$$

Se $F(\pi^*) = N_E$ temos por definição de G'_1 e G'_2, que existem N_E parcelas dessa soma que valem 1. Para tanto, $c^1_{ij} c^2_{\pi(i)\pi(j)} = 1$, com $i, j = 1, \ldots, n$. Isto só ocorre quando as respectivas arestas existem nos grafos G_1 e G_2. Por definição de isomorfismo de grafos, existe uma bijeção π^* tal que:

$$\begin{aligned} \pi^* : \quad G_1 \quad & \rightarrow \quad G_2 \\ (i,j) \quad & \rightarrow \quad (\pi^*(i), \pi^*(j)) \end{aligned}$$

que satisfaz as condições (i) e (ii) da Seção 2.1. Portanto, $G_1 \approx G_2$.

($\Leftarrow$) Se $G_1 \approx G_2$, então, por definição do PIG, existe uma bijeção f tal que:

$$\begin{array}{rccc} f: & G_1 & \rightarrow & G_2 \\ & (i,j) & \rightarrow & (f(i), f(j)) \end{array}$$

que satisfaz as condições (i) e (ii) da Seção 2.1.

Se considerarmos o $PIG(G_1, G_2)$ como o $PQA(G_1', G_2')$ adaptado, e $f \approx \pi^*$ é uma solução do $PQA(G_1', G_2')$, temos que na soma (3.2), as parcelas que contribuirão para o valor total da soma são aquelas que representam a existência das arestas nos grafos originais G_1 e G_2, isto é, $c^1_{ij} c^2_{\pi^*(i)\pi^*(j)} = 1$. As demais parcelas terão valor zero, ou seja, $c^1_{ij} c^2_{\pi^*(i)\pi^*(j)} = 0$, pois ao menos uma das arestas não existe no grafo original, isto é, $c^1_{ij} = 0$ ou $c^2_{\pi^*(i)\pi^*(j)} = 0$. Sendo assim, o valor máximo da soma (3.2) é exatamente N_E, o número de arestas do grafo G_1 (ou G_2). ■

Exemplos:

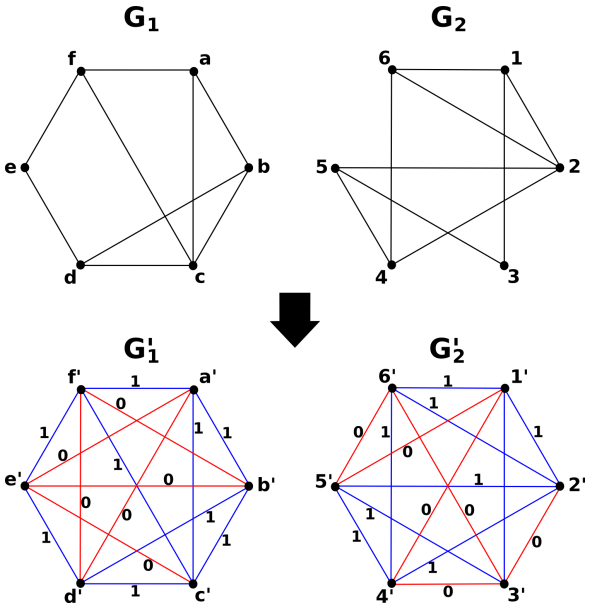


Figura 3.1: Aplicação da transcrição para dois grafos isomorfos

Nas Figuras 3.1 e 3.2 temos a ilustração da aplicação da formulação para dois casos: quando a entrada é composta de dois grafos isomorfos (Figura 3.1) e quando a entrada é composta de dois grafos não-isomorfos (Figura 3.2). Em ambos os casos realizamos a transcrição dos dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ para os grafos completos $G_1' = (V_1', E_1')$ e $G_2' = (V_2', E_2')$, respectivamente, como já descritos. Em seguida aplicamos a equação (3.1) para encontrar a alocação de custo máximo. Para isso fixamos os vértices do grafo G_1' e permutamos os vértices do grafo G_2'. Podemos observar que

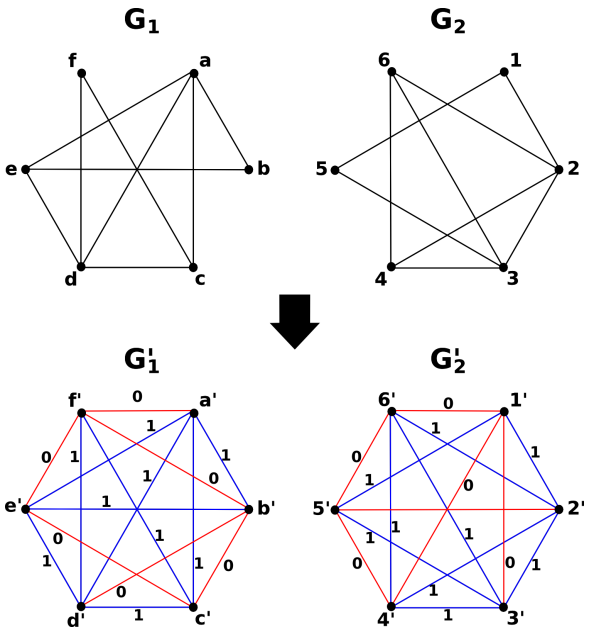


Figura 3.2: Aplicação da transcrição para dois grafos não-isomorfos

uma solução possível para o caso da Figura 3.1 é $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ d & c & e & a & f & b \end{pmatrix}$, de forma que: $\forall (a,b) \in E_1 \overset{\pi}{\Longrightarrow} \exists (\pi(a), \pi(b)) \in E_2$ e o valor da solução de acordo com a fórmula de custo do PQA é 9. Temos também que este valor é igual ao número de arestas de qualquer um dos grafos originais, como era esperado, dado que os grafos são isomorfos. Já para o caso da Figura 3.2, enumerando todas as possibilidades de alocação do grafo G_1 no grafo G_2, não encontramos nenhuma permutação que gere um custo igual ao número de arestas de um dos grafos originais (9 arestas), confirmando que os grafos não são isomorfos.

Capítulo 4

Espaço de soluções do PIG

No Problema de Isomorfismo de Grafos temos por objetivo encontrar uma associação biunívoca entre os vértices de dois grafos de forma a preservar suas adjacências. Uma característica deste problema é que os vértices associados precisam, necessariamente, ter o mesmo grau pois associações entre vértices de graus diferentes produzem soluções inviáveis.

Podemos descrever o espaço de soluções do problema para grafos de n vértices como sendo o conjunto de permutações dos n vértices de um dos grafos do problema, fixados os vértices do outro grafo. Assim, o tamanho do espaço de busca seria igual a $n!$. Porém, estão contidos nesse conjunto soluções inviáveis, isto é, soluções que possuem associações de vértices com graus diferentes. Se dividirmos os vértices em conjuntos de acordo com os seus graus e associarmos apenas vértices de mesmo grau, passaremos a analisar apenas soluções viáveis com relação a característica explorada, conseqüentemente teremos o espaço de busca reduzido. A seguir verificamos a veracidade da afirmação.

Considere dois grafos de n vértices $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, o primeiro representando os locais e as distâncias entre eles e o segundo, representando os departamentos e os fluxos de pessoas/objetos entre eles, onde $\#V_1 = \#V_2 = n$. Suponha que os vértices possam ser divididos em dois conjuntos de acordo com seus graus e que os dois grafos possuem a mesma quantidade de vértices com os mesmos graus, ou seja, ambos V_1 e V_2 podem ser divididos em dois conjuntos cada um de forma que a distribuição dos vértices de um grafo nos conjuntos seja igual à distribuição dos vértices do outro grafo, com relação ao número de vértices nos conjuntos. Seja $p \in \mathbb{Z}$ o número de vértices de um dos conjuntos, tal que $0 \leq p < n$. Então o outro conjunto terá $n - p$ vértices.

A Figura 4.1 ilustra a divisão dos vértices descrita. Nesse exemplo temos dois gra-

fos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, com $n = 8$, $V_1 = \{a, b, c, d, e, f, g, h\}$ e $V_2 = \{a', b', c', d', e', f', g', h'\}$. Em cada grafo podemos separar os vértices em dois conjuntos. No caso do grafo G_1 temos $A = \{g, h\}$ e $B = \{a, b, c, d, e, f\}$, e para o grafo G_2, temos $A' = \{a', c'\}$ e $B' = \{b', d', e', f', g', h'\}$, onde os conjuntos A e A' são compostos de vértices de grau 3 e os conjuntos B e B' são compostos de vértices de grau 4. Se apenas associarmos vértices de conjuntos equivalentes, ou seja, vértices de mesmo grau, então o número de soluções viáveis é dado por $p!(n-p)!$. Na ilustração associaremos apenas os vértices do conjunto A com o conjunto A' e vértices do conjunto B com o conjunto B', resultando em um espaço de soluções de tamanho igual a $2! \cdot 6! = 1.440$.

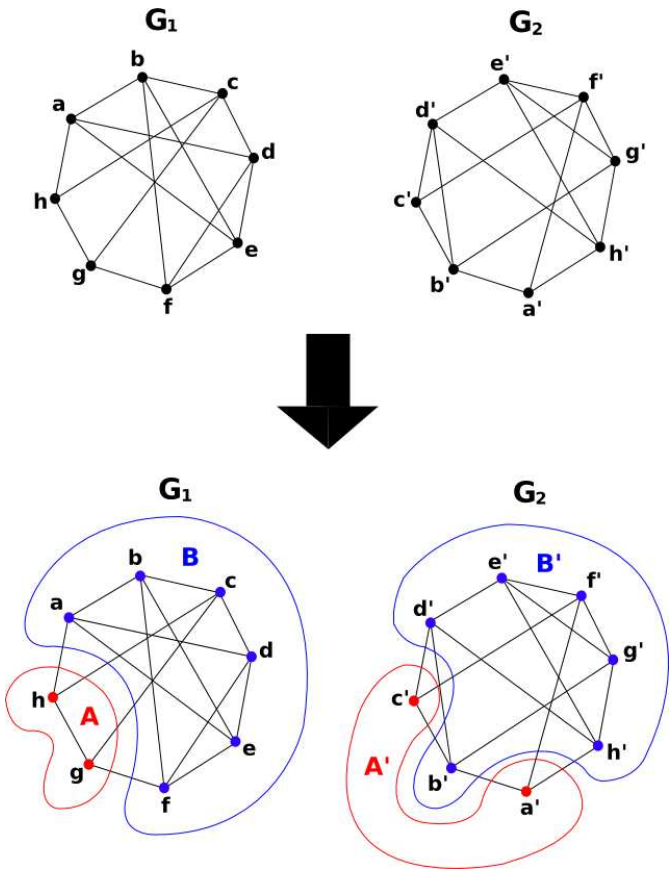


Figura 4.1: Exemplo de divisão dos vértices em conjuntos

Para estudar a influência da exploração dos graus dos vértices no tamanho do espaço de busca do problema, adotamos o termo *bloco* para nos referir aos conjuntos de vértices mencionados no exemplo.

Lema 4.1 *Suponha que um grafo do PIG contenha n vértices e estes estão divididos em dois blocos, onde um deles possui p vértices. Assim, temos que $n, p \in \mathbb{Z}_+$ e $n \geq p$. Então $n! \geq p!(n-p)!$.*

PROVA: Por contradição, supomos que $n! < p!(n-p)!$. Podemos dividir os dois membros

por $(n-p)!$. Assim, obtemos:

$$\begin{aligned}
\frac{n!}{(n-p)!} &< p! \\
\frac{n(n-1)(n-2)\cdots(n-p-1)(n-p)!}{(n-p)!} &< p(p-1)(p-2)\cdots 1 \\
\underbrace{n(n-1)(n-2)\cdots(n-p-1)}_{p \quad termos} &< \underbrace{p(p-1)(p-2)\cdots 1}_{p \quad termos}
\end{aligned}$$

Podemos observar que temos o mesmo número de termos nos dois lados da inequação e, sendo $n \geq p$, cada termo do lado esquerdo é menor que o respectivo termo do lado direito, ou seja:

$$\begin{aligned}
n &< p \\
(n-1) &< p-1 \\
(n-2) &< p-2 \\
&\vdots \\
(n-p-1) &< 1.
\end{aligned}$$

Mas por suposição, temos $n \geq p$ (contradição). Logo $n! \geq p!(n-p)!$. ■

Exemplo: Para os grafos da Figura 4.1 temos:

$$8! = 40.320 > 2! \cdot 6! = 1.440$$

Como mostramos, a divisão dos vértices em dois blocos e a geração de soluções associando-se apenas vértices de mesmo grau reduz o espaço de soluções viáveis. Podemos observar no exemplo que a divisão dos vértices em dois blocos, onde um deles possui apenas dois vértices, mostra uma redução considerável no tamanho do espaço de busca do problema. Veremos a seguir a generalização para mais que dois blocos. Considere a divisão dos vértices em k blocos de vértices, onde $k \in \mathbb{Z}_+$ e $1 < k \leq n$. Quando temos apenas um bloco (grafos regulares) é trivial que $n! \geq n!$, por isso o Teorema 4.1 é válido para casos com dois blocos ou mais.

Teorema 4.1 *Sejam n o número de vértices e k o número de blocos, tais que $n, k \in \mathbb{Z}_+$, $1 < k \leq n$, e $P = \{p_1, p_2, \ldots, p_k\}$ é um conjunto de inteiros positivos, onde p_i representa*

o número de vértices no bloco i, de forma que $\sum_{i=1}^{k} p_i = n$. *Então* $n! \geq p_1! \cdots p_{k-1}!(n - \sum_{i=1}^{k-1} p_i)!$.

PROVA: Seja $P(k)$[1] a desigualdade $n! \geq p_1! \cdots p_{k-1}!(n - \sum_{i=1}^{k-1} p_i)!$, onde k é o número de termos p_i, $\sum_{i=1}^{k-1} p_i < n$ e $1 < k \leq n$.

Caso base: Para $k = 2$, P(k) é verdade, pois de acordo com o Lema 4.1 temos que $n! \geq p!(n - p)!$.

Passo Indutivo: Assuma que para $k = n-1$, P(k) é verdadeiro, ou seja, $n! \geq p_1! \cdots p_{n-2}!(n - \sum_{i=1}^{n-2} p_i)!$, onde $\sum_{i=1}^{n-2} p_i < n$. Então mostraremos que P(k) é verdadeiro para $k = n$, ou seja, $n! \geq p_1! \cdots p_{n-1}!(n - \sum_{i=1}^{n-1} p_i)!$, onde $\sum_{i=1}^{k-1} p_i < n$.

Pela hipótese de indução:

$$n! \geq p_1! \cdots p_{k-2}! \left(n - \sum_{i=1}^{k-2} p_i\right)!. \tag{4.1}$$

Aplicando o Lema 4.1 ao termo $\left(n - \sum_{i=1}^{k-2} p_i\right)!$, temos:

$$\left(n - \sum_{i=1}^{k-2} p_i\right)! \geq p_{k-1}! \left(\left(n - \sum_{i=1}^{k-2} p_i\right) - p_{k-1}\right)! \tag{4.2}$$

$$\left(n - \sum_{i=1}^{k-2} p_i\right)! \geq p_{k-1}! \left(n - \sum_{i=1}^{k-1} p_i\right)!\,, \qquad \sum_{i=1}^{k-1} p_i < n \tag{4.3}$$

Assim, temos que:

$$n! \geq p_1! \cdots p_{k-2}! \left(n - \sum_{i=1}^{k-2} p_i\right)! \geq p_1! \cdots p_{k-1}! \left(n - \sum_{i=1}^{k-1} p_i\right)!$$

Logo:

$$n! \geq p_1! \cdots p_{k-1}! \left(n - \sum_{i=1}^{k-1} p_i\right)!\,,\ 0 < k < n \ \ e \ \ \sum_{i=1}^{k} p_i < n$$

■

Exemplos:

$$\begin{aligned} 10! = 3.628.800 &> 2! \cdot 3! \cdot 5! = 1.440 \\ 20! = 2.432.902.008.176.640.000 &> 4! \cdot 6! \cdot 10! = 62.705.664.000 \\ 20! = 2.432.902.008.176.640.000 &> 3! \cdot 3! \cdot 4! \cdot 4! \cdot 6! = 14.929.920 \end{aligned}$$

[1] Observe que escrevemos o k-ésimo termo como sendo $(n - \sum_{i=1}^{k-1} p_i)$

Podemos observar que o tamanho do espaço de soluções viáveis também é influenciado pela distribuição dos vértices nos blocos, de forma que quanto mais homogênea for a distribuição dos vértices nos blocos, menor será o espaço de busca.

Suponha que os grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ com $\#V_1 = \#V_2 = n$, possuam o mesmo número k de blocos, com $0 < k \leq n$, e os vértices estão distribuídos entre os blocos de forma similar. Sejam p_i o número de vértices no bloco i, com $i = 1, \ldots, k$. Então quanto menor a diferença entre os valores de p_i, $i = 1, \ldots, k$ e $\sum_{i=1}^{k} p_i = n$, menor será o resultado do produto $p_1!p_2! \cdots p_k!$, conseqüentemente o tamanho do espaço de soluções será menor.

Corolário 4.1 *Sejam n o número de vértices, k o número de blocos, tais que $n, k \in \mathbb{Z}_+$, $0 < k \leq n$, $P = p_1, p_2, \ldots, p_k$, $p_i \in \mathbb{Z}_+$, $\sum_{i=1}^{k} p_1 = n$. Então quanto menor a diferença entre os valores de p_i, $i = 1, \ldots, k$, menor será o resultado do produto $p_1!p_2! \cdots p_k!$.*

PROVA:

Suponha sem perda de generalidade que $k = 2$. Sejam n, p_1, p_2, p'_1, p'_2, r, r' inteiros positivos, tal que:

$$p_1 + p_2 = n \tag{4.4}$$

$$p'_1 + p'_2 = n \tag{4.5}$$

$$r = p_2 - p_1 \tag{4.6}$$

$$r' = p'_2 - p'_1 \tag{4.7}$$

$$r' > r. \tag{4.8}$$

Então queremos demonstrar que:

$$p'_1!p'_2! > p_1!p_2!. \tag{4.9}$$

Isolando os termos p_1 e p'_1 das equações (4.4) e (4.5), respectivamente, obtemos as equações:

$$p_1 = n - p_2 \tag{4.10}$$

$$p'_1 = n - p'_2 \tag{4.11}$$

Substituindo as equações (4.10) e (4.11) nas equações (4.6) e (4.7), obtemos:

$$r = p_2 - (n - p_2) \Rightarrow r = 2p_2 - n \tag{4.12}$$

$$r' = p'_2 - (n - p'_2) \Rightarrow r' = 2p'_2 - n \tag{4.13}$$

Substituindo as equações (4.12) e (4.13) na equação (4.8), temos:

$$\begin{aligned} 2p_2' - n &> 2p_2 - n \\ p_2' > p_2 &\Rightarrow p_2'! > p_2! \end{aligned} \tag{4.14}$$

De forma análoga, encontramos que

$$p_1 > p_1' \Rightarrow p_1! > p_1'!. \tag{4.15}$$

Multiplicando a equação (4.14) por $p_1!$, temos:

$$p_1!p_2'! > p_1!p_2! \tag{4.16}$$

dividindo os dois membros da inequação por $p_2'!$:

$$p_1! > \frac{p_1!p_2!}{p_2'!} \tag{4.17}$$

subtraindo a inequação (4.15) da inequação (4.17), temos:

$$0 > \frac{p_1!p_2!}{p_2'!} - p_1'! \tag{4.18}$$

multiplicando os dois membros por $p_2'!$:

$$0 > p_1!p_2! - p_1'!p_2'! \tag{4.19}$$

somando $p_1'!p_2'!$ aos dois membros:

$$p_1'!p_2'! > p_1!p_2! \tag{4.20}$$

■

Exemplo:

$$\begin{aligned} 15! \cdot 5! = 156.920.924.160.000 &> 10! \cdot 10! = 13.168.189.440.000 \\ 10! \cdot 7! \cdot 3! = 109.734.912.000 &> 7! \cdot 7! \cdot 6! = 18.289.152.000 \\ 4! \cdot 4! \cdot 2! = 1.152 &> 4! \cdot 3! \cdot 3! = 864 \end{aligned}$$

Uma característica que advém do Teorema 4.1 e do Corolário 4.1 é o fato do tamanho do espaço de soluções do problema ser inversamente proporcional ao número de conjuntos nos quais os vértices estão divididos, quando possuímos uma distribuição homogênea dos vértices nos blocos. Ou seja, supondo que os vértices são distribuídos entre os conjuntos de forma homogênea, quanto maior o número de conjuntos, menor será o tamanho do espaço de busca do problema. Isso pode ser facilmente observado, dado que quanto maior o número de conjuntos, menor será o tamanho de cada um deles, pois o somatório dos termos será sempre igual ao número total de vértices. Portanto, o produto resultante tende a diminuir.

Lema 4.2 *Sejam n, $k \in \mathbb{Z}_+$, $n \geq k$, $P = \{p_1, p_2, \ldots, p_k\}$, $p_i \in \mathbb{Z}_+$ e $\sum_{i=1}^{k} p_i = n$. Então o número de termos do produto $p_1!p_2! \cdots p_k!$ é igual a n.*

PROVA: Sejam $n, k \in \mathbb{Z}_+$, $n \geq k$, $P = \{p_1, p_2, \ldots, p_k\}$, $p_i \in \mathbb{Z}_+$, $i = 1, \ldots, k$ e $\sum_{i=1}^{k} p_i = n$. Sabemos que $n! = n \cdot (n-1) \cdots 1$ é um produto de n termos, e:

$$\begin{gathered} p_1! \text{ é um produto de } p_1 \text{ termos.} \\ p_2! \text{ é um produto de } p_2 \text{ termos.} \\ \vdots \\ p_k! \text{ é um produto de } p_k \text{ termos.} \end{gathered}$$

Como $p_1 + p_2 + \ldots + p_k = n$, então $p_1!p_2! \cdots p_k!$ é um produto de n termos. ■

Corolário 4.2 *Sejam n, k, $k' \in \mathbb{Z}_+$, $n \geq k \geq k'$, $P = \{p_1, p_2, \ldots, p_k\}$, $p_i \in \mathbb{Z}_+$, $i = 1, \ldots, k$, $p_i \simeq \lfloor \frac{n}{k} \rfloor$, $\sum_{i=1}^{k} p_i = n$, $P' = \{p'_1, p'_2, \ldots, p'_{k'}\}$, $p'_j \in \mathbb{Z}_+$, $p'_j \simeq \lfloor \frac{n}{k'} \rfloor$ e $\sum_{j=1}^{k'} p'_j = n$. Então:*

$$p_1! \cdots p_k! \leq p'_1! \cdots p'_{k'}!. \tag{4.21}$$

PROVA: Sendo $\sum_{i=1}^{k} p_i = n$, $\sum_{j=1}^{k'} p'_j = n$, cada termo i do membro esquerdo possui o valor $p_i \simeq \lfloor \frac{n}{k} \rfloor$, cada termo j do membro direito possui o valor $p'_j \simeq \lfloor \frac{n}{k'} \rfloor$ e além disso, $k \geq k'$, então $\forall\, p_i \in P, \forall\, p'_j \in P', p_i \leq p'_j$.

Mas, pelo Lema 4.2, $p_1! \cdots p_k!$ e $p'_1! \cdots p'_{k'}!$ são produtos de n termos. Assim:

$$\begin{aligned} p_1! \cdots p_k! &\leq p'_1! \cdots p'_{k'}! \\ \underbrace{p_1(p_1-1)\cdots 1 \cdots p_k(p_k-1)\cdots 1}_{n \text{ termos}} &\leq \underbrace{p'_1(p'_1-1)\cdots 1 \cdots p'_{k'}(p'_{k'}-1)\cdots 1}_{n \text{ termos}} \end{aligned}$$

Analisando os elementos da inequação, observamos que $p_i! = p_i(p_i-1)\cdots 1$ é um produto de p_i termos, onde o maior elemento possui valor p_i, e $p'_j! = p'_j(p'_j-1)\cdots 1$ é um produto de p'_j termos, onde o maior elemento possui valor p'_j. Podemos observar que o maior termo do membro esquerdo será sempre menor ou igual ao maior termo do membro direito da inequação. Como temos o mesmo número de termos em ambos os lados, podemos concluir que o produto resultante do membro esquerdo será sempre menor ou igual ao resultado do membro direito. Assim, temos que a inequação (4.21) é verdadeira. ■

Exemplo:

$$864 = 4! \cdot 3! \cdot 3! \leq 5! \cdot 5! = 14.400$$

Capítulo 5

Algoritmos Propostos

Como vimos anteriormente, o PIG pode ser aplicado a diversos problemas práticos. Entre os algoritmos exatos que encontramos na literatura para resolvê-lo temos o algoritmo de Ullman [Ullmann, 1976] e o VF (isomorfismo de grafos direcionados) [Cordella et al., 2001]. Também encontramos várias meta-heurísticas que já foram trabalhadas para resolver o problema, entre elas temos o GRASP e o algoritmo genético tratados no trabalho de [Boeres and Sarmento, 2005].

Para validar a reformulação implementamos um algoritmo exato e dois algoritmos heurísticos, o GRASP e a Busca Tabu, aplicando as características do PIG descritas no Capítulo 4.

O GRASP foi inicialmente proposto por [Feo et al., 1994] para resolver o problema do conjunto máximo independente. Esta meta-heurística consiste em um processo iterativo, onde em cada iteração temos o refinamento de uma solução inicial construída de acordo com mecanismos probabilísticos. Em cada iteração o algoritmo constrói uma solução inicial de forma gulosa e aleatória, e em seguida aplica uma busca local a solução gerada. Posteriormente, este algoritmo foi aplicado ao Problema Quadrático de Alocação [Li et al., 1994], ao problema de satisfabilidade (SAT) [Resende and Feo, 1996], ao problema de planarização de grafos [Resende and Ribeiro, 1997], ao problema de escalonamento de tarefas [Binato et al., 2001], entre outros.

A meta-heurística Busca Tabu foi proposta por [Glover, 1986] e utiliza a metodologia de busca local na vizinhança de uma solução e impõe restrições para evitar que o processo de busca estacione em alguma solução não ótima. Além disso, se utiliza de mecanismos que lhe permite fugir de ótimos locais. Esse algoritmo teve sua aplicação em vários problemas, por exemplo, ao problema da clique máxima [Gendreau et al., 1993], ao pro-

blema do caixeiro viajante [Gendreau et al., 1994], ao Problema Quadrático de Alocação [Misevicius, 2005], entre outros.

Quanto ao algoritmo exato, utilizamos a metodologia de descida em árvore e aplicamos as caracterísiticas das instâncias do PIG analisadas no Capítulo 4. Os detalhes do algoritmo serão apresentados na Seção 5.3.

A seguir, apresentaremos as meta-heurísticas GRASP e Busca Tabu implementadas para testar a reformulação, descrevendo os mecanismos utilizados para explorar as características do PIG descritos no Capítulo 4.

5.1 GRASP aplicado à reformulação

Utilizamos o algoritmo GRASP proposto por [Li et al., 1994] devido a facilidade de implementação e por ser uma das meta-heurísticas mais aplicadas ao PQA de acordo com [Loiola et al., 2004]. Fizemos as adaptações necessárias para sua aplicação à reformulação proposta. As modificações realizadas serão explicadas a seguir na apresentação de cada etapa do algoritmo. O Algoritmo 1 mostra o pseudo-código do algoritmo GRASP implementado. Como se trata de uma adapatação do algoritmo proposto por [Li et al., 1994] para o PIG reformulado, então o chamaremos de GPIGPQA.

Algoritmo 1: GPIGPQA

Entrada: MaxIter, numeroArestas

```
1  melhorSolucao← ∅;
2  melhorSolucao.custo← 0;
3  construção da LRC;
4  para nIter = 1, ..., MaxIter faça
5      solucao ← Construção da solução inicial;
6      buscaLocal(solucao);
7      se solucao.custo = numeroArestas então
8          pare;
9      se solucao.custo > melhorSolucao.custo então
10         melhorSolucao ← solucao;
```

Saída: melhorSolucao

O GRASP é um processo iterativo, onde cada iteração consiste em duas fases: a construção de uma solução inicial e a aplicação de uma busca local. Na primeira, a solução inicial é construída inserindo a ela uma associação viável de cada vez. Esta fase

é dividida em duas etapas: a construção de uma lista restrita de candidatos (LRC) e a construção da solução propriamente dita, elemento a elemento.

A LRC é formada por sobreposições de arestas, onde ambas as arestas que compõem um elemento da lista possuem valor 1, ou seja, são arestas existentes no grafo original respectivo. Além disso, somente as sobreposições válidas serão inseridas na LRC, isto é, os vértices associados deverão ter o mesmo grau. Então cada elemento da LRC é composto de duas associações de vértices. Como as informações para a construção da LRC dependem apenas das matrizes de adjacências dos grafos, portanto não é modificada no decorrer do algoritmo, então ela é construída uma única vez antes de iniciar as iterações.

O objetivo é maximizar o valor da equação (3.1) apresentado no Capítulo 3. O valor máximo possível de ser obtido é igual ao número de arestas de um dos grafos originais. Como o valor da função objetivo para a solução ótima é sempre conhecido, então o algoritmo termina assim que encontrar a solução ótima ou quando o limite máximo de iterações for alcançado.

5.1.1 Construção da Lista Restrita de Candidatos

Para a geração da LRC criamos duas listas, uma contendo todas as arestas com valor 1 do grafo de fluxo (L_{fluxo}) e a outra contendo todas as arestas com valor 1 do grafo de distâncias (L_{dist}). Em seguida inserimos na LRC todas as sobreposições válidas entre as arestas, isto é, os vértices associados, resultante das sobreposições entre as arestas, devem ter o mesmo grau. O Algoritmo 2 apresenta o pseudo-código do processo de geração da LRC.

Nas linhas 4 e 5 temos a construção das listas de arestas dos grafos de distância e de fluxo, respectivamente. Da linha 7 à linha 11 temos o processo de geração da LRC. Para cada aresta (i, j) de L_{dist} (linha 7) analisamos todas as associações possíveis com arestas de L_{fluxo}, ou seja, $\forall(k, l) \in L_{fluxo}$ (linha 8). Na linha 9 verificamos a validade da associação das arestas (i, j) e (k, l), de forma que se os vértices i e k tiverem o mesmo grau ($valido(i, k)$) e o mesmo for verdade para os vértices j e l ($valido(j, l)$), então a associação da aresta (i, j) com a aresta (k, l) é inserida na LRC (linhas 10 e 11). O processo termina quando todas as associações possíveis de arestas dos dois grafos forem analisadas. A saída do algoritmo consiste na LRC e seu tamanho.

Algoritmo 2: Construção da LRC

Entrada: $G'_1 = (V'_1, E'_1)$, $G'_2 = (V'_2, E'_2)$

```
1  /*Listas contendo as arestas existentes nos grafos originais:*/
2  /*L_dist: lista de arestas do grafo de distâncias, ou seja,
   i, j ∈ V_2, (i, j) ∈ E_2 : c_ij = 1 ⇒ (i, j) ∈ L_dist*/
3  /*L_fluxo: lista de arestas do grafo de fluxos, ou seja,
   k, l ∈ V_1, (k, l) ∈ E_1 : c_kl = 1 ⇒ (k, l) ∈ L_fluxo*/
4  Construção de L_dist;
5  Construção de L_fluxo;
6  tam_LRC ← 0;
7  para cada (i, j) ∈ L_dist faça
8      para cada (k, l) ∈ L_fluxo faça
9          se valido(i,k) e valido(j,l) então
10             LRC ← LRC ∪ {(i, k), (j, l)};
11             tam_LRC ← tam_LRC + 1;
```

Saída: LRC, tam_{LRC}

5.1.2 Segunda etapa da construção da solução inicial

Após a geração da LRC iniciamos a segunda etapa da construção da solução inicial, construída elemento a elemento. O Algoritmo 3 apresenta o pseudo-código desse processo.

O algoritmo recebe como entrada os conjuntos de vértices dos grafos (V_{dist} e V_{fluxo}), a LRC e seu tamanho (tam_{LRC}), o número de vértices (N_V) e a lista de blocos (B) resultantes da divisão dos vértices dos grafos por grau de vértices. O Algoritmo 3 mostra esse processo de construção.

Inicialmente, escolhemos aleatoriamente uma sobreposição de arestas entre os $\beta \cdot tam_{LRC}$ primeiros elementos, onde $0 < \beta \leq 1$ e tam_{LRC} é o tamanho da LRC, isto é, escolheremos aleatoriamente um elemento $\{(i,k),(j,l)\} \in LRC[1 \cdots (\beta \cdot tam_{LRC})]$ (linha 1). Em seguida, inserimos as duas primeiras associações de vértices, resultantes da sobreposição de arestas escolhida, na solução inicial (π) e as excluímos de seus respectivos conjuntos de vértices (linhas 3 à 7). Então iniciamos a construção do restante da solução.

Após a inserção das duas primeiras associações de vértices, estes são excluídos das listas de vértices ainda não associados de cada grafo. Então, enquanto os conjuntos não estiverem vazios (linha 8), é criada uma lista com todas as associações válidas entre os vértices ainda não inseridos na solução (L_v) (linhas 9 à 16). Em seguida, ordenamos a

Algoritmo 3: Construção da solução inicial

Entrada: N_V, V_{dist}, V_{fluxo}, B, LRC, tam_{LRC}

$\{(i,k),(j,l)\} \leftarrow random(LRC[1 \cdots \beta \cdot tam_{LRC}])$;
/*Inserção dos vértices associados na solução*/
$\pi \leftarrow \emptyset$;
$\pi \leftarrow \{(i,k),(j,l)\}$;
$N_V \leftarrow N_V - 2$;
$V_{fluxo} \leftarrow V_{dist} \setminus \{i,j\}$;
$V_{dist} \leftarrow V_{fluxo} \setminus \{k,l\}$;
enquanto $N_V > 0$ **faça**
    m $\leftarrow 0$;
    $L_v \leftarrow \emptyset$;
    para cada $b \in B$ **faça**
        para cada $p \in V_{dist}[b.inicio...b.fim]$ **faça**
            para cada $q \in V_{fluxo}[b.inicio...b.fim]$ **faça**
                $c_{pq} \leftarrow \sum_{(i,k)\in\pi} f_{qi} d_{pk}$;
                $L_v \leftarrow L_v \cup c_{pq}$;
                $m \leftarrow m + 1$;
    ordena L_v em ordem decrescente de custo;
    /*escolha randômica de uma associação da lista L_v*/
    $\{(i,k)\} \leftarrow random(vet[1 \cdots \alpha m])$; /*$i \in V_{dist}$ e $k \in V_{fluxo}$*/
    $\pi \leftarrow \{(i,k)\}$;
    $V_{dist} \leftarrow V_{dist} \setminus \{i\}$;
    $V_{fluxo} \leftarrow V_{fluxo} \setminus \{k\}$;
    $N_V \leftarrow N_V - 1$;

Saída: π

lista gerada (L_v) em ordem decrescente de custo e escolhemos uma associação entre os $\alpha \cdot m$ primeiros, onde $0 < \alpha \leq 1$ e m é o tamanho da lista (linhas 17 à 19) . Os vértices da associação sorteada são inseridos na solução π e excluídos de seus respectivos conjuntos e o laço prossegue (linhas 20 à 23).

5.1.3 Busca Local

O próximo passo do algoritmo GPIGPQA é a aplicação de um algoritmo de busca local sobre a solução gerada, com objetivo de chegar a um ótimo local, na esperança de encontrar um ótimo global em alguma iteração do algoritmo.

O algoritmo de busca local é um processo iterativo onde a cada passo ele escolhe uma solução na vizinhança da solução corrente que gere o melhor resultado. Existem várias técnicas para gerar a vizinhança de uma solução. A mais comumente utilizada é a *k-troca*, que gera uma nova solução a partir da solução atual, trocando k elementos de posição. Em nosso trabalho utilizamos a técnica 2-trocas para gerar a vizinhança no processo de busca local. A seguir, ilustramos a aplicação dessa técnica para gerar da vizinhança de uma solução para os grafos transcritos da Figura 3.1 do Capítulo 3.

Exemplo: Suponha $\pi = \begin{pmatrix} e' & a' & b' & d' & f' & c' \\ 3' & 1' & 4' & 5' & 6' & 2' \end{pmatrix}$ uma solução inicial obtido pelo algoritmo GPIGPQA aplicado aos grafos da Figura 3.1 do Capítulo 3, onde os vértices estão divididos em blocos de acordo com o grau. Cada bloco está representado por uma cor, os vértices na cor magenta possuem grau 2, na cor azul possuem grau 3 e na cor vermelha possuem grau 4. A vizinhança $V(\pi)$ é gerada aplicando-se a técnica 2-trocas sobre a solução π entre elementos de mesmo bloco.

$$V(\pi) = \left\{ \begin{array}{l} \{3'\ 4'\ 1'\ 5'\ 6'\ 2'\}, \{3'\ 5'\ 4'\ 1'\ 6'\ 2'\}, \{3'\ 6'\ 4'\ 5'\ 1'\ 2'\}, \\ \{3'\ 1'\ 5'\ 4'\ 6'\ 2'\}, \{3'\ 1'\ 6'\ 5'\ 4'\ 2'\}, \{3'\ 1'\ 4'\ 6'\ 5'\ 2'\} \end{array} \right\}$$

Em $V(\pi)$ destacamos em vermelho os elementos que tiveram suas posicões trocadas.

O Algoritmo 4 apresenta o pseudo-código do processo de busca local. Ele recebe como entrada a solução inicial π e a lista de blocos de vértices B, e utiliza uma *flag* (*Melhora*) para indicar se o processo chegou ou não a um ótimo local. O algoritmo inicia com a solução atual igual a π e em cada iteração gera a vizinhança da solução atual utilizando a estrutura *2-troca* (linhas 4 à 11), escolhendo o melhor vizinho (linhas 10 e 11). Se este possuir um custo melhor que o custo da solução atual, então a troca (referente ao melhor vizinho) é realizada na solução atual e é atribuída o valor 1 à *flag Melhora* para indicar que o algoritmo não chegou a um ótimo local (linhas 12 à 13). O algoritmo pára quando não há nenhum vizinho à solução que tenha um custo melhor.

Algoritmo 4: Busca Local

```
   Entrada: π,B
1  Melhora ← 1;
2  enquanto Melhora = 1 faça
3      Melhora ← 0;
4      para cada k ∈ B faça
5          para i=k.inicio ... (k.fim-1) faça
6              para j=(i+1) ... k.fim faça
7                  /*D*: Diferença de custo da melhor troca com a solução atual*/
8                  /*D_N: Diferença de custo da solução gerada pela troca dos
9                  vértices i e j com a solução atual*/
10                 se D_N > D* então
11                     troca ← troca_ij
12     se D*>0 então
13         Realize a troca;
14         Melhora ← 1;
   Saída: π
```

Vale a pena ressaltar que tanto na construção da solução inicial como no processo de busca local, apenas vértices de mesmo grau são associados para compor a solução, devido a divisão dos vértices em blocos.

5.2 Busca Tabu

Além do algoritmo GPIGPQA, implementamos o algoritmo de Busca Tabu com o objetivo de realizar um estudo comparativo quanto a eficiência dos algoritmos e a qualidade das soluções.

O algoritmo de Busca Tabu é basicamente um algoritmo de busca que, ao mesmo tempo que impõe restrições na busca, permite sobrepujá-las em determinadas circunstâncias. O algoritmo proíbe a realização dos movimentos mais recentes para evitar que a busca entre em um ciclo de trocas. Para essa finalidade, tem-se uma lista para armazenar os movimentos mais recentes, chamada de lista tabu. Os elementos presentes nessa lista são chamados de movimentos tabu ou movimentos proibidos, mas o algoritmo nem sempre obedece a classificação tabu dado aos movimentos. A Busca Tabu permite intensificar a busca pelo ótimo através do critério de aspiração. Quando um movimento satisfaz

esse critério, ele é executado mesmo se ele estiver presente na lista tabu. O critério mais comumente utilizado é aquele em que o movimento gera uma solução melhor que a melhor solução encontrada até então (critério de aspiração por objetivo). Além desses mecanismos, existem outros citados em [Reeves, 1993].

Os elementos básicos do algoritmo de Busca Tabu são: a representação da solução, a função de custo, a vizinhança, a lista tabu, o(s) critério(s) de aspiração e o critério de parada. Para tratar o PIG reformulado, definimos esses elementos como a seguir:

- *Representação da solução:* uma permutação dos vértices de um dos grafos, fixados os vértices do outro grafo em uma ordem pré-definida, onde os vértices estão agrupados de acordo com o grau. Como descrito no Capítulo 3.
- *Função de custo:* somatório dos produtos fluxo-distância resultantes das sobreposições de arestas geradas pelas sobreposições de vértices da solução. Descrito no Capítulo 3.
- *Vizinhança:* utiliza a estrutura 2-trocas para geração dos vizinhos da solução corrente. Porém, apenas vértices de mesmo graus podem ser trocados, para garantir a viabilidade da solução.
- *Lista tabu:* a lista tabu é composta dos 4 últimos movimentos realizados no processo de busca.
- *Critério de aspiração:* utilizamos o critério de aspiração por objetivo.
- *Critério de parada:* o algoritmo pára quando encontrar a solução ótima (custo = número de arestas) ou após realizar o número máximo de iterações previamente definido.

Podemos observar que o critério de aspiração tem a finalidade de intensificar a busca pela solução ótima. Além desse mecanismo, implementamos um outro com o objetivo de diversificar a busca, descrito em [Reeves, 1993], que se trata de uma estrutura de memória complementar. Essa estrutura guarda a freqüência de execução dos movimentos. Utilizamos as informações dessa estrutura apenas nos casos onde não temos nenhuma troca que gere melhora no custo da solução em um determinado número de iterações. Então aplicamos uma penalidade sobre cada movimento na vizinhança proporcional à freqüência com que cada um ocorreu. Em [Reeves, 1993] esse mecanismo é um dos componentes da memória a longo prazo da Busca Tabu.

O Algoritmo 5 apresenta o pseudo-código da implementação do algoritmo de Busca Tabu implementado para resolução do PIG reformulado.

Algoritmo 5: Busca Tabu

Entrada: $MaxIter$, $numeroArestas$, s

1 $s^* \leftarrow s$;
2 $s^*.custo \leftarrow s.custo$;
3 **para** $nIter = 1, ..., MaxIter$ **faça**
4 geração da vizinhança $V^* \subset N(s, nIter)$;
5 Escolha o melhor vizinho s' de V^*;/* escolha feita entre os vizinhos de s que não são tabu ou que satisfazem o critério de aspiração */
6 $s \leftarrow s'$;
7 atualização da lista tabu;
8 **se** $s.custo > s^*.custo$ **então**
9 $s^* \leftarrow s$;
10 $s^*.custo \leftarrow s.custo$;
11 **se** $s^*.custo = numeroArestas$ **então**
12 pare;

Saída: s^*

O algoritmo recebe como entrada o número máximo de iterações ($MaxIter$), o número de arestas ($numeroArestas$) e a solução inicial (s). Esta última é uma solução cujos vértices são associados observando-se seus graus e é gerada de forma aleatória. O algoritmo começa inicializando a melhor solução com a solução inicial (linhas 1 e 2). Em cada iteração geramos a lista de vizinhos da solução atual (linha 4), escolhemos o melhor vizinho que não infrinja a classificação tabu ou que satisfaça o critério de aspiração (linha 5), atualizamos a solução atual com o melhor vizinho (linha 6) e atualizamos a lista tabu (linha 7). Caso a nova solução seja a melhor encontrada até o momento, atualizamos a variável de melhor solução (linhas 8 à 10). O algoritmo pára quando o custo da melhor solução encontrada for igual ao número de arestas de um dos grafos (linhas 11 à 12) ou até o algoritmo executar o número máximo de iterações (linha 3).

O processo de geração da solução vizinha é apresentado pelo Algoritmo 6. Utilizamos a estrutura de vizinhanças 2-trocas e exploramos a divisão dos vértices de acordo com seus graus (blocos de vértices) para otimizar a geração da lista V^*, de forma que apenas as trocas válidas serão incluídas.

Os parâmetros de entrada do algoritmo são a solução atual (s), a lista onde será armazenado todas as trocas válidas na solução atual (V^*), a lista de blocos de vértices (B) e o número de blocos (tam_{V^*}). Inicializamos a lista V^* vazia (linhas 1 e 2) e inserimos todas as trocas possíveis entre vértices de mesmo bloco (linhas 3 à 6). O algoritmo retorna

Algoritmo 6: Geração da Vizinhaça

Entrada: s, V^*, B, tam_B

```
1 V* ← ∅;
2 tam_V* ← 0;
3 para k = 1, ..., tam_B faça
4     para i = B[k].inicio, ..., B[k].fim − 1 faça
5         para j = i + 1, ..., B[k].fim faça
6             V* ← V* ∪ (s[i], s[j]);
```

Saída: V^*, tam_{V^*}

a lista V^* e o tamanho da lista (tam_{V^*}).

5.3 Algoritmo Exato

Assim como nos algoritmos GPIGPQA e na Busca Tabu, no algoritmo exato implementado visualizamos uma solução para o problema como sendo uma permutação dos vértices de um dos grafos, fixados os vértices do outro grafo. Objetivamos encontrar a solução de maior custo, e sempre saberemos o custo da solução ótima (que é número de arestas de um dos grafos). Para esse fim, utilizamos a metodologia de descida em árvore para realização da busca, e associamos apenas vértices de mesmo grau.

Como vimos no Capítulo 3, a solução ótima é composta de associações de vértices que gerem apenas associações de arestas de mesmo valor, ou seja, cada aresta de valor 1 de um grafo deve ser associada a uma única aresta de valor 1 do outro grafo, e cada aresta de valor 0 de um grafo deve ser associada a outra aresta de valor 0 do outro grafo. Assim, o custo da solução que possuir essas características será igual ao número de arestas de um dos grafos.

Como a solução é construída no decorrer da descida na árvore de busca, ou seja, a cada nível da árvore uma nova associação de vértices é agregada a solução, então a profundidade máxima da árvore de busca será igual ao número de vértices. A cada passo do algoritmo, inserimos à solução em construção mais uma associação de vértices e continuamos a descida na árvore a partir daquele nó se essa associação "levar a solução ótima", ou seja, quando ela agregar apenas associações de arestas de mesmo valor ao grafo resultante. Quando isso não ocorre, abandonamos a exploração no ramo da árvore que parte

daquele nó e partimos para exploração dos nós irmãos[1] ou tios[2] daquele. Dessa forma, quando o algoritmo alcançar uma folha na árvore de busca, significará que encontramos um isomorfismo, e o algoritmo pára.

Algoritmo 7: Algoritmo Exato

```
   Entrada: solucao,nBloco,otimo
1  se otimo = 0 então
2      se o bloco terminou então
3          se não é folha então
4              nBloco ← nBloco + 1; /* Passa para o próximo bloco */
5              Chamada recursiva do algoritmo;
6          senão
7              /* A busca chegou em uma folha – foi encontrada uma solução ótima */
8              otimo ← 1;
9              Pare;
10         Backtrack na árvore;
11     senão
12         enquanto não esgotar as tentativas no bloco nBloco faça
13             s ← próxima associação para o bloco nBloco;
14             se valido(s) então
15                 solucao ← s;
16                 Chamada recursiva do algoritmo;
17                 se otimo = 1 então
18                     Pare;
19             senão
20                 Backtrack na árvore;
21         Backtrack na árvore;
   Saída: solucao
```

O Algoritmo 7 apresenta o pseudo-código do processo de busca implementado. Ele recebe como entrada o vetor para armazenar a solução (*solucao*), o número do bloco que está sendo trabalhado (*nBloco*) e uma variável (*otimo*) para indicar se a solução ótima foi encontrada ou não. Esta variável poderá ter dois valores: 1 caso a solução ótima tenha sido encontrada e 0 caso contrário.

Para associar apenas vértices de mesmo grau, utilizamos a estrutura de blocos apresen-

[1]Nó irmão é um outro nó que advém do mesmo nó pai

[2]Nó tio é um nó irmão do nó pai

tada no Capítulo 4 limitando as associações possíveis entre os vértices. Implementamos o algoritmo recursivamente e em cada chamada do algoritmo, caso a solução ótima não tenha sido encontrada, geramos uma nova associação de vértices (s) e a inserimos na solução se ela for válida (*valido(s)*), isto é, se a inclusão de s na solução irá agregar apenas associações de arestas de mesmo valor ao subgrafo resultante (linhas 13 à 15). Em seguida, o algoritmo é chamado recursivamente para a inserção de mais uma associação de vértices (linha 16).

Quando a associação s não é válida então realizamos o *backtracking* na árvore, isto é, a descida na árvore pára neste nó e retornamos para o nó pai, para continuar a descida a partir dos nós irmãos (linhas 19 e 20). Se após explorarmos todos os nós, filhos de um mesmo nó pai, não conseguirmos descer até um nó folha, então é realizado um *backtracking* para um nível acima na árvore (linha 21) e continuamos a descida em algum nó tio.

Nas linhas 2 à 10 do Algoritmo 7 tratamos os casos em que todos os vértices de um bloco foram inseridos na solução. Na chamada do algoritmo em que isso ocorre, não inserimos uma nova associação de vértices à solução, mas realizamos a mudança de bloco quando a solução não está completa (isto é, a solução não é uma folha da árvore de busca) e então realizamos a chamada recursiva do algoritmo para a inserção dos vértices restantes (linhas 3 à 5). Quando a solução passada para o algoritmo é completa (isto é, encontramos um isomorfismo), a variável *otimo* recebe o valor 1 para indicar que a solução ótima foi encontrada e o algoritmo pára (linhas 6 à 9).

O exemplo abaixo ilustra os passos do algoritmo para determinar a existência ou não do isomorfismo entre dois grafos, utilizando o algoritmo exato aplicado ao problema reformulado.

Exemplo: Para explicar os passos do algoritmo vamos utilizar os grafos ilustrados na Figura 5.1. Os grafos possuem seis vértices e nove arestas cada um. Pelo Teorema 3.1, sabemos que aplicando a reformulação do problema a esses dois grafos, o custo da melhor solução será 9, caso eles sejam isomorfos.

Aplicando a transcrição descrita no Capítulo 3, obtemos os grafos $G_1^{'}$ e $G_2^{'}$ ilustrados na Figura 5.2. Podemos observar que os vértices podem ser divididos em três blocos de acordo com o grau. O primeiro bloco com vértices de grau 2 (A), o segundo com vértices de grau 3 (B) e o terceiro com vértices de grau 4 (C). Ilustramos cada bloco com uma cor diferente para facilitar a análise do processo de busca pela solução aplicando o algoritmo exato. A Figura 5.2 mostra a estrutura de armazenamento dos vértices utilizado pelo algoritmo exato. Utilizamos três vetores, um para armazenar os vértices do grafo $G_1^{'}$, um para armazenar a solução do problema (permutação dos vértices do grafo $G_2^{'}$), e um

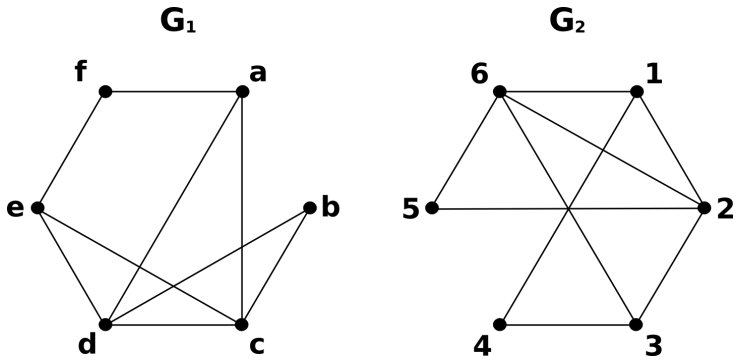


Figura 5.1: 2 grafos não completos com 6 vértices e 9 arestas cada um

terceiro para armazenar os vértices ainda não inseridos na solução. Durante a explicação do exemplo vamos nos referir a esse terceiro vetor como a lista de vértices candidatos.

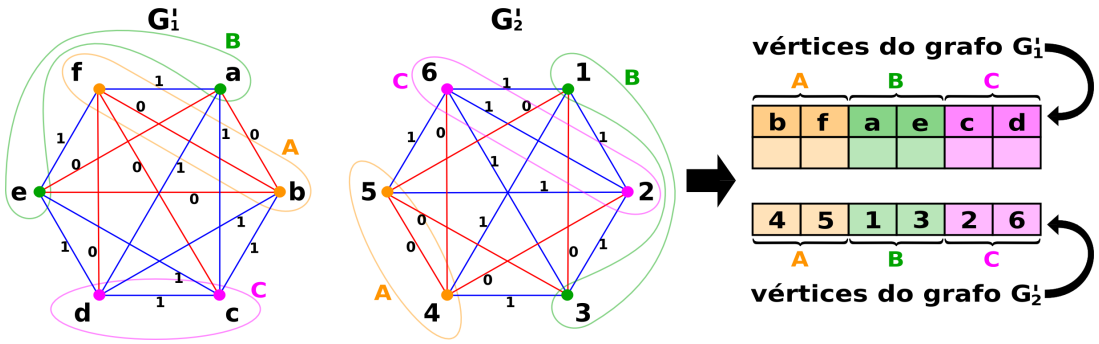
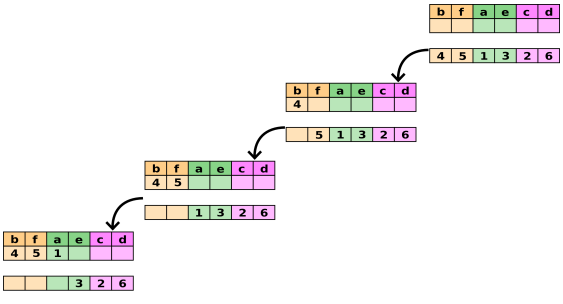


Figura 5.2: Transcrição dos grafos G_1 e G_2 para os grafos G_1' e G_2', e a representação dos dados para processamento do algoritmo exato

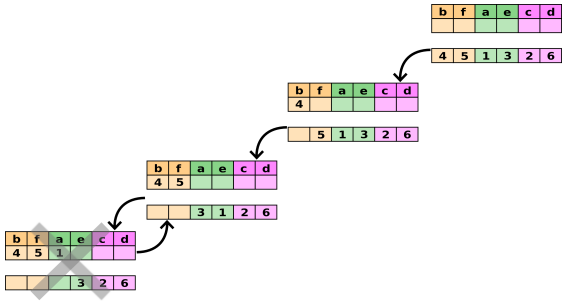
O algoritmo inicia com o vetor solução vazio e a cada nível da árvore inserimos uma nova associação de vértices. A Figura 5.3(a) ilustra esse procedimento. A nova associação apenas será inserida na solução se ela agregar ao grafo resultante sobreposições de arestas de valores iguais. A viabilidade dos graus dos vértices é garantida pelo fato de associarmos apenas vértices pertencentes ao mesmo bloco.

Podemos observar que a associação de vértices $(4-b)$ e $(5-f)$ geram a sobreposição de duas arestas de valor 0. Então essas associações são válidas. Na Figura 5.3(b) o algoritmo tenta inserir a associação $(1-a)$ à solução, mas esta gera duas sobreposições de arestas de valores diferentes, logo a associação é descartada e a busca a partir daquele nó é interrompido. Então o algoritmo realiza o *backtracking* na árvore, inserindo o vértice 1 no final do bloco B.

Em seguida, o algoritmo tenta inserir a associação $(3-a)$ (Figura 5.4(a)), mas esta também gera duas sobreposições de arestas de valores diferentes. Então ele realiza *backtracking* na árvore. Como todas as associações possíveis no bloco B não foram válidas,



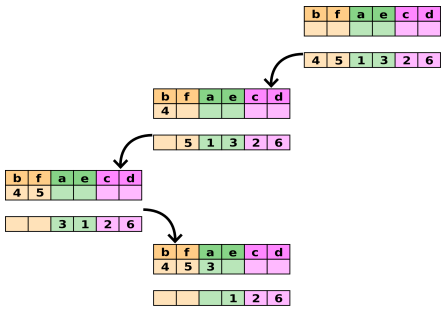
(a) Descida na árvore de busca



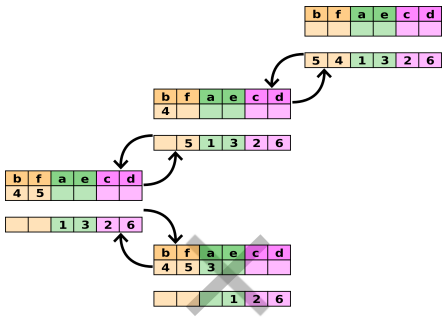
(b) Primeira associação do bloco B inválida (associação $(1-a)$), interrupção na descida e *backtracking* para o nó pai.

Figura 5.3: Execução do algoritmo exato

dadas as associações do bloco A, então a busca na árvore a partir do nó atual é interrompida e o algoritmo realiza outro *backtracking*, retornando a exploração ao bloco A. Neste momento, há apenas uma associação de vértices possível de ser realizada (associação $(5-f)$), que acaba de ser desfeita. Portanto, o algoritmo retorna mais um nível na árvore, desfazendo a associação $(4-b)$ e inserindo o vértice 4 ao final do bloco A (Figura 5.4(a)).



(a) Descida na árvore, testando a próxima associação de vértices do bloco B (associação $(3-a)$)



(b) Associação $(3-a)$ inválida, interrupção na descida e *backtracking* na árvore até a raiz.

Figura 5.4: Execução do algoritmo exato

A busca continua com as associações $(5-b)$, $(4-f)$ e assim por diante, seguindo a ordem mostrada na Figura 7. Como podemos observar, descendo seis níveis da árvore, conseguimos associar os seis vértices do grafo G_2' aos vértices do grafo G_1', encontrando o isomorfismo em um nó folha da árvore. Vale lembrar que a altura máxima da árvore é igual ao número de vértices de um dos grafos e, portanto, sempre que o algoritmo chegar em um nó folha, podemos concluir que os grafos são isomorfos. Caso os grafos não sejam isomorfos, o algoritmo terminará no nó raiz após esgotar as possibilidades de associações de vértices do primeiro bloco.

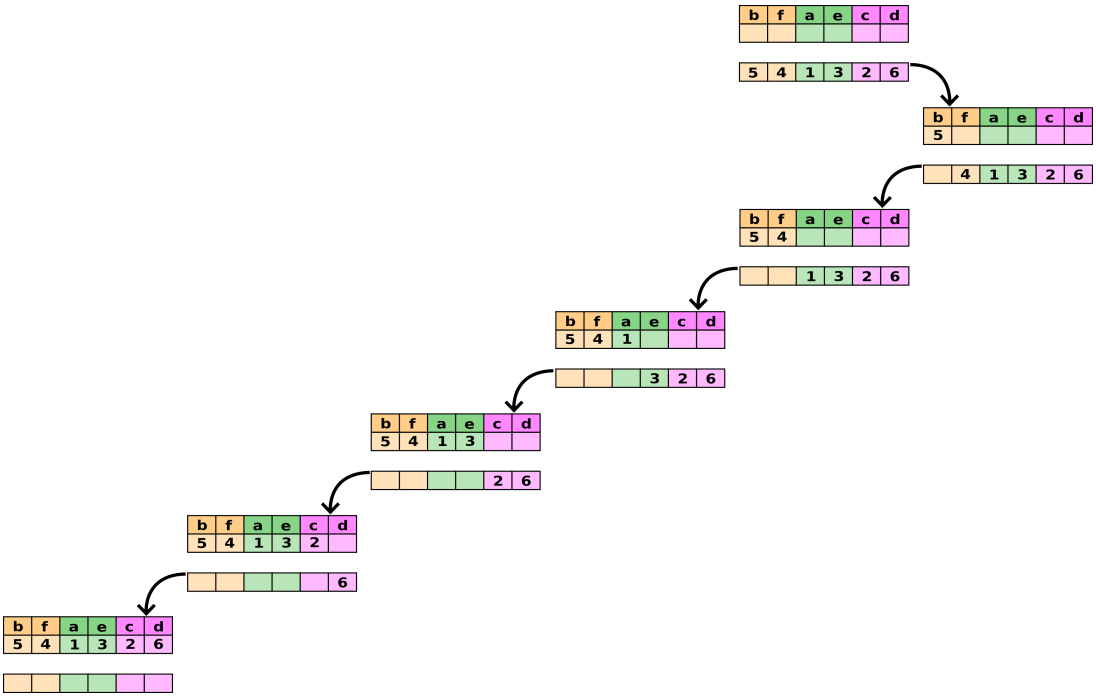


Figura 5.5: Execução do algoritmo exato: solução ótima

Como foi descrito acima, o *backtracking* na árvore consiste em passar o último vértice do vetor de solução para a lista de vértices candidatos na última posição do bloco ao qual ele pertence. Dessa forma podemos implementar o algoritmo utilizando apenas os três vetores e as matrizes de fluxos e de distâncias. Com isso a complexidade de espaço do algoritmo para tratar grafos de n vértices é $\Theta(n^2)$.

A seguir apresentaremos os resultados computacionais dos algoritmos implementados e os compararemos com os resultados obtidos da execução do algoritmo de Ullman implementado por [Messmer, 1996] e adaptado por [Boeres and Sarmento, 2005].

Capítulo 6

Resultados Computacionais

Neste capítulo apresentamos os resultados computacionais da aplicação dos algoritmos descritos no Capítulo 5 e do algoritmo de Ullmann sobre parte da base de dados da biblioteca VFLib. Utilizamos a linguagem C para implementação dos algoritmos GPIGPQA, Busca Tabu e Exato, com a biblioteca time.h para a obtenção dos tempos de execução. O compilador utilizado foi o gcc, versão 4.1.2. Os testes foram realizados em um processador AuthenticAMD, AMD Hammer Family processor - Model Unknown, 1600 MHz, com 447 MB de memória RAM, sistema operacional Linux Ubuntu 7.04 e kernel 2.6.20.

6.1 Base de dados

Para a realização dos testes dos algoritmos implementados e a análise de eficiência da reformulação, utilizamos parte das instâncias da base de dados da biblioteca VFLib. Esta foi sistematicamente gerada com o objetivo de servir como base de *benchmark* para algoritmos que se propõem a resolver o Problema de Isomorfismo de Grafos e suas variações (como o problema de isomorfismo de sub-grafos). Esta biblioteca foi desenvolvida pelo Laboratório de Sistemas Inteligentes e Visão Artificial (SIVALab) da Universidade de Nápoles "Federico II", Itália.

Essa base de dados é formada por pares de grafos divididos em categorias, totalizando 72.800 pares, dos quais 18.200 são pares de grafos isomorfos. Cada categoria é composta por grafos que se diferem na estrutura e no tamanho. Neste trabalho utilizamos a categoria dos grafos conectados aleatoriamente, que possui um total de 3.000 pares de grafos isomorfos, divididos em três grupos de 1.000 grafos de densidades $\eta = 0,01$, $\eta = 0,05$ e $\eta = 0,1$. Em cada grupo temos grafos de tamanhos que variam de 20 vértices até 1000

vértices, sendo 100 pares de grafos de cada tamanho.

Nos grafos dessa categoria as arestas conectam vértices sem nenhuma regra determinada. É assumido que a probabilidade de uma aresta conectar dois vértices do grafo é independente dos vértices. Para gerar esses grafos foi fixado o valor η de probabilidade de uma aresta estar presente entre dois vértices distintos i e i'. A distribuição de probabilidade é assumida como uniforme.

O parâmetro η significa que, se n é o número total de vértices do grafo, o número de suas arestas será igual a $\eta \cdot n \cdot (n-1)$. Entretanto, se este número não é suficiente para obter um grafo conectado (i.e. no mínimo $n-1$ arestas), mais arestas são adicionadas de forma adequada até que o grafo gerado seja conexo.

6.2 Algoritmo exato proposto por Ullmann

Para analisar a eficiência da reformulação, comparamos os resultados obtidos da execução do algoritmo exato apresentado na Seção 5.3 aplicado ao problema reformulado com os resultados do algoritmo exato proposto por [Ullmann, 1976] aplicado à formulação original do PIG. Este algoritmo está presente na biblioteca GUB (Graph matching toolkit of the University of Berne) implementado por [Messmer, 1996]. Para os testes, utilizamos uma simplificação da biblioteca GUB feita por [Boeres and Sarmento, 2005], pois a aplicação existente em GUB possui funcionalidades que vão além da identificação de um isomorfismo existente entre dois grafos.

O algoritmo de Ullman realiza uma busca enumerativa no espaço de soluções, utilizando o método de descida em profundidade para a exploração do espaço de busca. O algoritmo trabalha com os dois grafos ao mesmo tempo. Para isso, ele define uma matriz de isomorfismo para armazenar os vértices associados no decorrer da busca, além de utilizar as matrizes de adjacências de cada grafo.

A matriz de isomorfismo possui apenas células de valores 1 e 0. A idéia do algoritmo é definir uma célula com valor 1 por linha, de forma que em cada coluna haja apenas uma célula com valor 1 ao final do algoritmo. Cada linha da matriz representa uma associação de vértices. O algoritmo explora as adjacências dos vértices para reduzir o espaço de busca. A cada nível da árvore uma linha é definida, e antes de seguir para o próximo nível, o algoritmo descarta as associações de vértices não adjacentes aos recém-associados com outros adjacentes a eles.

O algoritmo abandona a busca em um ramo da árvore quando todas as células de

alguma linha da matriz de isomorfismo forem zeradas. A altura máxima da árvore de busca será igual ao número de vértices de um dos grafos. O algoritmo pára quando alcança uma folha, pois encontrou um isomorfismo entre os grafos, ou quando esgota as possibilidades de associações entre vértices (quando os grafos não são isomorfos).

6.3 Resultados obtidos

A seguir apresentaremos os resultados obtidos em cada algoritmo. Na Seção 6.4 analisamos as características das instâncias utilizadas e sua influência nos resultados. Na Seção 6.5 comparamos os resultados entre os algoritmos heurísticos. Na Seção 6.6 comparamos os resultados dos algoritmos exatos.

Da base de dados descrita na Seção 6.1 nomeamos cada grupo de grafos de acordo com a sua densidade de arestas, ou seja, r001 se refere ao grupo de grafos com densidade $\eta = 0,01$ e assim por diante. Para analisar os resultados obtidos, dividimos cada grupo em subgrupos de acordo com o número de vértices dos grafos. Assim, obtivemos 10 subgrupos de dados em cada grupo de grafos (20, 40, 60, 80, 100, 200, 400, 600, 800, 1000).

Outro fator importante a ser observado nos algoritmos apresentados no Capítulo 5 é a forma como tratamos as informações dos grafos. Como vimos no Capítulo 4, separamos os vértices em blocos para otimizar o processo de construção da solução e a busca local, de forma que apenas vértices de mesmo grau sejam associados. O número de arestas dos grafos apenas indica o valor máximo possível da função de custo, não influenciando diretamente na busca pela solução ótima.

Como os algoritmos GPIGPQA e Busca Tabu possuem fator aleatório, realizamos cinco execuções de cada algoritmo para cada instância da base de dados.

Os resultados serão apresentados através de gráficos. As tabelas com as informações referentes aos gráficos são apresentados no Apêndice A.

6.3.1 Resultados do GPIGPQA

O GPIGPQA requer que alguns parâmetros sejam definidos. Vimos na Seção 5.1 que o parâmetro β limita o intervalo de elementos da LRC possíveis de serem sorteados, e o parâmetro α limita o intervalo de elementos da lista de associações válidas de vértices (L_v) gerada na segunda etapa da construção da solução inicial. Como a LRC é formada apenas de associações de arestas com valor 1, então não há a necessidade de limitar o intervalo

de sorteio, logo adotamos $\beta = 1$. Já para o parâmetro α, testamos o desempenho do algoritmo em uma amostra da base de testes para α valendo 0,2, 0,3, 0,4, 0,5, 0,6 e 0,75. Obtivemos melhores resultados para $\alpha = 0,6$. Limitamos o número de iterações do algoritmo para 1000 iterações.

A seguir apresentamos os resultados computacionais obtidos. Analisamos o número médio de iterações, o tempo médio das iterações e a qualidade das soluções das instâncias em que não obtivemos a solução ótima.

• Número médio de iterações:

Analisamos a média dos números de iterações que o algoritmo GPIGPQA precisou para encontrar a solução ótima. Das cinco execuções realizadas para cada instância, extraímos o maior número de iterações, o menor e o número médio de iterações. Em seguida tiramos a média de cada um desses conjuntos, para cada subgrupo de instâncias. A Figura 6.1 mostra os gráficos obtidos das médias das iterações descritas acima.

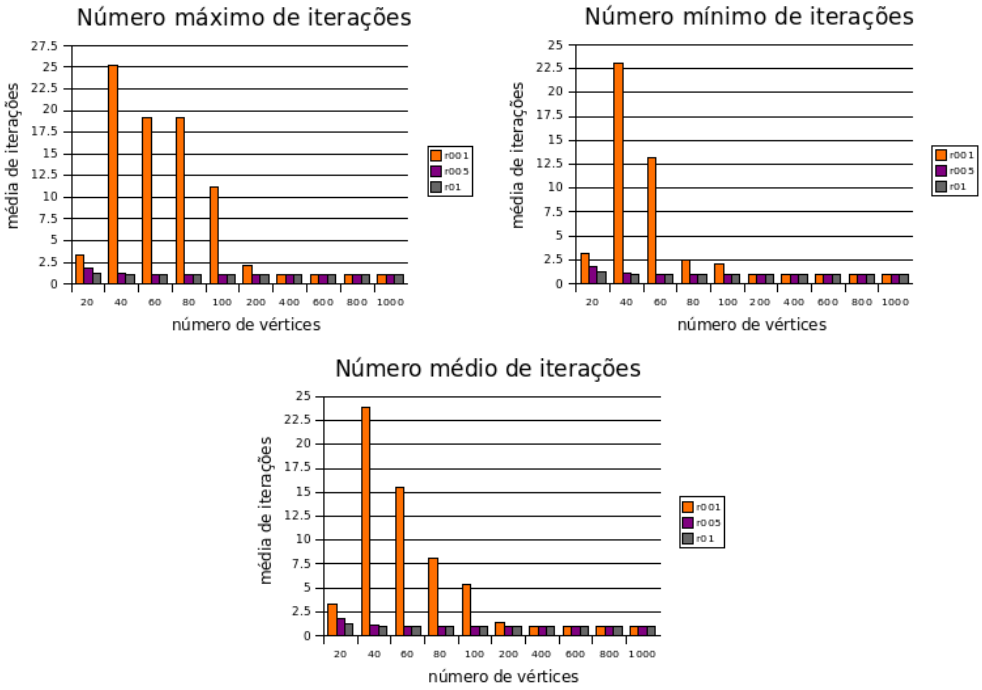


Figura 6.1: GPIGPQA – Gráficos das médias dos números de iterações

• Tempo médio das iterações:

Analisamos a média dos tempos das iterações do algoritmo GPIGPQA para a base de dados utilizada. Assim como fizemos para a análise da média de iterações, extraímos o maior valor de tempo médio, o menor valor e a média dos tempos médios de iterações das 5 execuções realizadas para cada instância. Em seguida tiramos a média de cada um desses conjuntos, para cada subgrupo de instâncias. A Figura 6.2 mostra os gráficos

obtidos das médias dos tempos analisados.

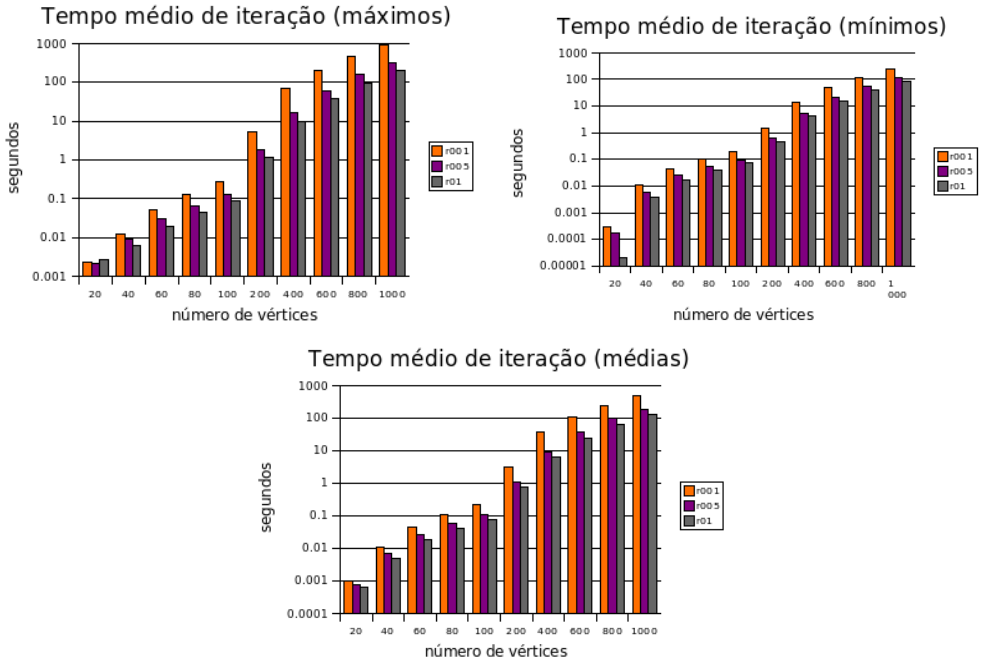


Figura 6.2: GPIGPQA – Gráficos das médias de tempo das iterações

• **Análise dos resultados:**

Observamos que dos três grupos analisados, obtivemos melhores resultados para os grafos dos grupos cuja densidade de arestas é maior. Analisamos os tempos necessários em cada etapa do processo, e observamos que houve uma melhora considerável de tempo na construção da solução inicial e na busca local para instâncias de maior densidade de arestas.

• **Qualidade das soluções:**

Das 3.000 instâncias testadas, não encontramos a solução ótima para três instâncias. Então analisamos o custo da melhor solução obtida e em qual iteração encontramos esse resultado. Na Tabela 6.1 são apresentadas, para cada uma das três instâncias, o número de vértices, o custo ótimo, o custo da melhor solução obtida, a iteração onde esta foi encontrada e sua distância em relação ao valor ótimo (em percentual).

Observamos que, além de ter obtido soluções próximas ao ótimo, o algoritmo GPIGPQA converge rapidamente para boas soluções.

Número de vértices	Custo ótimo	Custo da melhor solução	Iteração	Dist. do ótimo (%)
40	41	39	3	4.88
40	41	39	1	4.88
60	69	67	4	2.90

Tabela 6.1: Resultados obtidos das instâncias onde o algoritmo GPIGPQA não alcançou o ótimo.

6.3.2 Resultados da Busca Tabu

Como vimos na Seção 5.2 o algoritmo de Busca Tabu aplica uma penalidade ao movimentos da vizinhança de uma solução quando não temos nenhuma troca que gere uma melhora em seu custo. Essa penalidade é proporcional à freqüência de ocorrências de cada movimento, ou seja, a atratividade do movimento é subtraído do número de ocorrências multiplicado por um peso. A equação (6.1) apresenta a fórmula matemática para o cálculo da atratividade do movimento penalizado.

$$c_{ij}^{novo} = c_{ij} - \gamma \cdot f_{ij}. \tag{6.1}$$

Onde c_{ij} é o valor que o movimento agrega ao custo da solução, γ é o peso dado à freqüência do movimento e f_{ij} é o número de ocorrências da troca entre as células i e j no decorrer da busca.

Analisamos o desempenho do algoritmo para valores de γ iguais a 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8, 0,9 e 1,0, e encontramos melhores resultados para $\gamma = 0,5$. Definimos que a duração da classificação tabu é de 4 iterações e limitamos o número máximo de iterações do algoritmo para 5000 iterações. Executamos o algoritmo para as instâncias descritas na Seção 6.1 cujos tamanhos vão de 20 à 600 vértices, totalizando 2.400 instâncias. Para as instâncias de 800 e 1000 vértices não obtivemos os resultados em tempo viável.

Assim como fizemos para o algoritmo GPIGPQA, analisamos o desempenho do algoritmo de Busca Tabu quanto ao número médio de iterações, ao tempo médio de iterações e à qualidade das soluções em que não obtivemos a solução ótima. A seguir apresentamos os resultados computacionais obtidos.

• Número médio de iterações:

Analisamos a média dos números de iterações que o algoritmo de Busca Tabu precisou para encontrar a solução ótima. Repetimos os mesmos procedimentos realizados com os resultados do algoritmo GPIPQA, das cinco execuções realizadas para cada instância,

extraímos o maior número de iterações, o menor e o número médio de iterações. Em seguida tiramos a média de cada um desses conjuntos, para cada subgrupo de instâncias. A Figura 6.3 mostra os gráficos obtidos das médias das iterações descritas.

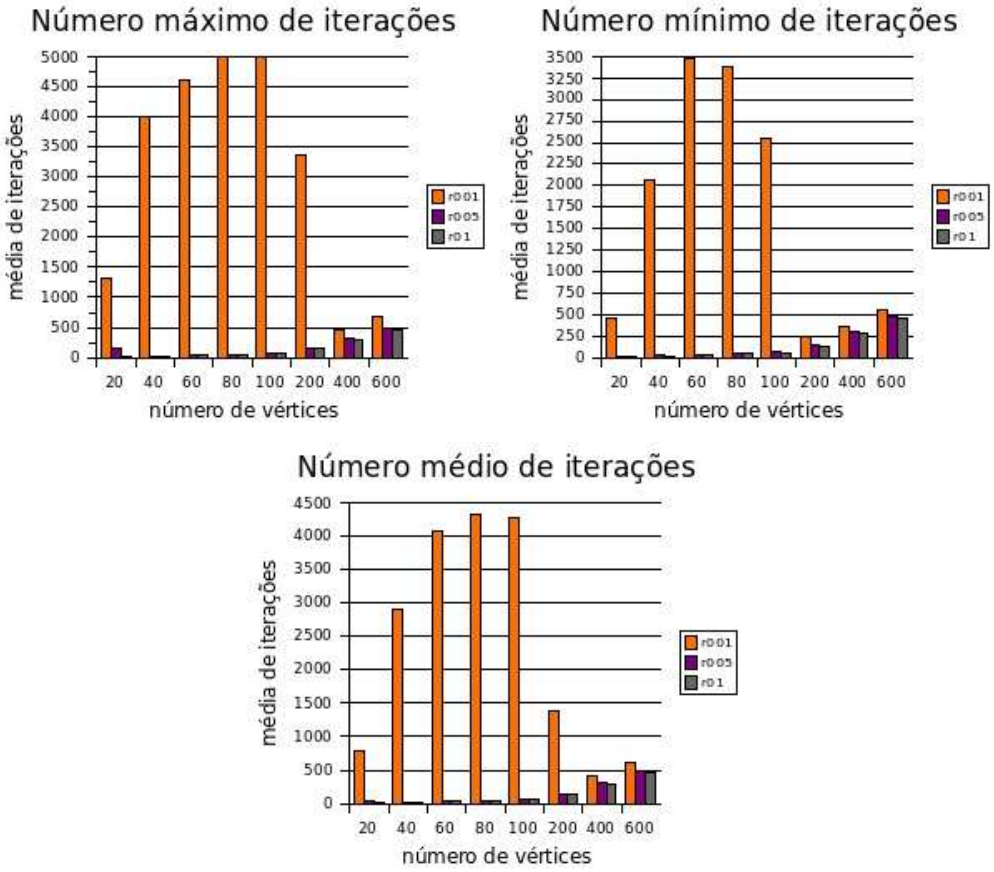


Figura 6.3: Busca Tabu – Gráficos das médias dos números de iterações

• **Tempo médio das iterações:**

Analisamos a média dos tempos das iterações do algoritmo de Busca Tabu para a base de dados utilizada. Extraímos o maior valor de tempo médio, o menor valor e a média dos tempos médios de iterações das 5 execuções realizadas para cada instância. Em seguida tiramos a média de cada um desses conjuntos, para cada subgrupo de instâncias. A Figura 6.4 mostra os gráficos obtidos das médias dos tempos analisados.

• **Análise dos resultados:**

Podemos observar que o algoritmo de Busca Tabu teve maior dificuldade para encontrar o ótimo nas instâncias de menor densidade de arestas. Isso vale tanto para o número de iterações do algoritmo como para o tempo médio gasto por iteração.

• **Qualidade das soluções:**

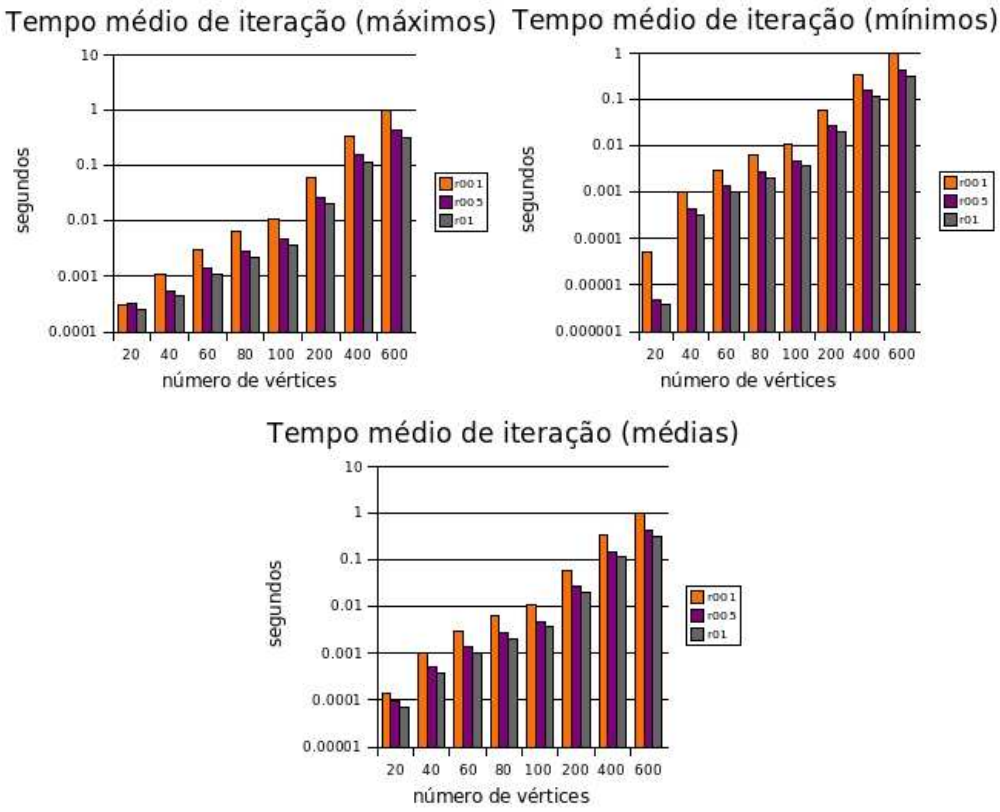


Figura 6.4: Busca Tabu – Gráficos das médias de tempo das iterações

Das 2.400 instâncias testadas, o algoritmo de Busca Tabu não encontrou o isomorfismo para 233 instâncias. Assim como fizemos para o algoritmo GPIGPQA, analisamos a qualidade das soluções encontradas com relação ao valor da melhor solução obtida e em qual iteração ela foi encontrada. Como se tratam de muitas instâncias, resolvemos analisar a média das diferenças entre o custo da solução ótima e o custo da solução obtida proporcionalmente ao número de vértices das instâncias. Na Tabela 6.2 apresentamos essa análise.

N_v	N_I	Med_{Iter}	$Med_{DifCusto}$	$Med_{DistOtimo}$ (%)
20	8	3,625	2,625	13,593993
40	38	5,868421	4,868421	11,547488
60	66	10,878788	9,878788	14,265398
80	71	23,366197	22,366197	21,964251
100	49	39,857143	38,857143	27,613092
200	1	299	298	67,135262

Tabela 6.2: Resultados obtidos das instâncias onde o algoritmo de Busca Tabu não alcançou o ótimo.

A primeira coluna (N_v) indica o número de vértices das instâncias, a segunda coluna

(N_I) indica o número de instâncias para as quais o algoritmo não encontrou o isomorfismo. A coluna Med_{Iter} mostra a média do número de iterações que o algoritmo precisou para chegar aos resultados obtidos. A coluna seguinte ($Med_{DifCusto}$) apresenta a média das diferenças de custo entre a solução ótima e a solução encontrada pelo algoritmo, e a coluna $Med_{DistOtimo}$ mostra o valor percentual das distâncias apresentadas em $Med_{DifCusto}$.

Observamos que o algoritmo de Busca Tabu não encontrou isomorfismo para 9,7% das instâncias testadas. Vimos pelas análises apresentadas que, para estas instâncias, as melhores soluções encontradas se distanciam do ótimo de aproximadamente 19,18% (média ponderada dos valores de $Med_{DistOtimo}$) e o algoritmo converge rapidamente para esses valores (observando-se a coluna Med_{Iter}). Em comparação ao algoritmo GPIGPQA a Busca Tabu não apresentou bom desempenho, tanto pela qualidade dos resultados quanto pelo tempo de execução do algoritmo.

6.3.3 Resultados do Algoritmo Exato aplicado à Reformulação

Dos três grupos de instâncias apresentados na Seção 6.1, não obtivemos resultados em tempo computacional viável para as instâncias do grupo r001. Então analisamos apenas os resultados obtidos da execução do algoritmo para as instâncias dos grupos r005 e r01 (totalizando 2.000 instâncias), que são apresentados na Figura 6.5.

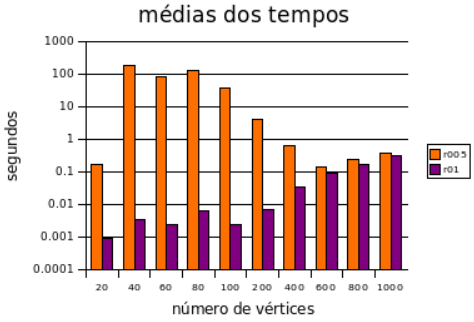


Figura 6.5: Gráfico das médias de tempo do Algoritmo Exato

Observamos que, assim como para as meta-heurísticas, o Algoritmo Exato obteve tempos melhores para as instâncias de maior densidade de arestas e encontrou maior dificuldade para encontrar a solução ótima para instâncias com um número de vértices considerado pequeno, obtendo melhores tempos para as instâncias com maior número de vértices. Na Seção 6.4 analisaremos com mais detalhes as características das instâncias e os motivos que levaram a esses resultados.

6.3.4 Resultados do Algoritmo de Ullmann

Executamos o Algoritmo de Ullmann para instâncias dos grupos r001, r005 e r01, com tamanhos que variam entre 20 até 400 vértices. Para as instâncias com 600, 800 e 1000 vértices tivemos problemas com a adaptação do algoritmo e também com relação à memória. A Figura 6.6 apresenta os resultados obtidos.

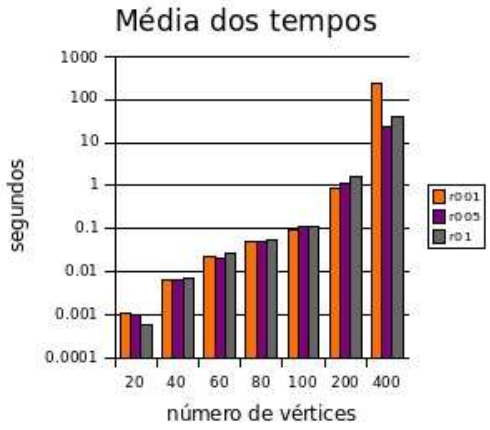


Figura 6.6: Gráfico das médias de tempo do Algoritmo de Ullmann

Observamos que o Algoritmo de Ullmann apresentou melhor desempenho para instâncias do grupo r001 com 40 à 200 vértices, mas aparentemente não mostrou o mesmo desempenho para dois subgrupos de instâncias (20 e 400 vértices). Observamos que a forma de exploração do espaço de busca pode influenciar o comportamento diferenciado do algoritmo, pois nesses dois casos observamos que para algumas instâncias o algoritmo demandou um elevado tempo de execução.

6.4 Análise das características das instâncias

Como vimos na seção anterior, obtivemos uma diferença notável de desempenho dos algoritmos apresentados no Capítulo 5 com relação à densidade das arestas dos grafos. Também vimos que os algoritmos tiveram mais dificuldade de encontrar o ótimo para as instâncias do grupo r001 com tamanhos entre 40 e 200 vértices. Então resolvemos analisar as características das instâncias entre os grupos e das instâncias entre si do grupo r001.

No Capítulo 4 estudamos a influência da qualidade da divisão dos vértices em blocos. Vimos que quanto mais homogêneo for a distribuição dos vértices nos blocos, menor é o espaço de soluções viáveis para o problema. Mas além disso, se os blocos forem homogêneos, então também teremos uma redução no tamanho do espaço de busca com

o aumento do número de blocos. Vale ressaltar que essas características dependem das instâncias do problema.

• Quanto à variação da densidade das arestas:

Para estudar as características dos blocos entre as instâncias dos grupos r001, r005 e r01, analisamos a média do número de blocos das instâncias, estas separadas de acordo com seus tamanhos. A Tabela 6.4 mostra os valores encontrados.

	r001	r005	r01
20	4,43	5,39	6,65
40	5,69	8,14	10,35
60	6,55	10,59	13,51
80	7,05	12,73	16,06
100	7,87	14,33	18,35
200	10,9	21,62	27,32
400	16,51	32,82	41,42
600	21,06	40,94	52,08
800	24,71	47,74	61,63
1000	27,76	54,61	69,5

Tabela 6.3: Médias do número de blocos em cada conjunto de instâncias.

Observamos que quanto maior a densidade de arestas, maior é a média do número de blocos. Mas apenas essa característica não é suficiente para podermos chegar a uma conclusão, então resolvemos analisar a distribuição dos vértices nos blocos. Observamos que as distribuições possuem uma forma semelhante quando analisamos instâncias de mesmo número de vértices mas de grupos diferentes. A Figura 6.4 mostra um exemplo das instâncias que analisamos.

Após a análise podemos ver que não temos uma distribuição homogênea dos vértices nos blocos, mas que quanto maior o número de blocos, menos vértices temos por bloco. Fator que contribuiu para a diminuição do espaço de busca (Corolário 4.2) e que justifica uma maior facilidade para encontrar o isomorfismo nas instâncias de maior densidade de arestas.

• Quanto à distribuição dos vértices nos blocos:

Como observamos os algoritmos apresentados no Capítulo 5 tiveram maior dificuldade para encontrar o isomorfismo para instâncias com 40 a 200 vértices. Então estudaremos agora as características dessas instâncias que levaram a esses resultados. A Tabela 6.4

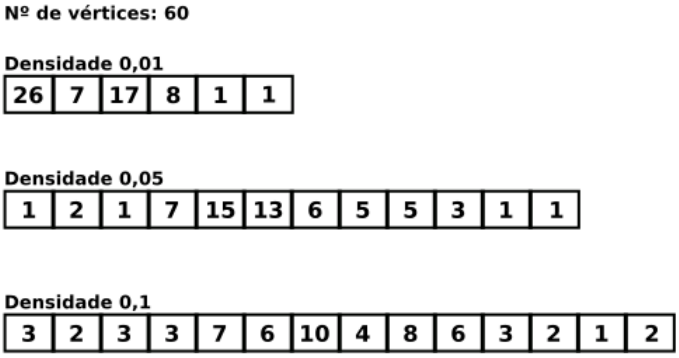


Figura 6.7: Distribuição dos vértices nos blocos para instâncias de 60 vértices dos grupos r001, r005 e r01.

mostra os coeficientes de distâncias médias entre os blocos, proporcionais ao número de vértices das instâncias. A primeira coluna da tabela indica o número de vértices dos grafos. As três colunas seguintes (r001, r005 e r01) apresentam os coeficientes de distância média dos grupos de instâncias indicados na primeira linha de cada coluna.

Para calcular esse coeficiente, calculamos a média dos desvios padrões[1] entre os blocos de uma instância. O desvio padrão (dv) é calculado da seguinte forma: seja $e_j = \frac{n_v}{n_b}$ o número esperado de vértices em um bloco de uma instância j, onde n_v é o número de vértices e n_b é o número de blocos, e seja b_i o número de vértices do bloco i. O desvio padrão das distâncias entre os blocos de uma instância j pode ser dado pela equação (6.2).

$$dv_j = \sqrt{\frac{\sum_{i=1}^{n_b}(b_i - e_j)^2}{n_b}}. \tag{6.2}$$

O coeficiente de distância média entre os blocos de um conjunto de instâncias (c_{dist}) é calculado através da equação (6.3)

$$c_{dist} = \frac{dv_{media} \cdot 100}{n_v} \tag{6.3}$$

onde dv_{media} é a média dos desvios padrões das instâncias de um mesmo conjunto.

Podemos observar que os grupos de instâncias r001, r005 e r01 possuem um mesmo comportamento quanto à distribuição de vértices nos blocos. As instâncias possuem uma distribuição menos homogênea quando temos um menor número de vértices. Mas podemos notar que a falta de homogeneidade é mais marcante nos conjuntos de instâncias do grupo r001, o que justifica a notável dificuldade dos algoritmos para tratar instâncias desse grupo. Com a análise da média do número de blocos em cada grupo feita anteriormente, podemos

[1] fórmula de desvio padrão pesquisada de [Morettin, 1999]

	r001	r005	r01
20	16.501625	10.40218	9.459655
40	15.000673	7.643865	6.272713
60	13.013323	6.44427	5.00947
80	11.211293	5.476966	4.30906
100	10.074744	4.957157	3.835184
200	6.927215	3.454112	2.640045
400	5.054707	2.419322	1.843277
600	4.153167	1.971817	1.497952
800	3.595204	1.710097	1.309857
1000	3.228591	1.526805	1.156056

Tabela 6.4: Coeficientes de distâncias entre os blocos de vértices.

observar que a quantidade de blocos das instâncias teve grande influência na distribuição dos vértices nos blocos.

6.5 Comparação entre as meta-heurísticas

Na Seção 6.3 apresentamos os resultados computacionais das meta-heurísticas isoladamente. Agora, apresentamos a comparação de desempenho entre elas. A Figura 6.8 apresenta os gráficos comparativos entre os dois algoritmos.

Os gráficos mostram as médias dos tempos totais de execução de cada algoritmo em cada grupo de instâncias. Comparamos o desempenho entre os algoritmos para cada um dos 10 subgrupos.

Podemos observar que para os subgrupos de instâncias com 40 ou mais vértices o GPIGPQA obteve melhores resultados. Como foi mencionado, no caso do algoritmo de Busca Tabu, realizamos testes para os subgrupos de instâncias com até 600 vértices, mas foi possível observar pelo tempo de execução de algumas instâncias dos subgrupos de 800 e 1000 vétices, que o desempenho do algoritmo de Busca Tabu foi pior que o desempenho do algoritmo GPIGPQA.

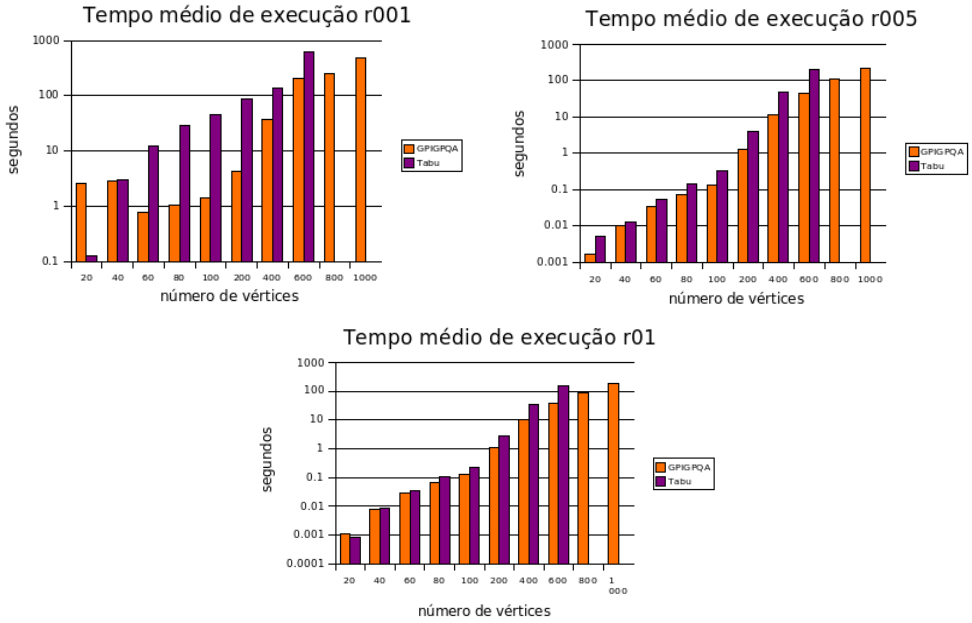


Figura 6.8: Gráficos das médias de tempo de execução dos algoritmos GPIGPQA (r001, r005, r01 e $n = 20$ *a* 1000) e Busca Tabu (r001, r005, r01 e $n = 20$ *a* 600)

6.6 Comparação entre os algoritmos exatos

Também apresentamos na Seção 6.3 os resultados do Algoritmo Exato e de Ullmann separadamente. Agora compararemos o desempenho entre os algoritmos e analisaremos os casos onde é mais vantajoso utilizar a reformulação do PIG ao invés da formulação tradicional. A Figura 6.9 mostra os gráficos comparativos com as médias dos tempos de execução dos algoritmos para cada grupo de instâncias. Estes são divididos em subgrupos, como foi explicado na Seção 6.3. Como foi mencionado na Seção 6.3, não conseguimos executar o Algoritmo de Ullmann para as instâncias com 600 vértices ou mais. Então realizamos a comparação de desempenho com os dados obtidos.

Podemos observar que para as instâncias do grupo r005 o Algoritmo Exato obteve melhores resultados para instâncias de 400 vértices. Mas pela ordem de crescimento do gráfico do Algoritmo de Ullmann e pelo fato de o Algoritmo Exato ter obtido uma média de tempo total de execução inferior a 0,5 segundo para as instâncias de 600, 800 e 1000 vértices, imaginamos que o primeiro obteria resultados piores que o Algoritmo Exato para essas instâncias.

Já para as instâncias do grupo r01, observamos que o algoritmo exato obteve melhores tempos de execução para instâncias com 40 ou mais vértices. Pela análise das características das instâncias, realizada na Seção 6.4, observamos que os subgrupos de instâncias com maior número de vértices possuem uma melhor distribuição dos vértices entre os blocos,

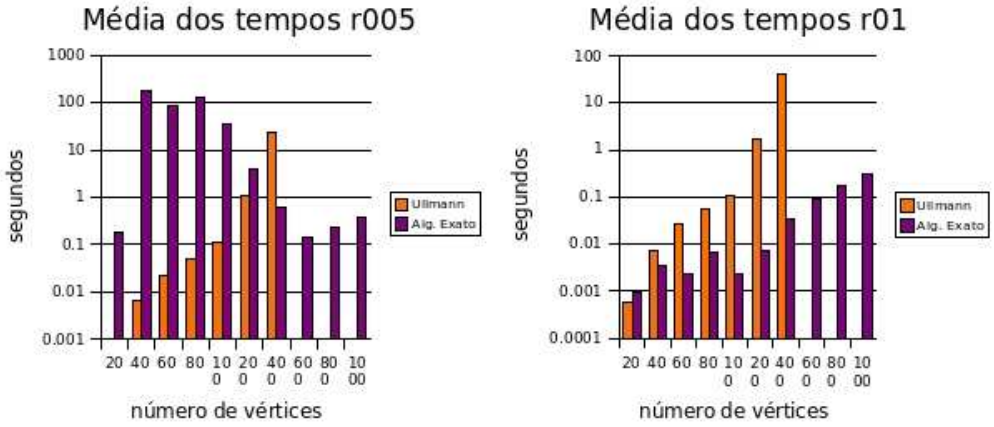


Figura 6.9: Gráficos comparativos de desempenho entre o Algoritmo Exato (r005, r01 e $n = 20, \ldots, 1000$) e o Algoritmo de Ullman (r001, r005, r01 e $n = 20, \ldots, 400$)

o que favoreceu a exploração da característica dos graus dos vértices e resultou no melhor desempenho do Algoritmo Exato para esses casos.

6.7 Comparação geral dos algoritmos

Agora faremos uma análise comparativa dos desempenhos de todos os algoritmos apresentados neste trabalho. A Figura 6.10 apresenta os gráficos com as médias de tempos totais de execução dos algoritmos GPIGPQA, Busca Tabu, Ullmann e Exato. Lembrando que o algoritmo de Busca Tabu foi executado para as instâncias dos grupos r001, r005 e r01 com tamanhos de 20 a 600 vértices, o algoritmo Exato não foi executado para as instâncias do grupo r001 e o algoritmo de Ullmann foi executado para as instâncias dos grupos r001, r005 e r01 com tamanhos de 20 a 400 vértices.

Com as análises realizadas sobre as características das instâncias e a influência dos graus dos vértices na dificuldade dos algoritmos apresentados no Capítulo 5 para encontrar o isomorfismo, observamos que os algoritmos GPIGPQA, Busca Tabu e Exato não obtiveram bons tempos de execução quando as instâncias possuem uma certa regularidade, ou seja, de acordo com a característica do isomorfismo explorada (graus dos vértices), quando os grafos possuem muitos vértices de mesmo grau. Porém, observamos que para os grafos gerados aleatoriamente, encontramos pouca regularidade com relação aos graus dos vértices para instâncias grandes, o que torna vantajoso a exploração dessa característica.

Observamos ainda que os algoritmos GPIGPQA e Busca Tabu obtiveram pior desempenho que os dois algoritmos exatos apresentados. Isso se deve ao fato de que em cada

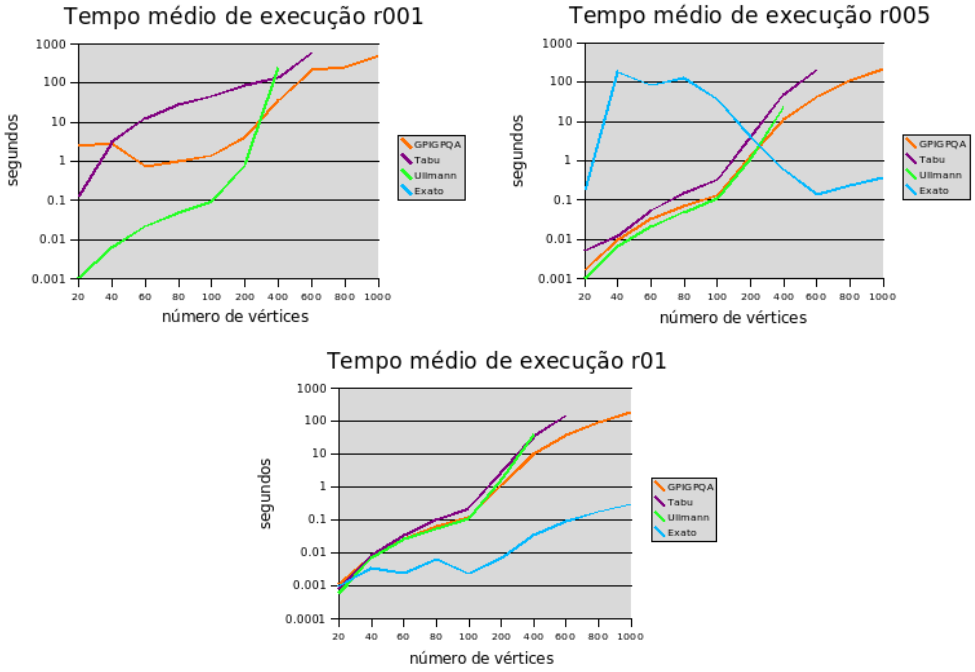


Figura 6.10: Gráficos comparativos de desempenho de todos os algoritmos (GPIGPQA – r001, r005, r01 e $n = 20, \ldots, 1000$; Busca Tabu – r001, r005, r01 e $n = 20, \ldots, 600$; Algoritmo de Ullmann – r001, r005, r01 e $n = 20, \ldots, 400$; Algoritmo Exato – r005, r01 e $n = 20, \ldots, 1000$)

iteração dessas meta-heurísticas geramos a lista de todos os vizinhos da solução corrente antes de realizar a escolha de um vizinho entre os melhores da vizinhança, e a geração dessa lista demanda muito tempo computacional.

Capítulo 7

Conclusão e trabalhos futuros

Neste trabalho apresentamos uma reformulação para o Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação e analisamos a aplicação das características do problema original no problema reformulado. Dentre as características destacamos a associação de vértices de mesmo grau para geração de soluções viáveis. Estudamos a influência dessa característica no tamanho do espaço de busca.

Validamos matematicamente a reformulação apresentada e implementamos três algoritmos para resolver o problema reformulado. Desses algoritmos dois são baseados em meta-heurísticas (GRASP e Busca Tabu) e o outro trata-se de um algoritmo exato baseado na técnica de descida em árvore. Além dos algoritmos implementados, utilizamos a versão do algoritmo de Ullmann apresentado no trabalho de [Boeres and Sarmento, 2005] para uma análise comparativa da eficiência da reformulação do PIG com a formulação tradicional do problema. Aplicamos esses algoritmos a um subconjunto da base de dados obtida da biblioteca VFLib.

A partir dos resultados obtidos, observamos que a densidade das arestas não influencia diretamente na eficiência dos algoritmos aplicados à reformulação, pois no processo de busca desses algoritmos focamos a atenção sobre as associações entre os vértices, e o número de arestas é utilizado apenas para a obtenção do custo da solução ótima. Porém, a densidade de arestas está ligada aos tamanhos dos blocos de vértices, e isto influencia no tamanho do espaço de busca do problema reformulado.

Observamos que, para os algoritmos apresentados no Capítulo 5, o fator que dificultou a busca foi o tamanho do espaço de soluções. Em uma análise geral, para as instâncias cuja distribuição de vértices nos blocos foi pouco homogênea e os tamanhos dos blocos foram "grandes" com relação ao número total de vértices, os algoritmos não encontraram

o ótimo em muitos casos.

Analisamos as divisões dos vértices nos grupos de grafos e observamos que os grafos de menor densidade de arestas possuem um menor número de blocos e uma distribuição menos homogênea dos vértices nos blocos. Enquanto os grafos com maior densidade de arestas possuem um número maior de blocos e uma quantidade reduzida de vértices por bloco.

No algoritmo GPIGPQA o tamanho dos blocos teve uma grande influência tanto no processo de geração da solução inicial como no processo de busca local, pois quando temos pequenos blocos de vértices, temos um número reduzido de associações válidas entre os vértices. Isso diminuiu o número de trocas realizadas no processo de busca local e o tamanho das listas de associações de vértices (L_v citado na Seção 5.1). Além da influência no tempo de processamento, o tamanho dos blocos de vértices influenciam na complexidade da busca pela solução, pois quanto maior é o tamanho dos blocos, maior será o espaço de soluções viáveis do problema, influenciando assim no número de execuções do algoritmo.

Nos algoritmos de Busca Tabu e Exato observamos que a dificuldade de encontrar o ótimo foi causado pelo aumento do espaço de busca. Para o primeiro algoritmo, isso influencia tanto no tempo médio das iterações como no número de iterações necessárias.

Comparamos os desempenhos dos algoritmos entre si, e vimos que entre os algoritmos GPIGPQA e o de Busca Tabu, o primeiro mostrou melhores resultados, tanto com relação ao tempo de execução como à qualidade das soluções encontradas. Entre os algoritmos Exato e Ullmann, vimos que para instâncias com maior densidade de arestas, o primeiro apresentou melhor desempenho que o segundo, mas não conseguiu processar as instâncias com menor densidade de arestas em tempo computacional viável. Realizamos uma análise geral dos algoritmos e vimos que os algoritmos implementados não obtiveram bons tempos de execução quando as instâncias possuem uma certa regularidade com relação aos graus dos vértices. Observamos também que os algoritmos GPIGPQA e Busca Tabu obtiveram um desempenho pior que os algoritmos exatos, e concluímos que esse comportamento se deve à forma de exploração da vizinhança da solução corrente e à estrutura de vizinhança adotada.

Com os conhecimentos obtidos, podemos decidir qual formulação do problema devemos adotar para a resolução do PIG, de acordo com as características dos grafos com relação aos graus dos vértices. Concluímos que a reformulação é vantajosa quando tratamos de grafos com uma grande diversidade de graus de vértices.

Para trabalhos futuros, pretendemos estudar uma outra característica dos grafos para resolução do PIG, que é a adjacência entre os vértices. Unindo a exploração dessa característica com a estudada neste trabalho, esperamos melhorar o desempenho da reformulação para grafos com pouca densidade de arestas, mantendo o bom desempenho apresentado para grafos com alta densidade de arestas. Além disso, pretendemos estudar uma outra estrutura de vizinhança para as meta-heurísticas implementadas.

Quanto aos resultados dos algoritmos, pretendemos executar o algoritmo de Busca Tabu para as instâncias de tamanhos 800 e 100 vértices, e ajustar o algoritmo de Ullman para realização dos testes para instâncias com mais de 400 vértices.

Neste trabalho analisamos o desempenho da reformulação para grafos gerados aleatoriamente, mas além desse grupo de grafos, pretendemos continuar a análise para grafos regulares e buscar mais características além daquelas já citadas.

Apêndice A

Tabelas dos resultados computacionais

A.1 Algoritmo GPIGPQA

Número médio de iterações:

Máximos	20	40	60	80	100	200	400	600	800	1000
r001	3.35	25.21	19.21	19.16	11.22	2.16	1.07	1	1	1
r005	1.83	1.18	1.03	1.01	1.02	1	1	1	1	1
r01	1.26	1.01	1	1	1	1	1	1	1	1

Tabela A.1: GPIGPQA: médias dos números máximos de iterações.

Mínimos	20	40	60	80	100	200	400	600	800	1000
r001	3.18	23.11	13.25	2.48	2.03	1	1	1	1	1
r005	1.78	1.18	1.03	1.01	1	1	1	1	1	1
r01	1.25	1.01	1	1	1	1	1	1	1	1

Tabela A.2: GPIGPQA: médias dos números mínimos de iterações.

Médias	20	40	60	80	100	200	400	600	800	1000
r001	3.264	23.924	15.566	8.068	5.342	1.36	1.014	1	1	1
r005	1.808	1.18	1.03	1.01	1.004	1	1	1	1	1
r01	1.254	1.01	1	1	1	1	1	1	1	1

Tabela A.3: GPIGPQA: médias dos números de iterações.

Tempo médio de iterações:

Máximos	r001	r005	r01
20	0.002304	0.002184	0.002800
40	0.012256	0.009027	0.006160
60	0.050160	0.030342	0.020081
80	0.129437	0.065244	0.046163
100	0.275394	0.127528	0.089246
200	5.337088	1.781991	1.142671
400	67.380031	15.945637	10.032107
600	206.507386	60.386294	39.439145
800	449.123668	157.407197	94.089720
1000	914.430108	313.749368	197.169362

Tabela A.4: GPIGPQA: médias dos tempos máximos de iterações.

Mínimos	r001	r005	r01
20	0.000283	0.000183	0.000020
40	0.010542	0.006087	0.003720
60	0.042119	0.024862	0.016561
80	0.099842	0.052823	0.039322
100	0.189156	0.093086	0.071524
200	1.490063	0.604518	0.476150
400	14.340456	5.545227	4.231425
600	51.992569	21.279290	15.787947
800	120.089625	54.899791	41.566678
1000	243.741273	113.194914	87.196889

Tabela A.5: GPIGPQA: médias dos tempos mínimos de iterações.

Médias	r001	r005	r01
20	0.000973	0.000773	0.000671
40	0.011428	0.007327	0.004976
60	0.045668	0.027230	0.018337
80	0.113563	0.058991	0.042451
100	0.229314	0.107151	0.080165
200	3.069109	1.047873	0.741110
400	35.794721	9.389499	6.597180
600	112.844556	36.204871	25.052486
800	243.964583	96.014609	62.923908
1000	482.620938	189.265468	129.857852

Tabela A.6: GPIGPQA: médias dos tempos médios de iterações.

A.2 Algoritmo de Busca Tabu

Número médio de iterações:

máximos	20	40	60	80	100	200	400	600
r001	1160.15	3560.56	4851.75	4950.69	5000	3519.26	451.29	688.96
r005	160.18	24.57	39.29	55.13	72.27	157.77	325.9	499.27
r01	9.56	21.08	33.94	48.62	63.86	143.63	304.02	471.94

Tabela A.7: Busca Tabu: médias dos números máximos de iterações.

mínimos	20	40	60	80	100	200	400	600
r001	411.86	1921.93	3321.29	3575.74	2502.4	240.72	368.75	556.74
r005	10.54	24.17	38.47	51.77	65.93	141.19	305.74	476.49
r01	9.52	20.92	33.3	46.62	59.48	130.97	287.22	452.72

Tabela A.8: Busca Tabu: médias dos números mínimos de iterações.

médias	20	40	60	80	100	200	400	600
r001	701.25	2726.41	4012.65	4508.68	4187.32	1434.61	405.66	604.58
r005	50.45	24.41	38.84	53.32	68.92	148.61	315.68	487.59
r01	9.54	21.01	33.64	47.67	61.49	137.15	295.53	462.07

Tabela A.9: Busca Tabu: médias dos números de iterações.

Tempo médio de iterações:

Máximos	r001	r005	r01
20	0.000257	0.000337	0.000262
40	0.001089	0.000557	0.000452
60	0.003126	0.001426	0.001087
80	0.006345	0.002779	0.002155
100	0.010977	0.004881	0.003694
200	0.059051	0.027144	0.020511
400	0.346072	0.15358	0.115767
600	1.018244	0.432982	0.322721

Tabela A.10: Busca Tabu: médias dos tempos máximos de iterações.

Mínimos	r001	r005	r01
20	0.000062	0.000005	0.000004
40	0.001014	0.000445	0.000321
60	0.003009	0.001346	0.000995
80	0.006144	0.002719	0.002087
100	0.010509	0.004816	0.003621
200	0.05674	0.026972	0.020377
400	0.340695	0.152587	0.115141
600	0.998317	0.428962	0.32073

Tabela A.11: Busca Tabu: médias dos tempos mínimos de iterações.

Médias	r001	r005	r01
20	0.000149	0.000096	0.00007
40	0.001054	0.000505	0.000394
60	0.003066	0.001382	0.001043
80	0.006261	0.00275	0.002122
100	0.010795	0.004846	0.003654
200	0.057654	0.027046	0.020435
400	0.343168	0.15301	0.115416
600	1.008171	0.430497	0.321621

Tabela A.12: Busca Tabu: médias dos tempos médios de iterações.

A.3 Algoritmo Exato

	r005	r01
20	0.175891	0.00096
40	179.998129	0.00356
60	84.199862	0.0024
80	127.725382	0.00648
100	36.610208	0.00236
200	3.973808	0.007081
400	0.615638	0.034082
600	0.137209	0.088806
800	0.237535	0.177131
1000	0.380784	0.301379

Tabela A.13: Algoritmo Exato: médias dos tempos de execução.

A.4 Algoritmo de Ullmann

	r001	r005	r01
20	0.00104	0.001	0.0006
40	0.00616	0.00664	0.00716
60	0.021641	0.021241	0.026481
80	0.048123	0.049643	0.055323
100	0.096166	0.111367	0.109687
200	0.834412	1.12643	1.646983
400	244.750696	23.775286	39.20149

Tabela A.14: Algoritmo Ullmann: médias dos tempos de execução

A.5 Comparação entre os algoritmos

A.5.1 GPIGPQA × Busca Tabu

r001	GPIGPQA	Tabu
20	2.55288	0.128280
40	2.903051	3.063079
60	0.755071	12.531015
80	1.024936	28.501773
100	1.406376	45.825176
200	4.194406	85.625423
400	36.666755	139.384367
600	210.365986	610.650307
800	247.789182	
1000	490.050202	

Tabela A.15: Comparações: GPIGPQA X Busca Tabu (r001).

r005	GPIGPQA	Tabu
20	0.001656	0.005112
40	0.009944	0.012560
60	0.032874	0.054243
80	0.072628	0.147201
100	0.134536	0.334685
200	1.273208	4.023876
400	11.330972	48.314699
600	42.890464	209.949073
800	112.026473	
1000	221.301807	

Tabela A.16: Comparações: GPIGPQA X Busca Tabu (r005).

r01	GPIGPQA	Tabu
20	0.001144	0.000800
40	0.007728	0.008432
60	0.027761	0.035466
80	0.065068	0.101646
100	0.124256	0.225182
200	1.130367	2.805511
400	9.921188	34.119164
600	36.508297	148.631345
800	89.879033	
1000	183.730914	

Tabela A.17: Comparações: GPIGPQA X Busca Tabu (r01).

A.5.2 Algoritmo Exato × Algoritmo de Ullmann

r005	Ullmann	Alg. Exato
20	0.001	0.175891
40	0.00664	179.998129
60	0.021241	84.199862
80	0.049643	127.725382
100	0.111367	36.610208
200	1.12643	3.973808
400	23.775286	0.615638
600		0.137209
800		0.237535
1000		0.380784

Tabela A.18: Comparações: Ullmann X Alg. Exato (r005).

r01	Ullmann	Alg. Exato
20	0.0006	0.00096
40	0.00716	0.00356
60	0.026481	0.0024
80	0.055323	0.00648
100	0.109687	0.00236
200	1.646983	0.007081
400	39.20149	0.034082
600		0.088806
800		0.177131
1000		0.301379

Tabela A.19: Comparações: Ullmann X Alg. Exato (r01).

A.5.3 Comparação entre todos os algoritmos

r001	GPIGPQA	Tabu	Ullmann	Exato
20	2.55288	0.128280	0.00104	
40	2.903051	3.063079	0.00616	
60	0.755071	12.531015	0.021641	
80	1.024936	28.501773	0.048123	
100	1.406376	45.825176	0.096166	
200	4.194406	85.625423	0.834412	
400	36.666755	139.384367	244.750696	
600	210.365986	610.650307		
800	247.789182			
1000	490.050202			

Tabela A.20: Comparações: desempenhos de todos os algoritmos (r001).

r005	GPIGPQA	Tabu	Ullmann	Exato
20	0.001656	0.005112	0.001	0.175891
40	0.009944	0.012560	0.00664	179.998129
60	0.032874	0.054243	0.021241	84.199862
80	0.072628	0.147201	0.049643	127.725382
100	0.134536	0.334685	0.111367	36.610208
200	1.273208	4.023876	1.12643	3.973808
400	11.330972	48.314699	23.775286	0.615638
600	42.890464	209.949073		0.137209
800	112.026473			0.237535
1000	221.301807			0.380784

Tabela A.21: Comparações: desempenhos de todos os algoritmos (r005).

r01	GPIGPQA	Tabu	Ullmann	Exato
20	0.001144	0.000800	0.0006	0.00096
40	0.007728	0.008432	0.00716	0.00356
60	0.027761	0.035466	0.026481	0.0024
80	0.065068	0.101646	0.055323	0.00648
100	0.124256	0.225182	0.109687	0.00236
200	1.130367	2.805511	1.646983	0.007081
400	9.921188	34.119164	39.20149	0.034082
600	36.508297	148.631345		0.088806
800	89.879033			0.177131
1000	183.730914			0.301379

Tabela A.22: Comparações: desempenhos de todos os algoritmos (r01).

Referências Bibliográficas

[Abreu et al., 2006] Abreu, N. M. M., Siqueira, A. S., Silveira, D. S., & Melo, V. A., 2006. Verificação heurística de isomorfismo de grafos através do problema quadrático de alocação.

[Berge, 1985] Berge, C., 1985. *Graphs*. North-Holland, Amsterdam.

[Bezerra et al., 2005] Bezerra, H., Velho, L., & Feijo, B., 2005. Sombreamento 3d para animacao 2d. Technical Report - VISGRAF Laboratory TR-2005-01, IMPA.

[Binato et al., 2001] Binato, S., Hery, W., Loewenstern, D., & Resende, M., 2001. A GRASP for job shop scheduling. In Hansen, P. & Ribeiro, C., eds, *Essays and surveys on metaheuristics*. Kluwer Academic Publishers.

[Boeres and Sarmento, 2005] Boeres, M. C. S. & 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*, Gramado -RS. XXXVII SBPO - Simpósio Brasileiro de Pesquisa Operacional, 2005.

[Brixius and Anstreicher, 2001] Brixius, N. W. & Anstreicher, K. M., 2001. The Steinberg wiring problem. Technical report, Department of Management Sciences, University of Iowa, Iowa City, Iowa 52242. To appear in *"The Sharpest Cut"*, edited by M. Grötschel, SIAM Publications.

[Burkard et al., 1984] Burkard, R. E., Çela, E., Pardalos, P. M., & Pitsoulis, L. S., 1984. Quadratic assignment problems. *European Journal of Operational Research*, vol. 15, pp. 283–289.

[BURKARD et al., 1998] BURKARD, R. E., ÇELA, E., PARDALOS, P. M., & PITSOULIS, L. S., 1998. The quadratic assignment problem. In *Handbook of combinatorial optimization, Vol. 3*, pp. 241–237. Kluwer Acad. Publ., Boston, MA.

[BURKARD et al., 1991] BURKARD, R. E., KARISCH, S., & RENDL, F., 1991. QAPLIB – A quadratic assignment problem library. *ejor*, vol. 55, pp. 115–119. www.opt.math.tu-graz.ac.at/qaplib/.

[Carvalho and Rahmann, 2006] Carvalho, Sérgio A. de, J. & Rahmann, S., 2006. Microarray layout as quadratic assignment problem. In et al., D. H., ed, *Lecture Notes in Informatics*, volume P-83, pp. 11–20. Gesellschaft für Informatik, Proceedings of the German Conference on Bioinformatics (GCB).

[Cordella et al., 1999] Cordella, L. P., Foggia, P., Sansone, C., & Vento, M., 1999. Performance evaluation of the vf graph matching algorithm. In *ICIAP '99: Proceedings of the 10th International Conference on Image Analysis and Processing*, pp. 1172, Washington, DC, USA. IEEE Computer Society.

[Cordella et al., 2001] Cordella, L. P., Foggia, P., Sansone, C., & Vento, M., 2001. An improved algorithm for matching large graphs. *Proc. of the 3rd IAPR TC-15 Workshop on Graphbased Representations in Pattern Recognition*, vol. , pp. 149–159.

[Corneil and Gotlieb, 1970] Corneil, D. G. & Gotlieb, C. C., 1970. An efficient algorithm for graph isomorphism. *J. ACM*, vol. 17, n. 1, pp. 51–64.

[Cross et al., 1997] Cross, A. D. J., Wilson, R. C., & Hancock, E. R., 1997. Inexact graph matching using genetic search. *Pattern Recognition*, vol. 30, n. 6, pp. 953–970.

[Dickey and Hopkins, 1972] Dickey, J. & Hopkins, J., 1972. Campus building arrangement using TOPAZ. *Transportation Research*, vol. 6, pp. 59–68.

[El-Sonbaty and Ismail, 1998] El-Sonbaty, Y. & Ismail, M. A., 1998. A new algorithm for subgraph optimal isomorphism. *Pattern Recognition*, vol. 31, n. 2, pp. 205–218.

[Elshafei, 1977] Elshafei, A. N., 1977. Hospital layout as a quadratic assignment problem. *orq*, vol. 28, pp. 167–179.

[Feo et al., 1994] Feo, T., Resende, M., & Smith, S., 1994. A greedy randomized adaptive search procedure for the maximum independent set. *Journal of Operations Research*, vol. 42, pp. 860–878.

[Ferreira et al., 2001] Ferreira, J. A., de Souza, P. A., & Moura, C. T. G., 2001. Proposta de implementaçáo de sistema de segurança utilizando sistemas biométricos.

[Foggia et al., 2001] Foggia, P., Sansone, C., & 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*, vol. , pp. 188–199.

[Fortin, 1996] Fortin, S., 1996. The graph isomorphism problem. Technical report.

[Gambardella et al., 1999] Gambardella, L., Taillard, É., & Dorigo, M., 1999. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, vol. 50, pp. 167–176.

[Gambardella and Taillard, 1997] Gambardella, L. M. & Taillard, D., 1997. Ant colonies for the quadratic assignment problem.

[Gendreau et al., 1994] Gendreau, M., Hertz, A., & Laporte, G., 1994. A tabu search heuristic for the vehicle routing problem. *Manage. Sci.*, vol. 40, n. 10, pp. 1276–1290.

[Gendreau et al., 1993] Gendreau, M., Soriano, P., & Salvail, L., 1993. Solving the maximum clique problem using a tabu search approach. *Ann. Oper. Res.*, vol. 41, n. 1-4, pp. 385–403.

[Glover, 1977] Glover, F., 1977. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, vol. 8, n. 1, pp. 156–166.

[Glover, 1986] Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, vol. 13, n. 5, pp. 533–549.

[Glover, 1989] Glover, F., 1989. Tabu search–Part I. *ORSA Journal on Computing*, vol. 1, n. 3, pp. 190–206.

[Glover, 1990] Glover, F., 1990. Tabu search– part II. *ORSA Journal on Computing*, vol. 2, n. 1, pp. 4–32.

[Goldberg, 1989] Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

[Koopmans and Beckmann, 1957] Koopmans, T. C. & Beckmann, M. J., 1957. Assignment problems and the location of economic activities. *Econometrica*, vol. 25, pp. 53–76.

[Lawler, 1963] Lawler, E. L., 1963. The quadratic assignment problem. *Management Science*, vol. 9, pp. 586–599.

[Li et al., 1994] Li, Y., Pardalos, P. M., & Resende, M. G. C., 1994. A greedy randomized adaptive search procedure for the quadratic assignment problem. In Pardalos, P. M. & Wolkowicz, H., eds, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 237–261. American Mathematical Society.

[Loiola et al., 2004] Loiola, E. M., de Abreu, N. M. M., & Boaventura, N. P. O., 2004. Uma revisão comentada das abordagens do problema quadrático de alocação. *Pesquisa Operacional*, vol. 24, pp. 73–109.

[Mathon, 1978] Mathon, R., 1978. Sample graphs for isomorphism testing. *Congressus Numerantium*, vol. 21, pp. 499 – 517.

[Messmer, 1996] Messmer, B., 1996. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. PhD thesis.

[Messmer and Bunke, 1999] Messmer, B. T. & Bunke, H., 1999. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, vol. 32, pp. 1979–1998.

[Misevicius, 2005] Misevicius, A., 2005. A tabu search algorithm for the quadratic assignment problem. *Comput. Optim. Appl.*, vol. 30, n. 1, pp. 95–111.

[Mladenovic and Hansen, 1997] Mladenovic, N. & Hansen, P., 1997. Variable neighborhood search. *Computers & Operations Research*, vol. 24, pp. 1097–1100.

[Morettin, 1999] Morettin, L. G., 1999. *Estatística Básica: Probabilidade*, volume 1.

[Reeves, 1993] Reeves, C. R., ed, 1993. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., New York, NY, USA.

[Resende and Ribeiro, 1997] Resende & Ribeiro, 1997. A GRASP for graph planarization. *NETWORKS: Networks: An International Journal*, vol. 29.

[Resende and Feo, 1996] Resende, M. G. C. & Feo, T. A., 1996. A GRASP for satisfiability. In Johnson, D. S. & Trick, M. A., eds, *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series In Discrete Mathematics and Theoretical Computer Science*, volume 26, pp. 499–520. AMS.

[Rohe, 2004] Rohe, O., 2004. Graphmatching problems and algorithms. Bachelor's thesis, University of Paderborn.

[Sahni and Gonzalez, 1976] Sahni, S. & Gonzalez, T., 1976. P-complete approximation problems. *Journal of the ACM*, vol. 23, pp. 555–565.

[Schaerf, 1999] Schaerf, A., 1999. A survey of automated timetabling. *Artif. Intell. Rev*, vol. 13, n. 2, pp. 87–127.

[Schmidt and Druffel, 1976] Schmidt, D. C. & Druffel, L. E., 1976. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *J. ACM*, vol. 23, n. 3, pp. 433–445.

[Taillard, 1991] Taillard, E., 1991. Robust tabu search for the quadratic assignment problem. *parcomput*, vol. 17, pp. 443–455.

[Ullmann, 1976] Ullmann, J. R., 1976. An algorithm for subgraph isomorphism. *J. ACM*, vol. 23, n. 1, pp. 31–42.

[Varela et al., 2006] Varela, M. C., Velho, L., Madeira, B., Bezerra, H., & Magalhaes, M., 2006. Muan: A stop motion animation system. Technical Report 02, IMPA.

[Wong, 1992] Wong, E. K., 1992. Model matching in robot vision by subgraph isomorphism. *Pattern Recognition*, vol. 25, n. 3, pp. 287–303.