Philippe Leal Freire dos Santos

HEURÍSTICAS

UMA APLICAÇÃO AO PROBLEMA DA PARTIÇÃO CROMÁTICO DE CUSTO MÍNIMO

Atena Editora
Ano 2026

Editora chefe
Profª Drª Antonella Carvalho de Oliveira
Editora executiva
Natalia Oliveira Scheffer
Assistente editorial
Flávia Barão
Bibliotecária
Janaina Ramos
Edição de arte
Yago Raphael Massuqueto Rocha

Heurísticas: uma aplicação ao problema da partição cromático de custo mínimo

Revisão: O autor
Indexação: Amanda Kelly da Costa Veiga

Dados Internacionais de Catalogação na Publicação (CIP)

S237 Santos, Philippe Leal Freire dos
Heurísticas: uma aplicação ao problema da partição cromático de custo mínimo / Philippe Leal Freire dos Santos - Ponta Grossa - PR: Atena, 2026.

Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
Inclui bibliografia
ISBN 978-65-258-3928-8
DOI: https://doi.org/10.22533/at.ed.288262601

1. Teoria dos grafos. 2. Algoritmos heurísticos. 3. Otimização combinatória. I. Santos, Philippe Leal Freire dos. II. Título.
CDD 511.5

Elaborado por Bibliotecária Janaina Ramos - CRB-8/9166

Atena Editora
Ponta Grossa - Paraná - Brasil
+55 (42) 3323-5493
+55 (42) 99955-2866
www.atenaeditora.com.br
contato@atenaeditora.com.br

Conselho Editorial

"Seja forte e corajoso; não temas, nem te desanimes, porque o Senhor, teu Deus, é contigo por onde quer que andares."

Josué 1:9

À minha esposa Renata.

Agradecimentos

Agradeço primeiramente a Deus por tudo o que Ele sempre fez por mim, por ter me sustentado e me mantido forte nos momentos mais difíceis. Seu amor por nós é inexplicável!

Ao meu orientador Professor Celso Carneiro Ribeiro, por me aceitar como seu aluno, pela orientação, dedicação, paciência, apoio e todo o aprendizado proporcionado. É sempre motivante observar a sua responsabilidade em realizar um trabalho de alta qualidade.

À minha esposa Renata pelo amor, carinho, oração, paciência, apoio, incentivo, dedicação e compreensão. Por ter entendido a minha ausência em muitos momentos e por ter abdicado de muita coisa para que eu pudesse me dedicar ao curso. Sem ela, a concretização deste sonho não seria possível.

Aos meus pais e à minha irmã Thaíza pelo incentivo, apoio e constantes orações. Por compreenderem a minha ausência em diversas situações durante todos estes anos.

Em especial, aos amigos Renata Mesquita e Tiago Santos (*in memoriam*), que foram muito importantes para a realização deste curso. Tiago, você foi um cara especial na minha vida, desde quando lhe conheci, antes do Doutorado. Um amigo que gostaria de ter para sempre ao meu lado. Aprendi muito com você durante todo esse tempo. Sua inteligência, amizade, alegria, dedicação e companheirismo eram admiráveis. Sua tranquilidade era contagiante, mesmo você dizendo que não passava de aparência. Caminhamos juntos esta longa jornada de estudo e posso afirmar que você fez um excelente trabalho, tornando-se um brilhante pesquisador. Deus te quis mais perto Dele, como Ele sempre faz com as pessoas especiais, mas gostaria de te agradecer por ter me dado o prazer de te chamar de amigo.

Aos amigos Leonardo Musmanno e Julliany Brandão por todo o apoio, amizade, sugestões e momentos de descontração, sem os quais o curso não teria sido o mesmo.

Aos amigos do Instituto Federal Fluminense, André Velasco, Diego Sales, Fábio Duncan, Francisco de Freitas, Frederico Galaxe, Mark Douglas Jacyntho, Rogério Avellar e

Vinícius Barcelos, pelo apoio e incentivo em todos os momentos desta jornada. Em especial, ao amigo Fernando Carvalho, por sua disponibilidade em me ajudar e generosidade de sempre.

Aos funcionários do IC/UFF, Teresa e Helio, pelos inúmeros auxílios durante o curso.

À CAPES pelo apoio financeiro, fundamental para a realização do Doutorado.

Resumo

O Problema da Partição Cromática de Custo Mínimo (PPCCM), considerado uma das diversas variantes do Problema Clássico de Coloração de Grafos (PCG), utiliza números reais como custos das cores, tendo como objetivo colorir os vértices de um grafo de modo que os adjacentes tenham cores diferentes e a soma dos custos das cores utilizadas seja mínima. Embora seja um problema NP-Difícil para grafos em geral, foram elaborados algoritmos polinomiais para algumas classes de grafos. Do ponto de vista prático, o mesmo foi empregado no projeto de circuitos VLSI e na solução de determinados problemas de escalonamento modelados como grafos de intervalo. Nesta tese são propostos algoritmos para o PPCCM considerando um grafo simples não-direcionado como entrada. Inicialmente foram desenvolvidas duas heurísticas baseadas na metaheurística Algoritmos Genéticos com Chaves Aleatórias Tendenciosas (*Biased Random Key Genetic Algorithms* - BRKGA, em inglês). Posteriormente, foi implementada uma heurística de trajetória que faz uso de duas estratégias de busca local seguidas por um procedimento de *path-relinking*. Para os experimentos computacionais foram geradas instâncias para o problema a partir de grafos comumente empregados no PCG.

Palavras-chave: Problema da Partição Cromática de Custo Mínimo, Coloração de Grafos, Heurísticas, Metaheurísticas, Busca Local, Algoritmos Genéticos.

Abstract

The Minimum Cost Chromatic Partition Problem (MCCPP) is one of several variants of the classical Graph Coloring Problem (GCP), in which there are real number as color costs and the aim is to color the vertices of a graph so that the adjacent ones have different colors and the sum of the costs of the used colors is minimal. Although the MCCPP is a NP-hard problem for general graphs, polynomial time algorithms were developed for some classes of graphs. From a practical point of view, the MCCPP has application in the design of VLSI circuits and in the solution of scheduling problems modeled as interval graphs. In this thesis, algorithms for the problem considering undirected simple graphs are proposed. Initially, two heuristics based on the metaheuristic Biased Random Key Genetic Algorithm (BRKGA) were developed. Following, we propose a trajectory search heuristic using local search and path-relinking. For computational experiments, instances for the problem from graphs commonly used in PCG were generated.

Keywords: Minimum Cost Chromatic Partition Problem, Graph Coloring, Heuristics, Metaheuristics, Local Search, Genetic Algorithms.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BRKGA	:	Biased Random Key Genetic Algorithm;
HBLPR	:	Heurística de Trajetória com Busca Local e Path-relinking;
PCG	:	Problema Clássico de Coloração de Grafos;
PPCCM	:	Problema da Partição Cromática de Custo Mínimo;
PSC	:	Problema da Soma Cromática;
RKGA	:	Random Key Genetic Algorithm;
RVNS	:	Reduced Variable Neighborhood Search;
TTT-Plot	:	Time-To-Target Plot;
VLSI	:	Very Large Scale Integration;

Sumário

Capítulo 1

Introdução

1.1 Motivação

O Problema Clássico de Coloração de Grafos (PCG) tem como objetivo colorir os vértices de um grafo G qualquer com um número mínimo de cores, chamado número cromático e representado por $\chi(G)$, de modo que sejam atribuídas cores distintas para vértices adjacentes. O PCG é amplamente estudado em virtude da sua importância teórica e aplicabilidade prática, comumente modelando problemas como grafos de conflitos, onde os vértices representam elementos distintos, as arestas definem conflitos entre esses elementos e cada cor determina um componente a ser otimizado. Como exemplo, são encontradas aplicações na área de escalonamento [Malkawi et al., 2008], redes de comunicação [Woo et al., 1991] e alocação de frequências [Narayanan e Shende, 2001]. Do ponto de vista teórico, por ser um problema NP-Difícil para grafos em geral [Garey e Johnson, 1979], ainda não foi possível elaborar um algoritmo em tempo polinomial que o resolva considerando grafos genéricos.

Estabelecido como uma das diversas variantes do PCG, o Problema da Soma Cromática (PSC) foi introduzido por Kubicka [Kubicka, 1989]. Dado um grafo não-direcionado G e números naturais em sequência como custos das cores, ele consiste em colorir os vértices de G de forma que tenham cores diferentes os que são adjacentes e a soma dos custos das cores dos vértices seja mínima. Esse problema não é equivalente ao PCG, no sentido de que não basta encontrar o número cromático do grafo G e, utilizando as $\chi(G)$ cores de menor custo, obter a menor soma possível.

O Problema da Partição Cromática de Custo Mínimo (PPCCM) é considerado uma generalização do PSC que, diferente deste, utiliza números reais como custos para as

cores. O mesmo foi apresentado em 1980 por Mead e Conway [Mead e Conway, 1980], que propuseram uma máquina hipotética com um grande número de processadores distribuídos como em uma árvore, a fim de aumentar sua capacidade computacional ao gerar paralelismo entre eles. Sendo ele um problema NP-Difícil, considerando um processamento sequencial, os autores apresentaram um algoritmo paralelo para a sua resolução na referida máquina, o que resultaria em um tempo de execução $O(n^2)$, dado um grafo não-direcionado com n vértices e um conjunto de n cores. No entanto, os mesmos verificaram que era necessário um número muito grande de processadores $(2n^n - 1)$ para alcançar tal tempo polinomial.

Apesar da complexidade do PPCCM para grafos em geral, foram elaborados algoritmos polinomiais para algumas classes de grafos, como árvores, grafos co-bipartidos e para o complementar do grafo sem triângulos. Do ponto de vista prático, Supowit [Supowit, 1987] e Sen et al. [Sen et al., 1992] o empregaram no projeto de circuitos VLSI, onde terminais precisam ser eletricamente conectados em diferentes camadas que possuem características distintas, tendo assim um custo associado ao posicionamento de um terminal em uma camada. O objetivo é particionar tais terminais em camadas, tentando não interceptá-los, de modo que o custo total seja mínimo. Problemas de escalonamento modelados como grafos de intervalo também são exemplos de aplicações para o problema [Kroon et al., 1997].

Devido à sua complexidade, inexiste algoritmo eficiente a fim de tratar o problema para o caso geral. A utilização de métodos exatos baseados em formulações simples, na tentativa de obter a melhor solução possível, torna-se impraticável para instâncias de tamanho médio e grande, por demandar elevado tempo computacional. Nesses casos, é sugerida a aplicação de métodos heurísticos, que permitem a obtenção de uma solução de boa qualidade, em um tempo de processamento aceitável.

1.2 Objetivo da Tese

Assim, o objetivo deste trabalho consiste em propor e avaliar algoritmos para o Problema da Partição Cromática de Custo Mínimo considerando como entrada um grafo simples não-direcionado. Para isso, inicialmente foram desenvolvidas duas heurísticas baseadas na metaheurística Algoritmos Genéticos com Chaves Aleatórias Tendenciosas (*Biased Random Key Genetic Algorithms* - BRKGA, em inglês), onde uma delas realiza, em cada um dos indivíduos elite, uma busca em vizinhança considerada uma modificação

da metaheurística VNS. Posteriormente, foi proposta uma heurística de trajetória que faz uso de duas estratégias de busca local, seguidas por um procedimento de *path-relinking* (reconexão por caminhos).

1.3 Organização da Tese

Este trabalho está organizado da seguinte forma, o PCG e algumas de suas variantes com custos nos vértices e nas cores são apresentados no próximo capítulo. O PPCCM é descrito no Capítulo 3, incluindo a sua formulação como um problema de programação inteira binária. As heurísticas baseadas na metaheurística BRKGA e os experimentos destas sobre um conjunto de instâncias desenvolvido para o problema são detalhados no Capítulo 4. O Capítulo 5 especifica a heurística de trajetória com busca local e *path-relinking*, bem como seus experimentos e os testes de comparação das três heurísticas propostas. Por fim, o Capítulo 6 apresenta as conclusões e sugestões para trabalhos futuros.

Capítulo 2

Problemas de Coloração de Grafos

Neste capítulo será especificado o Problema Clássico de Coloração de Grafos, com uma breve revisão dos principais algoritmos desenvolvidos para solucioná-lo e apresentação de determinadas variantes que consideram pesos nos vértices e nas cores, como o Problema da Soma Cromática, descrito na Seção 2.2.

2.1 O Problema Clássico de Coloração

Em 1852, Francis Guthrie conjecturou que seriam necessárias apenas quatro cores para colorir as regiões de qualquer mapa, de modo que regiões vizinhas não possuíssem a mesma cor, dando origem assim ao *Problema das Quatro Cores*. Apesar da sua aparente simplicidade, esse problema permaneceu em aberto por mais de cem anos, pois somente em 1976 conseguiu-se provar, utilizando computadores, que a conjectura estava correta, sendo instituído o *Teorema das Quatro Cores*.

Uma vez que é possível associar um mapa a um grafo planar, onde os vértices representam as regiões e as arestas a vizinhança entre as mesmas, outro problema estabelecido, intitulado *Problema de Coloração de Grafos*, tem como objetivo colorir um grafo qualquer com um número mínimo de cores de modo que sejam atribuídas cores distintas para vértices adjacentes.

De modo formal, considere um grafo não-direcionado $G = (V, E)$, onde V é o conjunto de vértices e E o conjunto de arestas. Uma *coloração própria* dos vértices de G consiste em atribuir cores diferentes para vértices adjacentes, sendo *imprópria* caso contrário. Assim, o Problema Clássico de Coloração de Grafos (PCG) consiste em encontrar uma coloração própria em G de modo que o número de cores utilizado seja mínimo. Este número mínimo

de cores é chamado *número cromático* de G e representado por $\chi(G)$. Uma *k-coloração* é uma coloração de G que utiliza k cores. A versão de decisão relacionada ao PCG é o Problema da k-coloração, que compreende determinar se é possível encontrar uma coloração própria de G com k cores. Como exemplo, considere o grafo G da Figura 2.1 que possui 7 vértices e 11 arestas. Foi possível encontrar uma 3-coloração própria em G, sendo esse o número mínimo de cores necessário para colori-lo, resultando em um número cromático $\chi(G) = 3$.

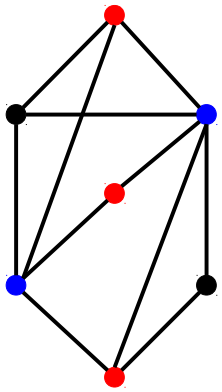


Figura 2.1: Grafo G com 7 vértices, 11 arestas e $\chi(G) = 3$.

O PCG desperta interesse de pesquisa em diversas áreas devido a sua aplicabilidade prática, comumente modelando problemas como grafos de conflitos, onde os vértices representam elementos distintos, as arestas definem conflitos entre esses elementos e cada cor um componente a ser otimizado. Como exemplo, são encontradas aplicações na área de escalonamento [Malkawi et al., 2008], redes de comunicação [Woo et al., 1991] e alocação de frequências [Narayanan e Shende, 2001], fluxo de tráfego aéreo [Barnier e Brisset, 2004], registradores [Pereira e Palsberg, 2005] e *timetabling* [Bello et al., 2008].

2.1.1 Algoritmos para o PCG

Do ponto de vista teórico, por ser o PCG um problema NP-Difícil para grafos em geral [Garey e Johnson, 1979], ainda não foi possível elaborar um algoritmo em tempo polinomial para solucioná-lo. Apesar disso, alguns métodos exatos foram desenvolvidos na tentativa de encontrar a solução ótima para o problema. O primeiro deles, aplicando a técnica de enumeração implícita, foi proposto em [Brown, 1972] e utiliza as cores já empregadas na coloração atual, ou uma nova cor, para colorir um vértice de cada vez, seguindo uma determinada ordem dos mesmos. Outro algoritmo, chamado DSATUR [Brélaz, 1979], fundamentado na mesma técnica e que propõe uma melhoria no algoritmo de Brown, utiliza como regra para a escolha do próximo vértice a ser colorido aquele que

tiver o maior *grau de saturação*, que é o número de cores distintas dos vértices adjacentes a um determinado vértice. Em caso de empate, o vértice escolhido é o que apresentar maior grau no subgrafo não colorido. Sewell [Sewell, 1996] promove melhorias no algoritmo DSATUR estabelecendo uma regra de desempate alternativa na seleção do vértice para *branching*, escolhendo aquele que gera a maior diminuição no número de cores disponíveis para os vértices restantes sem cor. Outros algoritmos empregando a estratégia de enumeração implícita são encontrados em [Kubale e Jackowski, 1985] e [Sager e Lin, 1991].

Um método baseado em geração de colunas para solucionar o PCG utilizando uma formulação de conjuntos independentes foi apresentado por Mehrotra e Trick [Mehrotra e Trick, 1996] que, embora necessite de sofisticadas regras de *branching* e a solução de um subproblema difícil, resolve rapidamente grafos de tamanho pequeno a médio. Visto como uma variação da formulação de Mehrotra e Trick, Campêlo et al. [Campêlo et al., 2004] estabeleceram um modelo para o problema, denominado Formulação de Representantes, que define um vértice representante para cada classe de cor, impondo que somente uma cor possa ser utilizada se a classe de cor for inicializada pelo vértice representante correspondente. Em [Campêlo et al., 2008] esta formulação foi revista a fim de eliminar soluções simétricas, visto que qualquer vértice de uma classe de cor poderia ser o representante dessa classe.

Malaguti et al. [Malaguti et al., 2011] propuseram um algoritmo *branch-and-price* também com base na formulação de cobertura de conjuntos do PCG apresentado em [Mehrotra e Trick, 1996]. No entanto, eles incorporaram a heurística MMT [Malaguti et al., 2008] a fim de gerar uma solução inicial viável e um conjunto de colunas para serem utilizadas pelo método exato. Uma abordagem baseada em modelos de programação inteira foi igualmente empregada no trabalho de Méndez-Díaz e Zabala [Méndez-Díaz e Zabala, 2008], utilizando algumas famílias de facetas do politopo 0/1 associadas a um desses modelos em um algoritmo de plano de corte, com o objetivo de remover soluções simétricas obtidas pela permutação das cores.

Tendo como base o algoritmo DSATUR e as melhorias propostas por Sewell [Sewell, 1996], Segundo [Segundo, 2012] desenvolveu um algoritmo exato introduzindo uma nova estratégia de desempate com o objetivo de reduzir o número de subproblemas gerados, sendo executada mais rápido do que a estratégia apresentada por Sewell, pois se restringe a um conjunto particular de vértices.

Devido à complexidade do PCG, grande parte dos algoritmos exatos tem a capacidade

de solucionar somente pequenas instâncias em tempo computacional aceitável, tipicamente com poucas centenas de vértices, sendo assim necessária a utilização de heurísticas e metaheurísticas para grafos que modelam aplicações do mundo real, comumente com milhares de vértices.

A heurística gulosa mais simples para o PCG, denominada Heurística Sequencial, assume que os vértices estão em uma determinada ordem de entrada e atribui a cada vértice a cor de menor índice que não tenha sido utilizada em vértices adjacentes. A versão gulosa do DSATUR [Brélaz, 1979] segue esse princípio, inicialmente organizando os vértices em ordem não-crescente dos graus e colorindo aquele de maior grau com a primeira cor. A coloração prossegue com a cor de menor índice possível, escolhendo como próximo vértice o que possui maior grau de saturação, que é atualizado a cada iteração. Outro que utiliza a primeira cor no vértice de grau maior é o algoritmo *Recursive Largest First* (RLF) de Leighton [Leighton, 1979], que a partir disso constrói uma classe de cor de cada vez, de maneira gulosa, separando os vértices ainda não coloridos em dois conjuntos: os que podem ser alocados na classe que está sendo criada e os que não podem.

Chams et al. [Chams et al., 1987] aplicaram a metaheurística *Simulated Annealing* à versão de decisão do PCG, explorando um conjunto de k-colorações (próprias ou não) e tendo como objetivo minimizar o número de arestas conflitantes (arestas que conectam vértices com a mesma cor). Uma solução vizinha é alcançada trocando a cor de um único vértice na solução corrente. Outra metaheurística de busca local aplicada ao problema foi a Busca Tabu em [Hertz e de Werra, 1987]. Esse algoritmo, intitulado TABUCOL, tem o mesmo espaço de soluções e a mesma função objetivo daquele apresentado em [Chams et al., 1987]. Contudo, somente os vértices adjacentes a arestas conflitantes são candidatos para a troca de cor. Quando a cor de um vértice é alterada, obtendo uma solução vizinha, esse vértice e sua cor anterior são armazenados em uma lista tabu, impossibilitando-o de receber tal cor por um determinado número de iterações, chamado prazo tabu. Essa restrição tabu não é considerada caso a troca de cor conduza a uma solução com valor de função objetivo menor do que um valor definido.

A comparação de três algoritmos *Simulated Annealing*, cada um com uma estrutura de vizinhança diferente, foi realizada em [Johnson et al., 1991], em que dois deles permitiam soluções completas inviáveis e o outro somente colorações completas próprias, tendo um número variável de classes de cor. Os autores também projetaram o algoritmo XRLF, que combina uma variante do RLF [Leighton, 1979] com a remoção de conjuntos independentes no grafo, mostrando-se competitivo com os outros algoritmos implementados.

Uma importante contribuição de Morgenstern [Morgenstern, 1996] foi a definição da vizinhança *Impasse Class Neighborhood*, utilizada para transformar uma coloração parcial em uma completa de mesmo valor. O autor apresentou um algoritmo *Simulated Annealing* empregando essa vizinhança juntamente com um método para a recombinação de soluções.

Um dos primeiros estudos a integrar busca local com algoritmos baseados em população, como os Algoritmos Genéticos (AGs), foi o trabalho de Costa et al. [Costa et al., 1995]. Esse tipo de algoritmo, conhecido como Algoritmo Evolucionário (AE), emprega uma população de soluções e um operador de cruzamento, como os AGs. Porém, o operador de mutação é substituído por uma busca local. No algoritmo desenvolvido, os autores utilizaram um método de descida simples como busca local. Na mesma época, Fleurent e Ferland [Fleurent e Ferland, 1996] elaboraram um AE utilizando o cruzamento uniforme padrão como operador de recombinação e uma versão com melhorias do algoritmo TABUCOL como operador de mutação. Em [Galinier e Hao, 1999], o algoritmo evolucionário híbrido (HEA) proposto igualmente utiliza uma versão atualizada do TABUCOL e um operador de cruzamento específico para o PCG, chamado *Greedy Partitioning Crossover* (GPX), que tem a característica de transmitir aos filhos gerados as estruturas dos pais. Segundo Malaguti e Toth [Malaguti e Toth, 2010], esse operador é o responsável por posicionar o HEA entre os melhores já desenvolvidos para o PCG.

No trabalho de Malaguti et al. [Malaguti et al., 2008] foi anunciado o algoritmo MMT, que realiza uma fase de inicialização, onde um limite superior e um inferior para o problema são obtidos, e duas etapas de otimização, que podem ser finalizadas assim que uma solução comprovadamente ótima é encontrada. Na primeira etapa, um AE é executado utilizando a combinação de uma Busca Tabu e o operador GPX, adaptado para a vizinhança *Impasse Class Neighborhood*, a fim de encontrar uma k-coloração que melhore a melhor solução encontrada na fase de inicialização. Na etapa seguinte, uma heurística soluciona o Problema de Cobertura de Conjuntos utilizando os conjuntos independentes armazenados durante a execução do AE na primeira etapa. Ainda que o AE resolva o problema para um valor fixo de k, o algoritmo MMT como um todo aborda a versão de otimização do PCG, sendo considerada uma das melhores heurísticas para o problema [Malaguti e Toth, 2010].

Em outro modelo de algoritmo evolucionário, denominado Algoritmo de Memória Adaptativa, a população constitui-se de partes das soluções, ao invés de soluções completas, e utiliza um método de recombinação dessas partes a fim de gerar novas soluções.

Fazendo uso desse modelo, Galinier et al. [Galinier et al., 2008] desenvolveram o algoritmo AMACOL para o problema com um número fixo de cores (k-coloração), tendo como população conjuntos independentes encontrados durante a execução e o TABUCOL como o operador de busca local.

Blöchliger e Zufferey [Blöchliger e Zufferey, 2008] apresentaram dois métodos de Busca Tabu considerando como solução uma k-coloração parcial viável, isto é, uma solução dividida em k conjuntos independentes e um conjunto de vértices ainda não coloridos. Embora baseada na vizinhança *Impasse Class Neighborhood*, a estratégia utilizada apresenta-se muito mais simples do que a proposta em [Morgenstern, 1996], tendo como objetivo simplesmente minimizar o número de vértices não coloridos. Além disso, os autores analisaram o uso de um prazo tabu dinâmico e outro reativo.

O desenvolvimento de um AE aplicando também uma Busca Tabu como método de busca local foi realizado por Lü e Hao [Lü e Hao, 2010]. Denominado MACOL, o algoritmo a princípio utiliza a Busca Tabu para melhorar as soluções da população inicial, que são k-colorações inviáveis, minimizando o número de arestas em conflito. A cada iteração, aplica-se o operador de cruzamento AMPaX (uma extensão do GPX) em duas ou mais soluções escolhidas aleatoriamente da população, tendo como resultado uma k-coloração, que também será melhorada pela busca local. Em seguida, para determinar se a solução resultante será inserida na população, o algoritmo avalia a estratégia definida para a atualização da mesma, estabelecendo ainda qual solução será substituída caso ocorra a inserção. Um dos autores utilizou o MACOL como segunda fase do algoritmo EXTRACOL em [Wu e Hao, 2012a]. Na primeira fase, ele utiliza uma Busca Tabu Adaptativa para identificar um conjunto independente máximo e tentar encontrar outros conjuntos independentes disjuntos do mesmo tamanho, fazendo em seguida a extração dos mesmos no grafo original. Esse processo se repete até não haver mais do que 800 vértices no grafo residual, que será colorido pelo MACOL na fase seguinte.

Recentemente, Moalic e Gondran [Moalic e Gondran, 2015] propuseram o algoritmo evolucionário HEAD, considerado uma variação do algoritmo HEA [Galinier e Hao, 1999]. Ambos utilizam como busca local uma versão atualizada do TABUCOL e o operador de cruzamento GPX, tendo como objetivo encontrar uma coloração com o número mínimo de arestas conflitantes. No entanto, HEAD propõe uma estratégia diferente para gerenciar a diversificação: reduzir o tamanho da população para somente duas soluções. Para tratar uma das principais desvantagens da utilização de uma população pequena, que é não proporcionar uma diversificação suficiente para o algoritmo evoluir, após um determinado

número de gerações (chamado pelos autores de ciclo), a melhor solução (solução elite) da população é armazenada e a solução elite de ciclos anteriores é reintroduzida na população, substituindo um dos seus dois membros. Como resultado, o algoritmo conseguiu reduzir o número de cores necessárias para colorir três grafos considerados difíceis na literatura.

Analisando os experimentos realizados pelos autores com as heurísticas mencionadas anteriormente, é possível identificar que TABUCOL, HEA, AMACOL, MMT, MACOL, EXTRACOL e HEAD são as que proporcionaram os melhores resultados na solução do PCG, sendo consideradas o estado da arte até o momento. Outros algoritmos utilizando outras estratégias para solucionar o problema encontram-se em [Laguna e Martí, 2001], [Chiarandini e Stützle, 2002], [Avanthay et al., 2003], [Hertz et al., 2008], [Plumettaz et al., 2010] e [Titiloye e Crispin, 2011]. Considerações adicionais sobre o PCG podem ser encontrados em [Galinier e Hertz, 2006], [Chiarandini et al., 2007], [Malaguti e Toth, 2010] e [Galinier et al., 2013].

2.1.2 Variantes do PCG com Pesos

Alguns problemas considerados variantes do PCG têm recebido atenção na literatura e motivado o desenvolvimento de algoritmos, tanto pela aplicação em situações reais quanto pela importância teórica, por serem igualmente NP-Difíceis. Dentre eles, encontram-se os que consideram pesos (ou custos) nos vértices e nas classes de cores do grafo. Esses pesos podem representar, por exemplo, as bandas de frequência a serem distribuídas em antenas de transmissão, de modo a evitar interferências, em um Problema de Alocação de Canais de Rádio [McDiarmid e Reed, 2000].

A modelagem para tal aplicação pode ser realizada pelo Problema de Multicoloração de Grafos (PMG), onde cada vértice $i \in V$ tem um peso positivo p_i associado, indicando o número de cores a serem atribuídas ao vértice i. Essas cores representam as bandas de frequência designadas às antenas, simbolizadas pelos vértices, sendo que, para cada aresta $(i, j) \in E$, a interseção das cores atribuídas aos vértices i e j tem que ser vazia, reproduzindo a tentativa de evitar interferências entre as bandas. Respeitando esta restrição, o objetivo é colorir o grafo com o menor número de cores possível.

Para solucionar o PMG, métodos exatos foram desenvolvidos, como em [Mehrotra e Trick, 2007], que apresentaram um algoritmo de geração de colunas para otimizar a relaxação linear de uma formulação para o problema, a qual utiliza uma variável para cada conjunto independente do grafo. Essa mesma abordagem foi aplicada no trabalho de Gualandi e Malucelli [Gualandi e Malucelli, 2012], onde os autores utilizaram programação

por restrições e novas técnicas de *branching* a fim de melhorar a performance do algoritmo.

Com a finalidade de solucionar instâncias maiores do problema, estratégias heurísticas também foram implementadas. Em [Lim et al., 2005], um algoritmo guloso constrói uma solução a partir de uma sequência de vértices gerada por duas metaheurísticas (*Squeaky Wheel Optimization* e Busca Tabu), que também são utilizadas posteriormente na tentativa de melhorar a solução. No Algoritmo Genético proposto em [Han e Kim, 2015] são aplicados dois operadores de cruzamento, sendo um desenvolvido especificamente para o problema. Além disso, o grafo de entrada não sofre transformação com a inclusão de novos vértices, como normalmente é feito para a resolução do PMG.

Outro exemplo de variante que admite pesos nos vértices é o Problema de Coloração de Grafos Ponderados (PCGP), no qual um peso positivo p_i é atribuído a cada vértice $i \in V$. De igual modo, cada classe de cor possui um custo associado, que é dado pelo peso máximo dos vértices coloridos com aquela cor. Diferentemente do PCG, onde o objetivo é colorir o grafo com o menor número de cores, no PCGP a intenção é encontrar uma coloração cuja soma dos custos das cores utilizadas seja a menor possível. Esse problema pode modelar aplicações reais, como o Problema de Escalonamento em Máquinas com Compatibilidade de Tarefas [Boudhar e Finke, 2000] e o Problema da Decomposição de Matrizes [Ribeiro et al., 1989,Prais e Ribeiro, 2000].

Como proposta de algoritmo para solucionar o PCGP, Malaguti et al. [Malaguti et al., 2009] apresentaram duas formulações de programação inteira com um número polinomial de variáveis e restrições. Um desses modelos é utilizado na inicialização de um algoritmo de duas fases, gerando um limite inferior para o problema. Na primeira fase, algumas heurísticas gulosas são aplicadas em sequência para produzir um grande número de conjuntos independentes. Na fase final, para melhorar a solução, uma formulação do Problema de Cobertura de Conjuntos (PCC), associada a alguns conjuntos gerados, é solucionada por uma heurística lagrangeana da literatura.

Uma heurística também segmentada em fases foi apresentada em [Oliveira et al., 2011]. Na primeira fase realiza-se um pré-processamento a fim de reduzir o tamanho do grafo de entrada. Na segunda, uma solução inicial é construída por uma heurística construtiva, aplicando na última fase, para melhorar a solução inicial, uma heurística VND que utiliza como busca local um algoritmo *backtracking*. Recentemente, uma abordagem exata foi proposta por Furini e Malaguti [Furini e Malaguti, 2012], que desenvolveram um algoritmo *branch-and-price* para o problema a partir de uma extensão da formulação do PCC apresentada no trabalho de Malaguti et al. [Malaguti et al., 2009].

Outro problema considerado uma variante do PCG, mas que admite custos somente para as classes de cores, o Problema da Soma Cromática (PSC) utiliza números naturais em sequência como tais custos. A sua descrição detalhada é apresentada na seção seguinte.

2.2 O Problema da Soma Cromática

O Problema da Soma Cromática (PSC) foi introduzido por Kubicka [Kubicka, 1989], que o definiu como: dado um grafo simples não-direcionado G e números naturais em sequência como custos das cores, deseja-se encontrar uma coloração própria, entre todas as colorações próprias de G, onde a soma total dos custos das cores dos vértices seja mínima. Essa soma total mínima é chamada de *soma cromática* de G e é denotada por $\Sigma(G)$. A *força* de G, $s(G)$, é o número mínimo de cores necessário para obter a sua soma cromática.

Esse problema não é equivalente ao PCG, no sentido de que não basta encontrar o número cromático do grafo e, utilizando o menor número de cores, obter a menor soma possível. Para ilustrar, considere a árvore T da Figura 2.2. Se somente duas cores com custos 1 e 2 forem utilizadas (Figura 2.2 (a)), a soma dos custos é igual a 12. No entanto, a melhor solução é encontrada inserindo mais uma cor de custo 3 (Figura 2.2 (b)), resultando em $\Sigma(T) = 11$, $s(T) = 3$. Assim, não é possível obter a coloração de soma ótima utilizando apenas duas cores, que é o valor de $\chi(T)$. De forma geral, considerando um grafo G qualquer, $s(G) \geq \chi(G)$ [Kokosiński e Kwarciany, 2007]. No exemplo da Figura 2.2, $s(T) > \chi(T)$.

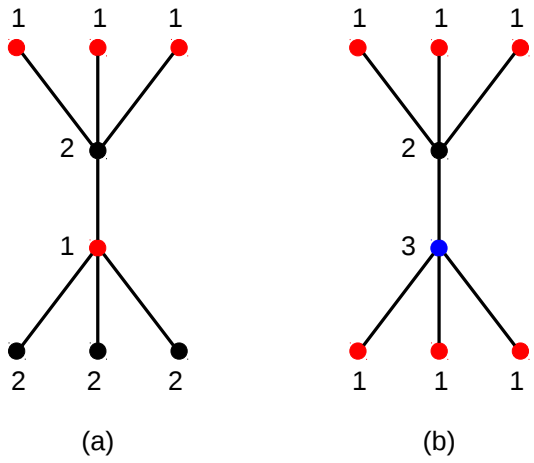


Figura 2.2: Árvore T: (a) colorida com duas cores, com soma igual a 12 e (b) colorida com três cores, resultando $\Sigma(T) = 11$ e $s(T) = 3$ (Extraída de [Kubicka e Schwenk, 1989]).

Um exemplo de sua aplicação é dado em [Bar-Noy et al., 1998] para o Problema de Alocação de Recursos Distribuídos. Para representar as restrições do problema, eles utilizam um grafo de conflito, onde os vértices representam os processadores e as arestas indicam uma concorrência pelos recursos. Assim, dois vértices são adjacentes se os processadores correspondentes não podem executar seus trabalhos simultaneamente. A intenção é minimizar o tempo médio de resposta do sistema ou, de forma equivalente, minimizar a soma dos tempos de execução dos trabalhos. Esse problema pode ser modelado como PSC assumindo um tempo de execução fixo para os trabalhos. Outras aplicações são encontradas em escalonamento [Halldórsson et al., 2003, Bar-noy e Kortsarz, 1998] e em projetos VLSI [Szkaliczki, 1999, Nicoloso et al., 1999].

2.2.1 Formulação

O PSC pode ser formulado como um problema de programação inteira binária. Considere um grafo não-direcionado $G = (V, E)$, onde V é o conjunto de n vértices e E o conjunto de arestas. Seja x_{ih} uma variável binária, tal que $x_{ih} = 1$ se o vértice $i \in V$ for colorido com a cor h e $x_{ih} = 0$ caso contrário. Uma formulação para o PSC como um Problema de Programação Inteira 0-1 é dada em [Wang et al., 2013]:

$$Min \quad \sum_{i=1}^{n} \sum_{h=1}^{n} h \cdot x_{ih} \tag{2.1}$$

sujeito a:

$$\sum_{h=1}^{n} x_{ih} = 1, \quad \forall\, i \in V \tag{2.2}$$

$$x_{ih} + x_{jh} \leq 1, \quad \forall\, i, j \in V : (i, j) \in E, \quad h = 1, \ldots, n \tag{2.3}$$

$$x_{ih} \in \{0, 1\}, \quad \forall\, i \in V, \quad h = 1, \ldots, n. \tag{2.4}$$

A função objetivo (2.1) minimiza a soma dos custos das cores utilizadas. A restrição (2.2) requer que cada vértice seja colorido por apenas uma cor, enquanto a restrição (2.3) impõe que, para cada par de vértices adjacentes, somente um deles receba a cor h, caso ela seja utilizada. Por fim, a restrição (2.4) indica a integralidade da variável x_{ih}.

Observa-se que essa formulação de programação inteira, que trata apenas de viabilidade, é muito fraca e que outras mais fortes podem ser desenvolvidas, baseadas, por exemplo, na ideia de formulação por representantes [Bahiense et al., 2014, Frota et al., 2010].

Solucionar o PSC para um grafo qualquer pertence à classe de problemas NP-Difícil, segundo Kubicka e Schwenk [Kubicka e Schwenk, 1989]. Nesse sentido, encontrar limites superiores e inferiores para $\Sigma(G)$ e $s(G)$ torna-se muito útil, uma vez que possibilita, por exemplo, verificar que uma determinada solução não apresenta resultado de boa qualidade quando seu valor ultrapassar algum limite superior. Na literatura são encontrados resultados para tais limites, que comumente são obtidos com base em determinadas características do grafo, como seu maior grau ($\Delta(G)$), número de vértices (n), número de arestas (m) e seu número cromático ($\chi(G)$). Alguns resultados serão apresentados, podendo a prova matemática ser encontrada no trabalho citado.

2.2.2 Limites para a Soma Cromática de um Grafo - $\Sigma(G)$

Um primeiro limite, embora não justo para a soma cromática de um grafo G, pode ser obtido analisando a estrutura do grafo [Kubicka e Schwenk, 1989]:

$$n \leq \Sigma(G) \leq \frac{n(n+1)}{2}. \tag{2.5}$$

O limite inferior é atingido se G não possuir arestas, resultando na necessidade de apenas uma cor para colorir todos os n vértices. Se G for um grafo completo, cada vértice terá que ser colorido com uma cor exclusiva, sendo o limite superior alcançado.

Um trabalho pioneiro que apresenta limites para a soma cromática de grafos em geral encontra-se em [Thomassen et al., 1989]. O primeiro limite apresentado pelos autores resulta da aplicação de um algoritmo guloso considerando uma ordenação qualquer dos vértices do grafo:

$$\Sigma(G) \leq n + m. \tag{2.6}$$

O segundo resultado mostra um limite inferior e superior para a soma cromática, para qualquer grafo conexo, em relação ao número de arestas:

$$\lceil \sqrt{8m} \rceil \leq \Sigma(G) \leq \lfloor 3(m+1)/2 \rfloor. \tag{2.7}$$

Ainda segundo Thomassen et al. [Thomassen et al., 1989], para qualquer grafo G sem vértices isolados:

$$\lceil \sqrt{8m} \rceil \leq \Sigma(G) \leq 3m. \tag{2.8}$$

Limites para a soma cromática também são exibidos em [Kokosiński e Kwarciany, 2007]:

$$\Sigma(G) \leq \frac{n(\chi(G)+1)}{2} \tag{2.9}$$

$$n + \frac{\chi(G)(\chi(G)-1)}{2} \leq \Sigma(G) \tag{2.10}$$

$$n + \frac{s(G)(s(G)-1)}{2} \leq \Sigma(G). \tag{2.11}$$

Kokosiński [Kokosiński, 2011] fez comparações de limites teóricos e experimentais para $\Sigma(G)$. Segundo ele, os melhores limites inferiores teóricos são os apresentados nas equações (2.7) e (2.10), tendo a segunda retornado melhores resultados em praticamente todos os experimentos. Ainda segundo o autor, as equações (2.6) e (2.9) exibem os melhores limites superiores teóricos, sendo o segundo melhor na maioria dos testes.

2.2.3 Limites para a Força de um Grafo - $s(G)$

No caso de limites para $s(G)$, um limite inferior e superior para um grafo G é [Kokosiński e Kwarciany, 2007]:

$$\chi(G) \leq s(G) \leq n. \tag{2.12}$$

Uma vez que são necessárias no mínimo $\chi(G)$ cores para colorir G, o limite inferior é alcançado. O limite superior é atingido quando for preciso atribuir uma cor diferente para cada vértice de G.

Outros limites para $s(G)$ são apresentados em [Hajiabolhassan et al., 2000]. O primeiro deles envolve o maior grau de G:

$$s(G) \leq \Delta(G) + 1. \tag{2.13}$$

Além disso, eles demonstram que a igualdade $s(G) = \Delta(G) + 1$ é atendida se, e somente se, G for um grafo completo ou um ciclo ímpar.

O terceiro limite exibido pelos autores envolve, além de $\Delta(G)$, outro parâmetro do

grafo chamado *número de coloração*, denotado por $col(G)$. O número de coloração de um grafo G é o menor número d tal que, dada alguma ordenação dos vértices de G, o número de arestas para todos os vértices listados antes de i ($i \in V$) é estritamente menor do que d. Assim, para qualquer grafo G:

$$s(G) \leq \lceil (col(G) + \Delta(G))/2 \rceil. \tag{2.14}$$

Ainda segundo os autores, considerando o número de coloração, para todo grafo G, $\chi(G) \leq col(G)$. No entanto, $s(G) \leq col(G)$ nem sempre é verdade. A desigualdade $\chi(G) \leq col(G) \leq \Delta(G)$ sempre se aplica, exceto para grafos regulares. O trabalho de Kokosiński e Kwarciany [Kokosiński e Kwarciany, 2007] também apresenta um limite para $s(G)$:

$$s(G) \leq \lfloor \sqrt{n(\chi(G)+1)} \rfloor. \tag{2.15}$$

Alguns resultados aplicam-se especificamente para determinadas classes de grafos. Kubicka [Kubicka, 2004] afirma que para todo inteiro positivo k, existe uma árvore T com $s(T) = k$. Além disso, Jiang e West [Jiang e West, 1999] garantem que para cada inteiro positivo k, existe uma árvore T_k com $s(T_k) = k$ e $\Delta(T_k) = 2k - 2$. Ainda para árvores, o seguinte resultado foi apresentado em [Hajiabolhassan et al., 2000] considerando uma árvore T:

$$s(T) \leq \lceil \Delta(T)/2 \rceil + 1. \tag{2.16}$$

Os mesmos autores melhoraram esse limite considerando o diâmetro ($d(T)$) de T:

$$s(T) \leq \lceil min(d(T), \Delta(T))/2 \rceil + 1. \tag{2.17}$$

Para grafos bipartidos, Malafiejski et al. [Malafiejski et al., 2004] afirmam que a soma cromática de um grafo conexo bipartido regular é igual a $3n/2$, para $n > 1$. Ainda, Kosowski [Kosowski, 2009] provou que o resultado (2.16) não se aplica somente para árvores, mas para qualquer grafo bipartido.

2.2.4 Algoritmos para o PSC

Como já mencionado, o PSC pertence à classe NP-Difícil para um grafo qualquer [Kubicka e Schwenk, 1989]. Esse resultado se mantém para determinadas classes de grafos,

como grafos de intervalo [Szkaliczki, 1999], grafos *split* [Kubicka, 2004] e grafos bipartidos [Salavatipour, 2003]. No entanto, para algumas classes é possível solucioná-lo em tempo polinomial, como árvores [Kubicka e Schwenk, 1989], grafos *k-split* [Salavatipour, 2003], grafos unicíclicos e *outerplanar* [Kubicka, 2005].

Em [Douiri e Elbernoussi, 2011], os autores propuseram uma heurística híbrida que combina um Algoritmo Genético (AG) com um método de restrições derivadas. Esse método é utilizado para construir a população inicial do AG, que tem como indivíduo uma atribuição de cores para todos os vértices do grafo. Como essa atribuição pode gerar conflitos entre vértices, uma vez que vértices adjacentes podem apresentar cores iguais, o objetivo do AG é minimizar o número de conflitos. Ele utiliza o método da roleta para seleção dos indivíduos que poderão passar pelo cruzamento de dois pontos e pela mutação, com probabilidade de ocorrência de 0.8 e 0.2, respectivamente. Para as instâncias testadas, o algoritmo melhorou o limite superior de quatro delas e alcançou os melhores resultados conhecidos para as demais.

O algoritmo proposto em [Helmar e Chiarandini, 2011], chamado MDS(5)+LS, emprega uma heurística construtiva na geração da solução inicial. Além disso, ele explora soluções que atribuem cores diferentes para vértices adjacentes, chamadas de *soluções próprias*, bem como as que não respeitam essa atribuição, denominadas *soluções impróprias*. Essa exploração é realizada utilizando as estruturas de vizinhança *Swap Neighborhood* e *One-Move Neighborhood*, considerando que esta pode aumentar o número de cores a fim de garantir o retorno de uma solução viável. Essa solução então é modificada de modo que os vértices sejam realocados nas menores classes de cores possível, sem que vértices adjacentes tenham a mesma cor, podendo eventualmente diminuir a quantidade de cores utilizadas. Em seguida, aplica-se um procedimento de perturbação à solução, que consiste em alterar a cor de uma fração do número de vértices, selecionando-os aleatoriamente. A nova cor desses vértices é escolhida também de forma aleatória de 1 até $k + 1$, sendo k a quantidade de cores da solução. O algoritmo finaliza a sua execução ao atingir um limite de tempo, ou um número máximo de iterações, ou um número de iterações sem melhoria. Comparando o MDS(5)+LS aos experimentos encontrados na literatura, ele melhora as soluções conhecidas de 27 das 38 instâncias analisadas e não apresenta resultado pior do que os demais métodos.

Uma heurística para o PSC que segue o método da metaheurística Busca Local Iterada foi desenvolvida por Benlic e Hao [Benlic e Hao, 2012] e denominada *Breakout Local Search* (BLS). A ideia básica é usar uma busca local para descobrir ótimos locais e empre-

gar um certo número de perturbações, cujo tipo é determinado adaptativamente, com a intenção de explorar melhor o espaço de soluções. O algoritmo começa com uma solução inicial randômica que, caso apresente conflitos entre vértices (coloração imprópria), sofre a aplicação de um método de busca local. Esse método avalia a variação no número de conflitos e a soma dos custos das cores considerando todas as possíveis trocas de cores para cada vértice do grafo, de maneira que torne a coloração própria. Caso a solução inicial não apresente conflitos, ou tenha passado pelo método anterior, são realizadas trocas de cores nos vértices, mantendo a coloração própria, para identificar a troca que mais decresce o valor da função objetivo. Esse processo é repetido até que um ótimo local seja alcançado.

Após a fase de busca local, o BLS aplica perturbações na solução para diversificar a busca e explorar outros pontos do espaço de soluções. O primeiro tipo de perturbação é a mais comum realizada para problemas de coloração: a troca de cor de cada vértice, mantendo a coloração própria. Cada vez que um vértice tem a cor alterada, ele é inserido em uma lista tabu que o proíbe de retornar para a cor anterior durante as próximas t iterações (o valor de t é determinado aleatoriamente em um dado intervalo). Essa proibição somente é desconsiderada se a troca conduzir a uma nova solução melhor que a melhor solução já encontrada. A segunda forma de perturbação consiste em realizar a troca válida de cores (que não causou conflitos entre vértices) mais recente que trouxe a maior redução no valor da função objetivo, desde que o conjunto da cor no qual o vértice estava não fique vazio. A última perturbação é a troca de cores aleatória. Primeiro seleciona-se randomicamente dois subconjuntos de cores S_i e S_j, tal que $|S_i| \leq |S_j|$, e em seguida move-se um vértice de S_i (também selecionado de forma aleatória) para o subconjunto S_j. Para variar entre os três tipos de perturbações, o BLS utiliza parâmetros que dependem do estado da busca, ou seja, do número de tentativas que não levaram à melhora de uma solução. Os testes realizados mostraram que o BLS melhorou o melhor resultado conhecido para quatro instâncias e atingiu o limite superior para outras 15, tendo falhado na obtenção do limite apenas em oito grafos.

O algoritmo EXSCOL [Wu e Hao, 2012b] tem por característica extrair iterativamente conjuntos independentes do grafo. Ele identifica inicialmente um conjunto independente de maior tamanho e em seguida procura o maior número possível de conjuntos independentes disjuntos daquele tamanho. Feito isso, retira-os do grafo e atribui a cada um a cor de menor custo disponível. Esse processo é repetido até que o grafo torne-se vazio. O fundamento dessa abordagem é que, pela extração de muitos conjuntos independentes disjuntos, naturalmente grandes classes de cores são construídas, o que reduz o número

necessário de cores e consequentemente a soma total dos custos das cores. Uma vez que encontrar um conjunto independente máximo de um grafo é NP-Difícil [Garey e Johnson, 1979], o EXSCOL utiliza uma heurística baseada em busca tabu para realizar essa tarefa. Como resultado da sua aplicação à instâncias do problema, o EXSCOL melhorou o limite superior de 17 grafos e alcançou o melhor resultado conhecido para nove deles, tendo falhado somente duas vezes na obtenção do limite.

No algoritmo memético MA-MSCP apresentado em [Moukrim et al., 2013], cada um dos 20 indivíduos (soluções) da população representa uma partição V_i $(i = 1, \ldots, k)$ do conjunto de vértices, de modo que $|V_1| \geq |V_2| \geq \ldots \geq |V_k|$. Na população inicial, 25% dos indivíduos são gerados por algoritmos gulosos e o restante aleatoriamente, não permitindo que dois indivíduos diferentes tenham o mesmo valor de função de avaliação para manter a diversidade dessa população. Para a evolução, quatro indivíduos são selecionados de forma aleatória na população, sendo o primeiro pai o melhor (menor valor de avaliação) dos dois primeiros selecionados e o segundo pai o melhor dos demais. Esses pais são submetidos ao operador de cruzamento, que é uma adaptação do GPX desenvolvido por Galinier e Hao [Galinier e Hao, 1999]. Em seguida, uma busca local é empregada utilizando duas estruturas de vizinhança a fim de melhorar o novo indivíduo produzido por esse operador. Para formar a próxima população, esse novo indivíduo substitui aquele com o mesmo valor de avaliação, se existir. No entanto, caso esse valor seja menor do que o de algum indivíduo, o novo é inserido na população e o pior indivíduo é excluído. Como critério de parada, o algoritmo utiliza um determinado tempo máximo. Com essa estratégia, o MA-MSCP encontrou o ótimo em 27 das 81 instâncias testadas para o PSC.

Outro algoritmo memético para o problema, denominado MASC, foi desenvolvido por Jin et al. [Jin et al., 2014]. Ele apresenta três componentes importantes, sendo o primeiro um procedimento de busca tabu com dupla vizinhança desenvolvido especialmente para o problema, que tem a finalidade de melhorar a qualidade de uma dada solução gerada pelo operador de *crossover*. Para isso, ele utiliza duas vizinhanças diferentes que são aplicadas alternadamente até que a melhor solução encontrada não consiga ser mais atualizada (intensificação). A primeira vizinhança é alcançada com a troca de alguns vértices de uma cor por outros vértices adjacentes de outra cor, enquanto que a segunda é obtida trocando apenas um vértice de uma cor por outra, sendo que em ambas o resultado das trocas tem que manter o grafo com uma coloração própria. Essas trocas de vértices entre classes de cores são inseridas na lista tabu e não podem se repetidas até um determinado número de iterações. A fim de escapar de ótimos locais, o procedimento utiliza uma fase de diversificação, onde, dada uma solução ótima local S^* com k classes de cores diferentes

e V_l a classe com maior número de vértices, ele cria uma classe adicional V_{k+1} e move aleatoriamente um terço dos vértices de V_l para V_{k+1}. Para prevenir que a busca retorne para S^*, V_l e V_{k+1} são inseridas na lista tabu e não podem fazer parte das vizinhanças durante um certo número de iterações.

O segundo componente é a utilização de um operador de *crossover* com vários pais. Esse operador gera somente um filho a partir de α pais escolhidos aleatoriamente da população, onde α varia de dois até quatro de acordo com o número de vértices e o número cromático do grafo. A intenção desse operador é transmitir grandes classes de cores dos pais para o filho, que sempre é uma coloração própria e pode ter um número de cores maior do que os seus pais.

O terceiro componente importante do MASC é o mecanismo de atualização da população. Ele utiliza duas funções, uma para medir a qualidade da solução (f) e a outra para analisar a sua diversidade (H). Considerando duas k-colorações próprias S_1 e S_2, pode-se dizer que S_1 é melhor do que S_2 se $f(S_1) < f(S_2)$, onde $f(S_i)$ indica a soma dos custos das cores da k-coloração própria S_i. Para estimar a diversidade de duas colorações S_i e S_j, o algoritmo utiliza a função $H_{i,j}$, que é o número de vértices em S_i e S_j que têm diferentes cores: $H_{i,j} = |\{v \in V : S_i(v) \neq S_j(v)\}|$. Um pequeno valor de $H_{i,j}$ indica uma alta similaridade entre S_i e S_j. Assim, o MASC combina as funções f e H para decidir se um filho substitui ou não um indivíduo na nova população. Aplicado à instâncias do problema, o algoritmo melhorou 17 limites superiores conhecidos, incluindo grafos com mais de 500 vértices, tendo também igualado 30 melhores resultados. Além disso, apresentou pela primeira vez limites superiores para 18 grafos.

Em [Jin e Hao, 2016], um novo algoritmo memético é aplicado ao PSC. Intitulado HESA, ele emprega uma heurística de obtenção de conjuntos independentes para gerar cada indivíduo da população inicial (de tamanho 20), de modo que não sejam inseridos indivíduos em duplicata na mesma. A cada geração, duas soluções da população, que não foram escolhidas em gerações passadas, são selecionadas aleatoriamente e utilizadas por um duplo procedimento de *crossover* para gerar duas novas soluções, que podem ser próprias ou não. Cada solução gerada é submetida a uma Busca Tabu de duas fases com o objetivo de melhorá-la, bem como torná-la própria caso ela não seja. A solução resultante passa pelo procedimento de atualização da população, onde a sua inserção será analisada, assim como definida a solução que será substituída, baseado na qualidade da solução e na distância entre as soluções da população. Esse processo de evolução acontece até o limite de tempo de duas horas. Como resultado, o algoritmo melhorou, de um total de

94 instâncias testadas, o limite superior de 24 delas e o limite inferior de 27 grafos.

Ao analisar de forma geral os resultados apresentados pelos algoritmos desenvolvidos para solucionar o PSC, Jin et al. [Jin et al., 2017] concluem que os seis últimos apresentados anteriormente são os que possibilitam alcançar as melhores soluções, embora não possam ser comparados estatisticamente por não terem sido testados com a mesma quantidade de instâncias.

Uma possibilidade de generalização do PSC é admitir números reais como custos das cores, sem qualquer sequência para os mesmos, mantendo o objetivo de encontrar uma coloração própria no grafo, onde a soma total desses custos seja mínima. Tal problema denomina-se Problema da Partição Cromática de Custo Mínimo (PPCCM), que é objeto de estudo desta tese e especificado detalhadamente no capítulo seguinte.

Capítulo 3

O Problema da Partição Cromática de Custo Mínimo

Neste capítulo será descrito em detalhes o Problema da Partição Cromática de Custo Mínimo, objeto de estudo desta tese. São apresentados sua definição, uma formulação como um problema de programação inteira binária, sua complexidade para grafos em geral e classes específicas, exemplos de sua aplicação, bem como algoritmos aproximativos existentes para determinados tipos de grafos.

3.1 Introdução

O Problema da Partição Cromática de Custo Mínimo (PPCCM) foi formulado por Mead e Conway [Mead e Conway, 1980], que propuseram uma hipotética máquina com um grande número de processadores distribuídos como em uma árvore, onde cada nó corresponderia a um processador, a fim de aumentar sua capacidade computacional ao gerar paralelismo entre eles. Sendo ele um problema NP-Difícil para grafos em geral [Sen et al., 1992], os autores apresentaram um algoritmo paralelo para a sua resolução na referida máquina, o que resultaria em um tempo de execução da ordem de $O(n^2)$, dado um grafo não-direcionado com n vértices e um conjunto de n cores. No entanto, embora ocorresse uma redução na complexidade do problema, seria necessário um número muito grande de processadores $(2n^n - 1)$ para alcançar tal tempo polinomial.

Apesar disso, para algumas classes de grafos, o PPCCM pode ser solucionado em tempo polinomial, como árvores, grafos co-bipartidos e o complementar de grafos sem triângulos [Kroon et al., 1997, Jansen, 1996]. O projeto de circuitos VLSI [Supowit, 1987, Sen et al., 1992] e a solução de um problema de escalonamento sobre grafos de

intervalos [Kroon et al., 1997] são exemplos de sua aplicação.

O PPCCM se diferencia do PSC quanto aos custos das cores. Enquanto neste tais custos obrigatoriamente são valores naturais sequenciais, no PPCCM os mesmos admitem valores reais, sem qualquer sequência. Para ilustrar essa diferença, dado o grafo G da Figura 3.1, uma solução viável do PSC em G é mostrada na Figura 3.1 (a). Considerando cores com custos 3.9, 4.4, 2.2, 3.6, 1.4 e 3.5, uma coloração viável do PPCCM no mesmo grafo é apresentada na Figura 3.1 (b).

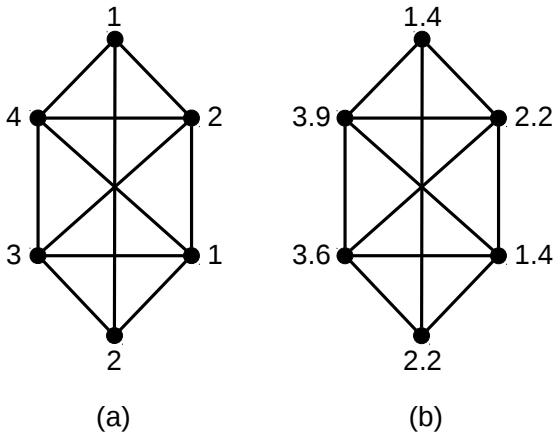


Figura 3.1: Grafo G: (a) coloração viável para o PSC e (b) coloração viável para o PPCCM.

3.2 Formulação e Complexidade

Seja $G = (V, E)$ um grafo simples não-direcionado, onde V é o conjunto de vértices e E o conjunto de arestas. Considere um conjunto de cores $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$ e um custo $w_c \geq 0$ associado a cada cor $c \in \mathcal{C}$. Seja x_{ic} uma variável binária, tal que $x_{ic} = 1$ se o vértice $i \in V$ for colorido com a cor $c \in \mathcal{C}$ e $x_{ic} = 0$ caso contrário. Sen et al. [Sen et al., 1992] formularam o PPCCM como o seguinte Problema de Programação Inteira 0-1:

$$Min \sum_{i \in V} \sum_{c \in \mathcal{C}} w_c \cdot x_{ic} \tag{3.1}$$

sujeito a:

$$\sum_{c \in \mathcal{C}} x_{ic} = 1, \ \forall \ i \in V \tag{3.2}$$

$$x_{ic} + x_{jc} \leq 1, \ \forall \ i, j \in V : (i, j) \in E, \ \forall \ c \in \mathcal{C} \tag{3.3}$$

$$x_{ic} \in \{0, 1\}, \ \forall \ i \in V, \ c \in \mathcal{C}. \tag{3.4}$$

A função objetivo (3.1) minimiza a soma dos custos das atribuições de cores. A restrição (3.2) impõe que cada vértice tenha apenas uma cor associada a ele e a restrição (3.3) exige que, dados dois vértices adjacentes, somente um deles possa receber a cor $c \in \mathcal{C}$, caso ela seja utilizada. Por fim, a restrição (3.4) indica que cada variável x_{ic} é binária.

Assim como observado para o PSC na Seção 2.2.1, essa formulação de programação inteira é muito simples por tratar apenas de viabilidade, sendo possível apresentar outras mais fortes, como uma baseada em formulação por representantes [Bahiense et al., 2014, Frota et al., 2010].

Importante notar que, como no PSC, para solucionar o PPCCM não basta encontrar o número cromático do grafo e obter a menor soma possível utilizando o número mínimo de cores. Para exemplificar, considere a mesma árvore T da Figura 2.2 (Seção 2.2) e cores com custos 5.9, 4.4, 2.5, 4.6, 1.2, 3.5, 5.4 e 6.7. Caso fossem utilizadas somente as duas cores de menor custo 1.2 e 2.5 (Figura 3.2 (a)), a soma dos custos seria igual a 14.8. No entanto, a melhor combinação é encontrada usando também a cor de custo 3.5 (Figura 3.2 (b)), resultando em uma solução ótimo de custo 13.2.

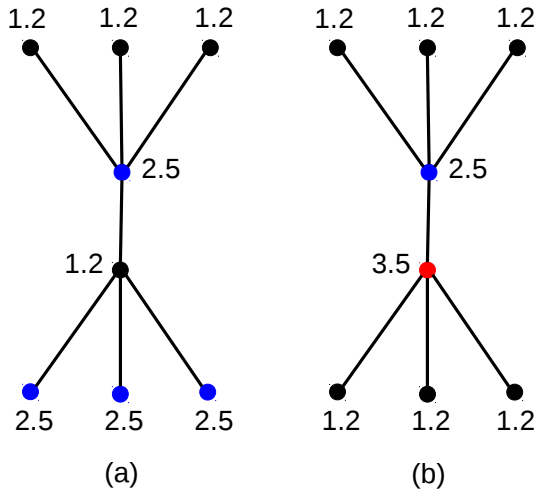


Figura 3.2: Árvore T: (a) colorida com as duas cores de menor custo, com soma igual a 14.8 e (b) colorida com as três cores de menor custo, resultando na solução ótima de custo 13.2.

Apesar da sua relevância teórica, o PPCCM também possui aplicabilidade prática, como em [Supowit, 1987] e [Sen et al., 1992] para projetos de circuitos VLSI. Nesses projetos, a conexão de dois ou mais terminais é chamada de *net*. Um conjunto predeterminado de *nets* deve ser distribuído em camadas, de modo que as *nets* que se interceptam não sejam alocadas na mesma camada. Considerando que cada camada possui um custo associado, o objetivo é distribuir tais *nets* nas camadas com o menor custo possível.

Esse problema torna-se equivalente ao PPCCM utilizando a representação por grafos, onde cada vértice representa uma *net*, sendo dois vértices adjacentes caso as respectivas *nets* se interceptem, e uma cor indicando cada camada. Desse modo, ao solucionar o PPCCM associado, é possível determinar a utilização de camadas com o custo mínimo.

Para exemplificação dessa modelagem, considere um conjunto de *nets* que, se forem distribuídos em uma mesma camada, provoca a interceptação de alguns deles, como apresentado na Figura 3.3 (a). A representação desse problema por grafo é ilustrada na Figura 3.3 (b), onde pode ser encontrada uma coloração viável para o mesmo utilizando, por exemplo, as duas cores (camadas) de menor custo. Por fim, a Figura 3.3 (c) apresenta a visualização em camadas dessa solução.

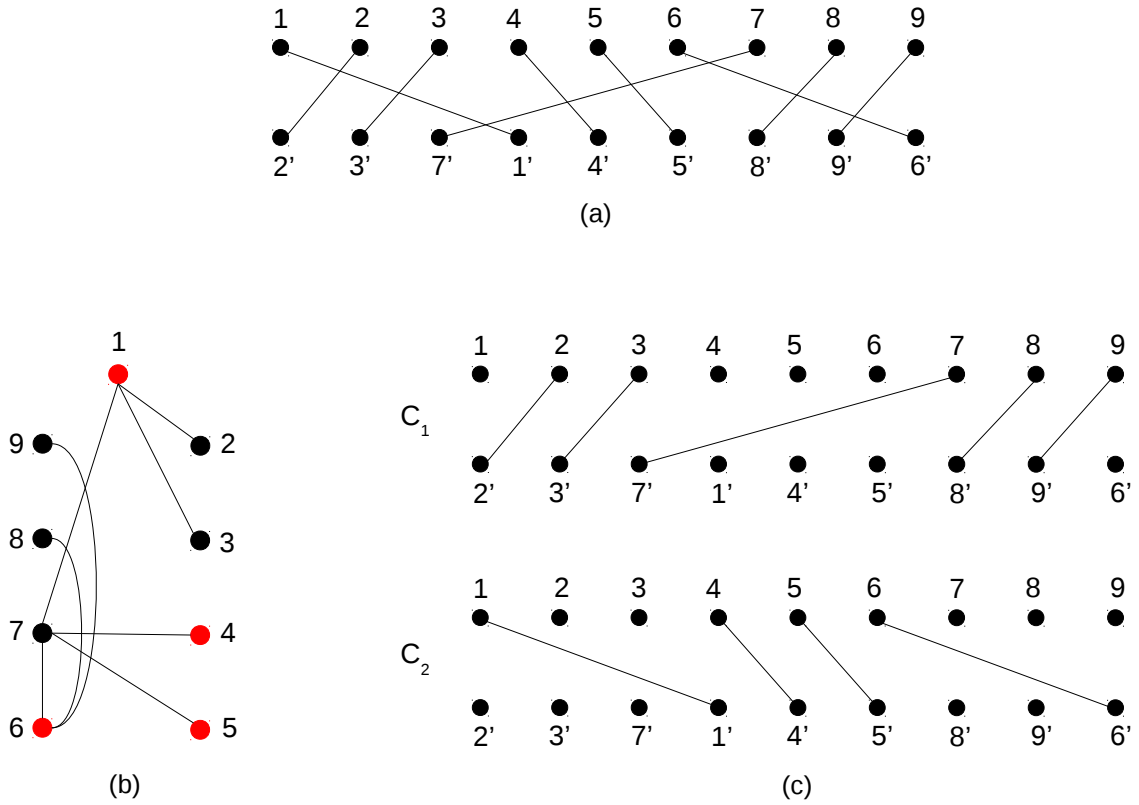


Figura 3.3: (a) Exemplo de um conjunto de *nets* distribuídos em uma camada, (b) o problema modelado por grafo e (c) a solução representada em camadas.

Outra aplicação é dada no trabalho de Kroon et al. [Kroon et al., 1997], onde os autores mostraram que o PPCCM para grafos de intervalo é equivalente ao Problema de Escalonamento de Intervalos Fixos com custo de processamento dependente da máquina. Nesse tipo de escalonamento, cada tarefa t requer processamento durante um intervalo de tempo fixo (s_t, f_t). Assume-se que um número suficiente de máquinas está disponível e que cada tarefa tem que ser executada por uma das máquinas. Como o custo de processamento depende da máquina, se a tarefa t for executada pela máquina m, então

o custo de processamento associado é c_m. O objetivo é encontrar um escalonamento não-preemptivo viável para todas as tarefas de modo que o custo total de processamento seja mínimo.

Modelando esse problema como um grafo de intervalo, cada vértice corresponde ao tempo do intervalo (s_t, f_t) e cada aresta conecta dois vértices se os intervalos correspondentes se sobrepõem. Assim, o PPCCM pode ser considerado como o problema de colorir os correspondentes intervalos (vértices), onde as cores representam as máquinas, de maneira que intervalos sobrepostos obtenham cores diferentes (máquinas diferentes) e que o custo total da coloração (custo total de processamento) seja mínimo.

Sen et al. [Sen et al., 1992] verificaram a complexidade do PPCCM restringindo-o ao Problema da Soma Cromática, que foi provado ser NP-Difícil para grafos em geral [Kubicka e Schwenk, 1989]. Para isso, consideraram números naturais em sequência como custos das cores. Desse modo, o PPCCM também é NP-Difícil para tais tipos de grafos.

Kroon et al. [Kroon et al., 1997] mostraram que o PPCCM pode ser resolvido em tempo linear para árvores. Considerando grafos de intervalos, provaram também que existe um algoritmo polinomial para o problema caso haja somente dois valores diferentes para os custos das cores e que o mesmo é NP-Difícil se esses custos tiverem quatro ou mais valores distintos.

Supondo ainda que os custos das cores tenham dois valores distintos, Jansen [Jansen, 1996] afirma que ainda assim o PPCCM é NP-Difícil para k-árvores com k não limitado, grafos *split*, bem como para grafos caminho não-direcionados e seus complementares.

Ainda, o autor mostra que o problema pode ser resolvido em tempo polinomial para algumas classes de grafos (tais como, cografos, grafos co-bipartidos e o complementar dos grafos sem triângulos). Porém, segundo Sen et al. [Sen et al., 1992], o PPCCM é NP-Difícil para grafos circulares.

3.3 Algoritmos para o PPCCM

Jansen [Jansen, 2000] provou que não existe algoritmo aproximativo polinomial com razão $O(|V|^{0.5-\varepsilon})$, com $\varepsilon > 0$, para o PPCCM restrito a grafos bipartidos e de intervalo, a menos que P = NP. Contudo, ele propôs algoritmos com razão $O(|V|^{0.5})$ para ambas as classes. Além disso, ele demonstrou que, para grafos *split* e cordais, não há algoritmos aproximativos polinomiais com razão $O(|V|^{1-\varepsilon})$, com $\varepsilon > 0$, a menos que P = NP.

Nos experimentos que serão relatados nas Seções 4.2.1.1 e 4.2.1.2, foi utilizado o resolvedor CPLEX na tentativa de obter soluções ótimas para o PPCCM. No entanto, o mesmo não conseguiu encontrar tais soluções para instâncias com algumas centenas de vértices, nem se quer soluções viáveis para algumas com mais de 900 vértices dentro do limite de tempo de 3600 segundos. Nesse caso, é sugerida a aplicação de métodos heurísticos que permitem a obtenção de soluções de qualidade, em tempos de processamento aceitáveis.

Assim, nesta tese inicialmente foram desenvolvidas duas heurísticas baseadas na metaheurística Algoritmos Genéticos com Chaves Aleatórias Tendenciosas para solucionarem o PPCCM, ambas descritas no capítulo seguinte. Posteriormente, uma heurística de trajetória que faz uso de duas estratégias de busca local, seguidas por um procedimento de *path-relinking*, também foi desenvolvida para tratar o problema, sendo esta detalhada no Capítulo 5.

Capítulo 4

Algoritmos Genéticos com Chaves Aleatórias Tendenciosas

Neste capítulo são desenvolvidas heurísticas baseadas nos Algoritmos Genéticos com Chaves Aleatórias Tendenciosas (BRKGA) para o PPCCM, com a descrição de cada um dos seus componentes, como o decodificador utilizado pelas mesmas, e os experimentos computacionais sobre um conjunto de instâncias desenvolvidas para o problema. Também é detalhada a busca em vizinhança utilizada por uma das heurísticas.

4.1 Introdução

Em um Algoritmo Genético (AG), as soluções para o problema tratado são representadas pelos *indivíduos* (ou cromossomos), compostos por *genes*, reunidos em uma população que evolui a cada geração utilizando os operadores genéticos de *cruzamento*, para recombinar indivíduos e gerar novos, e de *mutação*, a fim de diversificar a nova população e evitar que o processo de evolução fique estagnado em ótimos locais. A qualidade (ou aptidão) de cada indivíduo é dada pelo seu *fitness*, cujo valor é resultado da aplicação de uma função de avaliação sobre ele.

Com a intenção de evitar a produção de filhos inviáveis a partir de pais viáveis no processo de cruzamento, uma nova forma de representar um indivíduo foi proposta por Bean [Bean, 1994] ao utilizar, para cada *alelo* (valor de um gene), um número real gerado aleatoriamente no intervalo $[0, 1)$, denominado *chave aleatória*. Ele empregou, então, um algoritmo determinístico, que chamou de *decodificador*, para associar um indivíduo definido por chaves aleatórias a uma solução viável do problema, sendo o valor de *fitness* o custo dessa solução. Esse novo método foi intitulado Algoritmos Genéticos com Chaves

Aleatórias (*Random Key Genetic Algorithms* - RKGA, em inglês).

Nesse algoritmo, os P indivíduos da população inicial são gerados de maneira randômica. A população da geração corrente k é dividida em um pequeno grupo de P_e indivíduos elite, que possuem os melhores valores de *fitness*, e o restante em P_{ne} indivíduos não-elite. Portanto, $P = P_e + P_{ne}$. Com o objetivo de evoluir a população, todos os indivíduos elite são copiados sem qualquer modificação para a população da geração $k+1$, onde também é introduzido um pequeno número de P_m *mutantes*, que são indivíduos gerados do mesmo modo que aqueles da população inicial e com finalidade idêntica ao operador de mutação presente nos AGs. Os demais indivíduos ($P - P_e - P_m$) são gerados pelo cruzamento de outros dois, que são selecionados aleatoriamente de toda a população corrente (elite e não-elite). A Figura 4.1 ilustra esse processo de evolução da população.

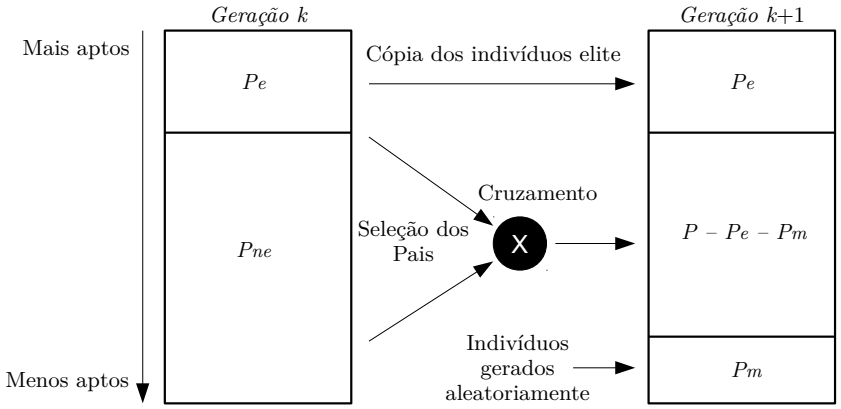


Figura 4.1: Processo de evolução da população no RKGA.

A partir dessa estrutura estabelecida pelo RKGA, Ericsson et al. [Ericsson et al., 2002] propuseram uma alteração na escolha de um dos dois indivíduos que serão submetidos ao cruzamento, dando origem aos Algoritmos Genéticos com Chaves Aleatórias Tendenciosas (*Biased Random Key Genetic Algorithms* - BRKGA, em inglês). Assim, a única diferença entre eles é que, no BRKGA, um desses indivíduos é escolhido de forma aleatória do grupo elite e outro do grupo não-elite da população corrente, diferentemente do algoritmo elaborado por Bean, onde ambos são selecionados de toda a população. Com essa modificação foi possível obter melhores resultados do que os apresentados pelo RKGA [Gonçalves et al., 2014].

Como o algoritmo de Bean, o BRKGA utiliza o método Uniforme Parametrizado de Spears e Jong [Spears e Jong, 1991] como operador de cruzamento. Esse método recorre a uma probabilidade ρ do descendente herdar o alelo do indivíduo proveniente do grupo elite e $1 - \rho$ de herdar do outro indivíduo. Para o seu funcionamento, um número real r_i aleatório no intervalo $[0, 1)$ é gerado para cada alelo i ($i = 1, \ldots, n$) e comparado com

a probabilidade ρ. Se $r_i < \rho$, o alelo é herdado do indivíduo elite, sendo recebido do indivíduo não-elite caso contrário. A Figura 4.2 apresenta um exemplo da execução desse método, adotando $\rho = 0.7$ e $n = 5$. Nesse exemplo, o descendente herdou o alelo do indivíduo elite para $i = 1, 3$ e 5.

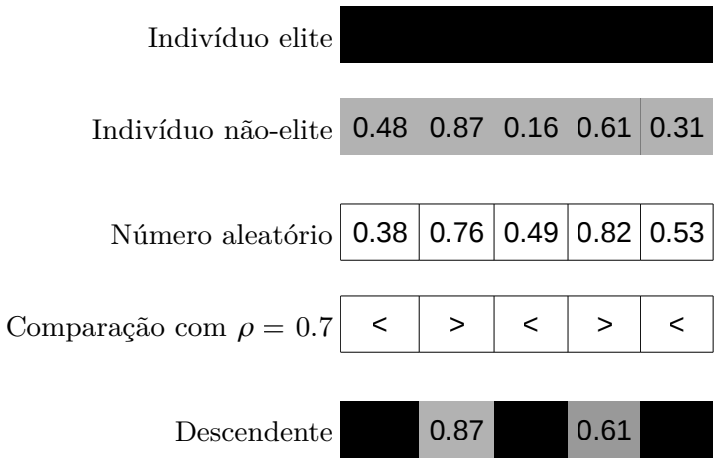


Figura 4.2: Exemplo de cruzamento uniforme parametrizado.

Um ponto importante observado na estrutura desses dois algoritmos de chaves aleatórias é que ambos possuem componentes que são totalmente independentes do problema a ser solucionado. O único componente dependente é o decodificador, que é responsável por associar o algoritmo ao problema tratado, retornando uma solução viável para o problema e o seu valor de *fitness* dado um indivíduo de chaves aleatórias. A Figura 4.3 apresenta o fluxograma de um BRKGA, destacando os componentes que dependem e aquele que não depende do problema.

Na literatura são encontradas aplicações do BRKGA aos mais variados problemas da área de otimização combinatória, como em telecomunicações [Ericsson et al., 2002, Buriol et al., 2007, Noronha et al., 2011, Resende, 2012], carregamento de *containers* [Gonçalves e Resende, 2012, Gonçalves e Resende, 2013, Zheng et al., 2015], planejamento de tráfego [Buriol et al., 2010, Stefanello et al., 2017], otimização em redes [Fontes e Gonçalves, 2007, Fontes e Gonçalves, 2013, Andrade et al., 2015], entre outros.

A título de exemplo, Gonçalves et al. [Gonçalves et al., 2005] empregaram o BRKGA para solucionar o Problema de Escalonamento *Job Shop*, que considera um conjunto de n *jobs* e um grupo de m máquinas, onde cada *job* é composto por uma sequência de x operações. Toda operação precisa ser executada em uma única máquina durante um período de tempo fixo e ininterrupto, sendo que cada máquina processa, no máximo, uma operação por vez. As operações de um *job* têm que ser processadas em uma dada or-

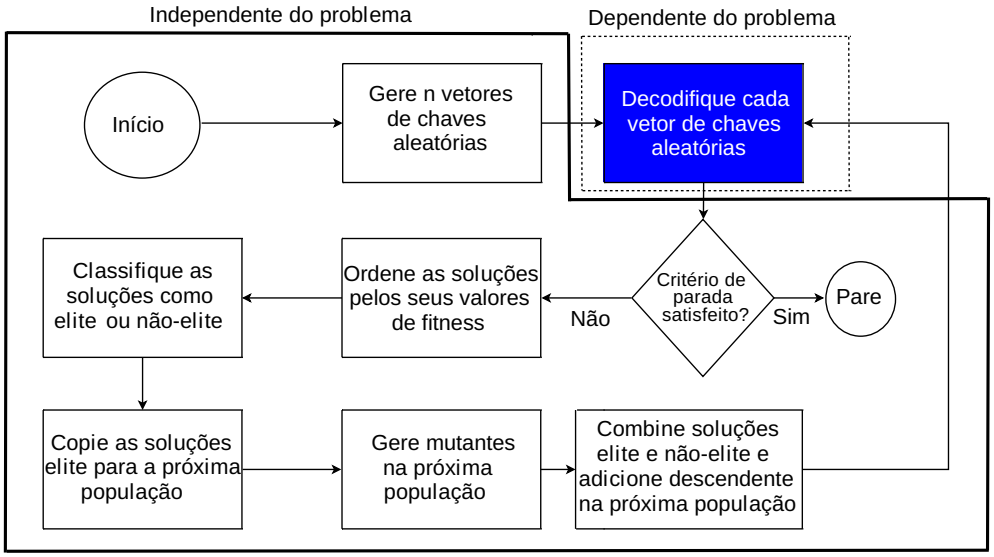


Figura 4.3: Fluxograma de um BRKGA, destacando os componentes que dependem e aquele que não depende do problema (Extraída de [Gonçalves e Resende, 2011a]).

dem. Perante tais restrições, o problema consiste em encontrar um escalonamento das operações nas referidas máquinas que minimize o tempo de término da última operação completa do escalonamento, conhecido como *makespan*. Os testes realizados compararam o algoritmo com outros 12 sobre um conjunto de 43 instâncias da literatura. Os experimentos revelaram que a heurística proposta alcançou o melhor resultado conhecido em 72% das instâncias testadas, com um desvio relativo médio de 0.39% para a melhor solução conhecida.

Outra aplicação pode ser encontrada em [Gonçalves e Resende, 2011b], onde os autores apresentam uma heurística híbrida que combina o BRKGA com um novo método de posicionamento para solucionar o Problema de Empacotamento Bidimensional Não-guilhotinado Restrito. Nesse problema, uma quantidade fixa de pequenas peças retangulares têm que ser dispostas em um grande retângulo plano, de forma a maximizar o valor das peças alocadas. Especialmente nesse caso, as peças não podem sofrer rotação e devem ser posicionadas sempre com suas arestas paralelas às arestas do retângulo maior. A heurística proposta foi testada sobre um conjunto de 703 instâncias da literatura e comparada a outros quatro algoritmos. Os testes demonstraram que a heurística obteve excelentes resultados em termos de qualidade de solução, apresentando robustez em relação aos demais algoritmos.

O BRKGA também foi empregado na resolução do Problema de Alocação Tática de Berços no trabalho de Lalla-Ruiz et al. [Lalla-Ruiz et al., 2014]. Este problema se

manifesta no contexto de terminais portuários e consiste em alocar navios aos berços do terminal, atribuindo perfis de guindastes para atendê-los no carregamento e descarregamento de cargas. Os seus objetivos são minimizar os custos do serviço de transporte de cargas entre os navios ancorados e maximizar o total de perfis de guidastes designados a eles. Esses perfis são uma representação do número de guindastes que serão atribuídos para um certo navio enquanto ele estiver ancorado, tendo cada perfil um valor específico que reflete aspectos técnicos. Para avaliar a eficiência da heurística desenvolvida, testes computacionais foram realizados com instâncias elaboradas pelos autores, além das clássicas encontradas na literatura. Os resultados foram comparados aos de outros três algoritmos e aos reportados por um modelo matemático para o problema, que foi submetido a um resolvedor. Os experimentos indicaram que o BRKGA foi capaz de prover soluções de alta qualidade em um reduzido tempo de processamento, bem como de poder se adaptar a problemas de tamanho real. Em razão da sua flexibilidade em resolver instâncias de diferentes dimensões, também foi possível verificar que o esforço computacional requerido não é altamente influenciado pelo tamanho das mesmas.

Em [Lalla-Ruiz et al., 2016] outro importante problema da área de otimização foi tratado utilizando uma heurística híbrida baseada no BRKGA. Considerando um conjunto de n facilidades, que requer um certo fluxo simétrico entre cada par, e outro de n localizações, com uma distância simétrica entre duas, o Problema de Alocação Quadrática tem como objetivo minimizar o custo derivado da distância e fluxos entre as facilidades. Na abordagem proposta, cada gene do indivíduo representa uma facilidade, que será atribuída a uma localização disponível. Esta atribuição será dada seguindo a ordem das facilidades após os genes do indivíduo serem ordenados de maneira não-decrescente pelas suas chaves.

Os testes computacionais foram realizados sobre um conjunto de instâncias clássicas da literatura divididas em esparsas e densas. Foram analisados os resultados da heurística híbrida com outros dois algoritmos e com um BRKGA sem a utilização da vizinhança. A qualidade das soluções da heurística indica a sua grande eficácia quando comparada a um dos algoritmos, independentemente da densidade das instâncias. Além disso, ela apresentou, na média, uma performance melhor do que os demais métodos, sendo capaz de retornar novas melhores soluções para três instâncias. Um estudo adicional também foi realizado, a fim de verificar o desempenho da aplicação da busca em vizinhança na melhoria dos indivíduos elite. Nesse estudo, os autores comparam a heurística implementada com o BRKGA sem a vizinhança e com duas versões de um RKGA, uma com o emprego da busca e a outra sem o seu uso. Os resultados demonstraram que a incorporação da busca melhorou a qualidade das soluções e que os dois métodos que a utilizam apresenta-

ram comportamentos semelhantes, embora com uma performance um pouco melhor para a heurística híbrida proposta. Essa pequena vantagem também é vista na convergência para a melhor solução, que acontece de maneira mais rápida do que o outro método.

Recentemente, Brandão et al. [Brandão et al., 2017] empregaram o BRKGA na resolução do Problema de Escalonamento de Cargas Divisíveis em Múltiplos Períodos. Uma carga divisível é uma quantidade de trabalho computacional que pode ser dividida e distribuída de forma aleatória entre processadores distintos para serem executadas em paralelo. Esses processadores estão dispostos de modo que o processador central, denominado *master*, armazena e divide a carga em porções de tamanhos arbitrários para serem transmitidos aos demais P processadores, chamados de operários. Dentre as restrições para o problema, o processador *master* pode enviar uma carga somente para um operário de cada vez, que só pode iniciar o processamento da mesma após recebê-la completamente. Assim, com o objetivo de minimizar o *makespan*, o problema tratado consiste em selecionar um subconjunto $A \subseteq P$ de n operários que executarão as cargas, chamados operários ativos; definir uma ordem, intitulada ordem de ativação, pela qual as cargas serão transmitidas para cada um deles; definir o número m de períodos de transmissão que serão utilizados; e decidir a quantidade de carga que será transmitida para cada operário $i \in A$ em cada período $k \in \{1, \ldots, m\}$. A fim de investigar a qualidade das soluções obtidas pelo BRKGA, os resultados dos experimentos realizados, sobre um conjunto de seis instâncias, foram comparados aos de outras duas heurísticas para o problema, sendo uma a melhor da literatura até então. Os testes demonstraram que a heurística proposta encontra melhores soluções mais rápido que os demais métodos e que converge para o melhor valor antes de um segundo para todas as instâncias, além de melhorar os *makespans* em 11.68%, na média.

4.2 BRKGA Aplicado ao PPCCM

Como a associação do BRKGA ao problema tratado é feita exclusivamente pelo decodificador, sendo necessário para cada problema em particular, esse componente é essencial para o êxito do algoritmo. Na heurística baseada no BRKGA proposta neste trabalho para a resolução do PPCCM, o decodificador recebe um indivíduo de n chaves aleatórias, onde cada gene corresponde a um dos n vértices do grafo tratado. Para alcançar uma solução viável, inicialmente ele obtém uma determinada sequência dos vértices ordenando de maneira não-decrescente as n chaves. Em seguida, inicia a coloração dos mesmos a partir da ordem dada por essa ordenação, utilizando a cor de menor custo. Quando mais

nenhum vértice puder ser colorido com essa cor e considerando que ela não será novamente empregada, a cor com o segundo menor custo é selecionada a fim de colorir os vértices que ainda não tenham cor associada. Essa estratégia prossegue até que todos os vértices do grafo tenham sido coloridos, sempre respeitando a ordem dos mesmos de acordo com as chaves e colorindo os adjacentes com cores distintas. Nessa abordagem, os indivíduos mais aptos são os que apresentam o menor valor de *fitness*, que é a soma dos custos das cores atribuídas a cada vértice.

O Algoritmo 1 apresenta o pseudocódigo do decodificador proposto. A solução S que será gerada é inicializada nas linhas 1–4. C_i denota as classes de cores da solução S associadas às cores $i = 1, \ldots, n$, sendo indexadas na ordem não-decrescente pelos seus custos na linha 5. Na linha 6, os vértices são copiados para um conjunto auxiliar V', sendo ordenados na linha 7 de maneira não-decrescente pelas suas chaves aleatórias correspondentes. A alocação dos vértices nas classes de cores é realizada no laço 9–19 seguindo a ordenação dos mesmos, iniciando pela classe de menor custo C_1 (linha 8). Se o vértice que está sendo analisado ainda não foi colorido (linha 10), o mesmo é atribuído à classe atual C_i na linha 11. No laço interno 12–16, os demais vértices são alocados a esta mesma classe, desde que ainda não tenham sido coloridos e que não sejam adjacentes aos vértices pertencentes à ela. Uma vez que a classe de cor atual não será mais utilizada, a classe de menor custo seguinte é selecionada (linha 17), a fim de colorir os vértices ainda sem classe associada. Finalizada a alocação de todos os vértices, o valor de *fitness* do indivíduo I é calculado na linha 20.

Na tentativa de aprimorar a qualidade dos indivíduos presentes no grupo elite, uma variação do BRKGA proposto, fazendo uso do mesmo decodificador, também foi desenvolvida. A nova heurística, denominada BRKGA+RVNS, diferencia-se da primeira apenas pela aplicação de uma busca em vizinhança sobre cada indivíduo que fará parte daquele grupo na geração seguinte, realizando a substituição do indivíduo caso encontre outro com valor de *fitness* menor. A Figura 4.4 demonstra o processo de evolução da população dessa heurística.

A estratégia de busca utilizada foi a Busca em Vizinhança Variável Reduzida (*Reduced Variable Neighborhood Search* - RVNS, em inglês) [Hansen e Mladenović, 1999], considerada uma modificação da metaheurística Busca em Vizinhança Variável (VNS, do inglês *Variable Neighborhood Search*) [Mladenović e Hansen, 1997], onde o procedimento de busca local não é aplicado. No RVNS, dado um conjunto de vizinhanças N_h ($h = 1, \ldots, h_{max}$), uma solução S' é obtida aleatoriamente em uma vizinhança $N_h(S)$ de

Algoritmo 1: Decodificador

Entrada: Indivíduo I.
Saída: Valor de *fitness* do indivíduo I.

```
1  S : ⟨C_1, ..., C_n⟩;
2  para i = 1, ..., n faça
3  |  C_i ← ∅;
4  fim para
5  Ordene as classes de cores da solução S pelos seus custos: w_{C_i} ≤ w_{C_{i+1}}, 1 ≤ i < n;
6  V' ← V;
7  Ordene os vértices de V' pelas suas chaves aleatórias: I_{V'_j} ≤ I_{V'_{j+1}}, 1 ≤ j < n;
8  i ← 1;
9  para j = 1, ..., n faça
10 |  se V'_j ∉ C_k, 1 ≤ k ≤ i então
11 |  |  C_i ← C_i ∪ {V'_j};
12 |  |  para ℓ = j + 1, ..., n : V'_ℓ ∉ C_k, 1 ≤ k ≤ i faça
13 |  |  |  se V'_ℓ não é adjacente a u, ∀u ∈ C_i então
14 |  |  |  |  C_i ← C_i ∪ {V'_ℓ};
15 |  |  |  fim se
16 |  |  fim para
17 |  |  i ← i + 1;
18 |  fim se
19 fim para
20 fitness ← f(S);
```

uma solução corrente S. Se S' for melhor do que S, a exploração prossegue a partir de S' e recomeça pela primeira estrutura de vizinhança $N_1(S')$. Caso contrário, a busca avança para a vizinhança seguinte $N_{h+1}(S)$ da solução corrente. Quando todas as vizinhanças de uma solução forem exploradas, a busca retorna à primeira, reiniciando o processo. Essas etapas acontecem até que um critério de parada seja satisfeito. A aplicação do RVNS para a melhoria dos indivíduos elite também foi empregada por Ma et al. [Ma et al., 2017], porém com uma estratégia diferente para a alteração dos alelos, ao fazer somente um tipo de operação (mutação) em genes selecionados aleatoriamente.

No procedimento RVNS proposto neste trabalho, as estruturas de vizinhança correspondem a alterações nos alelos dos indivíduos, que são as suas chaves aleatórias. O Algoritmo 2 descreve esse procedimento, que recebe como entrada cada indivíduo elite I. Inicialmente, na linha 1 uma cópia de I é efetuada, para que a mesma armazene o melhor indivíduo I^{best} (com o menor valor de *fitness*) encontrado durante a busca. O laço externo das linhas 3–24 permite que o processo de melhoria seja executado duas vezes. No laço interno das linhas 5–22, este processo explora a vizinhança do indivíduo I^{aux}, que é uma cópia do melhor indivíduo até o momento (linha 6).

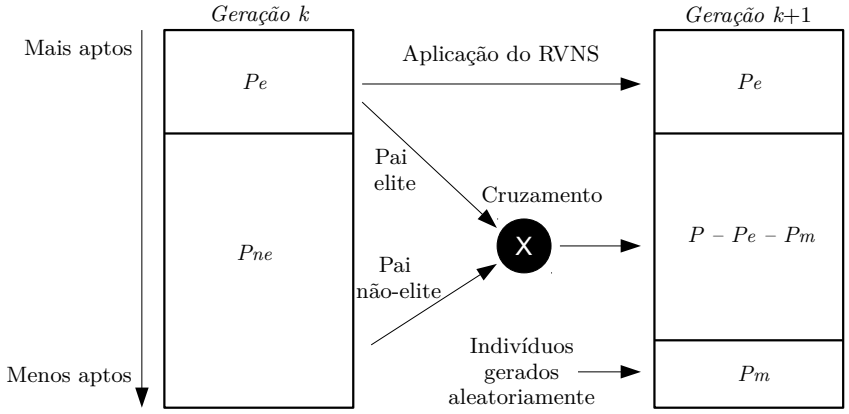


Figura 4.4: Processo de evolução da população no BRKGA+RVNS.

Definido o indivíduo a ser examinado, acontece a busca na vizinhança h ($h = 1, \ldots, 4$) do mesmo, que consiste em alterar o valor de h genes do indivíduo. Iniciando pela primeira vizinhança ($h = 1$), no laço das linhas 7–15 são selecionados h genes aleatoriamente e diferentes entre si para receberem, como alteração, uma nova chave aleatória na linha 11 ou a operação (1 − o valor da chave atual) na linha 13. A escolha de uma dessas alterações é feita com probabilidade igual a 0.5 (linha 10). Essas duas mudanças pretendem proporcionar, ao vértice associado a cada gene, a variação da sua posição no processo de coloração, uma vez que este processo ocorre seguindo a ordenação das chaves do indivíduo. Na primeira, essa posição é dada aleatoriamente pela geração da nova chave. Na segunda alteração, a posição é trocada utilizando a chave atual, podendo o vértice associado passar a ser colorido no início ou no fim do processo, dependendo da sua posição original.

Após as modificações em h genes de I^{aux}, caso o seu valor de *fitness* seja menor que o do melhor indivíduo (linha 16), este último é atualizado na linha 17 e a busca reinicia sobre ele na primeira vizinhança (linha 18). Caso contrário, a exploração segue para a vizinhança seguinte na linha 20. Esse processo de busca acontece até que todas as vizinhanças sejam examinadas (linhas 5–22), sendo retomado novamente na primeira delas na linha 4 para a segunda e última repetição. Ao final, caso o melhor indivíduo encontrado na busca seja mais apto do que o de entrada, este é atualizado nas linhas 25–26.

4.2.1 Experimentos Computacionais

Os experimentos realizados têm como finalidade ajustar os parâmetros das heurísticas BRKGA e BRKGA+RVNS propostas, bem como avaliar a qualidade das soluções produzidas por elas sobre um conjunto de instâncias. A biblioteca *brkgaAPI* [Toso e Resende,

Algoritmo 2: RVNS

Entrada: Indivíduo I do grupo elite.
Saída: Indivíduo I.

```
1  I^best ← I;
2  cont ← 1;
3  enquanto cont ≤ 2 faça
4  |  h ← 1;
5  |  enquanto h ≤ 4 faça
6  |  |  I^aux ← I^best;
7  |  |  para i de 1 até h faça
8  |  |  |  g ← aleatório[1, n], g ∈ ℤ e ainda não escolhido;
9  |  |  |  a ← aleatório[0, 1), a ∈ ℝ;
10 |  |  |  se a < 0.5 então
11 |  |  |  |  I_g^aux ← nova chave aleatória;
12 |  |  |  senão
13 |  |  |  |  I_g^aux ← 1 − I_g^aux;
14 |  |  |  fim se
15 |  |  fim para
16 |  |  se f(I^aux) < f(I^best) então
17 |  |  |  I^best ← I^aux;
18 |  |  |  h ← 1;
19 |  |  senão
20 |  |  |  h ← h + 1;
21 |  |  fim se
22 |  fim enquanto
23 |  cont ← cont + 1;
24 fim enquanto
25 se f(I^best) < f(I) então
26 |  I ← I^best;
27 fim se
```

2015], que é um *framework* na linguagem C++ para o desenvolvimento de BRKGAs, foi utilizada para implementar os dois algoritmos, sendo compilados com g++ versão 5.4.0. O computador empregado nos testes dispõe de um processador Intel Core i7-4790K de 4 GHz, com 16 GB de memória RAM e sistema operacional Linux Ubuntu 16.04 LTS 64 bits. Para a avaliação dos resultados obtidos nesses dois conjuntos de experimentos, foram empregadas medidas de qualidade aplicadas em [Ribeiro et al., 2002], [Resende et al., 2010], [Pessoa et al., 2013], [Brandão et al., 2015], entre outros:

- *Best*: valor da melhor solução obtida para uma determinada instância, considerando todas as execuções até então realizadas.
- *Sum Best*: número de execuções para as quais o valor *Best* foi alcançado por uma

determinada heurística. Quanto maior o valor de *Sum Best*, melhor é a performance do algoritmo.

- *#Best*: número de instâncias para as quais o valor *Best* foi alcançado por uma determinada heurística. Quanto maior o valor de *#Best*, melhor é a performance do algoritmo.

- *Dev*: desvio relativo entre *Best* e o valor da solução obtida em uma execução de uma instância por uma dada heurística. Quanto menor o valor de *Dev*, melhor é a performance do algoritmo.

- *Avg Dev*: valor médio de *Dev* sobre todas as instâncias e execuções de uma dada heurística. Quanto menor o valor de *Avg Dev*, melhor é a performance do algoritmo.

- *#Score*: para cada instância, representa o número de algoritmos que encontram uma solução melhor do que uma heurística específica. Caso duas ou mais heurísticas apresentem o mesmo resultado, todas recebem valor igual de *#Score*, indicando o número de algoritmos estritamente melhores do que todas elas.

- *Score*: soma dos *#Score* de uma heurística específica, sobre todas as instâncias do experimento. Quanto menor o valor de *Score*, melhor é a performance do algoritmo.

4.2.1.1 Ajustes de Parâmetros

Para esse experimento foram selecionados aleatoriamente 20 grafos de *benchmarks* amplamente utilizados para o PCG, obtidos em `http://mat.gsia.cmu.edu/COLOR03` e `http://mat.gsia.cmu.edu/COLOR/instances.html`. Como no PPCCM os custos das cores podem assumir qualquer valor real, para cada instância foram gerados custos aleatórios com valores inteiros variando de 1 até 1000, que, embora sejam inteiros, mantém o problema diferente do PSC, uma vez que neste tais valores são números naturais em sequência. Foi empregado o gerador Mersenne Twister [Matsumoto e Nishimura, 1998] para a geração de números aleatórios.

Os grafos utilizados são apresentados na Tabela 4.1, acompanhados do número de vértices (n), de arestas (m) e o valor ótimo (em negrito) de cada um. Este último foi obtido aplicando o resolvedor CPLEX 12.8.0 com o modelo das Equações (3.1) a (3.4) (Seção 3.2) e ajuste automático de parâmetros, adotando como critério de parada o tempo máximo de execução de 3600 segundos. Este experimento foi realizado em uma máquina virtual VMware ESXi 6.5 dispondo de 8 núcleos do processador Intel Xeon E5-2690 v4 2.6

GHz, 48 GB de memória RAM e sistema operacional Linux Ubuntu 16.04 LTS 64 bits. Para as instâncias que o valor ótimo não foi determinado pelo CPLEX dentro do limite de tempo, a tabela informa o melhor valor conhecido para a instância ao longo de todos os experimentos realizados nesta tese, por todos os algoritmos e variantes desenvolvidos.

Instância	n	m	Ótimo ou melhor valor
myciel3	11	20	**3097**
myciel4	23	71	**2068**
queen5_5	25	160	**3960**
2-Insertions_3	37	72	**695**
myciel5	47	236	**2212**
queen7_7	49	476	**2590**
2-FullIns_3	52	201	**832**
3-Insertions_3	56	110	**914**
huck	74	301	**1967**
jean	80	254	**2307**
david	87	406	**2819**
myciel6	95	755	**1657**
mug100_25	100	166	**3177**
games120	120	638	**4351**
anna	138	493	**1137**
DSJC125.1	125	736	1081
DSJC250.1	250	3218	1941
DSJC500.1	500	12458	5848
2-FullIns_5	852	12201	1319
DSJC1000.1	1000	49629	10587

Tabela 4.1: Instâncias utilizadas para o ajuste de parâmetros das heurísticas BRKGA e BRKGA+RVNS.

Uma vez definida a quantidade de alelos de cada indivíduo, considerada igual ao número de vértices do grafo tratado (Seção 4.2), de acordo com Gonçalves e Resende [Gonçalves e Resende, 2011a], os parâmetros necessários para a execução de um BRKGA são: o tamanho da população (P), a quantidade de indivíduos que pertencerão ao grupo elite (P_e), o número de indivíduos mutantes que serão gerados (P_m) e a probabilidade do descendente herdar um alelo do indivíduo elite no procedimento de cruzamento (ρ).

No entanto, para as heurísticas propostas, também há a necessidade de análise do parâmetro r, uma vez que será utilizada a técnica de *restarts* (reinicializações), por reconhecidamente contribuir com a redução do tempo necessário para alcançar boas soluções em algoritmos randomizados [Resende e Ribeiro, 2011]. Em ambas as heurísticas, duas estratégias de *restarts* foram testadas para a reconstrução total da população: (a) após r gerações sem melhoria da melhor solução e (b) após r gerações ininterruptas. Na

Parâmetro	Valor	BRKGA	BRKGA+RVNS
P	100, 150, 200	150	150
P_e	$0.20\times P$, $0.25\times P$	$0.25\times P$	$0.20\times P$
P_m	$0.10\times P$, $0.20\times P$	$0.10\times P$	$0.10\times P$
ρ	0.70, 0.80	0.70	0.70
r	$50^{(a)}$, $50^{(b)}$, $100^{(a)}$, $100^{(b)}$	$100^{(a)}$	$100^{(b)}$

Tabela 4.2: Valores dos parâmetros utilizados para ajuste e os melhores valores obtidos para as heurísticas BRKGA e BRKGA+RVNS.

Tabela 4.2, o valor utilizado para *restarts* é acompanhado dessas letras para indicar a estratégia aplicada.

Os valores analisados dos parâmetros, especificados na segunda coluna da Tabela 4.2, com exceção dos valores para r, seguiram como referência Gonçalves e Resende [Gonçalves e Resende, 2011a], tendo a combinação dos mesmos originado 96 versões de cada heurística. A fim de aplicar um critério de parada justo para todas elas, utilizou-se o tempo máximo de processamento, onde cada versão faz dez execuções independentes (utilizando sementes distintas para o gerador de números aleatórios) para cada instância com tal limite de tempo. Para a obtenção desse limite, a heurística BRKGA+RVNS foi executada uma única vez para cada instância por 200 gerações, usando como parâmetros $P = 200$, $P_e = 0.25\times P$, $P_m = 0.10\times P$, $\rho = 0.70$ e $r = 50^{(b)}$.

Na avaliação dos resultados das versões, foram adotadas as medidas *Sum Best* e *Avg Dev*, utilizando como critério de qualidade um alto valor para a primeira e, em caso de empate, o menor valor para a segunda. As Figuras 4.5 e 4.6 mostram gráficos com os valores dessas medidas para BRKGA e BRKGA+RVNS, respectivamente. Nesses gráficos, cada ponto corresponde a uma ou mais versões, onde a coordenada de cada ponto são as duas medidas adotadas. Os pontos mais acima e à esquerda dos gráficos representam aquelas versões que alcançaram os melhores valores para as referidas medidas. Observa-se que, para o BRKGA, a versão que atingiu *Sum Best* = 160 e *Avg Dev* = 0.041 foi a que obteve a melhor combinação de parâmetros, segundo os critérios definidos. Esses parâmetros são identificados na terceira coluna da Tabela 4.2. Para o BRKGA+RVNS, a melhor versão alcançou *Sum Best* = 160 e *Avg Dev* = 0.034, tendo os parâmetros apresentados na quarta coluna da Tabela 4.2.

4.2.1.2 Análise de Qualidade das Soluções

Com o objetivo de avaliar a qualidade das soluções apresentadas pelas heurísticas BRKGA e BRKGA+RVNS, empregando os melhores parâmetros determinados nos ex-

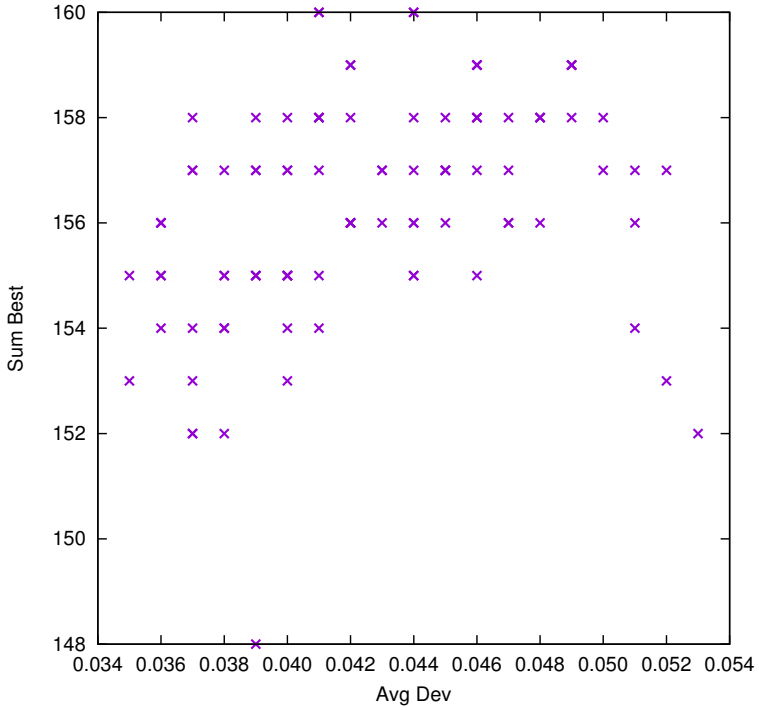


Figura 4.5: Resultados das medidas *Sum Best* e *Avg Dev* para as versões de ajuste de parâmetros do BRKGA. O critério de qualidade adotado foi apresentar um alto valor para a primeira medida e, em caso de empate, o menor valor na segunda. Neste caso, a melhor versão obteve *Sum Best* = 160 e *Avg Dev* = 0.041.

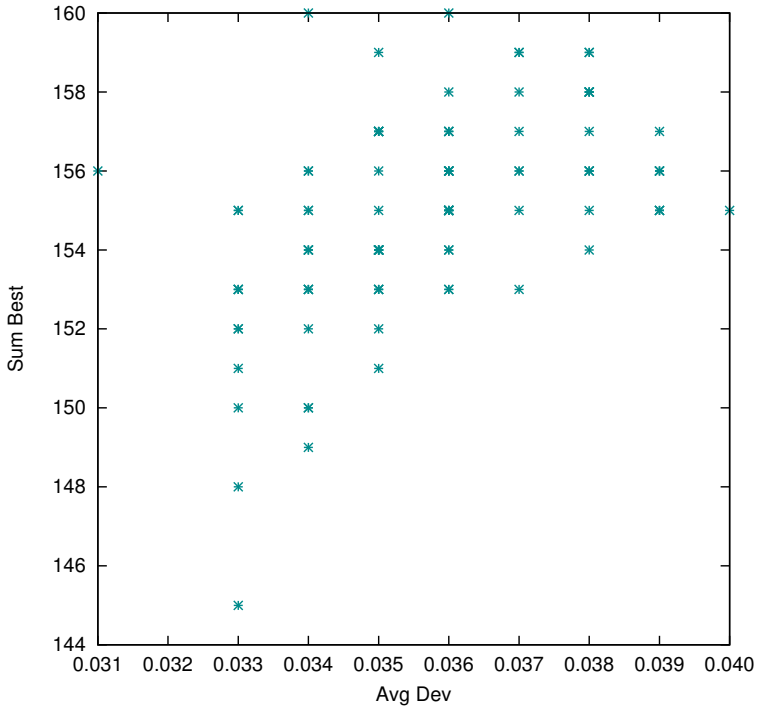


Figura 4.6: Resultados das medidas *Sum Best* e *Avg Dev* para as versões de ajuste de parâmetros do BRKGA+RVNS. O critério de qualidade adotado foi apresentar um alto valor para a primeira medida e, em caso de empate, o menor valor na segunda. Neste caso, a melhor versão obteve *Sum Best* = 160 e *Avg Dev* = 0.034.

perimentos de ajuste (Tabela 4.2), foram geradas 50 novas instâncias do mesmo modo como descrito na Seção 4.2.1.1. Sobre cada uma delas, cada heurística realizou dez execuções independentes, tendo, como critério de parada, um limite de tempo máximo para cada execução, que foi obtido pela aplicação do BRKGA+RVNS uma única vez por 200 gerações, utilizando os melhores parâmetros identificados na etapa de ajuste para essa heurística. Na tentativa de obter a solução ótima de cada instância, novamente foi utilizado o resolvedor CPLEX, sob as mesmas condições especificadas na Seção 4.2.1.1.

Os resultados detalhados dos experimentos são descritos nas Tabelas A.1 e A.2 (Apêndice A), que apresentam, para cada instância, o número de vértices (n), de arestas (m), o valor da melhor solução conhecida ao longo de todos os experimentos realizados, por todos os algoritmos e variantes desenvolvidos, e o valor da melhor solução alcançada pelo CPLEX, sublinhando os valores das soluções ótimas e sinalizando com o símbolo '−' caso uma solução viável não tenha sido obtida no tempo máximo de execução (3600 segundos). As colunas seguintes indicam, para cada heurística, o valor da melhor solução obtida, indicando em negrito quando este é igual ao melhor conhecido, a média dos valores das soluções alcançadas nas 10 execuções, o número de vezes que o melhor valor conhecido foi atingido, o desvio relativo médio percentual entre o valor da solução obtida e o valor da melhor solução conhecida, e o valor médio do índice da geração em que a melhor solução é encontrada. A última coluna indica o tempo (em segundos) utilizado como critério de parada em cada execução das heurísticas.

A Tabela 4.4 resume os resultados obtidos pelos dois algoritmos, apresentando para cada um deles somente o melhor valor de solução e o desvio relativo médio percentual. Analisando essas informações é possível observar que o CPLEX alcançou o melhor valor conhecido para 18 instâncias (em negrito), a solução ótima para 12 e não obteve solução viável, considerando o limite de tempo, para outras 13. Para seis instâncias, as duas heurísticas não encontraram um valor de solução tão bom quanto o encontrado pelo CPLEX.

Além disso, verifica-se que BRKGA atinge o melhor valor conhecido para 16 instâncias e o ótimo para nove delas, enquanto BRKGA+RVNS atinge o melhor valor conhecido para 14, sendo 10 ótimos. Este último alcançou o melhor valor conhecido em todas as 10 execuções para seis instâncias e BRKGA para quatro. A instância em que ambos não atingiram o melhor valor conhecido e que apresentaram maior dificuldade para tal, foi a mesma para os dois algoritmos (flat1000_60_0), porém BRKGA+RVNS alcançou desvio médio de 26.24%, inferior aos 28.86% do BRKGA.

	BRKGA	BRKGA+RVNS
Avg Dev (%)	**8.70**	8.72
Sum Best	92	**93**
#Best	**16** (9 ótimos)	14 (10 ótimos)
Score	**16**	21

Tabela 4.3: Comparativo da performance das heurísticas BRKGA e BRKGA+RVNS.

Analisando a performance de cada heurística utilizando os resultados das medidas de qualidade, resumidos na Tabela 4.3, pode-se verificar que BRKGA encontrou o melhor valor conhecido em 92 execuções e BRKGA+RVNS em 93, tendo BRKGA encontrado tal valor para duas instâncias a mais. Nota-se também que este apresentou um desvio médio de todo o experimento ligeiramente menor, bem como um valor de *Score* inferior.

Além dessas medidas, com o intuito de avaliar a distribuição do tempo de processamento de ambas as heurísticas, foram utilizados os gráficos *Time-To-Target* [Feo et al., 1994], ou TTT-Plots, que apresentam a probabilidade de um algoritmo encontrar uma solução de custo, no mínimo, tão bom quanto um dado valor alvo, em um determinado tempo. Para isso, seguindo a metodologia proposta em [Aiex et al., 2002] e [Aiex et al., 2007], cada heurística realiza 200 execuções independentes sobre uma instância, cada uma finalizada ao alcançar uma solução de custo menor ou igual a um valor alvo ou ao atingir um dado limite de tempo, quando este recurso for utilizado. Em seguida, os tempos dessas execuções são ordenados de maneira não-decrescente e a probabilidade $p_i = (i - \frac{1}{2})/200$ é associada ao i-ésimo tempo de processamento t_i, permitindo então plotar os pontos $d_i = (t_i, p_i)$, para $i = 1, \ldots, 200$. Para a análise dos resultados, quanto mais à esquerda a curva de um determinado algoritmo, melhor ele é, pois indica que o mesmo atinge o valor alvo mais rapidamente.

Em conjunto a esses gráficos, a fim de se obter um resultado numérico da comparação das duas heurísticas, foi empregada a ferramenta *tttplots-compare* desenvolvida por Ribeiro e Rosseti [Ribeiro e Rosseti, 2015], cujo suporte teórico encontra-se em [Ribeiro et al., 2012]. Considerando dois algoritmos estocásticos A_1 e A_2, T_{A_1} (resp. T_{A_2}) é uma variável aleatória contínua que representa o tempo para atingir o alvo do algoritmo A_1 (resp. A_2). Assim, a ferramenta informa a probabilidade $Pr(T_{A_1} \leq T_{A_2})$ de o algoritmo A_1 convergir de maneira mais rápida do que o algoritmo A_2.

Os TTT-Plots para a instância 3-FullIns_4 são encontrados na Figura 4.7, onde foi definido como alvo o valor da melhor solução conhecida (2951) e 1000 segundos como tempo de execução máximo. BRKGA consegue 100% de probabilidade de atingir o alvo

em menos de seis segundos de processamento, enquanto BRKGA+RVNS necessita de 16.3 segundos para obter a mesma probabilidade. Para essa instância, BRKGA apresenta $Pr(T_{BRKGA} \leq T_{BRKGA+RVNS}) = 0.886$. As Figuras 4.8 e 4.9 mostram, respectivamente, que a população do BRKGA e do BRKGA+RVNS convergem para o melhor valor durante os quatro segundos iniciais de execução, sendo possível notar que nesse período o processo de *restarts* de nenhuma das heurísticas foi executado.

Os resultados para a instância inithx.i.1 são apresentados na Figura 4.10. Nesse experimento foi empregada como alvo a média dos valores das soluções encontradas pelo BRKGA (3937) e 1000 segundos como tempo máximo. BRKGA+RVNS para essa instância atinge o valor alvo com probabilidade 100% em 209 segundos, sendo de 402.5 segundos para BRKGA. Ainda, BRKGA+RVNS obteve $Pr(T_{BRKGA+RVNS} \leq T_{BRKGA}) = 0.529$, indicando que tem maior probabilidade de convergir mais rapidamente para o alvo do que a outra heurística. Na evolução da população durante 50 segundos iniciais de execução, BRKGA atingiu 3935 como custo de solução (Figura 4.11), tendo BRKGA+RVNS alcançado o melhor valor conhecido 3934 (Figura 4.12). No tempo analisado, somente para BRKGA verificou-se a execução do processo de *restarts*.

4.2.1.3 Conclusões

Definindo um tempo de processamento fixo como critério de parada para as heurísticas, verificou-se que a heurística BRKGA encontrou soluções ligeiramente melhores do que BRKGA+RVNS, com valores, em média, 8.70% e 8.72%, respectivamente, acima dos melhores valores conhecidos para as instâncias examinadas. Embora BRKGA tenha encontrado estes valores para um maior número de instâncias, BRKGA+RVNS atingiu 10 valores ótimos, contra nove do BRKGA.

Esse equilíbrio também foi observado nos experimentos para avaliar a probabilidade de um algoritmo convergir de maneira mais rápida para um valor alvo. Percebeu-se também que, apesar de o procedimento RVNS tornar mais custosa as gerações do BRKGA+RVNS em termos de tempo de execução, ele foi capaz de auxiliar na redução do número de gerações para a obtenção das soluções. Desse modo, considerando os experimentos realizados e os resultados obtidos, não é possível determinar qual heurística apresenta-se superior a outra, uma vez que ambas se alternam em qualidade dependendo da instância tratada.

Assim, com o propósito de desenvolver um outro algoritmo para solucionar o PPCCM, foi proposta uma heurística de trajetória que faz uso de duas estratégias de busca local seguidas por um procedimento de *path-relinking*, sendo descrita no capítulo seguinte.

Instância	n	m	Melhor valor	CPLEX (3600 s)	BRKGA		BRKGA+RVNS		Tempo de execução heurísticas (s)
					Melhor valor	Avg Dev (%)	Melhor valor	Avg Dev (%)	
school1-nsh	352	14612	7647.00	13193.00	8999.00	20.47	9020.00	25.13	30.95
school1	385	19095	7158.00	14687.00	7723.00	12.13	7733.00	14.87	36.03
3-FullIns_4	405	3524	2951.00	**2951.00**	**2951.00**	0.00	**2951.00**	0.00	29.85
fpsol2.i.3	425	8688	3738.00	**3738.00**	**3738.00**	0.00	**3738.00**	0.00	35.15
le450_5c	450	9803	2610.00	**2610.00**	2642.00	2.34	2645.00	3.28	45.98
le450_5d	450	9757	2700.00	3421.00	2710.00	1.59	2711.00	1.61	47.64
le450_15c	450	16680	9556.00	11335.00	11131.00	17.05	10683.00	12.94	55.07
le450_15d	450	16750	10799.00	13011.00	12629.00	20.42	12441.00	15.93	54.71
le450_25a	450	8260	9730.00	**9730.00**	10258.00	6.63	10485.00	8.33	38.63
le450_25b	450	8263	7564.00	**7564.00**	8028.00	7.08	8241.00	9.43	42.11
le450_25c	450	17343	10447.00	11776.00	11940.00	14.77	11643.00	12.14	54.25
le450_25d	450	17425	11676.00	12791.00	13074.00	12.74	12816.00	10.44	54.21
fpsol2.i.2	451	8691	4694.00	**4694.00**	**4694.00**	0.02	**4694.00**	0.03	38.47
4-Insertions_4	475	1795	999.00	**999.00**	1001.00	0.75	**999.00**	0.67	27.18
fpsol2.i.1	496	11654	8364.00	**8364.00**	**8364.00**	0.00	**8364.00**	0.00	28.50
DSJC500.5	500	62624	18333.00	22845.00	22935.00	25.72	21892.00	21.41	152.42
C500.9	500	112332	63147.00	77015.00	71823.00	17.21	74360.00	18.83	276.35
DSJC500.9	500	112437	65373.00	78869.00	72535.00	12.72	74194.00	15.28	290.00
DSJR500.1	500	3555	6253.00	**6253.00**	6646.00	6.87	6724.00	8.33	46.68
DSJR500.1c	500	121275	27395.00	35554.00	27575.00	1.30	27462.00	1.49	149.89
DSJR500.5	500	58862	54392.00	64892.00	56453.00	5.72	58269.00	9.05	191.11
2-Insertions_5	597	3936	2999.00	**2999.00**	**2999.00**	0.13	**2999.00**	0.29	58.76
1-Insertions_6	607	6337	1347.00	1367.00	**1347.00**	0.24	**1347.00**	0.30	63.88
inithx.i.3	621	13969	3633.00	**3633.00**	**3633.00**	0.02	**3633.00**	0.00	96.46
inithx.i.2	645	13979	4073.00	**4073.00**	**4073.00**	0.01	**4073.00**	0.00	95.31
ash331GPIA	662	4185	1513.00	**1513.00**	1537.00	3.19	1539.00	3.45	65.85
4-FullIns_4	690	6650	2443.00	**2443.00**	**2443.00**	0.02	**2443.00**	0.07	82.17
will199GPIA	701	7065	4829.00	5428.00	4919.00	3.07	4948.00	3.87	79.01
inithx.i.1	864	18707	3934.00	**3934.00**	**3934.00**	0.05	**3934.00**	0.01	106.34
qg.order30	900	26100	11940.00	**11940.00**	12084.00	1.38	12027.00	0.94	270.78
latin_sqr_10	900	307350	48822.00	—	57949.00	20.27	56677.00	16.76	620.77
wap05	905	43081	12593.00	14181.00	13803.00	10.17	14039.00	12.31	254.86
wap06	947	43571	18453.00	20434.00	19495.00	6.39	19905.00	8.23	267.59
DSJC1000.5	1000	249826	46790.00	—	58453.00	25.90	57538.00	23.49	920.84
flat1000_50_0	1000	245000	41915.00	—	51719.00	25.22	51032.00	22.90	934.90
flat1000_60_0	1000	245830	40468.00	—	51900.00	28.86	50629.00	26.24	908.07
flat1000_76_0	1000	246708	41729.00	—	51205.00	23.59	50425.00	21.34	920.24
DSJC1000.9	1000	449449	103906.00	—	130627.00	27.59	129308.00	25.12	1869.64
C1000.9	1000	450079	105709.00	—	134237.00	28.22	131064.00	25.47	1874.78
5-FullIns_4	1085	11395	2212.00	**2212.00**	**2212.00**	0.00	**2212.00**	0.01	192.97
ash608GPIA	1216	7844	3859.00	4215.00	**3859.00**	0.42	3866.00	1.32	254.75
3-Insertions_5	1406	9695	1406.00	**1406.00**	**1406.00**	0.00	**1406.00**	0.00	262.40
abb313GPIA	1557	65390	4597.00	—	**4597.00**	0.63	4655.00	3.87	562.43
qg.order40	1600	62400	15280.00	—	15486.00	1.48	15438.00	1.13	1053.18
wap07	1809	103368	13380.00	—	15113.00	13.40	15205.00	14.24	1071.05
wap08	1870	104176	14497.00	—	15790.00	9.30	15855.00	10.43	1106.13
ash958GPIA	1916	12506	2886.00	3171.00	**2886.00**	0.71	2927.00	2.19	606.65
3-FullIns_5	2030	33751	4082.00	5845.00	**4082.00**	0.09	**4082.00**	0.31	872.03
wap01	2368	110871	18719.00	—	20584.00	10.48	20917.00	12.41	1842.69
wap02	2464	111742	17439.00	—	18853.00	8.47	19145.00	10.28	1920.90

Tabela 4.4: Resultados resumidos das heurísticas BRKGA e BRKGA+RVNS.

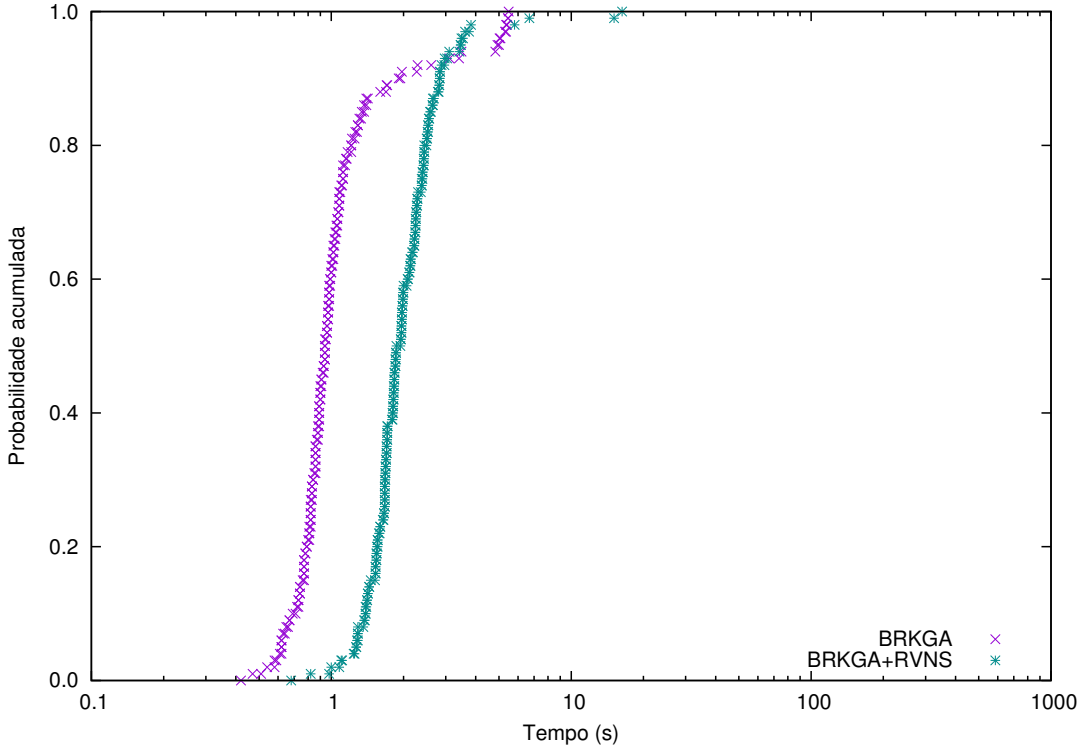


Figura 4.7: TTT-Plots para a instância 3-FullIns_4, com alvo 2951 e tempo máximo de 1000 segundos.

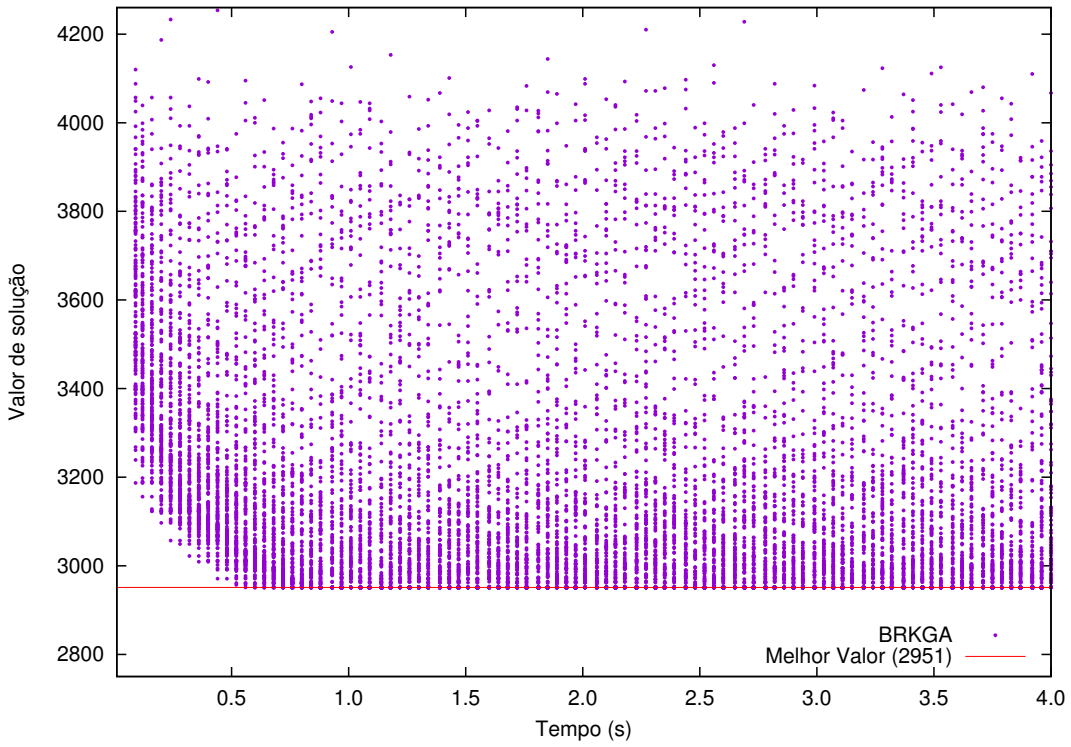


Figura 4.8: Evolução da população do BRKGA para a instância 3-FullIns_4 durante os quatro segundos iniciais de processamento. A heurística encontrou o melhor valor conhecido (2951) nesse período.

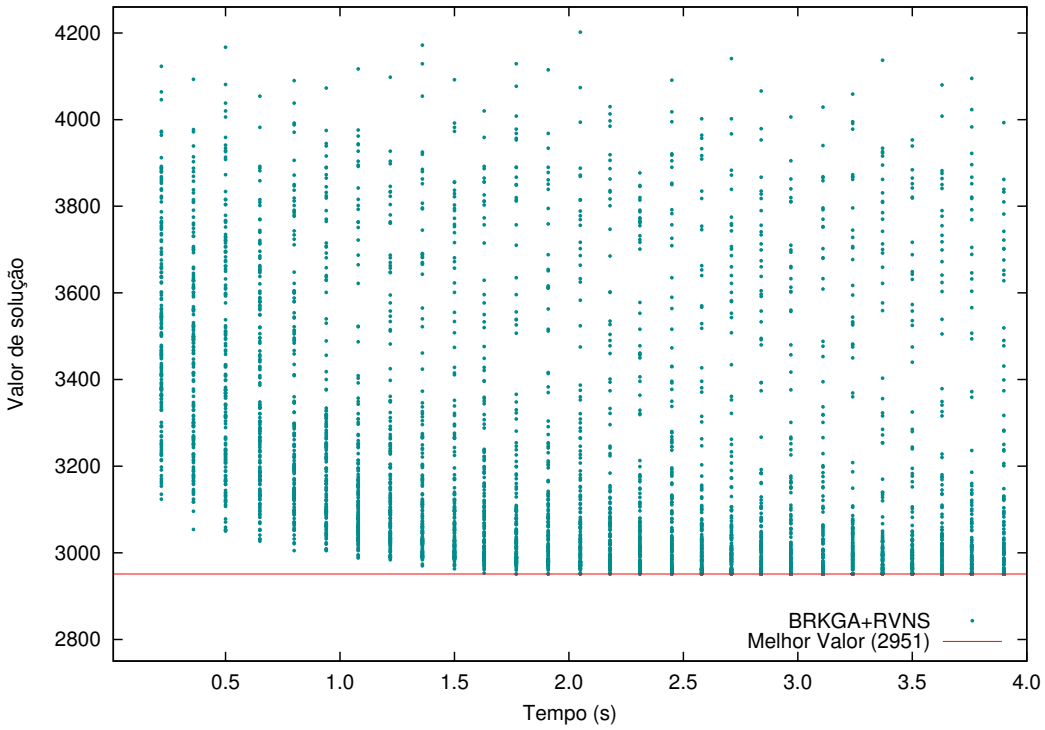


Figura 4.9: Evolução da população do BRKGA+RVNS para a instância 3-FullIns_4 durante os quatro segundos iniciais de processamento. A heurística encontrou o melhor valor conhecido (2951) nesse período.

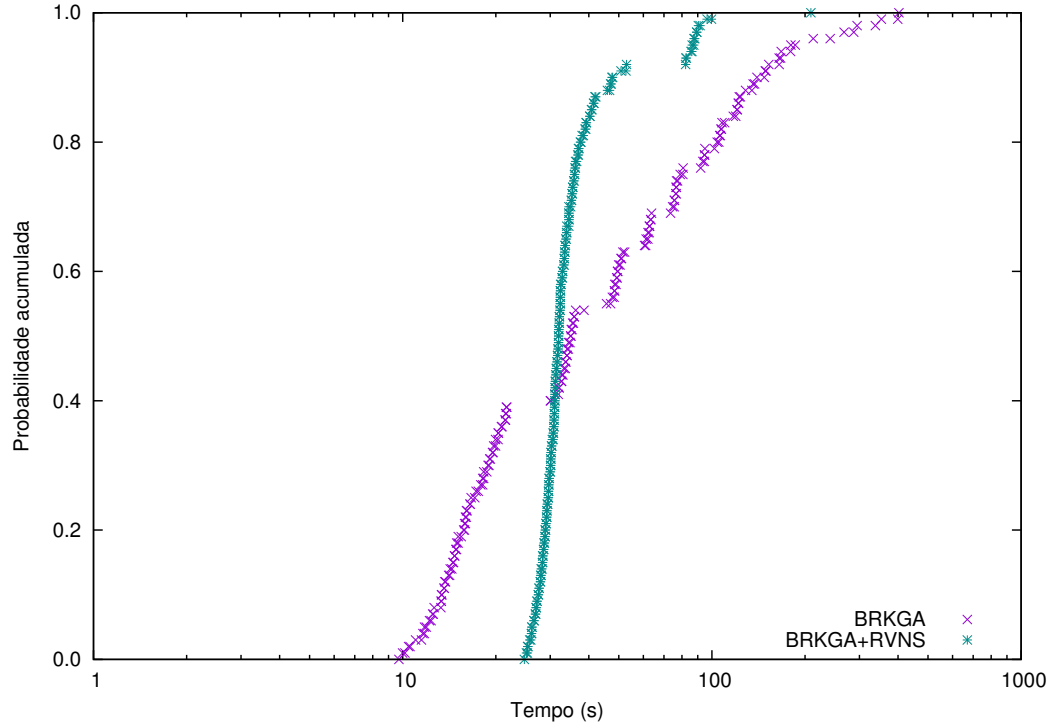


Figura 4.10: TTT-Plots para a instância inithx.i.1, com alvo 3937 e tempo máximo de 1000 segundos.

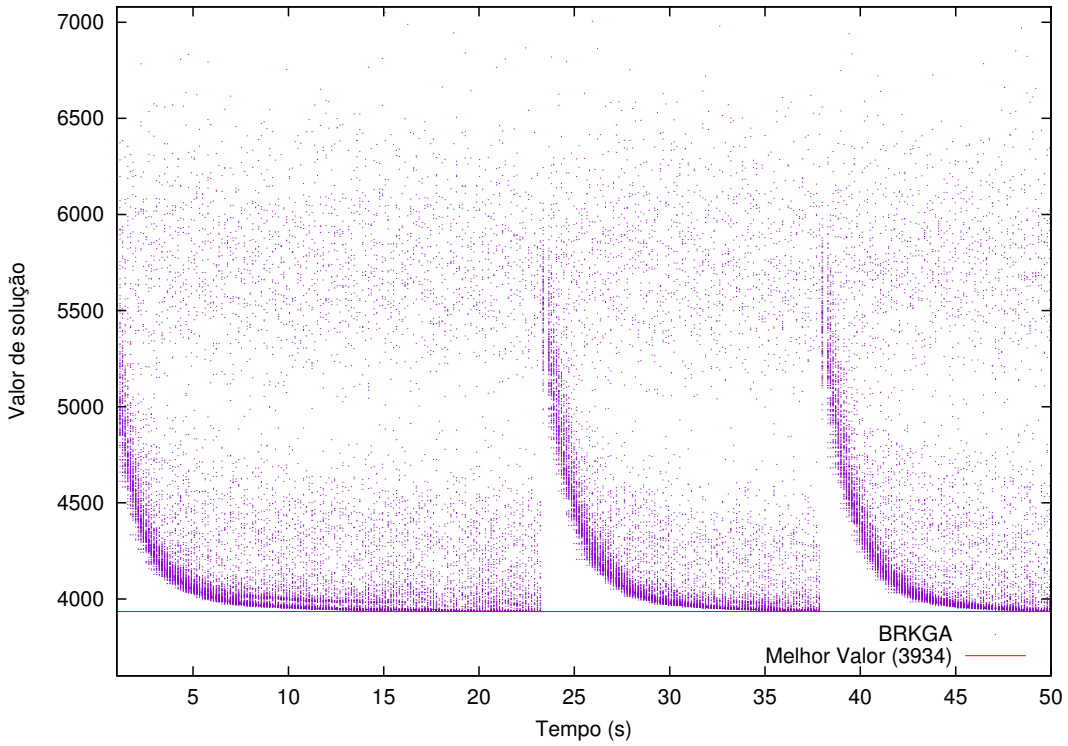


Figura 4.11: Evolução da população do BRKGA para a instância inithx.i.1 durante os 50 segundos iniciais de processamento. A heurística não encontrou o melhor valor conhecido (3934) nesse período, tendo atingido 3935.

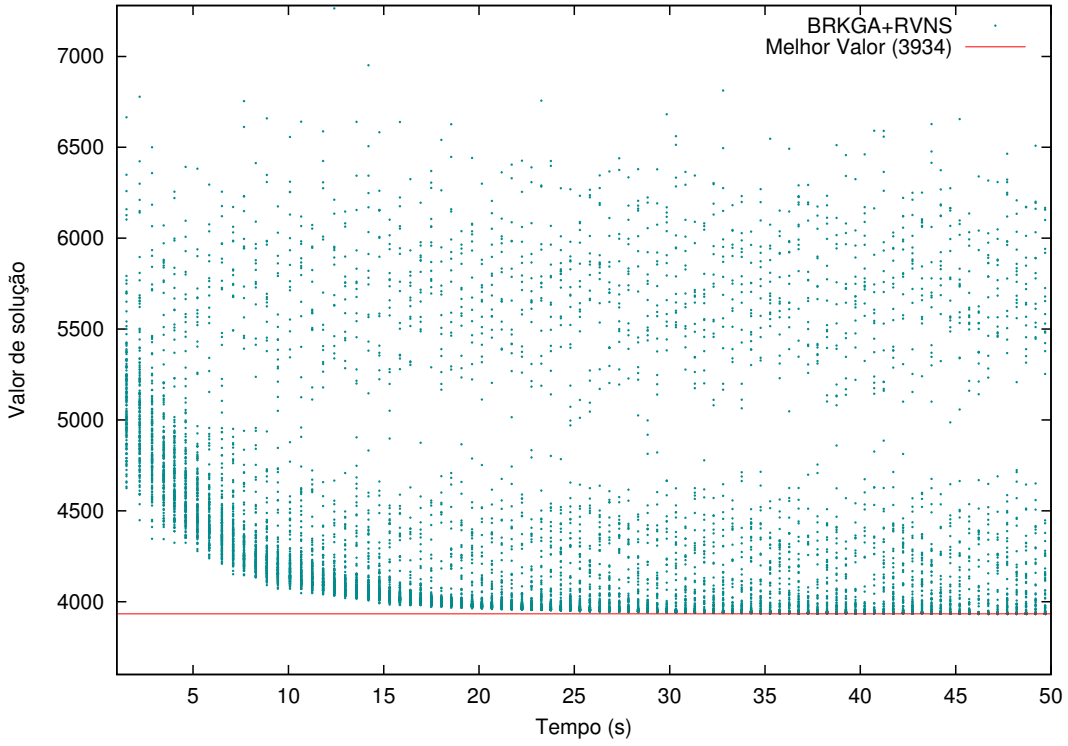


Figura 4.12: Evolução da população do BRKGA+RVNS para a instância inithx.i.1 durante os 50 segundos iniciais de processamento. A heurística encontrou o melhor valor conhecido (3934) nesse período.

Capítulo 5

Heurística de Trajetória com Busca Local e Path-relinking

Neste capítulo serão descritos os componentes da heurística de trajetória com busca local e *path-relinking* desenvolvida para solucionar o PPCCM, seu pseudocódigo, bem como resultados dos experimentos da sua aplicação em um conjunto de instâncias teste e da sua comparação com os algoritmos BRKGAs propostos no capítulo anterior.

5.1 Introdução

A heurística HBLPR utiliza duas estratégias de busca local, uma a fim de tornar viável a solução recebida e outra para tentar uma melhoria na solução corrente, seguidas pela aplicação de um procedimento de *path-relinking*. Uma perturbação na solução corrente é realizada como técnica de diversificação, sendo utilizada uma lista tabu para que essa solução não seja novamente visitada na próxima iteração. Todos os componentes da HBLPR são detalhados nas seções seguintes.

5.2 Função de Avaliação

A função de avaliação utilizada na HBLPR foi obtida a partir de uma modificação nos custos das cores da função proposta por Helmar e Chiarandini [Helmar e Chiarandini, 2011], uma vez que no PPCCM esses valores são reais. Considera-se uma solução, qualquer coloração S (própria ou não) que utiliza k cores e atribui exatamente uma cor para cada vértice. A função que avalia a qualidade da solução S, a qual deve ser minimizada, é dada

por:

$$f(S) \;=\; \sum_{c=1}^{k} (w_c \cdot |C_c| + M \cdot |E(C_c)|), \tag{5.1}$$

onde w_c representa o custo atribuído à cor c ($w_c \in \mathbb{R}$), C_c é o conjunto de vértices com a cor c (também chamado de classe de cor c), $E(C_c)$ conjunto de arestas conflitantes (conectando vértices coloridos com a mesma cor c) e M um valor positivo. O primeiro termo da Função (5.1) calcula o custo da classe de cor c de acordo com o número de vértices coloridos com essa cor e o segundo termo, para garantir viabilidade, penaliza esse valor caso existam arestas conflitantes nessa classe.

5.3 Procedimento para Obter a Solução Inicial

De modo a obter uma solução inicial não totalmente aleatória, mas já com algumas classes de cores, foi empregada a heurística RLF (*Recursive Largest First*) [Leighton, 1979], que implementa uma estratégia similar à de encontrar conjuntos independentes maximais, colorindo os vértices, uma classe de cada vez, seguindo uma estratégia gulosa.

Considerando que o algoritmo RLF sempre retornará uma coloração própria dado um grafo G, na HBLPR foi realizada uma adaptação: quando restarem $\lceil 0.10 \times |V| \rceil$ vértices ainda não coloridos, a execução do RLF é finalizada e tais vértices são atribuídos aleatoriamente às classes de cores anteriormente criadas. Com isso, a solução gerada pode ou não ser uma coloração própria. Essa geração não é completamente gulosa para poder randomizar o algoritmo e o fator de randomização poderia ser de 100%, mas foi decisão de projeto deixá-lo em 10%.

O pseudocódigo do procedimento de geração da solução inicial é apresentado no Algoritmo 3. O número de classes de cores e o conjunto de vértices não coloridos são inicializados nas linhas 1 e 2, respectivamente. O laço externo nas linhas 3–15 é executado enquanto o critério de parada não for atingido. A cada iteração, a próxima classe de cor C_k a ser gerada é inicializada nas linhas 4–5. O vértice ainda não colorido v_0 com o maior número de vértices adjacentes em V' é selecionado na linha 6 e atribuído à classe de cor C_k na linha 7. Todos os vértices não coloridos adjacentes a v_0 são movidos para um conjunto temporário de vértices não coloridos U na linha 8. O laço interno nas linhas 9–13 repete os passos acima para completar a classe de cor C_k, diferenciando apenas no conjunto a partir do qual o vértice v selecionado terá o maior número de vértices adjacentes, que neste caso é o conjunto U. Os vértices não coloridos temporariamente mantidos em U, que não podem ser movidos para a classe C_k, são novamente atribuídos a V' na

linha 14 e uma nova iteração do laço externo é executada. O algoritmo é interrompido quando há $\lceil 0.10 \times |V| \rceil$ vértices sem cor. Neste momento, estes vértices são atribuídos aleatoriamente às classes de cores já criadas na linha 16 e as demais são inicializadas na linha 17. Tipicamente, a solução inicial obtida será uma k-coloração imprópria.

Algoritmo 3: SoluçãoInicial

Entrada: Grafo $G = (V, E)$.
Saída: Coloração $S : \langle C_1, \ldots, C_n \rangle$.

```
1  k ← 0;
2  V' ← V;
3  enquanto |V'| ≥ ⌈0.10 × |V|⌉ faça
4  |   k ← k + 1;
5  |   C_k ← ∅;
6  |   Selecione o vértice v_0 ∈ V' com o maior número de vértices adjacentes em V';
7  |   Mova v_0 de V' para C_k;
8  |   U ← todos os vértices em V' adjacentes a v_0;
9  |   enquanto V' ≠ ∅ faça
10 |   |   Selecione o vértice v ∈ V' com o maior número de vértices adjacentes em U;
11 |   |   Mova v de V' para C_k;
12 |   |   U ← U∪ todos os vértices em V' adjacentes a v;
13 |   fim enquanto
14 |   V' ← U;
15 fim enquanto
16 Atribua aleatoriamente os vértices não coloridos em V' às classes de cores
   C_1, ..., C_k;
17 C_i ← ∅, i = k + 1, ..., n;
```

5.4 Busca Local por uma Coloração Própria

Considere S uma k-coloração imprópria, i a classe de cor do vértice $v \in V$ $(S(v) = i)$, j uma classe de cor diferente de i e $N_j(v)$ os vértices adjacentes a v na classe de cor j. Na estrutura de vizinhança utilizada por essa fase da heurística HBLPR, denominada *Critical One-Move Neighborhood*, aplicada em problemas de coloração [Hertz e de Werra, 1987, Avanthay et al., 2003, Chiarandini et al., 2007, Bouziri e Jouini, 2010], troca-se em S a cor do vértice v de i para j, sendo $1 \leqslant j \leqslant k+1$, $i \neq j$, $|N_j(v)| = 0$ e v um vértice conflitante (adjacente a outro(s) vértice(s) na mesma classe de cor). O vértice v escolhido para a troca é o que promove a maior redução no valor da função de avaliação de S. Além disso, o par $\langle v, j \rangle$ não pode estar classificado como movimento tabu, oriundo da Fase de Perturbação, que será descrita na Seção 5.8. No entanto, esse movimento é permitido se ele conduzir a uma solução vizinha melhor do que a melhor solução encontrada até o

momento.

Um vértice conflitante v é movido a cada vez que esse procedimento é empregado, sendo ele executado até que uma coloração própria seja alcançada, o que pode eventualmente aumentar o número de cores utilizado por S, pois v sempre será movido para uma classe de cor que não contenha vértices adjacentes a ele. Um exemplo de movimento na vizinhança *Critical One-Move Neighborhood* pode ser visto na Figura 5.1, onde o vértice v_7 é trocado da classe de cor C_i para C_j, uma vez que ele proporciona, entre os vértices conflitantes, maior redução no valor de avaliação da solução, considerando que $w_{C_j} \leq w_{C_i}$ e $\langle v, C_j \rangle$ não é tabu.

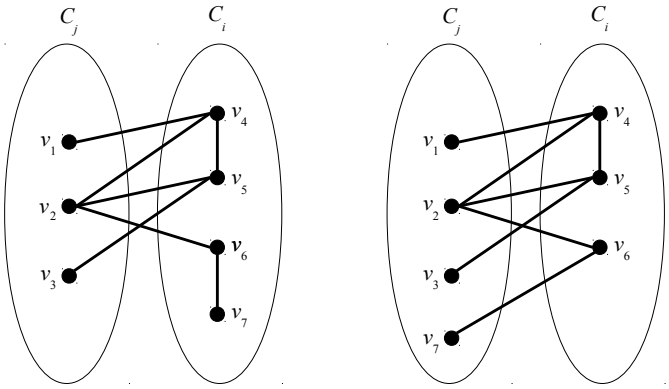


Figura 5.1: Exemplo de movimento na estrutura de vizinhança *Critical One-Move Neighborhood*

O custo de uma solução vizinha S', considerando a troca do vértice v da classe de cor i para a j, é determinado pela seguinte função:

$$f(S') \;=\; f(S) - (w_i + M \cdot |N_i(v)|) + w_j, \tag{5.2}$$

onde $f(S)$ é o custo da solução S, w_i é o custo da classe de cor i, M um valor positivo, $N_i(v)$ é o conjunto de vértices adjacentes a v na classe de cor i e w_j é o custo da classe de cor j. Esse cálculo retira do valor de S o custo da classe de cor atual de v, juntamente com a penalidade gerada por suas arestas conflitantes, e adiciona o custo da classe de cor j, uma vez que o mesmo será movido para uma classe que não contenha vértices adjacentes a ele ($|N_j(v)| = 0$), anulando assim o fator de penalização para essa classe.

O Algoritmo 4 apresenta o pseudocódigo do procedimento de busca por uma coloração própria, que investiga cada vértice em conflito na solução corrente imprópria S e seleciona o melhor movimento na vizinhança *Critical One-Move Neighborhood*. A solução S^* denota a melhor solução própria já encontrada pelo algoritmo. S_{melhor} é a melhor solução vizinha de S e é inicializada na linha 1. O laço externo nas linhas 2–11 investiga os movimentos

originados por cada vértice em conflito na vizinhança *Critical One-Move Neighborhood*. Para cada vértice em conflito v investigado, a solução corrente S é temporariamente copiada para a solução vizinha S' na linha 3 e a classe de cor i do vértice v em S é salva na linha 4. No laço interno 5–10, os movimentos do vértice v para cada classe de cor j, $j = 1, \ldots, k+1$: $j \neq i$, na qual seus vértices não estejam envolvidos em conflitos com v, são avaliadas. O vértice v é temporariamente movido para a classe de cor j em S' na linha 6. Se (a) o movimento do vértice v para a classe de cor j não é proibido e a solução S' é melhor do que a melhor solução vizinha já encontrada na vizinhança de v ou (b) a solução S' é estritamente melhor do que a melhor solução encontrada S^*, então a melhor vizinha S_{melhor} é atualizada na linha 8. O laço externo prossegue até que todos os vértices em conflito sejam examinados uma vez. O procedimento retorna a melhor solução vizinha S_{melhor} com no mínimo um conflito a menos do que a solução inicial S.

Algoritmo 4: BuscaColoraçãoPrópria

Entrada: k-coloração S e a melhor solução S^*.
Saída: Coloração S_{melhor}.

```
1  S_melhor ← S;
2  para cada vértice conflitante v ∈ V faça
3  |  S' ← S;
4  |  i ← S(v);
5  |  para j = 1, ..., k+1 : |N_j(v)| = 0, j ≠ i faça
6  |  |  S'(v) ← j;
7  |  |  se (⟨v, j⟩ não é um movimento tabu e f(S') < f(S_melhor)) ou f(S') < f(S*)
   |  |  então
8  |  |  |  S_melhor ← S';
9  |  |  fim se
10 |  fim para
11 fim para cada
```

5.5 Busca Local para Melhoria da Solução

Após uma coloração própria S ser encontrada pela aplicação do movimento de vizinhança *Critical One-Move Neighborhood*, ocorre a Fase de Melhoria em S, que realoca os vértices à menor classe de cor possível na tentativa de reduzir o custo de S, de modo que a coloração permaneça própria. Essa fase segue a seguinte estratégia [Helmar e Chiarandini, 2011]: as cores atribuídas aos conjuntos de vértices criados são rearranjadas, de maneira que a cor de menor custo é atribuída ao conjunto de vértices de maior cardinalidade e, seguindo a ordem não-crescente de cardinalidade dos conjuntos, os demais

vértices são reposicionados nessa classe ou em uma nova, de modo que não sejam criadas arestas conflitantes.

A fim de exemplificação, considere uma coloração própria S_1 resultante da aplicação da vizinhança *Critical One-Move Neighborhood* (Figura 5.2 (a)), onde $w_1 = 1.2$, $w_2 = 2.4$, $w_3 = 3.8$ e $w_4 = 6.3$, representam, respectivamente, os custos das classes de cores C_i, $i = 1, \ldots, 4$, resultando em $f(S_1) = 22.5$, de acordo com a Função (5.1). Seguindo a estratégia de melhoria, a cor de menor custo ($i = 1$) é atribuída ao conjunto de vértices de maior cardinalidade (v_4, v_6 e v_7). Em seguida, para cada vértice do segundo conjunto de maior cardinalidade (v_1 e v_3), ocorre a tentativa de reposicioná-lo na classe anteriormente criada, porém ambos geram arestas conflitantes (v_1 é adjacente a v_4 e v_6, e v_3 é adjacente a v_4), fazendo com que os mesmos recebam a cor com segundo menor custo ($i = 2$). Os demais conjuntos de vértices têm a mesma cardinalidade, o que torna indiferente por qual vértice iniciar a tentativa de realocação (v_5 ou v_2). Desse modo, procura-se realocar o vértice v_5 nas classes ora criadas C_1 e C_2. No entanto, em ambas ele origina arestas conflitantes, ficando o mesmo na classe de cor C_3, que tem custo $w_3 = 3.8$. Na tentativa de realocar o vértice v_2 nas classes já estabelecidas, percebe-se que arestas conflitantes são geradas nas classes C_1 e C_2, mas não na classe C_3, uma vez que ele não é adjacente ao vértice v_5. Assim, ele é inserido nesta última classe. Com isso, essa nova solução S_2, apresentada na Figura 5.2 (b), tem valor $f(S_2) = 16$. Essa Fase de Melhoria retornará a melhor solução (menor valor de função de avaliação) entre a solução gerada por ela e a que é recebida como entrada.

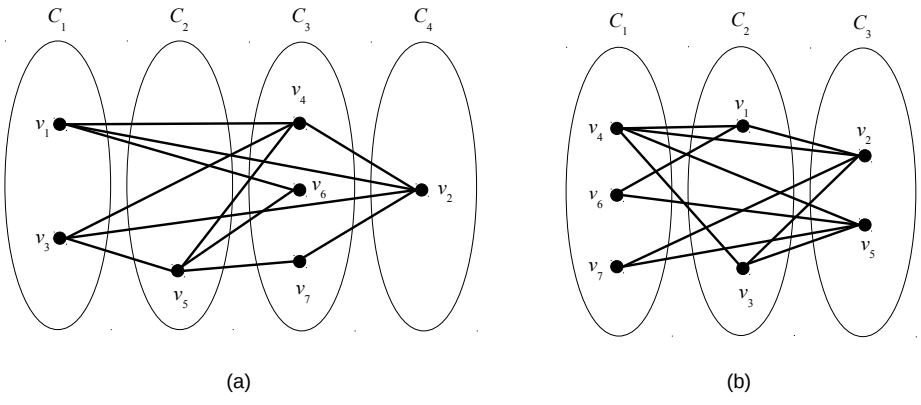


Figura 5.2: (a) Solução S_1 recebida como entrada com $f(S_1) = 22.5$; (b) Solução S_2 gerada pela fase de melhoria com $f(S_2) = 16$.

O Algoritmo 5 descreve em detalhes o procedimento de melhoria da solução, que inicia a partir de uma k-coloração própria S. C_i e C'_i denotam as classes de cores das colorações S e S' associadas às cores $i = 1, \ldots, n$, respectivamente. As classes de cores $C_1, \ldots, C_k$ da solução S são indexadas na ordem não-crescente pelas suas cardinalidades na linha 1. O

laço nas linhas 2–4 cria classes de cores inicialmente vazias $C'_1, \ldots, C'_n$ da nova solução S' indexadas na ordem não-decrescente pelos custos das cores. Os índices i e j são utilizados para visitar todas as classes de cores da solução inicial e da nova solução, respectivamente, S e S'. A classe de cor de maior cardinalidade da solução S é copiada para a classe de cor de menor custo da solução S' nas linhas 5 e 6. O laço nas linhas 7–24 realoca os vértices de cada classe de cor restante C_i, $i = 2, \ldots, C_k$, da solução S. O laço nas linhas 8–23 é executado para cada vértice $v \in C_i$. O laço interno 10–16 determina a classe de cor de menor custo $C'_\ell \in S'$ sem qualquer conflito com o vértice v. Se nenhuma classe for encontrada, uma nova C'_{j+1} formada inicialmente pelo vértice v é criada nas linhas 18–19. Caso contrário, v é movido para a classe de cor C'_ℓ na linha 21. Finalizada a análise de todas as classes de cor da solução S, se a nova solução S' for melhor do que S, então S é atualizada nas linhas 25 e 26.

5.6 Procedimento de Path-relinking

A técnica de *path-relinking* foi originalmente proposta por Glover [Glover, 1997] como uma estratégia de intensificação que explora trajetórias que ligam soluções de boa qualidade no espaço de busca, introduzindo características de uma ou mais soluções guia à solução inicial [Glover et al., 2003]. O emprego bem sucedido dessa técnica pode ser verificado em [Bastos e Ribeiro, 2002], [Ho e Gendreau, 2006], [Lai e Hao, 2015] e [Resende e Ribeiro, 2016], entre outros.

Galinier e Hao [Galinier e Hao, 1999] introduziram uma nova medida para determinar a distância entre duas colorações, que interpreta as soluções como partições de vértices. A distância entre elas corresponde ao número de transformações elementares necessárias para transformar uma solução em outra. Como cada transformação elementar significa mover um vértice de uma classe de cor para outra, Hamiez e Hao [Hamiez e Hao, 2002] chamaram essa medida de *Move Distance* (*MD*). Na heurística HBLPR, o valor *MD* entre duas soluções é utilizado para determinar se o *path-relinking* será executado. Seguindo Ribeiro e Resende [Ribeiro e Resende, 2012], essa execução ocorrerá somente se a distância *MD* entre a solução inicial (S^{inicial}) e a solução guia (S^{guia}) for maior ou igual a quatro, o que garante uma busca por soluções melhores do que ambas.

O algoritmo para calcular a distância *Move Distance* entre duas colorações próprias é descrito na Seção 5.6.1. Este algoritmo será utilizado na Seção 5.6.2 como parte do procedimento empregado para executar o *path-relinking* entre duas colorações próprias.

Algoritmo 5: MelhoriaSolução

Entrada: k-coloração $S : \langle C_1, \ldots, C_n \rangle$.
Saída: Coloração S.

```
1  Ordene as classes de cores da solução S pelas suas cardinalidades: |C_i| ≥ |C_{i+1}|,
   1 ≤ i < k;
2  para i = 1, ..., n faça
3  |  C'_i ← ∅;
4  fim para
5  j ← 1; i ← 1;
6  C'_j ← C_i;
7  para i = 2, ..., k faça
8  |  para cada v ∈ C_i faça
9  |  |  ℓ ← 1; encontrado ← falso;
10 |  |  enquanto ℓ ≤ j e encontrado = falso faça
11 |  |  |  se v é adjacente a qualquer vértice em C'_ℓ então
12 |  |  |  |  ℓ ← ℓ + 1;
13 |  |  |  senão
14 |  |  |  |  encontrado ← verdadeiro;
15 |  |  |  fim se
16 |  |  fim enquanto
17 |  |  se encontrado = falso então
18 |  |  |  j ← j + 1;
19 |  |  |  C'_j ← v;
20 |  |  senão
21 |  |  |  C'_ℓ ← C'_ℓ ∪ {v};
22 |  |  fim se
23 |  fim para cada
24 fim para
25 se f(S') < f(S) então
26 |  S ← S';
27 fim se
```

5.6.1 Move Distance

Considerando duas colorações próprias S^{inicial} e S^{guia}, a distância *Move Distance* (MD) entre elas corresponde ao número de vértices que devem ser movidos de uma classe de cor de S^{inicial} para outra até que seja obtida uma solução com os vértices organizados como em S^{guia} [Galinier e Hao, 1999, Hamiez e Hao, 2002].

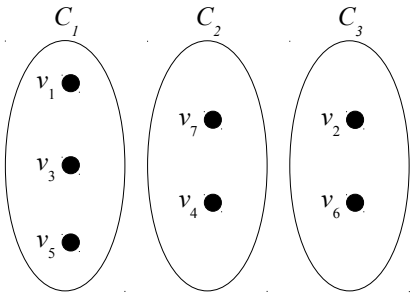
O exemplo da Figura 5.3 ilustra o cálculo da distância *Move Distance* entre S^{inicial} na Figura 5.3 (a) e S^{guia} na Figura 5.3 (b). A solução inicial é estabelecida como $S' = S^{\text{inicial}}$ na Figura 5.3 (c), com $C_1 = \{v_1, v_3, v_5\}$, $C_2 = \{v_4, v_7\}$ e $C_3 = \{v_2, v_6\}$. As classes de cores de S^{guia} são examinadas em ordem não-crescente de suas cardinalidades: $C_2^{\text{guia}} = \{v_3, v_5, v_7\}$ é a primeira a ser examinada. Uma vez que os vértices v_3 e v_5 já estão na classe

C_1 de S', mas o vértice v_7 não, v_7 deverá ser movido para C_1, gerando a primeira solução modificada S' na Figura 5.3 (d) (o movimento do vértice v_7 é a primeira transformação elementar). Uma vez que as duas classes restantes de S^{guia} têm a mesma cardinalidade, pode-se prosseguir a partir de qualquer uma delas. Seja $C_1^{\text{guia}} = \{v_1, v_4\}$ a próxima classe a ser examinada. Vértices v_1 e v_4 estão em classes distintas da solução S' na Figura 5.3 (d). O vértice v_4 não pode ser movido para a mesma classe de v_1, porque os vértices v_3, v_5 e v_7 já estão definitivamente posicionados nesta classe. No entanto, uma vez que não há outro vértice na segunda classe C_2 de S' exceto v_4, o vértice v_1 pode ser movido para C_2, resultando em uma nova segunda solução modificada S', mostrada na Figura 5.3 (e) (o movimento do vértice v_1 é a segunda transformação elementar). A última classe de S^{guia} a ser examinada é $C_3^{\text{guia}} = \{v_2, v_6\}$. Uma vez que todos os seus vértices estão na classe C_3 da solução S', eles não tem que ser movidos. A Figura 5.3 (f) ilustra a solução final S', onde os vértices estão organizados como em S^{guia}.

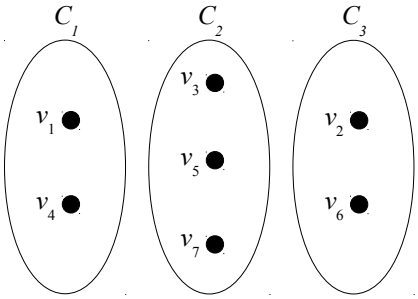
O Algoritmo 6 apresenta o pseudocódigo do procedimento que calcula a distância *Move Distance* entre S^{inicial} e S^{guia}. Denota-se por C_i^{inicial} e C_i^{guia} as classes de cores das soluções S^{inicial} e S^{guia}, $i = 1, \ldots, n$. A solução inicial $S' = S^{\text{inicial}}$ é estabelecida na linha 1 e será progressivamente modificada pelas transformações elementares, até que todos os vértices estejam organizados como em S^{guia}. O conjunto A é inicializado na linha 2 e contém os vértices que estão definitivamente organizados como em S^{guia}. O conjunto Δ é inicializado na linha 3 e contém os vértices que serão movidos de uma classe para outra. As classes de cores em S^{guia} são indexadas em ordem não-crescente de suas cardinalidades na linha 4. Cada uma delas será tratada em uma iteração do laço externo nas linhas 5–27. A cada iteração, a maior classe de cor ainda não tratada C_i^{guia} guiará os movimentos dos vértices que ainda estão em suas posições incorretas em S'. Uma partição P_j dos vértices em C_i^{guia} é criada na linha 6, com cada conjunto P_j contendo os vértices $C_j \cap C_i^{\text{guia}}$, $C_j \in S'$. Os conjuntos P_j são indexados em ordem não-crescente de suas cardinalidades na linha 7. Um indicador de movimento é fixado como `falso` na linha 8. Cada conjunto P_j, $j = 1, \ldots, n$, é analisado no laço interno nas linhas 9–18. Na linha 10, é verificado se os vértices em $C_i^{\text{guia}} \setminus P_j$ podem ser movidos para a classe de cor $C_j \in S'$. Isto será possível se (a) os vértices em P_j e C_j coincidem ou (b) os vértices em $C_j \setminus P_j$ ainda não estão definitivamente organizados como em S^{guia}. Sendo possível a movimentação, o indicador de movimento é atualizado na linha 11 e todos os vértices em $C_i^{\text{guia}} \setminus P_j$ são movidos para a classe C_j na solução S' nas linhas 12–14 e inseridos no conjunto Δ de vértices movimentados na linha 15. Todos os vértices em C_i^{guia} são inseridos no conjunto A na linha 16, uma vez que todos eles agora estão organizados em S' como em S^{guia}. Se

nenhum movimento possível é encontrado, os vértices em C_i^{guia} são movidos para uma classe vazia nas linhas 19–26. Após todas as classes de cores de S^{guia} serem examinadas, a distância MD é determinada na linha 28.

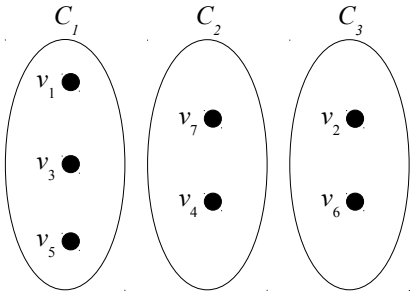
Observou-se que a distância *Move Distance* ($MD = |\Delta|$) calculada pelo Algoritmo 6 apresenta, na verdade, uma aproximação no número de transformações elementares que são necessárias para transformar S^{inicial} em S^{guia}. Isto acontece porque o procedimento calcula o número de vértices que têm que ser movidos entre as classes de cores de S^{inicial} até os vértices do grafo serem organizados como em S^{guia}, mas não necessariamente nas mesmas classes de cores.



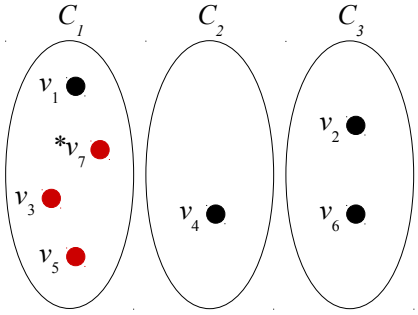
(a) S^{inicial}



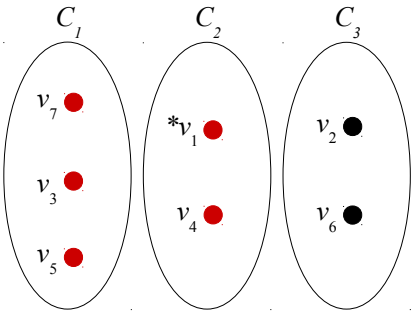
(b) S^{guia}



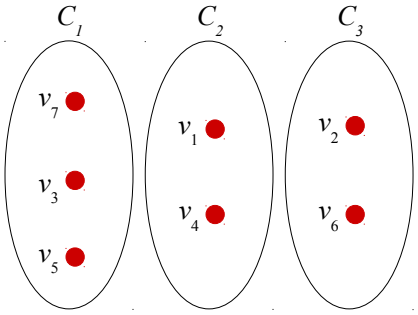
(c) Início dos cálculos $S' = S^{\text{inicial}}$



(d) 1ª solução modificada S': C_2^{guia} examinada e v_7 movido para C_1 em S'



(e) 2ª solução modificada S': C_1^{guia} examinada e v_1 movido para C_2 em S'



(f) Solução final S' com as mesmas classes de S^{guia}: C_3^{guia} examinada, sem necessidade de movimentos

Figura 5.3: Soluções S^{inicial} e S^{guia} com $MD = 2$ (vértices v_1 e v_7 são movidos).

Algoritmo 6: MoveDistance

Entrada: Coloração S^{inicial} e coloração S^{guia}.
Saída: Coloração S', conjunto Δ de vértices movidos e *Move Distance* (MD).

$S' : \langle C_1, \ldots, C_n \rangle \leftarrow S^{\text{inicial}}$;
$A \leftarrow \emptyset$;
$\Delta \leftarrow \emptyset$;
Ordene as classes de cores da solução S^{guia} pelas suas cardinalidades: $|C_i^{\text{guia}}| \geq |C_{i+1}^{\text{guia}}|$, $1 \leq i < n$;
para $i = 1, \ldots, n : C_i^{\text{guia}} \neq \emptyset$ **faça**
    Crie uma partição $P_j = C_j \cap C_i^{\text{guia}}, j = 1, \ldots, n$, dos vértices em V_i^{guia};
    Ordene os conjuntos P_j pelas suas cardinalidades: $|P_j| \geq |P_{j+1}|, 1 \leq j < n$;
    $move \leftarrow$ `falso`;
    para $j = 1, \ldots, n : P_j \neq \emptyset$ ***e*** $move =$ `falso` **faça**
        se $P_j = C_j$ ***ou*** $(C_j \setminus P_j) \cap A = \emptyset$ **então**
            $move \leftarrow$ `verdadeiro`;
            para cada $v \in C_i^{\text{guia}} \setminus P_j$ **faça**
                $S'(v) \leftarrow j$;
            fim para cada
            $\Delta \leftarrow \Delta \cup C_i^{\text{guia}} \setminus P_j$;
            $A \leftarrow A \cup C_i^{\text{guia}}$;
        fim se
    fim para
    se $move =$ `falso` **então**
        $\ell \leftarrow \text{argmin}\{w_j, j = 1, \ldots, n : C_j = \emptyset\}$;
        para cada $v \in C_i^{\text{guia}}$ **faça**
            $S'(v) \leftarrow \ell$;
        fim para cada
        $\Delta \leftarrow \Delta \cup C_i^{\text{guia}}$;
        $A \leftarrow A \cup C_i^{\text{guia}}$;
    fim se
fim para
$MD \leftarrow |\Delta|$;

5.6.2 Path-relinking

O procedimento de *path-relinking* entre duas soluções na HBLPR será sempre precedido pelo cálculo da distância *Move Distance* entre elas pelo Algoritmo 6, que também retorna uma solução S' com os vértices organizados como em S^{guia} e o conjunto Δ dos vértices para serem movidos entre as classes de cores até que S' seja obtida. Importante observar que o *path-relinking* proposto apresenta alteração na sua concepção a fim de eliminar as simetrias naturalmente geradas nos problemas de coloração.

O Algoritmo 7 descreve o pseudocódigo do procedimento de *path-relinking*. A solução

corrente S^{path} e a melhor solução $\overline{S}$ visitada pelo *path-relinking* são inicializadas com S^{inicial} nas linhas 1 e 2, respectivamente. O laço externo nas linhas 3–18 é executado enquanto a solução final S' não é alcançada, ou seja, enquanto o conjunto de movimentos Δ não está vazio. O melhor vértice para ser movido é determinado nas linhas 4–12. O laço interno nas linhas 5–12 investiga todos os possíveis movimentos. Na linha 7, cada vértice disponível v é provisoriamente movido para a mesma classe de cor que ele ocupa na solução final S'. O melhor movimento é atualizado nas linhas 8–11. Uma vez que o melhor vértice v^* para ser movido nesta iteração foi determinado, a solução corrente S^{path} é atualizado na linha 13 e o conjunto de movimentos é atualizado na linha 14. Se a nova solução corrente melhora a melhor solução $\overline{S}$ encontrada pelo *path-relinking*, então $\overline{S}$ é atualizado nas linhas 15 e 16.

Algoritmo 7: PathRelinking

Entrada: Coloração S^{inicial}, coloração S^{guia}, coloração S' e o conjunto Δ de movimentos.
Saída: Coloração $\overline{S}$.

```
1  S^path ← S^inicial;
2  S̄ ← S^inicial;
3  enquanto Δ ≠ ∅ faça
4      f_min ← ∞;
5      para cada v ∈ Δ faça
6          S^temp ← S^path;
7          S^temp(v) ← S'(v);
8          se f(S^temp) < f_min então
9              f_min ← f(S^temp);
10             v* ← v;
11         fim se
12     fim para cada
13     S^path(v*) ← S'(v*);
14     Δ ← Δ \ {v*};
15     se f(S^path) < f(S̄) então
16         S̄ ← S^path;
17     fim se
18 fim enquanto
```

5.7 Procedimento de Atualização da População Elite

A heurística HBLPR faz uso de uma população com um número fixo de soluções de boa qualidade, chamada população elite. Essa população é preenchida com as soluções próprias encontradas enquanto seu número máximo de soluções possível não for atingido,

passando a ser atualizada a partir do momento que este valor é alcançado. Para isso, a solução corrente (S) substitui a pior solução (de maior custo) da população se S não for similar a qualquer solução da população e caso tenha custo menor do que a pior solução, sendo duas soluções consideradas estruturalmente similares se a *Move Distance* entre elas for menor do que $\lfloor 0.10 \times |V| \rfloor$, seguindo Lai et al. [Lai et al., 2014].

O pseudocódigo para o procedimento de atualização da população elite é apresentado no Algoritmo 8. A solução elite de maior custo S_{maior} é determinada na linha 1. Um indicador de similaridade é inicializado com `falso` na linha 2. O laço nas linhas 3–8 determina se a solução S é similar a, no mínimo, uma solução elite S_{elite}, caso em que é atribuído `verdadeiro` ao indicador na linha 6. Se S não é similar a qualquer solução $S_{\text{elite}} \in \mathcal{E}$ e é melhor do que S_{maior}, a população elite é atualizada na linha 10.

Algoritmo 8: AtualizaçãoPopulaçãoElite

Entrada: Coloração S e a população elite $\mathcal{E}$.
Saída: População elite $\mathcal{E}$.

```
1  Seja S_maior a solução de maior custo em E;
2  similar ← falso;
3  para cada S_elite ∈ E e similar = falso faça
4  |  MoveDistance(S, S_elite, S', Δ, MD);
5  |  se MD ≤ ⌊0.10 × |V|⌋ então
6  |  |  similar ← verdadeiro;
7  |  fim se
8  fim para cada
9  se similar = falso e f(S) < f(S_maior) então
10 |  E ← (E \ S_maior) ∪ S;
11 fim se
```

5.8 Procedimento de Perturbação

A estratégia de diversificação utilizada pela heurística HBLPR, com a intenção de explorar novas regiões do espaço de soluções, foi realizar uma técnica de perturbação na solução corrente S. Essa técnica consiste em selecionar aleatoriamente, a partir de uma k-coloração própria, um determinado número de vértices do grafo e realocá-los, também de maneira aleatória, em $k+1$ classes de cor, se $k < |V|$, ou em k classes, caso contrário, sendo a nova classe de cada vértice distinta da atual e k o número de classes utilizadas por S. Com isso, a solução resultante pode ser uma coloração própria ou imprópria.

Além disso, para auxiliar a exploração desse espaço evitando a repetição de S na próxima iteração, foi utilizada uma lista tabu de tamanho fixo, onde os vértices realocados,

juntamente com as suas cores anteriores à perturbação, são inseridos. Dessa forma, na fase de busca local para obtenção da uma coloração própria (Seção 5.4) da iteração seguinte, tais vértices não podem retornar às suas cores se estiverem presentes na lista, exceto se esses movimentos produzirem uma solução melhor do que a melhor encontrada até então. Os vértices são inseridos na lista somente neste procedimento de perturbação, sendo ela apenas consultada na fase de busca por uma coloração própria e esvaziada ao fim desta.

O Algoritmo 9 descreve em detalhes o procedimento de perturbação. O laço nas linhas 1–13 aplica perturbações a exatamente *numPerturbacoes* vértices da k-coloração própria inicial. Na linha 2 um vértice v ainda não perturbado é selecionado. O índice da classe de cor atual i do vértice v é armazenado na linha 3. O laço interno nas linhas 4–10 é repetido até que uma nova classe de cor j, distinta de i, seja determinada e atribuída ao vértice v na linha 11. O movimento associado a atribuir novamente a classe de cor i ao vértice v é inserido na lista tabu na linha 12.

Algoritmo 9: Perturbação

Entrada: k-coloração S e o número de perturbações *numPerturbacoes*.
Saída: Coloração S.

```
1  para z = 1, ..., numPerturbacoes faça
2  |  v ← aleatório[1, |V|], v ainda não selecionado;
3  |  i ← S(v);
4  |  repita
5  |  |  se k < |V| então
6  |  |  |  j ← aleatório[1, k + 1];
7  |  |  senão
8  |  |  |  j ← aleatório[1, k];
9  |  |  fim se
10 |  até i ≠ j ;
11 |  S(v) ← j;
12 |  Insira na lista tabu ⟨v, i⟩;
13 fim para
```

5.9 Pseudocódigo da Heurística HBLPR

O Algoritmo 10 descreve em detalhes a heurística HBLPR. Na linha 1, o algoritmo ordena as cores de maneira não-decrescente pelos seus custos. Uma solução inicial S é gerada na linha 2 pelo Algoritmo 3 – $SoluçãoInicial(G, S)$ – descrito na Seção 5.3. A melhor solução S^* e seu custo $f(S^*)$ são inicializados na linha 3. O contador de soluções elite e a população elite são inicializados na linha 4, sendo ambos atualizados na linha 6 caso a solução inicial S seja uma coloração própria. O laço externo nas linhas 8–34

é executado enquanto um critério de parada não é satisfeito. Se a solução corrente S é uma coloração própria, o laço nas linhas 9–12 é ignorado. Caso contrário, o Algoritmo 4 – $BuscaColoraçãoPróprias(S, S^*, S_{\text{melhor}})$ – apresentado na Seção 5.4 é aplicado na linha 10 para reduzir a inviabilidade da solução corrente S, que é substituída pela sua solução vizinha S_{melhor} na linha 11 até se tornar viável. A lista tabu com os movimentos proibidos é esvaziada na linha 13. O Algoritmo 5 – $MelhoriaSolução(S)$ – apresentado na Seção 5.5 é aplicado à coloração própria corrente S na linha 14 na tentativa de melhorá-la.

Na linha 15 é verificado se a população elite $\mathcal{E}$ está completa. Caso não esteja, o contador de soluções elite é atualizado e a solução corrente S é simplesmente inserida na população na linha 16. Caso contrário, a solução S é utilizada na linha 18 para atualizar a população $\mathcal{E}$ empregando o Algoritmo 8 – $AtualizaçãoPopulaçãoElite(S, \mathcal{E})$ – apresentado na Seção 5.7. O procedimento de *path-relinking* é aplicado somente se a população elite apresentar um mínimo de cinco soluções. Neste caso, uma solução elite S_e é aleatoriamente selecionada da população na linha 21. A solução S^{inicial} é determinada como a melhor entre S e S_e na linha 22 e S^{guia} como a outra. O Algoritmo 6 – $MoveDistance(S^{\text{inicial}}, S^{\text{guia}}, S', \Delta, MD)$ – descrito na Seção 5.6.1 é aplicado na linha 23 para calcular a distância MD entre S^{inicial} e S^{guia}, assim como a solução S' com o conjunto de vértices organizados como em S^{guia} e o conjunto Δ de vértices que serão efetivamente movidos de uma classe para outra. Resende e Ribeiro [Resende e Ribeiro, 2016] mostraram que o *path-relinking* deve ser aplicado entre duas soluções somente se o número de transformações elementares entre elas é maior ou igual a quatro. Portanto, na linha 24 é descartada a aplicação do *path-relinking* se a distância MD entre as soluções S^{inicial} e S^{guia} for menor do que quatro. Caso contrário, o Algoritmo 7 – $PathRelinking(S^{\text{inicial}}, S^{\text{guia}}, S', \Delta, \overline{S})$ – apresentado na Seção 5.6.2 é aplicado na linha 25 e a solução corrente S é atualizada na linha 26 com a solução $\overline{S}$ obtida pelo *path-relinking* e, novamente, submetida ao Algoritmo 5 – $MelhoriaSolução(S)$ – na linha 27. A melhor solução S^* é atualizada nas linhas 30–31 e na linha 33 o Algoritmo 9 – $Perturbação(S, numPerturbacoes)$ – descrito na Seção 5.8 é aplicado para gerar uma solução perturbada para a iteração seguinte.

5.10 Experimentos Computacionais

Para a heurística HBLPR proposta, foram realizados experimentos computacionais para o ajuste de parâmetros, bem como para avaliar a qualidade das soluções produzidas por ela sobre um conjunto de instâncias. A mesma foi implementada utilizando a lingua-

Algoritmo 10: Heurística de Trajetória com Busca Local e Path-relinking (HBLPR)

Entrada: Grafo $G = (V, E)$ e os custos w das cores.
Saída: Coloração própria S^*.

Ordene as cores pelos seus custos: $w_i \leq w_{i+1}$, $1 \leq i < n$;
SoluçãoInicial(G, S);
$S^* \leftarrow S$; $f(S^*) \leftarrow f(S)$;
contElite $\leftarrow 0$; $\mathcal{E} \leftarrow \emptyset$;
se *S é uma coloração própria* **então**
    contElite $\leftarrow$ *contElite* $+ 1$; $\mathcal{E} \leftarrow \{S\}$;
fim se
enquanto *critério de parada não for satisfeito* **faça**
    enquanto *S não é uma coloração própria* **faça**
        BuscaColoraçãoPróprio$(S, S^*, S_{\text{melhor}})$;
        $S \leftarrow S_{\text{melhor}}$;
    fim enquanto
    Esvazie a Lista Tabu;
    MelhoriaSolução(S);
    se *contElite* $<$ *maxElite* **então**
        contElite $\leftarrow$ *contElite* $+ 1$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
    senão
        AtualizaçãoPopulaçãoElite$(S, \mathcal{E})$;
    fim se
    se *contElite* > 5 **então**
        Selecione uma solução $S_e \in \mathcal{E}$ aleatoriamente;
        $S^{\text{inicial}} \leftarrow \text{melhor}\{S, S_e\}$; $S^{\text{guia}} \leftarrow \text{pior}\{S, S_e\}$;
        MoveDistance$(S^{\text{inicial}}, S^{\text{guia}}, S', \Delta, MD)$;
        se $MD \geq 4$ **então**
            PathRelinking$(S^{\text{inicial}}, S^{\text{guia}}, S', \Delta, \overline{S})$;
            $S \leftarrow \overline{S}$;
            MelhoriaSolução(S);
        fim se
    fim se
    se $f(S) < f(S^*)$ **então**
        $S^* \leftarrow S$; $f(S^*) \leftarrow f(S)$;
    fim se
    Perturbação$(S, numPerturbacoes)$;
fim enquanto

gem C e compilada com gcc versão 5.4.0. Em todos os testes, o valor de M, utilizado nas Funções 5.1 e 5.2, seguiu a estratégia de Helmar e Chiarandini [Helmar e Chiarandini, 2011] para o PSC, ao empregar os custos das cores para determinar esse valor, que deve ser grande o suficiente para descartar a presença de arestas conflitantes em qualquer solução de minimização. Uma vez que o PPCCM admite valores reais para tais custos, esse

valor foi definido como $M = (|w_{max}| + |w_{min}| + 10)$, onde $w_{max} = max\{w_c : c = 1, \ldots, n\}$ e $w_{min} = min\{w_c : c = 1, \ldots, n\}$, sendo w_c o custo da classe de cor c. O ambiente computacional empregado nesses experimentos foi o mesmo descrito na Seção 4.2.1.

5.10.1 Ajustes de Parâmetros

Uma vez que todos os vértices realocados na Fase de Perturbação (Seção 5.8) serão inseridos na Lista Tabu, um parâmetro a ser determinado é o tamanho dessa lista, que ao mesmo tempo indica o número de vértices a serem perturbados. Outro parâmetro que necessita ser ajustado é o tamanho da População Elite (Seção 5.7), que corresponde à quantidade de indivíduos que farão parte desse conjunto.

A Tabela 5.1 apresenta, na segunda coluna, os valores determinados para o ajuste desses dois parâmetros, cuja combinação deu origem a 18 versões de teste do algoritmo. As instâncias utilizadas nesses testes e o tempo máximo de processamento, aplicado como critério de parada para cada uma das dez execuções independentes de cada versão, foram os mesmos empregados nesta etapa de ajuste das heurísticas BRKGA e BRKGA+RVNS, todos detalhados na Seção 4.2.1.1.

Como para os BRKGAs, na avaliação dos resultados dessas versões, foram adotadas as mesmas medidas *Sum Best* e *Avg Dev*, utilizando como critério de qualidade um alto valor para a primeira medida e, em caso de empate, o menor valor para a segunda. O gráfico da Figura 5.4 apresenta os valores dessas medidas obtidos por cada uma das versões. Os pontos mais acima e à esquerda do gráfico representam aquelas que alcançaram os melhores valores para as referidas medidas. Pode-se observar que a versão que atingiu *Sum Best* = 120 e *Avg Dev* = 0.013 foi a que obteve a melhor combinação de parâmetros, apresentados na última coluna da Tabela 5.1.

Parâmetro	Valores avaliados	HBLPR
Lista Tabu	$\lfloor\sqrt{n}\rfloor$, $\lfloor\log_{10} n\rfloor$, $\lfloor 10 + \log_{10} n\rfloor$, $\lceil 0.03 \times n\rceil$, $\lceil 0.05 \times n\rceil$, $\lceil 0.10 \times n\rceil$	$\lfloor 10 + \log_{10} n\rfloor$
População Elite	10, 20, 30	30

Tabela 5.1: Valores dos parâmetros utilizados para ajuste e os melhores valores obtidos para o algoritmo HBLPR.

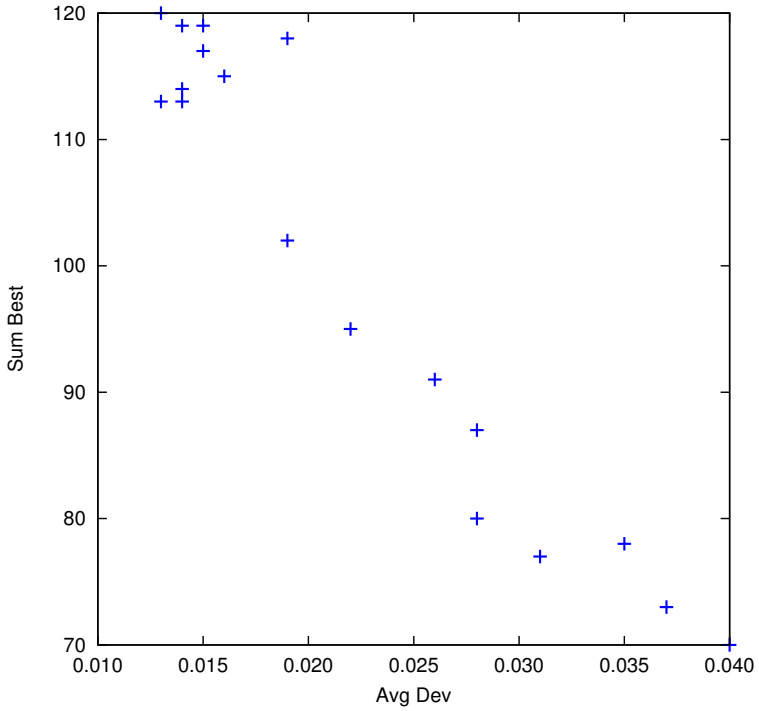


Figura 5.4: Resultados das medidas *Sum Best* e *Avg Dev* para as versões de ajuste de parâmetros do HBLPR. O critério de qualidade adotado foi apresentar um alto valor para a primeira medida e, em caso de empate, o menor valor na segunda. Neste caso, a melhor versão obteve $Sum\ Best = 120$ e $Avg\ Dev = 0.013$.

5.10.2 Análise de Qualidade das Soluções

A fim de avaliar a qualidade das soluções produzidas pela heurística HBLPR, empregando os melhores valores de parâmetros identificados na etapa de ajuste, foram utilizadas as 50 instâncias desenvolvidas para os experimentos com as heurísticas BRKGA e BRKGA+RVNS, bem como os resultados obtidos pelo CPLEX para as mesmas. Sobre cada instância, a HBLPR realizou dez execuções independentes, utilizando igualmente, como critério de parada, o limite de tempo máximo aplicado por aquelas duas heurísticas para cada execução. O processo de desenvolvimento dessas instâncias e da obtenção desses limites de tempo, assim como dos resultados do CPLEX, são descritos na Seção 4.2.1.2.

Os resultados detalhados dos experimentos são apresentados na Tabela 5.2, que fornece, para cada instância, o número de vértices (n), de arestas (m), o valor da melhor solução conhecida ao longo de todos os experimentos realizados, por todos os algoritmos e variantes desenvolvidos, e o valor da melhor solução obtida pelo CPLEX, sublinhando-se os valores das soluções ótimas e sinalizando-se com o símbolo '−' caso uma solução viável não tenha sido encontrada no tempo máximo de execução (3600 segundos). As colunas seguintes descrevem o valor da melhor solução obtida pela heurística, indicando em negrito

quando este é igual ao melhor conhecido, a média dos valores das soluções alcançadas nas dez execuções, o número de vezes que o melhor valor conhecido foi alcançado, o desvio relativo médio percentual entre o valor da solução obtida e o valor da melhor solução conhecida, e o índice médio da iteração na qual a melhor solução foi encontrada. A última coluna indica o tempo (em segundos) aplicado como critério de parada para cada execução da heurística.

Analisando-se os resultados, pode-se observar que HBLPR alcançou o melhor valor conhecido para 38 das 50 instâncias tratadas, tendo encontrado a solução ótima para sete delas. Em três instâncias, tal valor foi atingido em todas as dez execuções, sendo que para duas delas (le450_5c e 3-Insertions_5), o algoritmo convergiu na primeira iteração. A instância em que o mesmo não encontrou o melhor valor conhecido e apresentou maior dificuldade para isso foi abb313GPIA, onde obteve um desvio médio de 6.13%. Esses experimentos mostraram ainda que os valores das soluções encontradas pela heurística são, em média, apenas 1.08% acima dos melhores valores conhecidos para todas as instâncias examinadas.

Assim como os BRKGAs, a heurística foi submetida a testes para analisar a distribuição do seu tempo de execução na busca por uma solução com um valor tão bom quanto um determinado alvo. As instâncias utilizadas para mostrar esse desempenho foram inithx.i.1, qg.order30 e qg.order40, todas com alvos 0.10% e 0.15% acima dos melhores valores conhecidos para as mesmas (3934, 11940 e 15280, respectivamente). Os TTT-Plots para essas instâncias são apresentados nas Figuras 5.5 a 5.7, onde pode ser observado que a distribuição empírica se apresenta abaixo da distribuição teórica para probabilidades superiores a 80%, sendo um indicativo de estagnação no aprimoramento das soluções, segundo Stützle e Hoos [Stützle e Hoos, 1999]. É possível que esse comportamento ocorra devido à convergência da população elite empregada pelo procedimento de *path-relinking* (Seção 5.6). Como a mesma tende a se apresentar mais diversificada no início da busca e mais homogênea ao longo da execução, essa característica pode comprometer a melhoria das soluções ao final do processo.

Instância	n	m	Melhor valor	CPLEX (3600 s)	HBLPR					Tempo de execução heurística (s)
					Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor iteração	
school1-nsh	352	14612	7647.00	13193.00	**7647.00**	7658.90	6	0.16	17514.3	30.95
school1	385	19095	7158.00	14687.00	**7158.00**	7159.20	5	0.02	12003.1	36.03
3-FullIns_4	405	3524	2951.00	**2951.00**	**2951.00**	2951.60	9	0.02	2282.7	29.85
fpsol2.i.3	425	8688	3738.00	**3738.00**	**3738.00**	3738.30	7	0.01	2318.5	35.15
le450_5c	450	9803	2610.00	**2610.00**	**2610.00**	2610.00	10	0.00	1.0	45.98
le450_5d	450	9757	2700.00	3421.00	**2700.00**	2711.40	4	0.42	2502.9	47.64
le450_15c	450	16680	9556.00	11335.00	**9556.00**	9729.90	1	1.82	31159.0	55.07
le450_15d	450	16750	10799.00	13011.00	**10799.00**	10965.20	1	1.54	30700.0	54.71
le450_25a	450	8260	9730.00	**9730.00**	9875.00	9962.60	0	2.39	26305.4	38.63
le450_25b	450	8263	7564.00	**7564.00**	7765.00	7816.80	0	3.34	23571.0	42.11
le450_25c	450	17343	10447.00	11776.00	**10447.00**	10586.10	1	1.33	30043.7	54.25
le450_25d	450	17425	11676.00	12791.00	**11676.00**	11755.80	1	0.68	32653.7	54.21
fpsol2.i.2	451	8691	4694.00	**4694.00**	**4694.00**	4695.40	4	0.03	5424.0	38.47
4-Insertions_4	475	1795	999.00	**999.00**	1035.00	1035.00	0	3.60	665.5	27.18
fpsol2.i.1	496	11654	8364.00	**8364.00**	8365.00	8365.00	0	0.01	1832.7	28.50
DSJC500.5	500	62624	18333.00	22845.00	**18333.00**	18678.20	1	1.88	32214.5	152.42
C500.9	500	112332	63147.00	77015.00	**63147.00**	63806.90	1	1.05	19492.4	276.35
DSJC500.9	500	112437	65373.00	78869.00	**65373.00**	65904.60	1	0.81	22769.1	290.00
DSJR500.1	500	3555	6253.00	**6253.00**	6484.00	6518.00	0	4.24	29492.8	46.68
DSJR500.1c	500	121275	27395.00	35554.00	**27395.00**	27569.40	1	0.64	18217.3	149.89
DSJR500.5	500	58862	54392.00	64892.00	**54392.00**	54660.20	1	0.49	15783.7	191.11
2-Insertions_5	597	3936	2999.00	**2999.00**	3080.00	3087.70	0	2.96	20442.8	58.76
1-Insertions_6	607	6337	1347.00	1367.00	**1347.00**	1360.00	8	0.97	11714.6	63.88
inithx.i.3	621	13969	3633.00	**3633.00**	**3633.00**	3636.90	3	0.11	2820.9	96.46
inithx.i.2	645	13979	4073.00	**4073.00**	**4073.00**	4077.00	8	0.10	4387.5	95.31
ash331GPIA	662	4185	1513.00	**1513.00**	1554.00	1574.90	0	4.09	33298.0	65.85
4-FullIns_4	690	6650	2443.00	**2443.00**	2447.00	2450.00	0	0.29	7678.9	82.17
will199GPIA	701	7065	4829.00	5428.00	**4829.00**	4923.70	2	1.96	30939.5	79.01
inithx.i.1	864	18707	3934.00	**3934.00**	**3934.00**	3935.20	4	0.03	1257.0	106.34
qg.order30	900	26100	11940.00	**11940.00**	**11940.00**	11945.50	1	0.05	18618.6	270.78
latin_sqr_10	900	307350	48822.00	–	**48822.00**	49108.00	1	0.59	16879.1	620.77
wap05	905	43081	12593.00	14181.00	**12593.00**	12652.90	1	0.48	26149.4	254.86
wap06	947	43571	18453.00	20434.00	**18453.00**	18565.90	1	0.61	23501.2	267.59
DSJC1000.5	1000	249826	46790.00	–	**46790.00**	47272.40	1	1.03	33067.0	920.84
flat1000_50_0	1000	245000	41915.00	–	**41915.00**	42441.50	1	1.26	32380.6	934.90
flat1000_60_0	1000	245830	40468.00	–	**40468.00**	41110.60	1	1.59	29948.8	908.07
flat1000_76_0	1000	246708	41729.00	–	**41729.00**	42040.00	1	0.75	30091.8	920.24
DSJC1000.9	1000	449449	103906.00	–	**103906.00**	104748.70	1	0.81	25742.4	1869.64
C1000.9	1000	450079	105709.00	–	**105709.00**	106807.80	1	1.04	23889.3	1874.78
5-FullIns_4	1085	11395	2212.00	**2212.00**	**2212.00**	2214.10	7	0.09	578.5	192.97
ash608GPIA	1216	7844	3859.00	4215.00	3922.00	3931.60	0	1.88	36678.9	254.75
3-Insertions_5	1406	9695	1406.00	**1406.00**	**1406.00**	1406.00	10	0.00	1.0	262.40
abb313GPIA	1557	65390	4597.00	–	4847.00	4878.80	0	6.13	24751.0	562.43
qg.order40	1600	62400	15280.00	–	15285.00	15289.00	0	0.06	24849.8	1053.18
wap07	1809	103368	13380.00	–	**13380.00**	13463.50	1	0.62	40110.5	1071.05
wap08	1870	104176	14497.00	–	**14497.00**	14564.90	1	0.47	34883.6	1106.13
ash958GPIA	1916	12506	2886.00	3171.00	2897.00	2909.10	0	0.80	40967.1	606.65
3-FullIns_5	2030	33751	4082.00	5845.00	**4082.00**	4082.00	10	0.00	2569.4	872.03
wap01	2368	110871	18719.00	–	**18719.00**	18798.70	1	0.43	43886.8	1842.69
wap02	2464	111742	17439.00	–	**17439.00**	17487.60	1	0.28	36614.2	1920.90

Tabela 5.2: Resultados detalhados da heurística HBLPR.

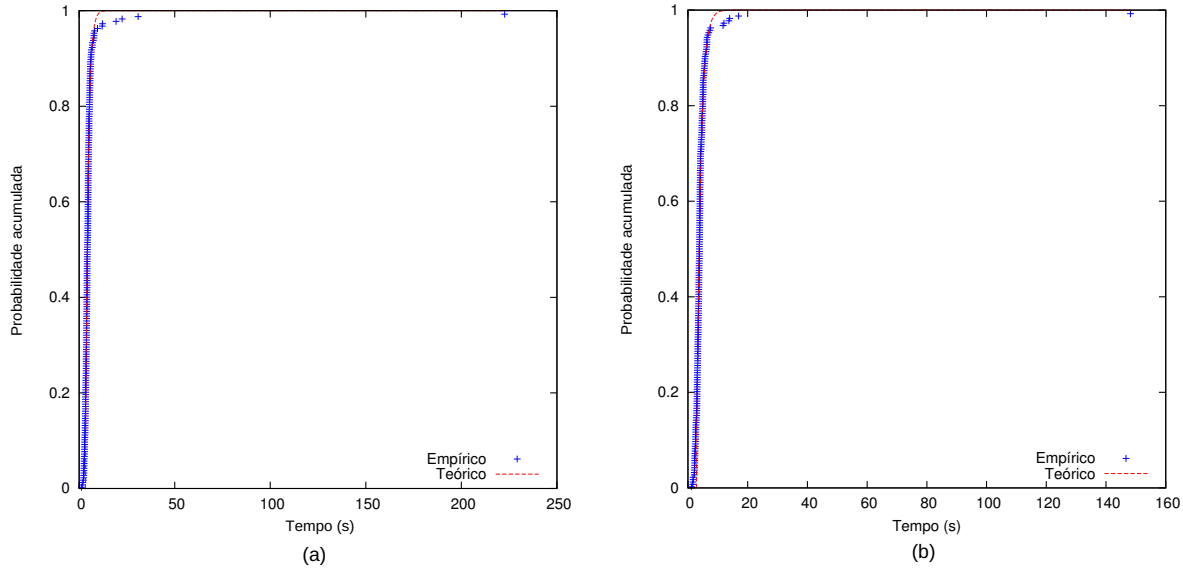


Figura 5.5: TTT-Plots para a instância inithx.i.1 com alvos (a) 0.10% e (b) 0.15% acima do melhor valor conhecido (3934).

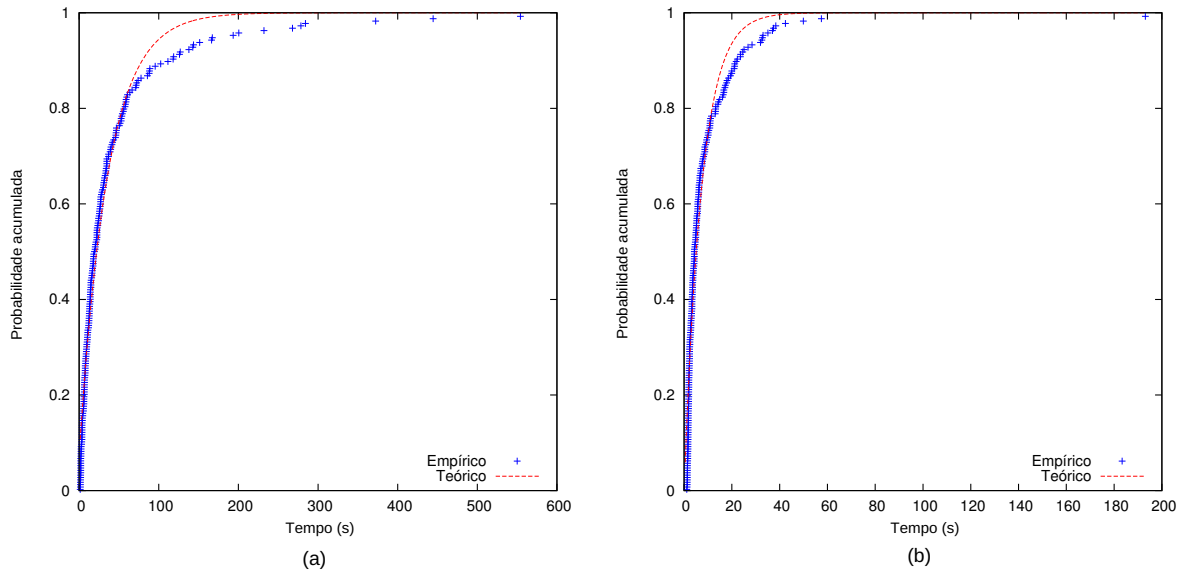


Figura 5.6: TTT-Plots para a instância qg.order30 com alvos (a) 0.10% e (b) 0.15% acima do melhor valor conhecido (11940).

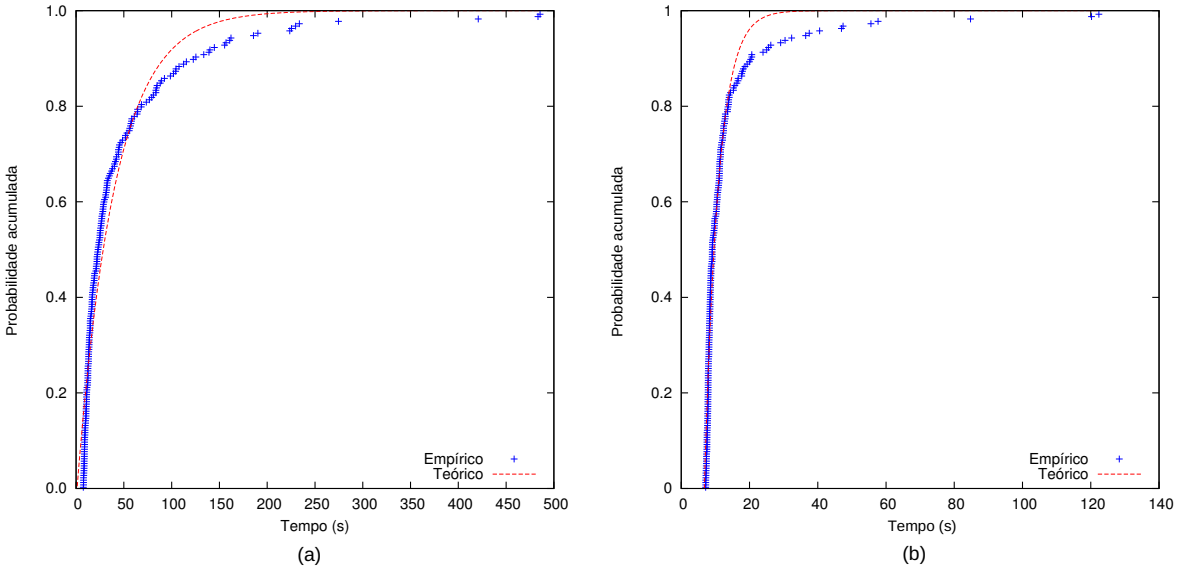


Figura 5.7: TTT-Plots para a instância qg.order40 com alvos (a) 0.10% e (b) 0.15% acima do melhor valor conhecido (15280).

5.11 Comparação das Heurísticas HBLPR, BRKGA e BRKGA+RVNS

Com o intuito de comparar diretamente as três heurísticas propostas neste trabalho, foram utilizados os resultados obtidos sobre as 50 instâncias tratadas nos experimentos anteriores, descritos nas Seções 4.2.1.2 (BRKGA e BRKGA+RVNS) e 5.10.2 (HBLPR).

Na Tabela 5.4 encontram-se os resultados de cada heurística apresentados de maneira resumida, com apenas o valor da melhor solução alcançada, o desvio relativo médio percentual e o tempo limite para cada execução. Outras informações do experimento são fornecidas nas Tabelas B.1 e B.2 (Apêndice B). Os resultados mostram que a heurística HBLPR encontrou o melhor valor conhecido para 38 instâncias e a solução ótima para sete delas. BRKGA atingiu esse valor para 16 instâncias e o valor ótimo para nove, enquanto BRKGA+RVNS para 14, sendo dez soluções ótimas.

Para 28 instâncias, HBLPR encontrou uma solução que não foi alcançada por nenhuma das outras duas heurísticas, ao passo que em sete instâncias pelo menos um dos BRKGAs atingiu um valor não obtido pela HBLPR e em cinco instâncias nenhuma das três heurísticas conseguiu atingir o melhor valor conhecido. A heurística HBLPR alcançou este valor em todas as execuções para três instâncias, BRKGA para quatro e BRKGA+RVNS para seis. No entanto, para apenas uma instância as três heurísticas alcançaram esse valor em

todas as dez execuções (3-Insertions_5). O desvio médio máximo obtido pelo BRKGA foi de 28.86%, BRKGA+RVNS alcançou 26.24% e HBLPR atingiu apenas 6.13%.

Também foram produzidos TTT-Plots para avaliar o comportamento das três heurísticas na busca por um valor alvo. Na Figura 5.8 são encontrados esses gráficos para a instância 3-FullIns_4 com alvo 2951, custo da melhor solução conhecida, e 1000 segundos como tempo limite. Nota-se que a heurística HBLPR consome o maior tempo de processamento para atingir 100% desse alvo, 793.5 segundos. BRKGA e BRKGA+RVNS gastam 5.5 e 16.3 segundos, respectivamente. No entanto, HBLPR apresentou maior probabilidade de convergir mais rapidamente para o alvo do que as outras duas heurísticas, uma vez que $Pr(T_{HBLPR} \leq T_{BRKGA}) = 0.547$ e $Pr(T_{HBLPR} \leq T_{BRKGA+RVNS}) = 0.611$. Esse comportamento pode ser observado na Figura 5.9, onde, dada a evolução da melhor solução no decorrer de quatro segundos iniciais de processamento, todas as três heurísticas atingiram o melhor valor conhecido nesse período, sendo que HBLPR de forma mais rápida.

Os TTT-Plots para a instância inithx.i.1 são apresentados na Figura 5.10. Nesse experimento, o valor médio das soluções obtidas por BRKGA (3937) foi definido como alvo, sendo o tempo máximo de execução limitado a 1000 segundos. Para essa instância, HBLPR atinge o valor alvo, com 100% de probabilidade, em 134.8 segundos, ao passo que BRKGA+RVNS e BRKGA necessitam, respectivamente, de 209.0 e 402.5 segundos. Novamente a heurística HBLPR atinge o valor alvo antecipadamente, apresentando $Pr(T_{HBLPR} \leq T_{BRKGA+RVNS}) = 0.995$ e $Pr(T_{HBLPR} \leq T_{BRKGA}) = 0.989$. A evolução da melhor solução, durante os 50 segundos iniciais de processamento, é mostrada na Figura 5.11, onde verifica-se que somente HBLPR, em menor tempo, e BRKGA+RVNS convergiram para o melhor valor conhecido (3934), uma vez que BRKGA obteve 3935 durante essa execução.

5.11.1 Conclusões

Um resumo dos resultados desses experimentos pode ser observado na Tabela 5.3, que apresenta um comparativo da performance dos algoritmos conforme as medidas de qualidade descritas na Seção 4.2.1. Analisando esses resultados, é possível concluir que a heurística HBLPR mostrou-se mais eficaz nos experimentos sobre as 50 instâncias tratadas, encontrando o melhor valor conhecido para um número maior de instâncias. Além disso, os valores de suas soluções são, em média, 1.08% acima dos melhores valores conhecidos, ao passo que essa média para BRKGA e BRKGA+RVNS é de 8.70% e 8.72%,

respectivamente. Por fim, a medida *Score* ratifica o domínio da HBLPR sobre as demais, uma vez que apresenta o menor valor entre os três algoritmos.

Finalizadas as comparações das heurísticas propostas, no próximo capítulo são realizadas as considerações finais sobre este trabalho e apresentadas possibilidades de investigações futuras.

	BRKGA	BRKGA+RVNS	HBLPR
Avg Dev (%)	8.70	8.72	**1.08**
Sum Best	92	93	**120**
#Best	16	14	**38**
Score	48	54	**15**

Tabela 5.3: Comparativo da performance dos algoritmos BRKGA, BRKGA+RVNS e HBLPR sobre as 50 instâncias.

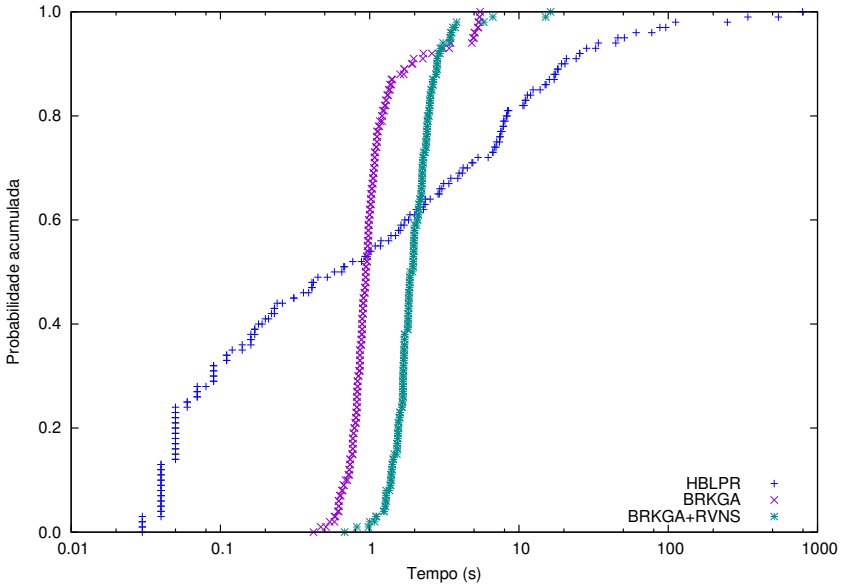


Figura 5.8: TTT-Plots para a instância 3-FullIns_4, com alvo 2951 e tempo máximo de 1000 segundos. Utilizando a ferramenta *tttplots-compare*:
$Pr(T_{HBLPR} \leq T_{BRKGA}) = 0.547$ e $Pr(T_{HBLPR} \leq T_{BRKGA+RVNS}) = 0.611$.

					BRKGA		BRKGA+RVNS		HBLPR		Tempo de
Instância	n	m	Melhor valor	CPLEX (3600 s)	Melhor valor	Avg Dev (%)	Melhor valor	Avg Dev (%)	Melhor valor	Avg Dev (%)	execução heurísticas (s)
school1-nsh	352	14612	7647.00	13193.00	8999.00	20.47	9020.00	25.13	**7647.00**	0.16	30.95
school1	385	19095	7158.00	14687.00	7723.00	12.13	7733.00	14.87	**7158.00**	0.02	36.03
3-FullIns_4	405	3524	2951.00	**2951.00**	**2951.00**	0.00	**2951.00**	0.00	**2951.00**	0.02	29.85
fpsol2.i.3	425	8688	3738.00	**3738.00**	**3738.00**	0.00	**3738.00**	0.00	**3738.00**	0.01	35.15
le450_5c	450	9803	2610.00	**2610.00**	2642.00	2.34	2645.00	3.28	**2610.00**	0.00	45.98
le450_5d	450	9757	2700.00	3421.00	2710.00	1.59	2711.00	1.61	**2700.00**	0.42	47.64
le450_15c	450	16680	9556.00	11335.00	11131.00	17.05	10683.00	12.94	**9556.00**	1.82	55.07
le450_15d	450	16750	10799.00	13011.00	12629.00	20.42	12441.00	15.93	**10799.00**	1.54	54.71
le450_25a	450	8260	9730.00	**9730.00**	10258.00	6.63	10485.00	8.33	9875.00	2.39	38.63
le450_25b	450	8263	7564.00	**7564.00**	8028.00	7.08	8241.00	9.43	7765.00	3.34	42.11
le450_25c	450	17343	10447.00	11776.00	11940.00	14.77	11643.00	12.14	**10447.00**	1.33	54.25
le450_25d	450	17425	11676.00	12791.00	13074.00	12.74	12816.00	10.44	**11676.00**	0.68	54.21
fpsol2.i.2	451	8691	4694.00	**4694.00**	**4694.00**	0.02	**4694.00**	0.03	**4694.00**	0.03	38.47
4-Insertions_4	475	1795	999.00	**999.00**	1001.00	0.75	**999.00**	0.67	1035.00	3.60	27.18
fpsol2.i.1	496	11654	8364.00	**8364.00**	**8364.00**	0.00	**8364.00**	0.00	8365.00	0.01	28.50
DSJC500.5	500	62624	18333.00	22845.00	22935.00	25.72	21892.00	21.41	**18333.00**	1.88	152.42
C500.9	500	112332	63147.00	77015.00	71823.00	17.21	74360.00	18.83	**63147.00**	1.05	276.35
DSJC500.9	500	112437	65373.00	78869.00	72535.00	12.72	74194.00	15.28	**65373.00**	0.81	290.00
DSJR500.1	500	3555	6253.00	**6253.00**	6646.00	6.87	6724.00	8.33	6484.00	4.24	46.68
DSJR500.1c	500	121275	27395.00	35554.00	27575.00	1.30	27462.00	1.49	**27395.00**	0.64	149.89
DSJR500.5	500	58862	54392.00	64892.00	56453.00	5.72	58269.00	9.05	**54392.00**	0.49	191.11
2-Insertions_5	597	3936	2999.00	**2999.00**	**2999.00**	0.13	**2999.00**	0.29	3080.00	2.96	58.76
1-Insertions_6	607	6337	1347.00	1367.00	**1347.00**	0.24	**1347.00**	0.30	**1347.00**	0.97	63.88
inithx.i.3	621	13969	3633.00	**3633.00**	**3633.00**	0.02	**3633.00**	0.00	**3633.00**	0.11	96.46
inithx.i.2	645	13979	4073.00	**4073.00**	**4073.00**	0.01	**4073.00**	0.00	**4073.00**	0.10	95.31
ash331GPIA	662	4185	1513.00	**1513.00**	1537.00	3.19	1539.00	3.45	1554.00	4.09	65.85
4-FullIns_4	690	6650	2443.00	**2443.00**	**2443.00**	0.02	**2443.00**	0.07	2447.00	0.29	82.17
will199GPIA	701	7065	4829.00	5428.00	4919.00	3.07	4948.00	3.87	**4829.00**	1.96	79.01
inithx.i.1	864	18707	3934.00	**3934.00**	**3934.00**	0.05	**3934.00**	0.01	**3934.00**	0.03	106.34
qg.order30	900	26100	11940.00	**11940.00**	12084.00	1.38	12027.00	0.94	**11940.00**	0.05	270.78
latin_sqr_10	900	307350	48822.00	-	57949.00	20.27	56677.00	16.76	**48822.00**	0.59	620.77
wap05	905	43081	12593.00	14181.00	13803.00	10.17	14039.00	12.31	**12593.00**	0.48	254.86
wap06	947	43571	18453.00	20434.00	19495.00	6.39	19905.00	8.23	**18453.00**	0.61	267.59
DSJC1000.5	1000	249826	46790.00	-	58453.00	25.90	57538.00	23.49	**46790.00**	1.03	920.84
flat1000_50_0	1000	245000	41915.00	-	51719.00	25.22	51032.00	22.90	**41915.00**	1.26	934.90
flat1000_60_0	1000	245830	40468.00	-	51900.00	28.86	50629.00	26.24	**40468.00**	1.59	908.07
flat1000_76_0	1000	246708	41729.00	-	51205.00	23.59	50425.00	21.34	**41729.00**	0.75	920.24
DSJC1000.9	1000	449449	103906.00	-	130627.00	27.59	129308.00	25.12	**103906.00**	0.81	1869.64
C1000.9	1000	450079	105709.00	-	134237.00	28.22	131064.00	25.47	**105709.00**	1.04	1874.78
5-FullIns_4	1085	11395	2212.00	**2212.00**	**2212.00**	0.00	**2212.00**	0.01	**2212.00**	0.09	192.97
ash608GPIA	1216	7844	3859.00	4215.00	**3859.00**	0.42	3866.00	1.32	3922.00	1.88	254.75
3-Insertions_5	1406	9695	1406.00	**1406.00**	**1406.00**	0.00	**1406.00**	0.00	**1406.00**	0.00	262.40
abb313GPIA	1557	65390	4597.00	-	**4597.00**	0.63	4655.00	3.87	4847.00	6.13	562.43
qg.order40	1600	62400	15280.00	-	15486.00	1.48	15438.00	1.13	15285.00	0.06	1053.18
wap07	1809	103368	13380.00	-	15113.00	13.40	15205.00	14.24	**13380.00**	0.62	1071.05
wap08	1870	104176	14497.00	-	15790.00	9.30	15855.00	10.43	**14497.00**	0.47	1106.13
ash958GPIA	1916	12506	2886.00	3171.00	**2886.00**	0.71	2927.00	2.19	2897.00	0.80	606.65
3-FullIns_5	2030	33751	4082.00	5845.00	**4082.00**	0.09	**4082.00**	0.31	**4082.00**	0.00	872.03
wap01	2368	110871	18719.00	-	20584.00	10.48	20917.00	12.41	**18719.00**	0.43	1842.69
wap02	2464	111742	17439.00	-	18853.00	8.47	19145.00	10.28	**17439.00**	0.28	1920.90

Tabela 5.4: Resultados resumidos das heurísticas BRKGA, BRKGA+RVNS e HBLPR para as 50 instâncias.

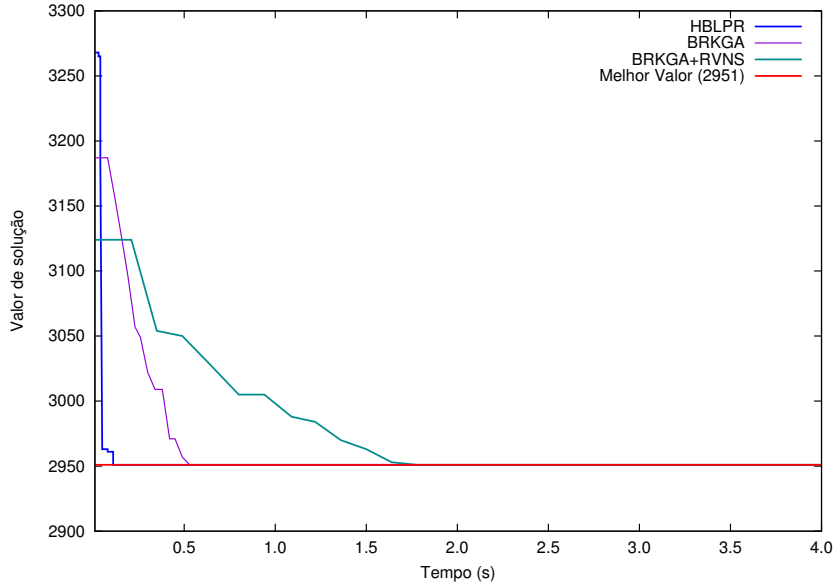


Figura 5.9: Evolução da melhor solução para a instância 3-FullIns_4. Todas as heurísticas alcançaram o melhor valor conhecido (2951) ao longo dos quatro segundos iniciais de processamento.

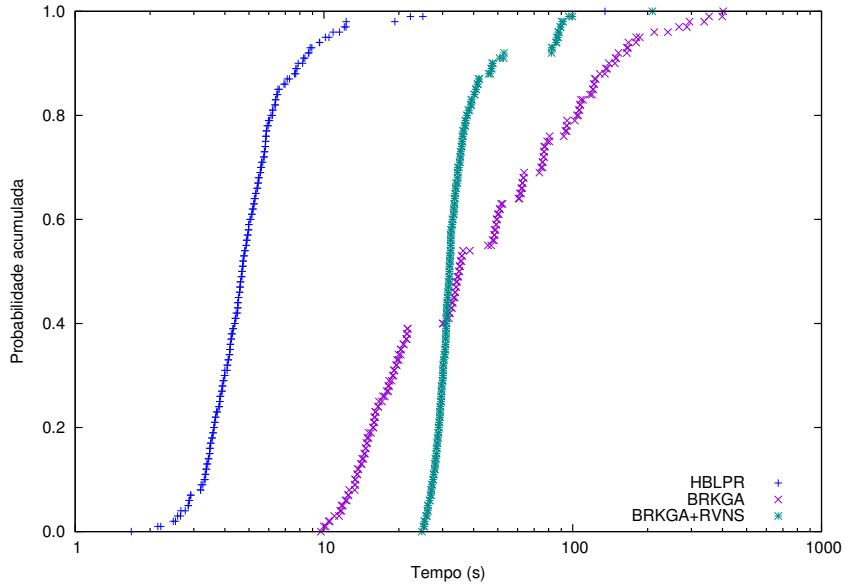


Figura 5.10: TTT-Plots para a instância inithx.i.1, com alvo 3937 e tempo máximo de 1000 segundos. Utilizando a ferramenta *tttplots-compare*: $Pr(T_{HBLPR} \leq T_{BRKGA+RVNS}) = 0.995$ e $Pr(T_{HBLPR} \leq T_{BRKGA}) = 0.989$.

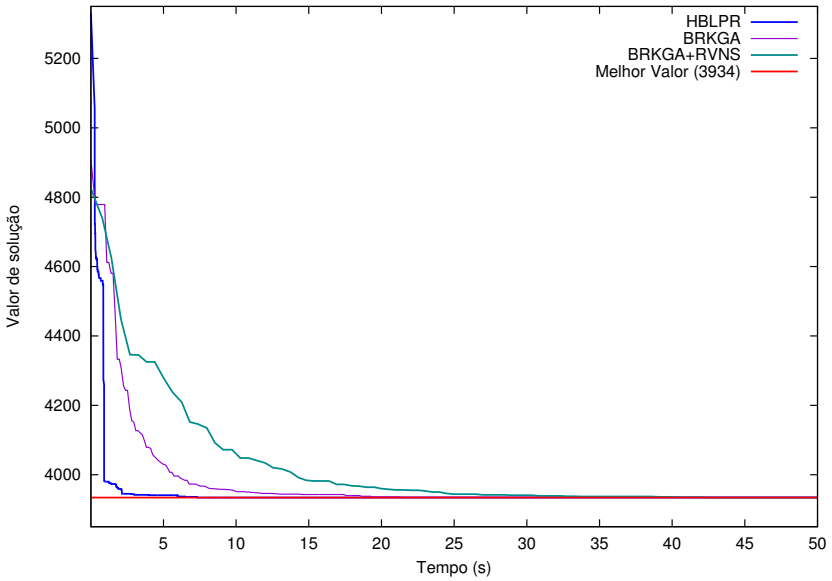


Figura 5.11: Evolução da melhor solução para a instância inithx.i.1. Somente as heurísticas HBLPR e BRKGA+RVNS alcançaram o melhor valor conhecido (3934) ao longo dos 50 segundos iniciais de processamento, tendo o BRKGA obtido 3935 nesse mesmo período.

Capítulo 6

Conclusão e Trabalhos Futuros

Nesta tese foi abordado o Problema da Partição Cromática de Custo Mínimo (PPCCM), considerado uma generalização do Problema da Soma Cromática (PSC). Diferentemente deste, que utiliza números naturais em sequência como custos das cores, o PPCCM considera custos reais, tendo como objetivo colorir os vértices de um grafo de modo que vértices adjacentes tenham cores diferentes e a soma dos custos das cores utilizadas seja mínima. Sendo NP-Difícil para grafos em geral, a utilização de algoritmos exatos para obter a melhor solução possível torna-se impraticável para instâncias de grande porte, por necessitar de elevado tempo computacional, sendo sugerido o emprego de métodos heurísticos para a sua resolução.

Foram propostos algoritmos aproximados para solucionar o PPCCM considerando um grafo simples não-direcionado. Inicialmente foram desenvolvidas duas heurísticas baseadas na metaheurística Algoritmos Genéticos com Chaves Aleatórias Tendenciosas, denominadas BRKGA e BRKGA+RVNS. Ambas fazem uso do mesmo codificador, porém a segunda aplica a estratégia de busca em vizinhança RVNS, considerada uma modificação da metaheurística VNS, nos indivíduos que farão parte do grupo elite na geração seguinte, com a intenção de aprimorar sua qualidade.

Posteriormente, foi desenvolvida a heurística HBLPR, que faz uso de duas estratégias de busca local. Tais buscas são seguidas por um procedimento de *path-relinking*, que explora a trajetória de conexão entre duas soluções. Por último, ocorre uma perturbação nos vértices da solução corrente, ocasionando a inclusão dos mesmos em uma lista tabu, a fim de evitar que essa solução se repita na iteração seguinte.

Para avaliar o desempenho das heurísticas implementadas, foram criadas instâncias para o problema a partir de grafos selecionados aleatoriamente de *benchmarks* amplamente

utilizados do Problema de Coloração de Grafos (PCG).

Os experimentos mostraram que as heurísticas BRKGA e BRKGA+RVNS apresentaram praticamente a mesma performance sobre o conjunto de instâncias testado, não sendo possível definir a predominância de uma sobre a outra. Na análise dos resultados também verificou-se que a heurística HBLPR foi a que se mostrou mais eficaz, encontrando o melhor valor conhecido para 76% das instâncias tratadas, além de obter soluções com custo médio 1.08% acima dos melhores valores conhecidos.

Assim, as principais contribuições deste trabalho foram a retomada do estudo de um problema da literatura para o qual inexistia algoritmo eficiente (exato ou aproximado) para o caso geral, bem como o desenvolvimento das primeiras heurísticas para tratar o problema.

Como trabalho futuro, pretende-se investigar a aplicação de uma estratégia de *restarts* após certo limite para tentar evitar a estagnação do algoritmo, como sugerido em [Stützle e Hoos, 1999] para esses casos. Também como pesquisa futura, deseja-se avaliar a utilização de outras técnicas de diversificação na HBLPR que inclua uma lista tabu adaptativa, que possa variar seu tamanho de acordo com a evolução da melhor solução.

Outra oportunidade de investigação é o desenvolvimento de algoritmos exatos para o PPCCM com base em técnicas de programação inteira, com a intenção de hibridizá-los com metaheurísticas para tratar instâncias de tamanhos maiores em tempos computacionais viáveis. Além disso, também podem ser gerados modelos de programação inteira mais fortes baseados em formulações por representantes de classes de cores.

Referências

[Aiex et al., 2002] Aiex, R. M., Resende, M. G. e Ribeiro, C. C. (2002). Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, 8:343–373.

[Aiex et al., 2007] Aiex, R. M., Resende, M. G. C. e Ribeiro, C. C. (2007). tttplots: A perl program to create time-to-target plots. *Optimization Letters*, 1:355–366.

[Andrade et al., 2015] Andrade, C. E., Resende, M. G., Zhang, W., Sinha, R. K., Reichmann, K. C., Doverspike, R. D. e Miyazawa, F. K. (2015). A biased random-key genetic algorithm for wireless backhaul network design. *Applied Soft Computing*, 33:150–169.

[Avanthay et al., 2003] Avanthay, C., Hertz, A. e Zufferey, N. (2003). A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151:379–388.

[Bahiense et al., 2014] Bahiense, L., Frota, Y., Noronha, T. F. e Ribeiro, C. C. (2014). A branch-and-cut algorithm for the equitable coloring problem using a formulation by representatives. *Discrete Applied Mathematics*, 164:34 – 46. Combinatorial Optimization.

[Bar-Noy et al., 1998] Bar-Noy, A., Bellare, M., Halldósson, M. M., Shachnai, H. e Tamir, T. (1998). On chromatic sums and distributed resource allocation. *Information and Computation*, 140:183–202.

[Bar-noy e Kortsarz, 1998] Bar-noy, A. e Kortsarz, G. (1998). Minimum color sum of bipartite graphs. *Journal of Algorithms*, 28:339–365.

[Barnier e Brisset, 2004] Barnier, N. e Brisset, P. (2004). Graph coloring for air traffic flow management. *Annals of Operations Research*, 130:163–178.

[Bastos e Ribeiro, 2002] Bastos, M. P. e Ribeiro, C. C. (2002). Reactive tabu search with path-relinking for the Steiner problem in graphs. In Ribeiro, C. C. e Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 39–58. Springer, Boston.

[Bean, 1994] Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160.

[Bello et al., 2008] Bello, G. S., Boeres, M. C. S. e Rangel, M. C. (2008). An approach for the class/teacher timetabling problem using graph coloring. In *Proceedings of the 7th Internacional Conference on the Practice and Theory of Automated Timetabling*, volume 1, pages 1–6.

[Benlic e Hao, 2012] Benlic, U. e Hao, J.-K. (2012). A study of breakout local search for the minimum sum coloring problem. In Bui, L., Ong, Y., Hoai, N., Ishibuchi, H. e Suganthan, P., editors, *Simulated Evolution and Learning*, volume 7673 of *Lecture Notes in Computer Science*, pages 128–137. Springer, Berlin.

[Blöchliger e Zufferey, 2008] Blöchliger, I. e Zufferey, N. (2008). A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35:960–975.

[Boudhar e Finke, 2000] Boudhar, M. e Finke, G. (2000). Scheduling on a batch machine with job compatibilities. *Belgian Journal of Operations Research, Statistics and Computer Science*, 40:69–80.

[Bouziri e Jouini, 2010] Bouziri, H. e Jouini, M. (2010). A tabu search approach for the sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:915–922.

[Brandão et al., 2015] Brandão, J. S., Noronha, T. F., Resende, M. G. C. e Ribeiro, C. C. (2015). A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions in Operational Research*, 22:823–839.

[Brandão et al., 2017] Brandão, J. S., Noronha, T. F., Resende, M. G. C. e Ribeiro, C. C. (2017). A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *International Transactions in Operational Research*, 24:1061–1077.

[Brélaz, 1979] Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256.

[Brown, 1972] Brown, J. R. (1972). Chromatic scheduling and the chromatic number problem. *Management Science*, 19:456–463.

[Buriol et al., 2007] Buriol, L., Resende, M. e Thorup, M. (2007). Survivable IP network design with OSPF routing. *Networks*, 49:51–64.

[Buriol et al., 2010] Buriol, L. S., Hirsch, M. J., Pardalos, P. M., Querido, T., Resende, M. G. C. e Ritt, M. (2010). A biased random-key genetic algorithm for road congestion minimization. *Optimization Letters*, 4:619–633.

[Campêlo et al., 2008] Campêlo, M., Campos, V. A. e Corrêa, R. (2008). On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156:1097–1111.

[Campêlo et al., 2004] Campêlo, M., Corrêa, R. e Frota, Y. (2004). Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, 89:159–164.

[Chams et al., 1987] Chams, M., Hertz, A. e de Werra, D. (1987). Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32:260–266.

[Chiarandini et al., 2007] Chiarandini, M., Dumitrescu, I. e Stützle, T. (2007). Stochastic local search algorithms for the graph colouring problem. In Gonzalez, T. F., editor, *Handbook of Approximation Algorithms and Metaheuristics*, Computer & Information Science Series, pages 63.1–63.17. Chapman & Hall/CRC, Boca Raton.

[Chiarandini e Stützle, 2002] Chiarandini, M. e Stützle, T. (2002). An application of iterated local search to graph coloring. In *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, Ithaca.

[Costa et al., 1995] Costa, D., Hertz, A. e Dubuis, C. (1995). Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1:105–128.

[Douiri e Elbernoussi, 2011] Douiri, S. M. e Elbernoussi, S. (2011). New algorithm for the sum coloring problem. *International Journal of Contemporary Mathematical Sciences*, 6:453–463.

[Ericsson et al., 2002] Ericsson, M., Resende, M. e Pardalos, P. (2002). A genetic algorithm for the weight setting problem in ospf routing. *Journal of Combinatorial Optimization*, 6:299–333.

[Feo et al., 1994] Feo, T. A., Resende, M. G. C. e Smith, S. H. (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878.

[Fleurent e Ferland, 1996] Fleurent, C. e Ferland, J. A. (1996). Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461.

[Fontes e Gonçalves, 2013] Fontes, D. B. M. M. e Gonçalves, J. F. (2013). A multi-population hybrid biased random key genetic algorithm for hop-constrained trees in nonlinear cost flow networks. *Optimization Letters*, 7:1303–1324.

[Fontes e Gonçalves, 2007] Fontes, D. B. M. M. e Gonçalves, J. F. (2007). Heuristic solutions for general concave minimum cost network flow problems. *Networks*, 50:67–76.

[Frota et al., 2010] Frota, Y., Maculan, N., Noronha, T. F. e Ribeiro, C. C. (2010). A branch-and-cut algorithm for partition coloring. *Networks*, 55(3):194–204.

[Furini e Malaguti, 2012] Furini, F. e Malaguti, E. (2012). Exact weighted vertex coloring via branch-and-price. *Discrete Optimization*, 9:130–136.

[Galinier et al., 2013] Galinier, P., Hamiez, J.-P., Hao, J.-K. e Porumbel, D. (2013). Recent advances in graph vertex coloring. In Zelinka, I., Snásel, V. e Abraham, A., editors, *Handbook of Optimization*, volume 38 of *Intelligent Systems Reference Library*, pages 505–528. Springer, Berlin.

[Galinier e Hao, 1999] Galinier, P. e Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397.

[Galinier e Hertz, 2006] Galinier, P. e Hertz, A. (2006). A survey of local search methods for graph coloring. *Computers & Operations Research*, 33:2547–2562.

[Galinier et al., 2008] Galinier, P., Hertz, A. e Zufferey, N. (2008). An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156:267–279.

[Garey e Johnson, 1979] Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

[Glover, 1997] Glover, F. (1997). Tabu search and adaptive memory programming — advances, applications and challenges. In Barr, R. S., Helgason, R. V. e Kennington, J. L., editors, *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, pages 1–75. Springer, Boston.

[Glover et al., 2003] Glover, F., Laguna, M. e Marti, R. (2003). Scatter search and path relinking: Advances and applications. In Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, pages 1–35. Springer, Boston.

[Gonçalves et al., 2005] Gonçalves, J. F., de Magalhães Mendes, J. J. e Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95.

[Gonçalves e Resende, 2012] Gonçalves, J. F. e Resende, M. G. (2012). A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39:179–190.

[Gonçalves e Resende, 2013] Gonçalves, J. F. e Resende, M. G. (2013). A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics*, 145:500–510.

[Gonçalves et al., 2014] Gonçalves, J. F., Resende, M. G. e Toso, R. F. (2014). An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesquisa Operacional*, 34:143–164.

[Gonçalves e Resende, 2011a] Gonçalves, J. F. e Resende, M. G. C. (2011a). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525.

[Gonçalves e Resende, 2011b] Gonçalves, J. F. e Resende, M. G. C. (2011b). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22:180–201.

[Gualandi e Malucelli, 2012] Gualandi, S. e Malucelli, F. (2012). Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24:81–100.

[Hajiabolhassan et al., 2000] Hajiabolhassan, H., Mehrabadi, M. e Tusserkani, R. (2000). Minimal coloring and strength of graphs. *Discrete Mathematics*, 215:265–270.

[Halldórsson et al., 2003] Halldórsson, M. M., Kortsarz, G. e Shachnai, H. (2003). Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs. *Algorithmica*, 37:187–209.

[Hamiez e Hao, 2002] Hamiez, J.-P. e Hao, J.-K. (2002). Scatter Search for Graph Coloring. In Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E. e Schoenauer, M., editors, *Artificial Evolution*, Lecture Notes in Computer Science, pages 168–179. Springer, Berlin.

[Han e Kim, 2015] Han, K. e Kim, C. (2015). An evolutionary approach for graph multi-coloring problem. *Applied Mathematical Sciences*, 9:1677–1684.

[Hansen e Mladenović, 1999] Hansen, P. e Mladenović, N. (1999). An introduction to variable neighborhood search. In Voß, S., Martello, S., Osman, I. H. e Roucairol, C., editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Springer, Boston.

[Helmar e Chiarandini, 2011] Helmar, A. e Chiarandini, M. (2011). A local search heuristic for chromatic sum. In Gaspero, L. D., Schaerf, A. e Stützle, T., editors, *Proceedings of the 9th Metaheuristics International Conference*, pages 161–170, Udine.

[Hertz e de Werra, 1987] Hertz, A. e de Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing*, 39:345–351.

[Hertz et al., 2008] Hertz, A., Plumettaz, M. e Zufferey, N. (2008). Variable space search for graph coloring. *Discrete Applied Mathematics*, 156:2551–2560.

[Ho e Gendreau, 2006] Ho, S. C. e Gendreau, M. (2006). Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12:55–72.

[Jansen, 1996] Jansen, K. (1996). Complexity results for the optimum cost chromatic partition problem. *Universität Trier, Mathematik/Informatik, Forschungsbericht*, 96-41.

[Jansen, 2000] Jansen, K. (2000). Approximation results for the optimum cost chromatic partition problem. *Journal of Algorithms*, 34:54–89.

[Jiang e West, 1999] Jiang, T. e West, D. B. (1999). Coloring of trees with minimum sum of colors. *Journal of Graph Theory*, 32:354–358.

[Jin et al., 2017] Jin, Y., Hamiez, J.-P. e Hao, J.-K. (2017). Algorithms for the minimum sum coloring problem: A review. *Artificial Intelligence Review*, 47:367–394.

[Jin e Hao, 2016] Jin, Y. e Hao, J.-K. (2016). Hybrid evolutionary search for the minimum sum coloring problem of graphs. *Information Sciences*, 352 - 353:15–34.

[Jin et al., 2014] Jin, Y., Hao, J.-K. e Hamiez, J.-P. (2014). A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43:318–327.

[Johnson et al., 1991] Johnson, D. S., Aragon, C. R., McGeoch, L. A. e Schevon, C. (1991). Optimization by simulated annealing: An experimental evaluation; Part II, graph coloring and number partitioning. *Operations Research*, 39:378–406.

[Kokosiński, 2011] Kokosiński, Z. (2011). Parallel metaheuristics in graph coloring. In *The First International Conference on Automatic Control and Information Technology*, pages 209–214, Lviv.

[Kokosiński e Kwarciany, 2007] Kokosiński, Z. e Kwarciany, K. (2007). On sum coloring of graphs with parallel genetic algorithms. In Beliczynski, B., Dzielinski, A., Iwanowski, M. e Ribeiro, B., editors, *Adaptive and Natural Computing Algorithms*, volume 4431 of *Lecture Notes in Computer Science*, pages 211–219. Springer, Berlin.

[Kosowski, 2009] Kosowski, A. (2009). A note on the strength and minimum color sum of bipartite graphs. *Discrete Applied Mathematics*, 157:2552–2554.

[Kroon et al., 1997] Kroon, L., Sen, A., Deng, H. e Roy, A. (1997). The optimal cost chromatic partition problem for trees and interval graphs. In d'Amore, F., Franciosa, P. e Marchetti-Spaccamela, A., editors, *Graph-Theoretic Concepts in Computer Science*, volume 1197 of *Lecture Notes in Computer Science*, pages 279–292. Springer, Berlin.

[Kubale e Jackowski, 1985] Kubale, M. e Jackowski, B. (1985). A generalized implicit enumeration algorithm for graph coloring. *Communications of the ACM*, 28:412–418.

[Kubicka, 1989] Kubicka, E. (1989). *The Chromatic Sum and Efficient Tree Algorithms*. PhD thesis, Western Michigan University, Kalamazoo.

[Kubicka, 2004] Kubicka, E. (2004). The chromatic sum of a graph: History and recent developments. *International Journal of Mathematics and Mathematical Sciences*, 2004:1563–1573.

[Kubicka, 2005] Kubicka, E. (2005). Polynomial algorithm for finding chromatic sum for unicyclic and outerplanar graphs. *ARS COMBINATORIA*, 76:193–202.

[Kubicka e Schwenk, 1989] Kubicka, E. e Schwenk, A. J. (1989). An introduction to chromatic sums. In *Proceedings of the ACM Seventeenth Annual Computer Science Conference*, pages 39–45. ACM Press.

[Laguna e Martí, 2001] Laguna, M. e Martí, R. (2001). A GRASP for coloring sparse graphs. *Computational Optimization and Applications*, 19:165–178.

[Lai e Hao, 2015] Lai, X. e Hao, J.-K. (2015). Path relinking for the fixed spectrum frequency assignment problem. *Expert Systems with Applications*, 42:4755–4767.

[Lai et al., 2014] Lai, X., Lü, Z., Hao, J., Glover, F. e Xu, L. (2014). Path relinking for bandwidth coloring problem. Referência online em: `https://arxiv.org/abs/1409.0973`, último acesso em: 20/01/2018.

[Lalla-Ruiz et al., 2016] Lalla-Ruiz, E., Expósito-Izquierdo, C., Melián-Batista, B. e Moreno-Vega, J. M. (2016). A hybrid biased random key genetic algorithm for the quadratic assignment problem. *Information Processing Letters*, 116:513–520.

[Lalla-Ruiz et al., 2014] Lalla-Ruiz, E., González-Velarde, J. L., Melián-Batista, B. e Moreno-Vega, J. M. (2014). Biased random key genetic algorithm for the tactical berth allocation problem. *Applied Soft Computing*, 22:60–76.

[Leighton, 1979] Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84:489–506.

[Lim et al., 2005] Lim, A., Zhu, Y., Lou, Q. e Rodrigues, B. (2005). Heuristic methods for graph coloring problems. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 933–939, New York. ACM.

[Lü e Hao, 2010] Lü, Z. e Hao, J.-K. (2010). A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203:241–250.

[Ma et al., 2017] Ma, C., Kong, M., Pei, J. e Pardalos, P. M. (2017). BRKGA-VNS for parallel-batching scheduling on a single machine with step-deteriorating jobs and release times. In Nicosia, G., Pardalos, P. M., Giuffrida, G. e Umeton, R., editors, *Machine Learning, Optimization, and Big Data - Third International Conference, Volterra, 2017*, volume 10710 of *Lecture Notes in Computer Science*, pages 414–425. Springer.

[Malafiejski et al., 2004] Malafiejski, M., Giaro, K., Janczewski, R. e Kubale, M. (2004). Sum coloring of bipartite graphs with bounded degree. *Algorithmica*, 40:235–244.

[Malaguti et al., 2008] Malaguti, E., Monaci, M. e Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20:302–316.

[Malaguti et al., 2009] Malaguti, E., Monaci, M. e Toth, P. (2009). Models and heuristic algorithms for a weighted vertex coloring problem. *Journal of Heuristics*, 15:503–526.

[Malaguti et al., 2011] Malaguti, E., Monaci, M. e Toth, P. (2011). An exact approach for the vertex coloring problem. *Discrete Optimization*, 8:174–190.

[Malaguti e Toth, 2010] Malaguti, E. e Toth, P. (2010). A survey on vertex coloring problems. *International Transactions in Operational Research*, 17:1–34.

[Malkawi et al., 2008] Malkawi, M., Hassan, M. A.-H. e Hassan, O. A.-H. (2008). A new exam scheduling algorithm using graph coloring. *International Arab Journal of Information Technology*, 5:80–86.

[Matsumoto e Nishimura, 1998] Matsumoto, M. e Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30.

[McDiarmid e Reed, 2000] McDiarmid, C. e Reed, B. (2000). Channel assignment and weighted coloring. *Networks*, 36:114–117.

[Mead e Conway, 1980] Mead, C. e Conway, L. (1980). *Introduction to VLSI Systems*. Addison-Wesley, Boston.

[Mehrotra e Trick, 1996] Mehrotra, A. e Trick, M. A. (1996). A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354.

[Mehrotra e Trick, 2007] Mehrotra, A. e Trick, M. A. (2007). *A Branch-And-Price Approach for Graph Multi-Coloring*, pages 15–29. Springer, Boston.

[Méndez-Díaz e Zabala, 2008] Méndez-Díaz, I. e Zabala, P. (2008). A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156:159–179.

[Mladenović e Hansen, 1997] Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100.

[Moalic e Gondran, 2015] Moalic, L. e Gondran, A. (2015). The new memetic algorithm *HEAD* for graph coloring: An easy way for managing diversity. In Ochoa, G. e Chicano, F., editors, *Evolutionary Computation in Combinatorial Optimization: 15th European Conference*, volume 9026 of *Lecture Notes in Computer Science*, pages 173–183. Springer.

[Morgenstern, 1996] Morgenstern, C. (1996). Distributed coloration neighborhood search. In Johnson, D. S. e Trick, M. A., editors, *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 335–358. American Mathematical Society.

[Moukrim et al., 2013] Moukrim, A., Sghiouer, K., Lucet, C. e Y, L. (2013). Upper and lower bounds for the minimum sum coloring problem. Referência online em: `https://www.hds.utc.fr/~moukrim/dokuwiki/doku.php?id=en:mscp`, último acesso em: 14/01/2018.

[Narayanan e Shende, 2001] Narayanan, L. e Shende, S. M. (2001). Static frequency assignment in cellular networks. *Algorithmica*, 29:396–409.

[Nicoloso et al., 1999] Nicoloso, S., Sarrafzadeh, M. e Song, X. (1999). On the sum coloring problem on interval graphs. *Algorithmica*, 23:109–126.

[Noronha et al., 2011] Noronha, T. F., Resende, M. G. C. e Ribeiro, C. C. (2011). A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization*, 50:503–518.

[Oliveira et al., 2011] Oliveira, C., Noronha, T. F. e Urrutia, S. (2011). Heurística vnd com backtracking para o problema de coloração de vértices com pesos. In *Proceedings of the XLIII Simpósio Brasileiro de Pesquisa Operacional*, pages 1687–1698, Ubatuba.

[Pereira e Palsberg, 2005] Pereira, F. M. Q. e Palsberg, J. (2005). Register allocation via coloring of chordal graphs. In Yi, K., editor, *Programming Languages and Systems*, volume 3780 of *Lecture Notes in Computer Science*, pages 315–329. Springer, Berlin.

[Pessoa et al., 2013] Pessoa, L. S., Resende, M. G. e Ribeiro, C. C. (2013). A hybrid lagrangean heuristic with grasp and path-relinking for set k-covering. *Computers & Operations Research*, 40:3132–3146.

[Plumettaz et al., 2010] Plumettaz, M., Schindl, D. e Zufferey, N. (2010). Ant local search and its efficient adaptation to graph colouring. *Journal of the Operational Research Society*, 61:819–826.

[Prais e Ribeiro, 2000] Prais, M. e Ribeiro, C. C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176.

[Resende et al., 2010] Resende, M., Martí, R., Gallego, M. e Duarte, A. (2010). GRASP and path relinking for the max-min diversity problem. *Computers & Operations Research*, 37:498–508.

[Resende, 2012] Resende, M. G. C. (2012). Biased random-key genetic algorithms with applications in telecommunications. *TOP*, 20:130–153.

[Resende e Ribeiro, 2011] Resende, M. G. C. e Ribeiro, C. C. (2011). Restart strategies for GRASP with path-relinking heuristics. *Optimization Letters*, 5:467–478.

[Resende e Ribeiro, 2016] Resende, M. G. C. e Ribeiro, C. C. (2016). *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer.

[Ribeiro et al., 1989] Ribeiro, C. C., Minoux, M. e Penna, M. C. (1989). An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. *European Journal of Operational Research*, 41:232–239.

[Ribeiro e Resende, 2012] Ribeiro, C. C. e Resende, M. G. C. (2012). Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics*, 18:193–214.

[Ribeiro e Rosseti, 2015] Ribeiro, C. C. e Rosseti, I. (2015). tttplots-compare: A perl program to compare time-to-target plots or general runtime distributions of randomized algorithms. *Optimization Letters*, 9:601–614.

[Ribeiro et al., 2012] Ribeiro, C. C., Rosseti, I. e Vallejos, R. (2012). Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54:405–429.

[Ribeiro et al., 2002] Ribeiro, C. C., Uchoa, E. e Werneck, R. F. (2002). A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246.

[Sager e Lin, 1991] Sager, T. J. e Lin, S.-J. (1991). A pruning procedure for exact graph coloring. *ORSA Journal on Computing*, 3:226–230.

[Salavatipour, 2003] Salavatipour, M. R. (2003). On sum coloring of graphs. *Discrete Applied Mathematics*, 127:477–488.

[Segundo, 2012] Segundo, P. S. (2012). A new DSATUR-based algorithm for exact vertex coloring. *Computers & Operations Research*, 39:1724–1733.

[Sen et al., 1992] Sen, A., Deng, H. e Guha, S. (1992). On a graph partition problem with application to VLSI layout. *Information Processing Letters*, 43:87–94.

[Sewell, 1996] Sewell, E. C. (1996). An improved algorithm for exact graph coloring. In Johnson, D. S. e Trick, M. A., editors, *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 359–373. American Mathematical Society.

[Spears e Jong, 1991] Spears, W. e Jong, d. (1991). On the virtues of parameterized uniform crossover. In Belew, R. e Booker, L., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo. Morgan Kaufman.

[Stefanello et al., 2017] Stefanello, F., Buriol, L. S., Hirsch, M. J., Pardalos, P. M., Querido, T., Resende, M. G. C. e Ritt, M. (2017). On the minimization of traffic congestion in road networks with tolls. *Annals of Operations Research*, 249:119–139.

[Stützle e Hoos, 1999] Stützle, T. e Hoos, H. H. (1999). Analyzing the run-time behaviour of iterated local search for the TSP. In *III Metaheuristics International Conference*, pages 449–453, Angra dos Reis.

[Supowit, 1987] Supowit, K. (1987). Finding a maximum planar subset of a set of nets in a channel. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6:93–94.

[Szkaliczki, 1999] Szkaliczki, T. (1999). Routing with minimum wire length in the dogleg-free manhattan model is np-complete. *SIAM Journal on Computing*, 29:274–287.

[Thomassen et al., 1989] Thomassen, C., Erdös, P., Alavi, Y., Malde, P. J. e Schwenk, A. J. (1989). Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13:353–357.

[Titiloye e Crispin, 2011] Titiloye, O. e Crispin, A. (2011). Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8:376–384.

[Toso e Resende, 2015] Toso, R. e Resende, M. (2015). A c++ application programming interface for biased random-key genetic algorithms. *Optimization Methods & Software*, 30:81–93.

[Wang et al., 2013] Wang, Y., Hao, J., Glover, F. e Lü, Z. (2013). Solving the minimum sum coloring problem via binary quadratic programming. Referência online em: `https://arxiv.org/abs/1304.5876`, último acesso em: 10/01/2018.

[Woo et al., 1991] Woo, T.-K., Su, S. Y. W. e Newman-Wolfe, R. (1991). Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. *IEEE Transactions on Communications*, 39:1794–1801.

[Wu e Hao, 2012a] Wu, Q. e Hao, J.-K. (2012a). Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39:283–290.

[Wu e Hao, 2012b] Wu, Q. e Hao, J.-K. (2012b). An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39:1593–1600.

[Zheng et al., 2015] Zheng, J.-N., Chien, C.-F. e Gen, M. (2015). Multi-objective multi-population biased random-key genetic algorithm for the 3-d container loading problem. *Computers & Industrial Engineering*, 89:80–87.

APÊNDICE A – Resultados detalhados das heurísticas BRKGA e BRKGA+RVNS

					BRKGA					BRKGA+RVNS					Tempo de
Instância	n	m	Melhor valor	CPLEX (3600 s)	Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	execução heurísticas (s)
school1-nsh	352	14612	7647.00	13193.00	8999.00	9212.50	0	20.47	239.8	9020.00	9568.40	0	25.13	114.6	30.95
school1	385	19095	7158.00	14687.00	7723.00	8026.00	0	12.13	257.0	7733.00	8222.30	0	14.87	113.4	36.03
3-FullIns_4	405	3524	2951.00	**2951.00**	**2951.00**	2951.00	10	0.00	30.2	**2951.00**	2951.00	10	0.00	13.7	29.85
fpsol2.i.3	425	8688	3738.00	**3738.00**	**3738.00**	3738.00	10	0.00	170.4	**3738.00**	3738.00	10	0.00	52.0	35.15
le450_5c	450	9803	2610.00	**2610.00**	2642.00	2671.10	0	2.34	415.2	2645.00	2695.70	0	3.28	119.6	45.98
le450_5d	450	9757	2700.00	3421.00	2710.00	2742.80	0	1.59	314.8	2711.00	2743.60	0	1.61	132.6	47.64
le450_15c	450	16680	9556.00	11335.00	11131.00	11185.20	0	17.05	299.6	10683.00	10792.50	0	12.94	139.2	55.07
le450_15d	450	16750	10799.00	13011.00	12629.00	13004.60	0	20.42	278.2	12441.00	12518.80	0	15.93	139.5	54.71
le450_25a	450	8260	9730.00	**9730.00**	10258.00	10375.10	0	6.63	130.8	10485.00	10540.30	0	8.33	144.4	38.63
le450_25b	450	8263	7564.00	**7564.00**	8028.00	8099.60	0	7.08	173.2	8241.00	8277.10	0	9.43	144.6	42.11
le450_25c	450	17343	10447.00	11776.00	11940.00	11989.70	0	14.77	375.9	11643.00	11714.90	0	12.14	111.7	54.25
le450_25d	450	17425	11676.00	12791.00	13074.00	13164.00	0	12.74	333.9	12816.00	12894.50	0	10.44	102.0	54.21
fpsol2.i.2	451	8691	4694.00	**4694.00**	**4694.00**	4694.80	5	0.02	372.5	**4694.00**	4695.50	4	0.03	120.4	38.47
4-Insertions_4	475	1795	999.00	**999.00**	1001.00	1006.50	0	0.75	415.7	**999.00**	1005.70	2	0.67	109.7	27.18
fpsol2.i.1	496	11654	8364.00	**8364.00**	**8364.00**	8364.20	8	0.00	315.5	**8364.00**	8364.00	10	0.00	77.3	28.50
DSJC500.5	500	62624	18333.00	22845.00	22935.00	23048.20	0	25.72	277.8	21892.00	22258.50	0	21.41	108.7	152.42
C500.9	500	112332	63147.00	77015.00	71823.00	74012.10	0	17.21	281.1	74360.00	75039.80	0	18.83	112.5	276.35
DSJC500.9	500	112437	65373.00	78869.00	72535.00	73687.00	0	12.72	385.9	74194.00	75362.90	0	15.28	98.3	290.00
DSJR500.1	500	3555	6253.00	**6253.00**	6646.00	6682.40	0	6.87	276.0	6724.00	6773.70	0	8.33	145.1	46.68
DSJR500.1c	500	121275	27395.00	35554.00	27575.00	27752.50	0	1.30	200.1	27462.00	27804.10	0	1.49	98.0	149.89
DSJR500.5	500	58862	54392.00	64892.00	56453.00	57503.40	0	5.72	181.8	58269.00	59311.90	0	9.05	101.3	191.11
2-Insertions_5	597	3936	2999.00	**2999.00**	**2999.00**	3002.80	4	0.13	341.2	**2999.00**	3007.60	2	0.29	107.7	58.76
1-Insertions_6	607	6337	1347.00	1367.00	**1347.00**	1350.30	2	0.24	354.8	**1347.00**	1351.00	1	0.30	82.4	63.88
inithx.i.3	621	13969	3633.00	**3633.00**	**3633.00**	3633.80	7	0.02	259.2	**3633.00**	3633.00	10	0.00	67.2	96.46
inithx.i.2	645	13979	4073.00	**4073.00**	**4073.00**	4073.50	7	0.01	320.9	**4073.00**	4073.00	10	0.00	84.5	95.31

Tabela A.1: Resultados detalhados das heurísticas BRKGA e BRKGA+RVNS - Parte I.

Instância	n	m	Melhor valor	CPLEX (3600 s)	BRKGA				BRKGA+RVNS				Tempo de execução heurísticas (s)
					Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	
ash331GPIA	662	4185	1513.00	**1513.00**	1537.00	1561.20	0	3.19	420.4	1539.00	1565.20	0	3.45	121.1	65.85
4-FullIns_4	690	6650	2443.00	**2443.00**	**2443.00**	2443.40	9	0.02	191.8	**2443.00**	2444.80	5	0.07	76.5	82.17
will199GPIA	701	7065	4829.00	5428.00	4919.00	4977.40	0	3.07	212.9	4948.00	5015.90	0	3.87	144.5	79.01
inithx.i.1	864	18707	3934.00	**3934.00**	**3934.00**	3936.10	3	0.05	381.3	**3934.00**	3934.20	9	0.01	79.7	106.34
qg.order30	900	26100	11940.00	**11940.00**	12084.00	12105.00	0	1.38	308.3	12027.00	12052.30	0	0.94	76.5	270.78
latin_sqr_10	900	307350	48822.00	—	57949.00	58717.60	0	20.27	232.7	56677.00	57003.10	0	16.76	136.2	620.77
wap05	905	43081	12593.00	14181.00	13803.00	13873.30	0	10.17	201.3	14039.00	14143.00	0	12.31	103.7	254.86
wap06	947	43571	18453.00	20434.00	19495.00	19631.40	0	6.39	173.5	19905.00	19972.10	0	8.23	115.5	267.59
DSJC1000.5	1000	249826	46790.00	—	58453.00	58906.70	0	25.90	357.5	57538.00	57779.70	0	23.49	134.3	920.84
flat1000_50_0	1000	245000	41915.00	—	51719.00	52485.90	0	25.22	372.3	51032.00	51513.50	0	22.90	118.5	934.90
flat1000_60_0	1000	245830	40468.00	—	51900.00	52145.60	0	28.86	246.7	50629.00	51088.20	0	26.24	103.5	908.07
flat1000_76_0	1000	246708	41729.00	—	51205.00	51574.60	0	23.59	344.6	50425.00	50632.00	0	21.34	129.3	920.24
DSJC1000.9	1000	449449	103906.00	—	130627.00	132574.80	0	27.59	359.8	129308.00	130002.40	0	25.12	111.9	1869.64
C1000.9	1000	450079	105709.00	—	134237.00	135543.50	0	28.22	275.9	131064.00	132631.40	0	25.47	134.8	1874.78
5-FullIns_4	1085	11395	2212.00	**2212.00**	**2212.00**	2212.00	10	0.00	178.2	**2212.00**	2212.30	9	0.01	61.2	192.97
ash608GPIA	1216	7844	3859.00	4215.00	**3859.00**	3875.20	1	0.42	260.3	3866.00	3909.90	0	1.32	156.8	254.75
3-Insertions_5	1406	9695	1406.00	**1406.00**	**1406.00**	1406.00	10	0.00	36.3	**1406.00**	1406.00	10	0.00	3.7	262.40
abb313GPIA	1557	65390	4597.00	—	**4597.00**	4625.90	1	0.63	350.4	4655.00	4774.80	0	3.87	115.6	562.43
qg.order40	1600	62400	15280.00	—	15486.00	15506.10	0	1.48	354.4	15438.00	15453.00	0	1.13	116.7	1053.18
wap07	1809	103368	13380.00	—	15113.00	15173.30	0	13.40	307.9	15205.00	15285.20	0	14.24	133.0	1071.05
wap08	1870	104176	14497.00	—	15790.00	15845.60	0	9.30	260.5	15855.00	16009.50	0	10.43	117.5	1106.13
ash958GPIA	1916	12506	2886.00	3171.00	**2886.00**	2906.50	1	0.71	263.3	2927.00	2949.20	0	2.19	146.3	606.65
3-FullIns_5	2030	33751	4082.00	5845.00	**4082.00**	4085.80	4	0.09	289.5	**4082.00**	4094.60	1	0.31	82.8	872.03
wap01	2368	110871	18719.00	—	20584.00	20680.90	0	10.48	252.2	20917.00	21041.30	0	12.41	110.7	1842.69
wap02	2464	111742	17439.00	—	18853.00	18915.50	0	8.47	267.0	19145.00	19232.00	0	10.28	118.3	1920.90

Tabela A.2: Resultados detalhados das heurísticas BRKGA e BRKGA+RVNS - Parte II.

APÊNDICE B – Resultados detalhados das heurísticas BRKGA, BRKGA+RVNS e HBLPR

Instância	n	m	Melhor valor	CPLEX (3600 s)	BRKGA				BRKGA+RVNS				HBLPR				Tempo de execução heurísticas (s)
					Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor iteração	
school1-nsh	352	14612	7647.00	13193.00	8999.00	9212.50	0	20.47	239.8	9020.00	9568.40	0	25.13	114.6	**7647.00**	7658.90	6	0.16	17514.3	30.95
school1	385	19095	7158.00	14687.00	7723.00	8026.00	0	12.13	257.0	7733.00	8222.30	0	14.87	113.4	**7158.00**	7159.20	5	0.02	12003.1	36.03
3-FullIns_4	405	3524	2951.00	**2951.00**	**2951.00**	2951.00	10	0.00	30.2	**2951.00**	2951.00	10	0.00	13.7	**2951.00**	2951.60	9	0.02	2282.7	29.85
fpsol2.i.3	425	8688	3738.00	**3738.00**	**3738.00**	3738.00	10	0.00	170.4	**3738.00**	3738.00	10	0.00	52.0	**3738.00**	3738.30	7	0.01	2318.5	35.15
le450_5c	450	9803	2610.00	**2610.00**	2642.00	2671.10	0	2.34	415.2	2645.00	2695.70	0	3.28	119.6	**2610.00**	2610.00	10	0.00	1.0	45.98
le450_5d	450	9757	2700.00	3421.00	2710.00	2742.80	0	1.59	314.8	2711.00	2743.60	0	1.61	132.6	**2700.00**	2711.40	4	0.42	2502.9	47.64
le450_15c	450	16680	9556.00	11335.00	11131.00	11185.20	0	17.05	299.6	10683.00	10792.50	0	12.94	139.2	**9556.00**	9729.90	1	1.82	31159.0	55.07
le450_15d	450	16750	10799.00	13011.00	12629.00	13004.60	0	20.42	278.2	12441.00	12518.80	0	15.93	139.5	**10799.00**	10965.20	1	1.54	30700.0	54.71
le450_25a	450	8260	9730.00	**9730.00**	10258.00	10375.10	0	6.63	130.8	10485.00	10540.30	0	8.33	144.4	9875.00	9962.60	0	2.39	26305.4	38.63
le450_25b	450	8263	7564.00	**7564.00**	8028.00	8099.60	0	7.08	173.2	8241.00	8277.10	0	9.43	144.6	7765.00	7816.80	0	3.34	23571.0	42.11
le450_25c	450	17343	10447.00	11776.00	11940.00	11989.70	0	14.77	375.9	11643.00	11714.90	0	12.14	111.7	**10447.00**	10586.10	1	1.33	30043.7	54.25
le450_25d	450	17425	11676.00	12791.00	13074.00	13164.00	0	12.74	333.9	12816.00	12894.50	0	10.44	102.0	**11676.00**	11755.80	1	0.68	32653.7	54.21
fpsol2i.2	451	8691	4694.00	**4694.00**	**4694.00**	4694.80	5	0.02	372.5	**4694.00**	4695.50	4	0.03	120.4	**4694.00**	4695.40	4	0.03	5424.0	38.47
4-Insertions_4	475	1795	999.00	**999.00**	1001.00	1006.50	0	0.75	415.7	**999.00**	1005.70	2	0.67	109.7	1035.00	1035.00	0	3.60	665.5	27.18
fpsol2i.1	496	11654	8364.00	**8364.00**	**8364.00**	8364.20	8	0.00	315.5	**8364.00**	8364.00	10	0.00	77.3	8365.00	8365.00	0	0.01	1832.7	28.50
DSJC500.5	500	62624	18333.00	22845.00	22935.00	23048.20	0	25.72	277.8	21892.00	22258.50	0	21.41	108.7	**18333.00**	18678.20	1	1.88	32214.5	152.42
C500.9	500	112332	63147.00	77015.00	71823.00	74012.10	0	17.21	281.1	74360.00	75039.80	0	18.83	112.5	**63147.00**	63806.90	1	1.05	19492.4	276.35
DSJC500.9	500	112437	65373.00	78869.00	72535.00	73687.00	0	12.72	385.9	74194.00	75362.90	0	15.28	98.3	**65373.00**	65904.60	1	0.81	22769.1	290.00
DSJR500.1	500	3555	6253.00	**6253.00**	6646.00	6682.40	0	6.87	276.0	6724.00	6773.70	0	8.33	145.1	6484.00	6518.00	0	4.24	29492.8	46.68
DSJR500.1c	500	121275	27395.00	35554.00	27575.00	27752.50	0	1.30	200.1	27462.00	27804.10	0	1.49	98.0	**27395.00**	27569.40	1	0.64	18217.3	149.89
DSJR500.5	500	58862	54392.00	64892.00	56453.00	57503.40	0	5.72	181.8	58269.00	59311.90	0	9.05	101.3	**54392.00**	54660.20	1	0.49	15783.7	191.11
2-Insertions_5	597	3936	2999.00	**2999.00**	**2999.00**	3002.80	4	0.13	341.2	**2999.00**	3007.60	2	0.29	107.7	3080.00	3087.70	0	2.96	20442.8	58.76
1-Insertions_6	607	6337	1347.00	1367.00	**1347.00**	1350.30	2	0.24	354.8	**1347.00**	1351.00	1	0.30	82.4	**1347.00**	1360.00	8	0.97	11714.6	63.88
inithx.i.3	621	13969	3633.00	**3633.00**	**3633.00**	3633.80	7	0.02	259.2	**3633.00**	3633.00	10	0.00	67.2	**3633.00**	3636.90	3	0.11	2820.9	96.46
inithx.i.2	645	13979	4073.00	**4073.00**	**4073.00**	4073.50	7	0.01	320.9	**4073.00**	4073.00	10	0.00	84.5	**4073.00**	4077.00	8	0.10	4387.5	95.31

Tabela B.1: Resultados detalhados das heurísticas BRKGA, BRKGA+RVNS e HBLPR para as 50 instâncias - Parte I.

Instância	n	m	Melhor valor	CPLEX (3600 s)	BRKGA					BRKGA+RVNS					HBLPR					Tempo de execução heurísticas (s)
					Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor geração	Melhor valor	Valor médio	# Melhor valor	Avg Dev (%)	Média melhor iteração	
ash331GPIA	662	4185	1513.00	**1513.00**	1537.00	1561.20	0	3.19	420.4	1539.00	1565.20	0	3.45	121.1	1554.00	1574.90	0	4.09	33298.0	65.85
4-FullIns_4	690	6650	2443.00	**2443.00**	**2443.00**	2443.40	9	0.02	191.8	**2443.00**	2444.80	5	0.07	76.5	2447.00	2450.00	0	0.29	7678.9	82.17
will199GPIA	701	7065	4829.00	5428.00	4919.00	4977.40	0	3.07	212.9	4948.00	5015.90	0	3.87	144.5	**4829.00**	4923.70	2	1.96	30939.5	79.01
inithx.i.1	864	18707	3934.00	**3934.00**	**3934.00**	3936.10	3	0.05	381.3	**3934.00**	3934.20	9	0.01	79.7	**3934.00**	3935.20	4	0.03	1257.0	106.34
qg.order30	900	26100	11940.00	**11940.00**	12084.00	12105.00	0	1.38	308.3	12027.00	12052.30	0	0.94	76.5	**11940.00**	11945.50	1	0.05	18618.6	270.78
latin_sqr_10	900	307350	48822.00	-	57949.00	58717.60	0	20.27	232.7	56677.00	57003.10	0	16.76	136.2	**48822.00**	49108.00	1	0.59	16879.1	620.77
wap05	905	43081	12593.00	14181.00	13803.00	13873.30	0	10.17	201.3	14039.00	14143.00	0	12.31	103.7	**12593.00**	12652.90	1	0.48	26149.4	254.86
wap06	947	43571	18453.00	20434.00	19495.00	19631.40	0	6.39	173.5	19905.00	19972.10	0	8.23	115.5	**18453.00**	18565.90	1	0.61	23501.2	267.59
DSJC1000.5	1000	249826	46790.00	-	58453.00	58906.70	0	25.90	357.5	57538.00	57779.70	0	23.49	134.3	**46790.00**	47272.40	1	1.03	33067.0	920.84
flat1000_50_0	1000	245000	41915.00	-	51719.00	52485.90	0	25.22	372.3	51032.00	51513.50	0	22.90	118.5	**41915.00**	42441.50	1	1.26	32380.6	934.90
flat1000_60_0	1000	245830	40468.00	-	51900.00	52145.60	0	28.86	246.7	50629.00	51088.20	0	26.24	103.5	**40468.00**	41110.60	1	1.59	29948.8	908.07
flat1000_76_0	1000	246708	41729.00	-	51205.00	51574.60	0	23.59	344.6	50425.00	50632.00	0	21.34	129.3	**41729.00**	42040.00	1	0.75	30091.8	920.24
DSJC1000.9	1000	449449	103906.00	-	130627.00	132574.80	0	27.59	359.8	129308.00	130002.40	0	25.12	111.9	**103906.00**	104748.70	1	0.81	25742.4	1869.64
C1000.9	1000	450079	105709.00	-	134237.00	135543.50	0	28.22	275.9	131064.00	132631.40	0	25.47	134.8	**105709.00**	106807.80	1	1.04	23889.3	1874.78
5-FullIns_4	1085	11395	2212.00	**2212.00**	**2212.00**	2212.00	10	0.00	178.2	**2212.00**	2212.30	9	0.01	61.2	**2212.00**	2214.10	7	0.09	578.5	192.97
ash608GPIA	1216	7844	3859.00	4215.00	**3859.00**	3875.20	1	0.42	260.3	3866.00	3909.90	0	1.32	156.8	3922.00	3931.60	0	1.88	36678.9	254.75
3-Insertions_5	1406	9695	1406.00	**1406.00**	**1406.00**	1406.00	10	0.00	36.3	**1406.00**	1406.00	10	0.00	3.7	**1406.00**	1406.00	10	0.00	1.0	262.40
abb313GPIA	1557	65390	4597.00	-	**4597.00**	4625.90	1	0.63	350.4	4655.00	4774.80	0	3.87	115.6	4847.00	4878.80	0	6.13	24751.0	562.43
qg.order40	1600	62400	15280.00	-	15486.00	15506.10	0	1.48	354.4	15438.00	15453.00	0	1.13	116.7	15285.00	15289.00	0	0.06	24849.8	1053.18
wap07	1809	103368	13380.00	-	15113.00	15173.30	0	13.40	307.9	15205.00	15285.20	0	14.24	133.0	**13380.00**	13463.50	1	0.62	40110.5	1071.05
wap08	1870	104176	14497.00	-	15790.00	15845.60	0	9.30	260.5	15855.00	16009.50	0	10.43	117.5	**14497.00**	14564.90	1	0.47	34883.6	1106.13
ash958GPIA	1916	12506	2886.00	3171.00	**2886.00**	2906.50	1	0.71	263.3	2927.00	2949.20	0	2.19	146.3	2897.00	2909.10	0	0.80	40967.1	606.65
2-FullIns_5	2030	33751	4082.00	5845.00	**4082.00**	4085.80	4	0.09	289.5	**4082.00**	4094.60	1	0.31	82.8	**4082.00**	4082.00	10	0.00	2569.4	872.03
wap01	2368	110871	18719.00	-	20584.00	20680.90	0	10.48	252.2	20917.00	21041.30	0	12.41	110.7	**18719.00**	18798.70	1	0.43	43886.8	1842.69
wap02	2464	111742	17439.00	-	18853.00	18915.50	0	8.47	267.0	19145.00	19232.00	0	10.28	118.3	**17439.00**	17487.60	1	0.28	36614.2	1920.90

Tabela B.2: Resultados detalhados das heurísticas BRKGA, BRKGA+RVNS e HBLPR para as 50 instâncias - Parte II.