O PROBLEMA DO SUBGRAFO BICONEXO MÍNIMO GENERALIZADO: ALGORITMOS E FORMULAÇÕES

RAFAEL SANTOS COELHO

O PROBLEMA DO SUBGRAFO BICONEXO MÍNIMO GENERALIZADO: ALGORITMOS E FORMULAÇÕES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: ALEXANDRE SALLES DA CUNHA

Belo Horizonte

Março de 2012

Coelho, Rafael Santos
C672p O Problema do Subgrafo Biconexo Mínimo Generalizado: Algoritmos e Formulações / Rafael Santos Coelho. — Belo Horizonte, 2012
xxii, 86 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Alexandre Salles da Cunha

1. Computação – Teses. 2. Teoria dos grafos – Teses. 3. Redes de computadores – Teses. 4. Algoritmos de computador – Teses. I. Orientador. II. Título

CDU 519.6*62(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

O problema do subgrafo biconexo mínimo generalizado: Algoritmos e formulações

RAFAEL SANTOS COELHO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. ALEXANDRE SALLES DA CUNHA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. CARLOS ROBERTO VENÂNCIO DE CARVALHO
Departamento de Engenharia de Produção - UFMG

PROF. LUIDI GELABERT SIMONETTI
Centro Tecnológico - UFF

Belo Horizonte, 01 de março de 2012.

Dedico esta dissertação de mestrado a meus pais, Lúcia e Alberto, a minha irmã, Ana, e aos meus amigos e amigas.

Agradecimentos

Agradeço imensamente a minha família, meus pais Lúcia e Alberto, e minha irmã, Ana. Sem o amor incondicional deles, eu não teria chegado aonde cheguei hoje. Agradeço a todos meus amigos e amigas, especialmente Stélio Henriques, Ânderson Bispo, Anderson Gonzaga, Aline Bessa, Mariana Pandolfi e Thayla Fernandes pelo constante apoio que eles e elas me deram e continuam dando. Agradeço também aos meus colegas da UFMG, companheiros de caminhada, Leonardo Conegundes, Dilson Lucas, Tiago Januário, Amadeu Almeida, Vitor Andrade, Vinicius Morais, Franklin Assunção, Marco Túlio e muitos outros, pelos momentos de descontração que passamos juntos e por toda a ajuda que já recebi deles. Por fim, mas não menos importante, agradeço muito meu orientador, Alexandre, pela sua paciência, franqueza e camaradagem e, principalmente, por ter aberto minha mente a novos caminhos e desafios.

"My work consists of two parts: of the one which is here, and of everything which I have not written. And precisely this second part is the important one."

(Ludwig Wittgenstein)

Resumo

Dados um número inteiro $k > 2$ e um grafo simples não-direcionado $G = (V, E)$ com pesos reais positivos nas arestas, onde V está particionado em k subconjuntos (ou *clusters*), o Problema do Subgrafo Biconexo Mínimo Generalizado (PSBMG) equivale a encontrar um subgrafo biconexo de custo mínimo S de G que contenha exatamente um vértice de cada *cluster* de G. Neste trabalho, foram propostos um algoritmo exato *Branch-and-cut* e heurísticas para solucionar o PSBMG. O algoritmo *Branch-and-cut* separa quatro classes de desigualdades válidas para o problema em questão, todas elas generalizadas de outros problemas da literatura relacionados ao PSBMG. Para avaliar as abordagens de solução introduzidas, foram realizados testes computacionais em instâncias de *benchmark* densas e esparsas. Para um número considerável delas, foi possível prover novos certificados de otimalidade e melhores limites inferiores e superiores.

Palavras-chave: Biconexidade em Teoria de Grafos, Projeto de Redes Resilientes, Algoritmos Branch-and-cut.

Abstract

Given an integer number $k > 2$ and an undirected simple graph $G = (V, E)$ with positive real weights assigned to its edges, where V is partitioned into k subsets (or clusters), the Generalized Minimum Biconnected Subgraph Problem (GMBSP) amounts to finding a minimum-cost biconnected subgraph S of G, such that S contains exactly one vertex from every cluster of G. In this work, we propose an exact Branch-and-cut algorithm and some heuristics to solve PSBMG. Our Branch-and-cut algorithm separates four classes of valid inequalities for GMBSP, all of them adapted from valid inequalities for other problems related to GMBSP. In order to better assess our solution approaches, we conducted several computational experiments using dense and sparse benchmark instances. For a considerable fraction of those instances, we were able to provide new optimality certificates, as well as better upper and lower bounds.
Keywords: Biconnectivity in Graph Theory, Survivable Network Design Problems, Branch-and-cut Algorithms.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

No dia 8 de maio de 1991, o *Wall Street Journal* [Stoer, 1992] noticiou que o aeroporto internacional de O'Hare, em Chicago, nos Estados Unidos, havia entrado em colapso. Linhas de emergência 911 ficaram inalcançáveis e caixas eletrônicos inoperantes. O motivo do acidente foi um incêndio que danificou o maquinário de uma importante empresa de telecomunicações, ocasionando a desconexão da sua rede de telefonia. Episódios como esse não são raros. Uma lição que eles nos ensinam é a de que redes de comunicação precisam ser planejadas e construídas de modo a possuir *resiliência*.

Dizemos que uma rede de comunicação é considerada *resiliente* se ela é capaz de se manter em funcionamento mesmo após a ocorrência de falhas em seus componentes internos. A tarefa de se projetar uma rede resiliente é complicada e engloba múltiplas fases, uma delas, enfatizada neste trabalho, sendo o chamado desenho topológico da rede. Nessa etapa, o objetivo é decidir onde os nós da rede serão posicionados e, principalmente, como eles serão conectados uns aos outros. Para tanto, deve-se levar em conta, entre outros fatores, os custos de instalação e de intercomunicação dos nós.

No contexto de redes de computadores, com a popularização da tecnologia de fibra ótica devido a sua alta confiabilidade e largura de banda, métodos atuais de projeto de redes costumam favorecer topologias mais esparsas. Tal decisão aumenta o potencial risco de interrupção dos serviços providos na rede em decorrência de acidentes naturais ou danos físicos. Por essa razão, o projeto de redes resilientes ganha um papel crucial.

Do ponto de vista da Teoria de Grafos, a questão da resiliência em redes de computadores se espelha no conceito de conexidade. Uma forma de atingir resiliência, ao se projetar a topologia de uma rede, é garantir que existam múltiplos caminhos disjuntos entre qualquer par de nós da rede. Estudos já confirmaram que topologias biconexas conseguem, com baixo custo, prover um nível seguro de resiliência. Formalmente, um

grafo é chamado de *vértice-biconexo*, ou simplesmente *biconexo*, se existem, no mínimo, dois caminhos vértice-disjuntos entre quaisquer dois vértices distintos nesse grafo.

No âmbito da Otimização Combinatória, o problema do projeto de redes resilientes se encaixa em uma categoria mais ampla de problemas conhecidos na literatura acadêmica como problemas de projeto de redes. Dado um conjunto de restrições de conexidade e pesos associados aos vértices e arestas de um grafo G, o *Problema do Projeto de Redes* (PPR) busca a obtenção de um subgrafo de G que cumpra as restrições impostas e que possua *custo* mínimo, i.e., tal que a soma dos pesos dos vértices e arestas presentes na solução seja mínima.

O PPR tem recebido bastante atenção nos meios acadêmico e industrial recentemente. Em parte, isso se justifica pelo fato de que inúmeros problemas muito estudados em Otimização Combinatória, e.g., o *Problema do Caminho Mínimo*, o *Problema da Árvore Geradora Mínima*, entre outros, podem ser facilmente reduzidos a um caso particular do PPR, o que demonstra sua capacidade de generalização. Uma extensão relevante do PPR é o chamado *Problema do Projeto de Redes Generalizado* (PPRG), que se diferencia do PPR no aspecto de que suas restrições são expressas não para cada um dos vértices do grafo que modela a rede, mas para aglomerados de vértices (ou *clusters*). Diversos outros problemas ditos "generalizados", semelhantes ao PPRG, já foram estudados na literatura, entre eles citamos o *Problema da Árvore Geradora Mínima Generalizado* [Myung et al., 1995] e o *Problema do Caixeiro Viajante Generalizado* [Applegate et al., 2006].

Nesta dissertação, abordamos uma especialização NP-difícil do PPR, denominada *Problema do Subgrafo Biconexo Mínimo Generalizado* (PSBMG). Sendo G um grafo com pesos reais positivos nas arestas e com conjunto de vértices V particionado em k *clusters*, para algum inteiro $k > 2$, o PSBMG equivale a encontrar um subgrafo biconexo de G de custo mínimo contendo exatamente um vértice de cada *cluster*. Ao contrário da sua versão não-generalizada, i.e., do *Problema do Subgrafo Biconexo Mínimo* (PSBM), o PSBMG foi pouco estudado. O primeiro trabalho que propôs uma abordagem de solução para o PSBMG foi desenvolvido por Hu & Raidl [2009]. Desde então, pelo o que se tem conhecimento, somente mais um trabalho, conduzido por Pagacz et al. [2010], explorou o PSBMG.

1.1 Principais Contribuições

As principais contribuições desta dissertação são listadas a seguir na ordem em que aparecem no texto.

- Discussão de três formulações de Programação Inteira para o PSBMG. Duas formulações, uma direcionada e a outra não-direcionada, se baseiam em um número exponencial de restrições *cutsets* e em resultados de Teoria de Grafos que garantem sua validade. A terceira formulação usa um número polinomial, na quantidade de vértices e arestas da instância, de restrições de fluxos multi-produto.
- Apresentação de duas classes de desigualdades válidas para o politopo das soluções inteiras do PSBMG, adaptadas de outras desigualdades válidas para o PSBM.
- Desenvolvimento de um algoritmo exato *Branch-and-cut*, duas heurísticas construtivas, uma heurística de refinamento e duas estruturas de vizinhança de busca local para o PSBMG.
- Novos certificados de otimalidade, melhores limites inferiores e superiores para instâncias do PSBMG de pequeno, médio e grande porte também usadas em outros trabalhos da literatura.

1.2 Organização da Dissertação

Esta dissertação está estruturada em seis capítulos. O restante do texto está organizado da seguinte maneira:

Capítulo 2. [O Problema do Subgrafo Biconexo Mínimo Generalizado] O PSBMG é definido formalmente, apresentamos propriedades combinatórias das suas soluções ótimas e seus aspectos de complexidade computacional. Uma revisão bibliográfica com os trabalhos que já abordaram o problema e suas variações correlatas também é apresentada.

Capítulo 3. [Formulações de Programação Inteira para o PSBMG] Formulações de Programação Inteira para o PSBMG são introduzidas. São também apresentadas duas novas desigualdades lineares válidas para o problema.

Capítulo 4. [Um Método de Solução Exata para o PSBMG] Algoritmos de planos de corte para avaliar os limites de Programação Linear de uma das formulações propostas são introduzidos e utilizados em um algoritmo *Branch-and-cut* para o PSBMG.

Capítulo 5. [Métodos de Solução Heurística para o PSBMG] Duas heurísticas construtivas, uma heurística de refinamento e duas buscas locais são

propostas e avaliadas para o PSBMG isoladamente e em conjunção com o algoritmo *Branch-and-cut* discutido no Capítulo 4.

Capítulo 6. [Conclusões e Trabalhos Futuros]
As contribuições e limitações dessa dissertação são sintetizadas, e finalmente, conclusões e direções para trabalhos fututos são apresentadas.

Capítulo 2

O Problema do Subgrafo Biconexo Mínimo Generalizado

Neste capítulo, introduzimos algumas definições e a notação necessárias para o entendimento desta dissertação. Definimos formalmente o PSBMG e outros problemas a ele relacionados. Fazemos também uma revisão bibliográfica de trabalhos que já abordaram o PSBMG, ou algumas variações do mesmo, via algoritmos exatos, aproximativos e heurísticos. Em seguida, discutimos seus aspectos de complexidade computacional e suas principais propriedades combinatórias.

2.1 Conceitos Preliminares

Para conceitos elementares de Teoria de Grafos, usamos como referências os livros [West, 2001; Bollobás, 1998; Diestel, 2005]. Ao longo deste trabalho, a menos quando dito explicitamente, toda menção ao termo ***grafo*** significa um grafo finito, simples e não-direcionado. Denotamos um grafo por um par ordenado $G = (V, E)$, em que V é um conjunto de vértices e E é um conjunto de arestas. Quando V e E não estiverem subtendidos, escrevemos $V(G)$ e $E(G)$ para aludir, respectivamente, ao conjunto de vértices e arestas de G. Usamos a palavra ***digrafo*** para nos referir a um grafo direcionado. Para fins de clareza, reservamos o termo ***arco*** como um sinônimo de aresta direcionada.

Dizemos que G é ***vértice-biconexo*** se existem pelo menos dois caminhos vértice-disjuntos entre qualquer par de vértices distintos de G. O conceito de grafo ***aresta-biconexo*** é definido analogamente. Dizemos que $u \in V$ é um ***ponto de articulação*** de G se a sua remoção resulta em um grafo desconexo. Se não houver dúvida no contexto, a palavra biconexo será usada daqui em diante para designar qualquer grafo

vértice-biconexo.

Para todo subconjunto $X \subset V$, com $X \neq \emptyset$, o ***corte*** induzido por X em G, denotado por $\delta(X)$, é o conjunto de arestas de G com exatamente uma extremidade em X, i.e., $\delta(X) = \{\{u, v\} \in E : u \in X, v \notin X\}$. Quando necessário, para enfatizar a dependência em G, escrevemos $\delta_G(X)$ em vez que $\delta(X)$. Uma ***d-partição*** de um conjunto finito $W \neq \emptyset$, para algum inteiro $d > 0$, é um conjunto $P = \{P_1, \ldots, P_d\}$ de aglomerados (ou ***clusters***) em que $P_i \subset W$, $P_i \neq \emptyset$ para todo $i \in \{1, \ldots, d\}$, $P_i \cap P_j = \emptyset$ para quaisquer $i, j \in \{1, \ldots, d\}$ distintos e $\cup_{i=1}^{d} P_i = W$.

Para conceitos elementares de Teoria da Complexidade Computacional, nós nos apoiamos nas referências [Johnson & Garey, 1979; Arora & Barak, 2007]. Para um número real $\sigma \geq 1$ e um dado problema de otimização de minimização com solução ótima de custo w, um ***algoritmo σ-aproximativo*** para esse problema, ou um ***algoritmo de aproximação com fator de aproximação*** ou ***garantia de desempenho*** σ, é um algoritmo que encontra, para toda instância do problema, uma solução viável com custo, no máximo, igual a σw e, no pior caso, em tempo polinomial no tamanho da instância. O conceito de algoritmo de aproximação para problemas de otimização de maximização é definido analogamente.

Vamos denotar por $\mathbb{R}_+$, $\mathbb{R}_+^*$ e $\mathbb{Z}_+$, respectivamente, os conjuntos dos números reais não-negativos, dos reais positivos e dos inteiros não-negativos. Usamos o símbolo $\mathbb{B}$ para denotar o conjunto $\{0, 1\}$.

2.2 Definição do Problema e Problemas Relacionados

Muitos problemas em Otimização Combinatória possuem uma estreita relação com o PSBM(G). Visto que grande parte do que já se fez e já se sabe a respeito do PSBM(G) é oriundo do conhecimento associado a esses problemas, para fins de clareza e completude, definimos a seguir alguns problemas relacionados ao PSBMG que serão referenciados repetidas vezes ao longo da dissertação.

O primeiro deles é a chamada versão não-generalizada do PSBMG, a saber o *Problema do Subgrafo Biconexo Mínimo* (PSBM). Dado um grafo $G = (V, E)$ com pesos reais positivos $\{w_{ij} \in \mathbb{R}_+^* : \{i, j\} \in E\}$ nas arestas, o PSBM tem por objetivo encontrar um subgrafo gerador biconexo G' de G, tal que $c(G')$ seja mínimo, onde $c(G') = \sum_{\{i,j\} \in E(G')} w_{ij}$ denota o ***custo*** de G'. Quando conveniente, usamos também a notação w_{ij}^G para nos referir ao peso da aresta $\{i, j\}$ de G.

O segundo problema a ser citado é uma extenção do PSBM muito estudada e

abreviada por h-PSBM. Dada uma função $h : V \to \mathbb{Z}_+$, o h-PSBM procura um subgrafo G' de G com custo $c(G')$ mínimo, tal que exista(m) pelo menos $\min\{h(u), h(v)\}$ caminho(s) vértice-disjunto(s) para todo par de vértices distintos $u, v \in V$. Observamos que o PSBM é um caso particular do h-PSBM quando $h(u) = 2$ para todo $u \in V$.

O próximo problema que apresentamos é o *Problema do Projeto de Redes Resilientes* (PPRR). O PPRR pode ser brevemente definido como uma relaxação combinatória do h-PSBM que consiste em encontrar uma solução de custo mínimo contendo caminhos aresta-disjuntos. É imediato ver que o PPRR é uma relaxação combinatória do h-PSBM, uma vez que toda solução viável para o h-PSBM é também viável para o PPRR. Quando $h(u) = 2$ para todo vértice $u \in V$, o PPRR é também conhecido na literatura pelo nome de *Problema do Subgrafo Aresta-Biconexo Mínimo* (PSABM).

Podemos agora definir formalmente o *Problema do Subgrafo Biconexo Mínimo Generalizado* (PSBMG). Seja $Q = \{V_1, \ldots, V_k\}$, para $k = |Q| > 2$, uma k-partição de V. O PSBMG consiste em determinar um subgrafo biconexo G' de G com custo $c(G')$ mínimo e contendo exatamente um vértice de cada *cluster* de Q. Dada uma instância qualquer do PSBMG e uma solução viável $S = (V_S, E_S)$ para essa instância, dizemos que $u \in V$ é um ***vértice eleito*** (ou ***cluster-head***) em S se $u \in V_S$. Além disso, uma aresta $e \in E_S$ é dita ser ***redundante*** em relação a S se sua remoção preserva a viabilidade e diminui o custo de S. Daqui em diante, vamos nos referir a qualquer instância ou solução viável para o PSBMG e suas variações com os símbolos definidos neste parágrafo.

2.3 Revisão Bibliográfica

Em vista da similaridade entre os problemas definidos na seção anterior, a seguir apresentamos uma revisão bibliográfica para o PSBM(G) incluindo, quando convir, trabalhos que também lidam com o PPRR, h-PSBM ou PSABM. A fim de simplificar a notação, usamos $n = |V|$ e $m = |E|$ para representar, respectivamente, as cardinalidades dos conjuntos de vértices e arestas da instância G.

2.3.1 Algoritmos de Aproximação

Começamos citando abordagens de solução para o PSBM por meio de algoritmos de aproximação. Para o caso especial do PSBM restrito a arestas com pesos uniformes, i.e., arestas com pesos iguais, Khuller & Vishkin [1992] projetaram o primeiro algoritmo de aproximação que se tem registro. Tal algoritmo tem complexidade assintótica de tempo no pior caso de $O(n + m)$ e um fator de aproximação de $\frac{5}{3}$. Em resumo, ele

obtém soluções viáveis computando inicialmente uma árvore de busca de DFS (*depth-first search*) T de G. Em seguida, o algoritmo agrega a T um subconjunto das arestas de recuo (*back edges*), encontradas durante a DFS, pequeno o bastante para "cobrir" os pontos de articulação em T. Por fim, o algoritmo detecta e poda arestas redundantes da solução corrente. Para fazer a análise da garantia de desempenho do seu algoritmo, Khuller e Vishkin propuseram e utilizaram o conceito de escultura de um grafo (*graph carving*). Os autores realizaram experimentos computacionais em instâncias aleatórias e verificaram que, na média, o algoritmo por eles proposto obteve soluções de 10% a 12% piores que o ótimo para grafos esparsos e 2% piores para grafos densos.

Garg et al. [1993] melhoraram o algoritmo de Khuller-Vishkin apresentando um novo algoritmo $O(n+m)$ $\frac{3}{2}$-aproximativo, que opera em dois estágios. Seu primeiro estágio é idêntico ao algoritmo de Khuller-Vishkin. No segundo estágio, emprega-se um procedimento para particionar os vértices de G em blocos e, em seguida, são identificadas e retiradas arestas redundantes de cada bloco, mais especificamente, as chamadas arestas *cordas*, i.e., arestas que ligam vértices não-adjacentes de ciclos com mais de três vértices em G. A análise do fator de aproximação do algoritmo foi feita usando como suporte a observação de que para qualquer conjunto independente de G com i vértices, todo subgrafo gerador biconexo de G tem, no mínimo, $\max\{n, 2i\}$ arestas. Os autores, por fim, provaram que o fator de aproximação de $\frac{3}{2}$ é justo explicitando classes de instâncias do PSBM para as quais esse fator é atingido.

Em [Vempala & Vetta, 2000], foi proposto um algoritmo $\frac{4}{3}$-aproximativo com complexidade assintótica de tempo no pior caso de $O(n^2m)$, que é o melhor conhecido para o PSBM com pesos uniformes em termos de garantia de desempenho. Os autores constataram que o menor subgrafo gerador aresta-biconexo de G com grau mínimo igual a 2 provê um limite inferior para a solução ótima do PSBM com pesos uniformes. Vempala e Vetta chamaram de D2 o problema de se obter esse limite inferior e demonstraram que ele se reduz ao *Problema do Emparelhamento de Cardinalidade Máxima*. Em linhas gerais, o algoritmo de Vempala-Vetta primeiramente resolve D2 usando uma modificação do algoritmo $O(m\sqrt{n})$ de Micali-Vazirani [Micali & Vazirani, 1980], a implementação assintoticamente mais rápida que se conhece do algoritmo de Edmonds [Edmonds, 1965b]. Em seguida, ele cuidadosamente adiciona arestas à solução de D2 para assegurar que não existam mais pontos de articulação. O mérito do trabalho em questão foi justamente provar que é possível "vértice-biconectar" a solução de D2, i.e., transformar a solução de D2 em um grafo vértice-biconexo, usando um subconjunto das arestas de G com cardinalidade não maior que um terço da cardinalidade das arestas de D2.

Voltamos nossa atenção agora para o caso geral do PSBM. Frederickson & JaJa

[1981] desenvolveram um algoritmo $O(n^2)$ 3-aproximativo. A descrição do algoritmo de Frederickson-JaJa é bem complexa e extensa, porém, em alto nível, tal algoritmo inicialmente obtém uma árvore geradora mínima de G e, em seguida, vértice-biconecta essa árvore analisando a estrutura topológica da sua chamada árvore de blocos e pontos de articulação (*block-cutvertex tree*).

A mesma garantia de desempenho foi alcançada por um algoritmo $O(\min\{m, 2n\}n^2)$ de Ravi & Williamson [1995]. O algoritmo de Ravi-Williamson, como quase todos até aqui discutidos, também é bi-fásico. Na sua primeira fase, ele emprega um algoritmo primal-dual de Goemans & Williamson [1995a] para resolver em $O(n^2 \log n)$ com fator de aproximação igual a 2 o PSABM. Na sua segunda fase, o algoritmo ainda utiliza um esquema primal-dual para escolher em G um subconjunto de arestas de baixo custo total a fim de transformar a solução encontrada na primeira fase em um grafo vértice-biconexo.

No mesmo ano, Khuller & Raghavachari [1995] apresentaram um algoritmo $O(n^2m)$ com fator de aproximação $\left(2 + \frac{1}{n}\right)$. Tal algoritmo cria uma cópia direcionada D de G, com um vértice raiz artificial r e arcos adicionais com custo nulo saindo de r e incidindo sobre cada vértice de D. O algoritmo de Khuller-Raghavachari procede usando um algoritmo $O(n^2m)$ proposto por Frank & Tardos [1989] para encontrar um subdigrafo gerador de D com custo mínimo, tal que existam pelo menos dois caminhos direcionados vértice-disjuntos entre r e os vértices extremos da aresta e de menor custo em G. Khuller & Raghavachari [1995] provaram que a solução para esse problema unida à aresta e induz uma solução viável para o PBSM cujo custo é, no máximo, $\left(2 + \frac{1}{n}\right)$ vezes o ótimo.

Por fim, Fleischer [2001] projetou o melhor algoritmo de aproximação conhecido para PSBM, em termos de garantia de desempenho, com um fator de aproximação de 2. O algoritmo de Fleischer é um procedimento iterativo que a cada iteração resolve a relaxação linear de uma formulação de Programação Inteira para o PSBM. Uma vez calculada a solução ótima do programa linear definido em uma determinada iteração, todas as arestas de G cujas variáveis de decisão associadas assumirem valor maior ou igual a $\frac{1}{2}$ são retiradas do grafo de entrada e passam compor a solução corrente. Isso se repete até o momento em que a solução corrente se torna viável para o PSBM. Nesse caso, Fleischer mostrou que o custo dessa solução não ultrapassa o dobro do ótimo.

2.3.2 Heurísticas

Existem também trabalhos na literatura que lidam com o PSBM(G) por meio de métodos de solução heurística. Adiante, listamos e comentamos os principais.

Steiglitz et al. [1969] propuseram uma heurística que constrói uma solução viável para o problema escolhendo aleatoriamente um conjunto de arestas do grafo e, em seguinda, aprimorando tal solução por meio de uma rotina de busca local similar à 2-*opt* para o *Problema do Caixeiro Viajante* (PCV) [Applegate et al., 2006].

Monma & Shallcross [1989] introduziram as duas primeiras heurísticas construtivas combinatórias para o PSBM, a saber *two-tree dense* e *greedy-ears sparse*, juntamente com seis procedimentos de busca local, a maioria deles também inspirados em buscas locais previamente propostas para o PCV. A heurística *two-tree dense* obtém uma solução viável para o problema partindo de uma árvore geradora mínima T do grafo de entrada G e, em seguida, incrementando T com as arestas que formam uma árvore geradora mínima do subgrafo de G induzido pelas folhas de T. Já a heurística *greedy-ears sparse* encontra de forma gulosa uma decomposição auricular (*ear decomposition*) de G, i.e., uma partição das arestas de G em ciclos e caminhos, resolvendo uma sequência apropriada de instâncias do *Problema do Caminho Mínimo*. Os autores realizaram experimentos computacionais em 23 instâncias de teste, 20 das quais eram instâncias aleatoriamente geradas e 3 eram instâncias reais espelhadas em redes de computadores dos laboratórios da Bellcore. Em suma, as principais conclusões obtidas pelos autores foram as de que melhores resultados foram alcançados quando as buscas locais eram aplicadas a soluções iniciais geradas pela heurística *two-tree dense* e que, além disso, o tempo de execução gasto para atingir um ótimo local combinando todas as buscas locais em um arcabouço do tipo *Variable Neighborhood Descent* [Gendreau & Potvin, 2010] era cerca de três vezes maior que o tempo necessário para obter a solução inicial.

Clarke & Anandalingam [1995] projetaram a primeira heurística baseada em técnicas de Programação Matemática. Tal heurística emprega um algoritmo de planos de corte para resolver a relaxação linear de uma formulação de Programação Inteira para o PSBM. Na sequência, ela localiza e elimina arestas redundantes da chamada solução suporte e aplica três das seis rotinas de buscas locais propostas por Monma & Shallcross [1989]. Os autores executaram testes em 600 instâncias aleatórias divididas igualmente em seis classes de instâncias com 10, 25, 50, 75, 100 e 200 vértices. Nos resultados exibidos, nenhuma das instâncias com 200 vértices foram resolvidas na otimalidade dentro do limite de tempo de execução imposto. Mesmo assim, o *gap* de dualidade médio para essas instâncias foi de 3.5%, e a heurística levou, em média, quatro minutos para encontrar a solução final, sendo que aproximadamente 42% desse tempo foi gasto no cálculo do limite de Programação Linear. Para as instâncias de tamanho médio, i.e., com 75 e 100 vértices, os *gaps* de dualidade médios foram de, respectivamente, 2.2% e 2.5% e os tempos de execução médios para obter a solução

final foram cerca de, respectivamente, 16 e 37 segundos, metade dos quais foram usados para avaliar o limite de Programação Linear. Apenas para as instâncias com 10 e 25 vértices, a heurística em questão foi capaz de encontrar a solução ótima em todos os testes. Isso foi feito em menos de um segundo na média.

Pelo o que se tem conhecimento, os únicos trabalhos que até então abordaram diretamente o PSBMG foram [Hu & Raidl, 2009; Pagacz et al., 2010]. Hu & Raidl [2009] propuseram um algoritmo memético para o PSBMG que une operadores de mutação e cruzamento com dois procedimentos de busca local. O primeiro procedimento de busca local visa a otimizar a escolha do vértice eleito em cada *cluster*. Já o segundo procedimento tenta refinar a maneira como os *cluster-heads* estão interligados. Os autores introduziram uma técnica, chamada por eles de redução em grafos (*graph reduction*), parar acelerar consideravelmente as buscas locais. Testes computacionais foram realizados com 16 instâncias Euclidianas retiradas da TSPLIB [Reinelt, 1991] adaptadas com um procedimento de *clustering* geográfico descrito em [Fischetti et al., 1997]. As instâncias de teste têm entre 137 e 442 vértices e 30 e 189 *clusters*. Os autores também estenderam uma formulação de Programação Inteira Mista direcionada e compacta, originalmente proposta para o PSABM por Hu et al. [2010b], que usa restrições de fluxos multi-produto adicionando a ela restrições para garantir vértice-biconexidade.

Hu e Raidl usaram o *solver* CPLEX [IBM ILOG CPLEX Optimizer, 2012], versão 11.2, para calcular o limite de Programação Linear associado a essa formulação e para gerenciar os nós da árvore de enumeração. Dentro do limite de tempo de execução de uma hora, das 16 instâncias de teste consideradas, o CPLEX conseguiu achar soluções inteiras apenas para duas delas e obteve limites de Programação Linear para 11. Em todos os testes, os autores empregaram dois critérios de parada, a saber um limite máximo de 200 iterações do algoritmo sem melhorias na melhor solução corrente e um limite de tempo de execução máximo que variou de acordo com o tamanho de cada instância. Foram reportados para cada instância o custo da melhor solução encontrada e o valor médio do custo da solução em 30 execuções do algoritmo. Nos resultados obtidos, o *gap* médio das melhores soluções é de aproximadamente 31% e o tempo médio de execução é de dois minutos. Os autores fizeram um estudo experimental do impacto das buscas locais na eficácia do seu algoritmo e concluíram que, quando desativada, o método encontrava soluções, em média, 10% piores. Foi observado também que aumentar a probabilidade de aplicação dos mecanismos de busca local não contribuía substancialmente para melhores soluções, apenas elevava os tempos de execução. Vale ressaltar que para a formulação proposta por Hu e Raidl, o CPLEX conseguiu obter soluções ótimas somente para instâncias de pequeno porte, com, no máximo, 80 vértices e 16 *clusters*.

Pagacz et al. [2010] melhoraram o trabalho de Hu e Raidl acrescentando a ele dois mecanismos de gerenciamento de população a fim de escapar de ótimos locais. Em seus experimentos computacionais, Pagacz et al. usaram as mesmas instâncias de teste e o mesmo ambiente computacional que Hu & Raidl [2009]. Nos resultados relatados, os autores mostraram que os mecanismos de gerenciamento de população causaram maior efeito positivo na melhoria da qualidade das soluções quando eram usados conjuntamente, e não isoladamente. Além disso, ficou claro nos testes realizados por eles que o êxito do uso de tais mecanismos ficou fortemente dependente dos valores de determinados parâmetros de controle. Não houve melhorias consideráveis no tocante à diminuição dos tempos de execução em comparação com os resultados expostos em [Hu & Raidl, 2009]. Pagacz et al. não fizeram nenhum tipo de análise quantitativa dos resultados obtidos.

2.3.3 Formulações de Programação Inteira e Algoritmos Exatos

Formulações de Programação Inteira também têm sido amplamente investigadas para o (h-)PSBM. A primeira delas foi proposta por Grötschel et al. [1992b]. Essa formulação é não-direcionada e utiliza exponencialmente muitas restrições baseadas em *cutsets* para garantir a vértice-biconexidade das soluções viáveis. Os autores citam duas classes de desigualdades essenciais à validade da formulação, a saber *cutsets* e *node cutsets*, e três classes de desigualdades fortalecedoras, a saber desigualdades de partição, de partição de nós e de r-cobertura. Estudos detalhados da estrutura poliédrica do h-PSBM que contemplam todas essas classes desigualdades, incluindo comentários sobre a complexidade computacional dos seus respectivos problemas de separação, foram conduzidos em [Grötschel et al., 1992b; Stoer, 1992].

Mahjoub [1994] introduziu duas novas classes de desigualdades válidas para o PSABM, por conseguinte também válidas para o PSBM, as chamadas desigualdades *odd-wheel* e de F-partição. No seu trabalho, Mahjoub provou condições necessárias e suficientes para que tais desigualdades induzam facetas da envoltória convexa das soluções viáveis do problema. Mais do que isso, o autor também demonstrou que, para grafos série-paralelos cujos pesos nas arestas satisfazem a chamada desigualdade triangular, somente as desigualdades triviais em conjunto com as desigualdades baseadas em *cutsets* são suficientes para descrever por completo a envoltória convexa das soluções viáveis para o PSBM. Resultado semelhante foi obtido por Barahona & Mahjoub [1995], que mostraram que, para grafos de Halin, i.e., grafos cujas arestas podem ser divididas em uma árvore geradora e um ciclo Hamiltoniano visitando as folhas dessa árvore, as

desigualdades triviais, mais as desigualdades baseadas em *cutsets* unidas às de partição definem de forma exata a envoltória convexa das soluções viáveis para o PSBM.

Mahjoub & Nocq [1999] propuseram uma ordenação parcial dos pontos extremos fracionários do politopo dos vetores reais que satisfazem as desigualdades triviais, de *cutset* e *node cutset*. Os autores introduziram um algoritmo de tempo polinomial para separar, da envoltória convexa das soluções inteiras do PSBM, os chamados vértices minimais desse politopo, i.e., vértices que não são majorados por nenhum outro vértice de acordo com essa ordenação. Mahjoub e Nocq também apresentaram testes de redução que podem ser usados para acelerar tal algoritmo de separação.

Resultados de experimentos computacionais com algoritmos *Branch-and-cut* para o h-PSBM baseados em uma adaptação da formulação proposta por Grötschel et al. [1992b] foram relatados em [Grötschel et al., 1992a, 1995]. Por se tratarem de resultados que usam instâncias de teste aleatórias de pequeno porte não mais disponíveis na Internet e há muito ultrapassados em matéria de tempos de execução e da qualidade das soluções encontradas, preferimos não comentá-los e nos concentrar em trabalhos mais recentes.

Kerivin et al. [2004] também realizaram testes computacionais com um algoritmo *Branch-and-cut* para o PSBM. Tal algoritmo separa de forma exata desigualdades *cutset*, *node cutset* e de partição e, de forma heurística, desigualdades de F-partição. Os autores usaram 30 instâncias de teste Euclidianas da TSPLIB [Reinelt, 1991], o *solver* CPLEX versão 4.0 com o pacote MINTO [Savelsbergh & Nemhauser, 1998], versão 3.0, para a obter, respectivamente, a solução dos programas lineares e fazer o gerenciamento da árvore de enumeração. O limite de tempo de execução escolhido foi de três horas. Dentre as instâncias de teste consideradas, a menor de todas envolvia 50 vértices e a maior 318. Foram exibidos os *gaps* de dualidade na raiz em dois cenários, o primeiro considerando somente a separação das três primeiras desigualdades e o segundo a separação de todas elas, incluindo as de F-partição. Todas as instâncias foram resolvidas na otimalidade, três delas na raiz no primeiro cenário e 14 no segundo. Os *gaps* de dualidade médios em ambos os cenários foram da ordem de 0.01%. Para instâncias com, no máximo, 100 vértices (o que representa cerca de 55% das instâncias), o tempo de execução para encontrar o ótimo foi inferior a três segundos. Esse tempo de execução foi de dois minutos na média para instâncias com mais de 120 vértices, cerca de 20% das instâncias consideradas.

Os autores constataram a relevância das desigualdades de F-partição na eficácia e no tempo de execução do seu algoritmo. Para a grande maioria dos testes, o número de cortes encontrados dessa classe de desigualdades somava mais que 60% de todos os cortes detectados. Além disso, quando desigualdades de F-partição eram separadas e

adicionadas, foi observado que o *gap* de dualidade na raiz diminuiu, na média, por um fator de um terço. Um fenômeno interessante ressaltado pelos autores foi o de que, para aproximadamente 74% das instâncias de teste, as soluções ótimas encontradas para o PSBM eram ciclos Hamiltonianos de custo mínimo.

Continuando a revisão sobre formulações de Programação Inteira propostas para o PSBM, citamos o trabalho desenvolvido por Magnanti & Raghavan [2005]. Nele, os autores introduziram uma formulação direcionada compacta para o PSABM que usa restrições de fluxos multi-produto e que se torna mais forte que a formulação não-direcionada de Grötschel et al. [1992b] ao ser complementada com restrições para garantir a vértice-biconexidade das soluções viáveis. A validade da formulação de Magnanti e Raghavan segue como um corolário de um teorema de Nash-Williams [1960] que caracteriza grafos aresta-biconexos via orientações de arestas.

Similarmente ao trabalho de Magnanti & Raghavan [2005], Chimani et al. [2010] introduziram uma nova caracterização de grafos biconexos também via orientações de arestas e a utilizaram para deduzir duas formulações direcionadas para o h-PSBM, onde $h(u) \in \{0, 1, 2\}$ para todo $u \in V$. A primeira formulação, chamada de DCUT, contém exponencialmente muitas restrições baseadas em *directed cutsets*. A segunda formulação, chamada de DFLOW, usa restrições de fluxos multi-produto. Ambas as formulações são equivalentes no tocante à qualidade do limite de Programação Linear que elas induzem. Os autores provaram que a DCUT e DFLOW dominam a formulação apresentada por Magnanti & Raghavan [2005].

Chimani et al. realizaram experimentos computacionais a fim de comparar o desempenho de um algoritmo *Branch-and-cut* que separa *directed cutsets* e *directed node cutsets* da formulação DCUT com o desempenho do CPLEX como um *solver* de Programação Linear e um algoritmo exato *Branch-and-bound* que usa a formulação DFLOW. Pela primeira vez na literatura, os autores coletaram sistematicamente instâncias de testes para o (h)-PSBM e PPRR usadas em vários trabalhos e criaram um repositório de instâncias de *benchmark* chamado TSNDPLib (*Topological* $\{0, 1, 2\}$*-Survivable Network Design Problem Library*) [TSNDPLib, 2008]. Nos seus testes computacionais, as instâncias foram divididas em dois grupos. O primeiro grupo, chamado de grupo G, é formado por 30 grafos *grid* para cada valor de n, onde $n \in \{100, 400, 900, 1600, 2500, 3600, 4900\}$. O segundo grupo possui 34 instâncias com 100 e 400 vértices retiradas do repositório PCSTPLib (*Prize Collecting Steiner Tree Problem Library*) [Johnson et al., 2000]. As instâncias pertencentes ao segundo grupo foram separadas em dois subgrupos, com os nomes de K e P. Os autores usaram o *solver* CPLEX, versão 9.0, para solução dos programas lineares e gerenciamento da árvore de enumeração com um limite de tempo de execução de duas horas.

Para facilitar o comentário dos resultados, usamos um abuso de nomenclatura. Referenciamos por DCUT o desempenho o algoritmo *Branch-and-cut* que separa desigualdades da formulação DCUT e por DFLOW o desempenho do CPLEX para resolver o PSBM modelado pela formulação DFLOW. Chimani et al. observaram que para instâncias com até 100 vértices, DCUT conseguiu encontrar o ótimo em menos de dois segundos. Para instâncias com mais de 100 vértices até 900 vértices, DCUT levou, em média, um minuto. Para cerca de metade das instâncias do segundo grupo, DFLOW não conseguiu obter a solução ótima no limite de tempo de execução. Para as instâncias resolvidas na otimalidade, em sua grande maioria instâncias do primeiro grupo com até 900 vértices, DFLOW tomou, em média, de cem a mil vezes mais o que tempo de DCUT. No tocante à qualidade do limite de Programação Linear dessas formulações, para uma enorme fração das instâncias, cerca de 90% delas, o problema foi resolvido na raiz, uma vez que a solução encontrada da relaxação linear para a formulação DCUT era inteira. Para as instâncias que necessitaram de *branching*, o *gap* de dualidade médio na raiz foi de $0,16\%$. Apenas para aproximadamente 5% de todas as instâncias, DCUT não foi capaz de encontrar a solução ótima dentro do limite de tempo estabelecido. Esses resultados são os melhores resultados conhecidos no tocante a algoritmos *Branch-and-cut* para o PSBM que usam instâncias da literatura. Pelo nosso conhecimento, todos os algoritmos exatos para o PSBM baseados em técnicas de Programação Matemática propostos até então são algoritmos do tipo *Branch-and-cut*. Não há trabalhos nas linhas de Relaxação Lagrangeana ou Geração de Colunas.

Para livros, capítulos de livros separados e monografias sobre o PPRR e (h-)PSBM, referimos o leitor para [Stoer, 1992; Grötschel et al., 1995; Kerivin & Mahjoub, 2005]. É importante esclarecer que, segundo a nossa revisão bibliográfica, nenhum trabalho apresentou até o momento algoritmos exatos para solucionar o PSBMG.

2.4 O PSBMG e Suas Propriedades

Nesta seção, discutimos sobre aspectos da complexidade computacional do PSBMG e mencionamos algumas propriedades combinatórias das suas soluções ótimas. Todos os resultados teóricos provados a seguir foram generalizados a partir de resultados originalmente obtidos para o PSBM.

Apresentamos a seguir uma prova da NP-dificuldade do PSBM, por conseguinte do PSBMG também, visto que o PSBM é um caso especial do PSBMG quando $|Q| = |V|$. Por conveniência, para todo vértice $u \in V$, vamos denotar por $d(u) = |\delta(\{u\})|$ o ***grau***

de u. O seguinte lema vai ser útil na nossa prova.

Lema 1. *Para todo grafo biconexo G com n vértices e m arestas, é válida a desigualdade $m \geq n$.*

Prova. Da definição de grafo biconexo, podemos concluir que se G é biconexo, então, para todo $u \in V$, temos que $d(u) \geq 2$. Pelo Lema do Aperto de Mãos, que afirma que $\sum_{u \in V} d(u) = 2m$, concluímos que $m \geq n$. □

Grafos biconexos cujos vértices possuem grau igual a 2 são os conhecidos ciclos Hamiltonianos, i.e., como conseguência do Lema 1, dizemos que ciclos Hamiltonianos são grafos biconexos com a mínima quantidade possível de arestas. Provamos que o PSBM é NP-díficil mostrando que o chamado *Problema do Ciclo Hamiltoniano* (PCH) pode ser polinomialmente reduzido ao PSBM. Dado um grafo qualquer, o PCH é um problema de decisão NP-completo que pergunta se esse grafo possui um ciclo Hamiltoniano. Trata-se de um dos mais antigos e estudados problemas da Teoria da Complexidade Computacional e que figura na lista dos 21 famosos problemas de Karp [Karp, 1972].

Proposiçãrafo 1. *O PSBM é NP-difícil.*

Prova. Seja um grafo G com m arestas e n vértices uma instância para o PCH. Vamos demonstrar que existe um algoritmo de tempo polinomial no tamanho de G, i.e., em $m + n$, que reduz essa instância para o PCH em uma instância para o PSBM. Esse algoritmo gera uma instância $\widetilde{G}$ para o PSBM da seguinte maneira. Definimos $\widetilde{G}$ como um grafo completo com n vértices, tal que para todo par de vértices distintos $u, v \in V$, o peso da aresta $\{u, v\}$ em $\widetilde{G}$ é 1 se $\{u, v\}$ existe em G, caso contrário o peso de $\{u, v\}$ em $\widetilde{G}$ é $n+1$. É certo dizer que esse procedimento de redução, i.e., a descrição de como $\widetilde{G}$ é definido, requer $O(n^2)$ passos. Sabemos que existe um ciclo Hamiltoniano em G se, e somente se, o custo da solução ótima do PSBM em $\widetilde{G}$ for n. Em outras palavras, de posse de um algoritmo exato de tempo polinomial em $m + n$ para o PSBM, basta resolvermos o PSBM em $\widetilde{G}$ e verificar se o subgrafo gerador biconexo mínimo de $\widetilde{G}$ tem custo n. Em caso afirmativo, a resposta para o PCH em G é sim. Do contrário, respondemos não. □

Em virtude da grande semelhança entre o PSBM, PSABM e o PCV, alguns trabalhos da literatura já estudaram relações existentes entre as soluções ótimas ou entre as estruturas poliédricas desses problemas. Comentamos sobre alguns desses trabalhos a seguir. A princípio, merecem destaque os artigos de Frederickson & JaJa [1982] e Monma et al. [1990]. Dada uma instância H para o PSBM e o PCV, chamamos de $c_{PCV}(H)$ e $c_{PSBM}(H)$, respectivamente, os custos das soluções ótimas para o PCV e

o PSBM definidos em H. Quando H é um grafo planar, Frederickson e JaJa provaram que $\frac{3}{4}c_{PCV}(H) \leq c_{PSBM}(H)$. Mais do que isso, eles conjecturaram que essa relação é válida para qualquer instância do PSBM. A prova de tal conjectura só veio com o trabalho de Monma et al. [1990]. Partindo desse resultado, vamos generalizá-lo para o PSBMG, i.e., provamos que para qualquer instância G para o PCVG e PSBMG, segue que $\frac{3}{4}c_{PCVG}(G) \leq c_{PSBMG}(G)$, onde PCVG significa o *Problema do Caixeiro Viajante Generalizado*. Para isso, precisamos de alguns conceitos adicionais.

Definimos o PCVG analogamente ao PSBMG. Dada uma instância G para o PSBMG, o PCVG tem por objetivo encontrar um subgrafo G' de G de custo $c(G')$ mínimo e que seja um ciclo contendo exatamente um vértice de cada *cluster* em G. Chamamos de uma ***base*** de G um subconjunto $B \subset V$, onde $|B| = k$, tal que para todo *cluster* C, tem-se que $|C \cap B| = 1$. Além disso, dada uma base B de G, chamamos o subgrafo de G induzido por B de um ***moldura*** de G, denotado em diante por $\mathcal{M}(B)$ ou por $\mathcal{M}_G(B)$ quando quisermos enfatizar a dependência em G.

Proposiçãrafo 2. $\frac{3}{4}c_{PCVG}(G) \leq c_{PSBMG}(G)$.

Prova. Seja $S^* = (V^*, E^*)$ a solução ótima para o PSBMG em G. É sabido que $\frac{3}{4}c_{PCV}(\mathcal{M}(V^*)) \leq c_{PSBM}(G)$. Por definição, temos que $c_{PCVG}(G) \leq c_{PCV}(\mathcal{M}(V^*))$. Logo, $\frac{3}{4}c_{PCVG}(G) \leq c_{PSBMG}(G)$. □

Dois outros resultados teóricos dignos de nota foram obtidos por Monma et al. [1990]. Os autores demonstraram que a solução ótima para o problema de Programação Linear definido no chamado Politopo de Eliminação de Subciclos, uma relaxação linear clássica do PCV, também provê um limite inferior para a solução ótima do PSBM. Por fim, foi provado que o PSBM se reduz ao PSABM quando restrito às chamadas ***instâncias métricas***, i.e., instâncias cujos pesos das arestas obedecem à desigualdade triangular, i.e., quando para todo subgrafo completo com três vértices na instância em questão, a soma dos pesos de quaisquer duas arestas desse subgrafo vale, no mínimo, o peso da terceira aresta. Estendemos esse resultado para o PSBMG e o PSABMG, i.e., provamos que o PSBMG se reduz ao PSABMG em instâncias métricas, onde PSABMG significa o *Problema do Subgrafo Aresta-Biconexo Mínimo Generalizado*, definido analogamente ao PSBMG.

Proposiçãrafo 3. *O PSBMG se reduz ao PSABMG em instâncias métricas.*

Prova. Dada uma instância G métrica para o PSBMG e o PSABMG, denotamos por $\mathfrak{M}(G)$ o conjunto formado por todas as molduras de G. Observamos que se G é métrica, então toda moldura $\widehat{G} \in \mathfrak{M}(G)$ também é métrica. Mais ainda, qualquer moldura de G pode ser definida como uma instância para o PSBM e o PSABM. Dessa forma, a

instância G para o PSBMG e o PSABMG consiste na união de todas as instâncias para o PSBM e o PSABM em $\mathfrak{M}(G)$. De fato, $c_{PSBMG}(G) = \min\{c_{PSBM}(\widehat{G}) : \widehat{G} \in \mathfrak{M}(G)\}$ e $c_{PSABMG}(G) = \min\{c_{PSABM}(\widehat{G}) : \widehat{G} \in \mathfrak{M}(G)\}$. Sabemos que para qualquer moldura $\widehat{G} \in \mathfrak{M}(G)$, se S^* é a solução ótima para o PSBM em $\widehat{G}$, então S^* também é a solução ótima para o PSABM em $\widehat{G}$. Sendo assim, como o PSBM se reduz ao PSABM para cada moldura em $\mathfrak{M}(G)$, concluímos que o PSBMG se reduz ao PSABMG em G. □

Há também trabalhos na literatura que buscam caracterizar a estrutura combinatória das soluções ótimas para o PSBM. Dentre eles, citamos [Monma et al., 1990; Dirac, 1967]. É fácil constatar que a solução ótima para qualquer instância do PSBM é um grafo biconexo que não contém arestas redundantes. Formalmente, tais soluções são chamadas de aresta-minimais, uma vez que elas não contêm nenhum subgrafo gerador próprio que também seja uma solução viável para a instância em questão. Monma et al. [1990] estudaram a topologia de grafos biconexos aresta-minimais e provaram dois resultados interessantes. Em primeiro lugar, foi demonstrado que para toda instância métrica G do PSBM, a solução ótima para G é um grafo aresta-minimal cujos vértices possuem grau igual a 2 ou 3. Em segundo lugar, a remoção de qualquer aresta de G ocasiona o surgimento de pelo menos uma aresta ponte em G, i.e., uma aresta que se removida resulta em G desconexo. Uma caracterização alternativa de grafos biconexos aresta-minimais foi dada por Dirac [1967], que provou que um grafo biconexo é aresta-minimal se, e somente se, ele é biconexo e não possui arestas cordas. Claramente, todas essas propriedades, embora originalmente provadas no contexto do PSBM, também se aplicam às soluções ótimas do PSBMG.

Capítulo 3

Formulações de Programação Inteira para o PSBMG

Neste capítulo, apresentamos três formulações de Programação Inteira (PI) para o PSBMG. Todas as formulações aqui expostas baseiam-se em outras formulações de PI já existentes para o PSBM.

3.1 Formulação Não-Direcionada Baseada em *cutsets*

Introduzimos nesta seção uma formulação de PI para o PSBMG não-direcionada envolvendo um número exponencial de restrições baseadas em *cutsets*. Por praticidade, vamos nos referir a tal formulação pela notação $\mathcal{P}_{\mathrm{c}}$. A formulação $\mathcal{P}_{\mathrm{c}}$ é adaptada de uma formulação de PI para o PSBM proposta por Grötschel et al. [1992b].

Dada uma instância $G = (V, E)$ para o PSBMG com n vértices e m arestas, sejam $x \in \mathbb{B}^m$ e $y \in \mathbb{B}^n$ variáveis de decisão associadas, respectivamente, às arestas e vértices de G. Dada uma solução viável $S = (V_S, E_S)$ para G, para toda aresta $\{i, j\} \in E$, definimos

$$x_{ij} = \begin{cases} 1 & \text{se } \{i, j\} \in E_S, \\ 0 & \text{caso contrário.} \end{cases} \tag{3.1}$$

Analogamente para todo vértice $u \in V$, definimos

$$y_u = \begin{cases} 1 & \text{se } u \in V_S, \\ 0 & \text{caso contrário.} \end{cases} \tag{3.2}$$

Dados um conjunto finito $D \neq \emptyset$, uma função $f : D \to \mathbb{R}$ e um subconjunto $Z \subseteq D$, escrevemos $f(Z) = \sum_{z \in Z} f_z$, onde a notação f_z é simplesmente uma simplificação de $f(z)$. Seja $\mathcal{P}_c \subset \mathbb{R}^{n+m}$ o poliedro definido pelas desigualdades

$$y(V_i) = 1 \quad \text{para todo } i \in \{1, \ldots, k\}, \tag{3.3}$$

$$x(\delta(W)) \geq 2 \quad \text{para todo } W \subset Q, W \neq \emptyset, \tag{3.4}$$

$$x(\delta(W) \setminus \delta(V_i)) \geq 1 \quad \text{para todo } i \in \{1, \ldots, k\}, W \subset Q \setminus \{V_i\}, W \neq \emptyset, \tag{3.5}$$

$$y_u \geq x_{ij} \quad \text{para todo } u \in V,\ \{i,j\} \in \delta(\{u\}), \tag{3.6}$$

$$0 \leq y_u \leq 1 \quad \text{para todo } u \in V, \tag{3.7}$$

$$0 \leq x_{ij} \leq 1 \quad \text{para todo } \{i,j\} \in E. \tag{3.8}$$

Sendo assim, o PSBMG pode ser formulado como

$$\min \left\{ \sum_{\{i,j\} \in E} w_{ij} x_{ij} : (x,y) \in \mathcal{P}_c \cap (\mathbb{B}^m \times \mathbb{B}^n) \right\}. \tag{3.9}$$

As restrições (3.4) são as desigualdades *cutsets*. Elas garantem que existam pelo menos dois caminhos aresta-disjuntos entre qualquer par de *clusters* em G. As restrições (3.5) são as chamadas desigualdades *node cutsets*. Elas forçam que a solução viável continue sendo conexa após a remoção de um vértice qualquer. As restrições (3.3) obrigam que exatamente um vértice de cada *cluster* seja escolhido. As restrições (3.6) acoplam os dois tipos de variáveis de decisão. As desigualdades (3.7) e (3.8) são chamadas de triviais.

As desigualdades *cutsets* (3.4) e *node cutsets* (3.5) garantem concomitantemente a vértice-biconexidade das soluções viáveis para o PSBMG. A prova da validade dessas desigualdades decorre diretamente da definição de grafo bixonexo ou como uma consequência de um teorema de Menger [1927], que relaciona caminhos vértice-disjuntos a cortes em grafos.

A formulação $\mathcal{P}_c$ possui $n + m$ variáveis de decisão, $O(k)$ restrições (3.3), $O(2^k)$ restrições (3.4), $O(k2^k)$ restrições (3.5) e $O(m)$ restrições (3.6).

3.1.1 Novas Desigualdades Válidas

Com o intuito de fortalecer a formulação $\mathcal{P}_c$, acrescentamos duas novas classes de desigualdades válidas, a saber as desigualdades de F-partição e as desigualdades de

partição de nós. Discorremos sobre cada uma delas abaixo.

Dada uma t-partição $\{Q_1, \ldots, Q_t\}$ de Q, para todo $t \in \{2, \ldots, k-1\}$, e $F \subset \delta(Q_1)$, com $|F|$ ímpar, a classe de desigualdades de F-partição para o PSBMG é expressa como

$$x(\delta(Q_1, \ldots, Q_t) \setminus F) \geq t - \left\lceil \frac{|F|}{2} \right\rceil, \tag{3.10}$$

onde $\delta(Q_1, \ldots, Q_t) = \cup_{i=1}^{t} \delta(Q_i)$.

As desigualdades de F-partição são consideradas válidas para PSBMG porque podem ser derivadas como $\{0, \frac{1}{2}\}$-cortes de Chvátal-Gomory, i.e., cortes de Chvátal-Gomory que empregam somente pesos 0 e $\frac{1}{2}$, das desigualdades *cutsets* e triviais. Para isso, basta somarmos as *cutsets*

$$x(\delta(Q_p)) \geq 2 \quad \text{para todo } p \in \{1, \ldots, t\} \tag{3.11}$$

com as desigualdades

$$x_{ij} \geq 0 \quad \text{para todo } \{i, j\} \in \delta(Q_1) \setminus F, \tag{3.12}$$

$$-x_{ij} \geq -1 \quad \text{para todo } \{i, j\} \in F, \tag{3.13}$$

ponderando-as com peso $\frac{1}{2}$ e, em seguida, tomar o teto dos coeficientes da desigualdade resultante.

A segunda classe de desigualdades adicionadas à formulação $\mathcal{P}_c$ são as desigualdades de partição de nós, que generalizam as *node cutsets*. Para todo $i \in \{1, \ldots, k\}$ e para todas as t-partições $\{Q_1, \ldots, Q_t\}$ de $Q \setminus \{V_i\}$, com $t \in \{2, \ldots, k-2\}$, as desigualdades de partição de nós são expressas por

$$x(\delta(Q_1, \ldots, Q_t) \setminus \delta(V_i)) \geq t - 1. \tag{3.14}$$

É possível provar a validade das desigualdades de partição de nós usando um argumento de redução ao absurdo. Seja $S = (V_S, E_S)$ uma solução viável para G. Consideremos que S_u seja o subgrafo de S induzido por $V_S \setminus \{u\}$, para qualquer $u \in V_S$. Mostramos a seguir que para toda t-partição $\tilde{Q} = \{\tilde{Q}_1, \ldots, \tilde{Q}_t\}$ de $V(S_u)$, com $t \in \{2, \ldots, k-2\}$, é válida a desigualdade

$$x(\delta(\tilde{Q}_1, \ldots, \tilde{Q}_t) \setminus \delta(\{u\})) \geq t - 1. \tag{3.15}$$

Suponhamos, por contradição, que exista uma desigualdade de partição de nós violada

para S, i.e., que exista uma t-partição $\tilde{Q}$ de $V(S_u)$, para algum $u \in V_S$, tal que (3.15) seja violada. Vamos demonstrar que isso implica em uma contradição lógica. Definimos o grafo $G_{\tilde{Q}} = (V_{\tilde{Q}}, E_{\tilde{Q}})$, tal que $V_{\tilde{Q}} = \{1, \ldots, t\}$ e $\{i, j\} \in E_{\tilde{Q}}$ se, e somente se, $\{i, j\} \in \delta_G(\tilde{Q}_i) \cap \delta_G(\tilde{Q}_j)$. Se a desigualdade (3.15) não é satisfeita para $\tilde{Q}$ e u, então podemos concluir que $G_{\tilde{Q}}$ possui, no máximo, $t - 2$ arestas, i.e., concluímos que $G_{\tilde{Q}}$ é desconexo. Logo, se $G_{\tilde{Q}}$ é desconexo, então S_u também é desconexo, o que implica em uma contradição, uma vez que S_u precisa ser conexo, dado que, por hipótese, S é um grafo biconexo.

Finalmente, definimos

$$\mathcal{P}_{\mathrm{c}}^{+} = \mathcal{P}_{\mathrm{c}} \cap \left\{(x, y) \in \mathbb{R}^{n+m} : (x, y) \text{ satisfaz (3.10) e (3.14)}\right\}. \tag{3.16}$$

A nova formulação de PI para o PSBMG, incluindo as desigualdades de F-partição e partição de nós, pode ser escrita como

$$\min \left\{ \sum_{\{i,j\} \in E} w_{ij} x_{ij} : (x, y) \in \mathcal{P}_{\mathrm{c}}^{+} \cap (\mathbb{B}^m \times \mathbb{B}^n) \right\}. \tag{3.17}$$

3.2 Formulação Direcionada Baseada em *directed cutsets*

Introduzimos nesta seção uma formulação de PI para o PSBMG envolvendo exponencialmente muitas restrições baseadas em *directed cutsets*. Chamamos tal formulação de DCUT. Sua validade advém da caracterização de grafos biconexos via orientações de arestas apresentada em [Chimani et al., 2010]. Enunciamos abaixo esse resultado, mas sem prová-lo. Antes, porém, definimos o conceito de orientações de arestas em grafos. Dado um grafo $H = (V_H, E_H)$ qualquer, uma ***orientação das arestas*** de H é uma função $o : E_H \to V_H \times V_H$ que associa um sentido para cada aresta de G ou, informalmente falando, uma função o que transforma arestas de H em arcos.

Teorema ([Chimani et al., 2010])**.** *Seja $G = (V, E)$ um grafo arbitrário. Dizemos que G é biconexo se, e somente se, para qualquer $s \in V$, existir uma orientação das arestas de G segundo a qual, para todo $t \in V \setminus \{s\}$, haja dois caminhos direcionados em G, um de s para t e outro de t para s, sendo ambos vértice-disjuntos, de tal forma que exatamente um arco incida sobre s.*

Seja uma instância G para o PSBMG. Associamos a G um digrafo $\mathcal{D} = (V, A)$, com $A = \{(i, j), (j, i) : \{i, j\} \in E\}$. Para todo arco $(i, j) \in A$, o peso de (i, j) em

$\mathcal{D}$ é definido como $w^{\mathcal{D}}_{(i,j)} = w^{G}_{ij}$, onde $\{i,j\} \in E$. Por praticidade, estendemos para digrafos a notação de cortes em grafos introduzida no Capítulo 2. Para todo $W \subset V_B$, definimos $\delta^+(W) = \{(i,j) \in A : i \in W, j \notin W\}$, $\delta^-(W) = \{(i,j) \in A : i \notin W, j \in W\}$ e $\Delta(W) = \delta^+(W) \cup \delta^-(W)$.

Sejam $x \in \mathbb{B}^{2m}$ e $y \in \mathbb{B}^n$ variáveis de decisão associadas, respectivamente, aos arcos e vértices de $\mathcal{D}$. Dada uma solução viável $S = (V_S, E_S)$ para o PSBMG em G, para todo arco $(i,j) \in A$, definimos

$$x_{(i,j)} = \begin{cases} 1 & \text{se } (i,j) \text{ aparece na orientação das arestas de } G \text{ que induz } S, \\ 0 & \text{caso contrário.} \end{cases} \tag{3.18}$$

Analogamente para todo vértice $u \in V$, definimos

$$y_u = \begin{cases} 1 & \text{se } u \in V_S, \\ 0 & \text{caso contrário.} \end{cases} \tag{3.19}$$

Para $r \in \{1, \ldots, k\}$ fixo, seja $\mathcal{P}_{\mathrm{d}} \subset \mathbb{R}^{n+2m}$ o poliedro definido pelas desigualdades

$$y(V_i) = 1 \quad \text{para todo } i \in \{1, \ldots, k\}, \tag{3.20}$$

$$x(\delta^+(W)) \geq 1 \quad \text{para todo } W \subset Q, W \neq \emptyset, \tag{3.21}$$

$$x(\delta^-(W)) \geq 1 \quad \text{para todo } W \subset Q, W \neq \emptyset, \tag{3.22}$$

$$x(\Delta(W) \setminus \Delta(V_i)) \geq 1 \quad \text{para todo } i \in \{1, \ldots, k\} \setminus \{r\}, W \subset Q \setminus \{V_i\}, W \neq \emptyset, \tag{3.23}$$

$$x(\delta^-(V_r)) = 1, \tag{3.24}$$

$$x_{(i,j)} + x_{(j,i)} \leq 1 \quad \text{para todo } \{i,j\} \in E, \tag{3.25}$$

$$y_u \geq x_{(u,j)} + x_{(j,u)} \quad \text{para todo } u \in V, j \in \delta_G(\{u\}), \tag{3.26}$$

$$0 \leq y_u \leq 1 \quad \text{para todo } u \in V, \tag{3.27}$$

$$0 \leq x_{(i,j)} \leq 1 \quad \text{para todo } (i,j) \in A. \tag{3.28}$$

Sendo assim, o PSBMG pode ser formulado como

$$\min \left\{ \sum_{(i,j) \in A} w^{\mathcal{D}}_{(i,j)} x_{(i,j)} : (x,y) \in \mathcal{P}_{\mathrm{d}} \cap (\mathbb{B}^{2m} \times \mathbb{B}^n) \right\}. \tag{3.29}$$

As restrições (3.20) garantem que exatamente um vértice seja escolhido por *clus-*

ter. As restrições (3.21) obrigam que exista pelo menos um caminho direcionado saindo do *cluster* V_r rumo a cada um dos *clusters* restantes. As restrições (3.22) forçam que exista pelo menos um caminho direcionado chegando ao *cluster* V_r vindo de cada um dos demais *clusters*. As restrições (3.23) garantem que, ao ser removido qualquer *cluster* V_i de G, onde $i \in \{1, \ldots, k\} \setminus \{r\}$, ainda exista em $\mathcal{D}$ ou um caminho direcionado de V_r para W ou o contrário. As restrições (3.24) garantem que exatamente um arco incida sobre o *cluster* V_r. As restrições (3.25) obrigam que a orientação das arestas de G seja única. As restrições (3.26) fazem o acoplamento entre os dois tipos de variáveis da formulação. Por fim, as restrições (3.27) e (3.28) são as desigualdades triviais.

A formulação DCUT para o PSBMG possui $n + 2m$ variáveis de decisão, $O(k)$ restrições (3.20), $O(2^k)$ restrições (3.21) e (3.22), $O(k2^k)$ restrições (3.23), uma restrição (3.24), $O(m)$ restrições (3.25) e (3.26).

3.3 Formulação Direcionada Baseada em Fluxos Multi-produto

Introduzimos nesta seção uma formulação de Programação Inteira Mista (PIM) compacta e direcionada para o PSBMG baseada em fluxos multi-produto. Referenciamos essa formulação daqui em diante pela sigla FLOW. Sua validade advém da caracterização de grafos biconexos via orientações de arestas provada por Chimani et al. [2010] e enunciada na seção anterior.

Dada uma instância G para o PSBMG, definimos o ***backbone*** de G, denotado por $\mathcal{B}(G) = (V_\mathcal{B}, E_\mathcal{B})$, onde $V_\mathcal{B} = \{1, \ldots, k\}$ e $E_\mathcal{B} = \{\{i, j\} : \delta_G(V_i) \cap \delta_G(V_j) \neq \emptyset\}$. Associamos a $\mathcal{B}(G)$ o digrafo $\mathcal{D}_\mathcal{B} = (V_\mathcal{B}, A_\mathcal{B})$, onde $A_\mathcal{B} = \{(i, j), (j, i) : \{i, j\} \in E_\mathcal{B}\}$. Definimos também o conjunto de produtos $X = \{(r, t), (t, r) : t \in V_\mathcal{B} \setminus \{r\}\}$, onde $r \in V_\mathcal{B}$ é um vértice raiz arbitrário. A formulação FLOW se fundamenta na idéia de que para cada vértice $t \in V_\mathcal{B} \setminus \{r\}$, um produto é enviado a partir de r rumo a t e outro produto é enviado de t para r por caminhos direcionados vértice-disjuntos em $\mathcal{D}_\mathcal{B}$. Além disso, o número de arcos incidindo sobre r deve ser igual a 1.

Introduzimos variáveis de decisão $x \in \mathbb{B}^m$, $y \in \mathbb{B}^n$, $z \in \mathbb{B}^{|A_\mathcal{B}|}$ e $f \in \mathbb{R}_+^{|A_\mathcal{B}|} \times \mathbb{R}_+^{2k-2}$ associadas, respectivamente, às arestas e vértices de G, aos arcos de $\mathcal{D}_\mathcal{B}$ e ao fluxo dos produtos em X nos arcos de $\mathcal{D}_\mathcal{B}$. As variáveis de decisão $x \in \mathbb{B}^m$ e $y \in \mathbb{B}^n$ são definidas como explicado na seção anterior. Dada uma solução viável $S = (V_S, E_S)$

para o PSBMG em G, definimos para todo arco $(i,j) \in A_{\mathcal{B}}$

$$z_{(i,j)} = \begin{cases} 1 & \text{se } (i,j) \text{ é usado na orientação das arestas de } \mathcal{B}(G) \text{ que induz } S, \\ 0 & \text{caso contrário.} \end{cases} \tag{3.30}$$

Denotamos por $f_{(i,j)}^{(s,t)}$ a quantidade de fluxo relativo ao produto $(s,t) \in X$ que passa pelo arco $(i,j) \in A_{\mathcal{B}}$. Seja $\mathcal{P}_{\mathrm{f}} \subset \mathbb{R}^{n+m+2(k+|A_{\mathcal{B}}|-1)}$ o poliedro definido pelas desigualdades

$$y(V_i) = 1 \quad \text{para todo } i \in \{1,\ldots,k\}, \tag{3.31}$$

$$y_u \geq x_{ij} \quad \text{para todo } u \in V, \{i,j\} \in \delta_G(\{i\}), \tag{3.32}$$

$$\sum_{(i,j)\in\delta^-(\{j\})} f_{(i,j)}^{(s,t)} - \sum_{(j,i)\in\delta^-(\{j\})} f_{(j,i)}^{(s,t)} = \begin{cases} -1 & \text{se } j = s, \\ 1 & \text{se } j = t, \\ 0 & \text{caso contrário} \end{cases} \quad \text{para todo } (s,t) \in X, j \in V_{\mathcal{B}}, \tag{3.33}$$

$$z_{(i,j)} \geq f_{(i,j)}^{(s,t)} \quad \text{para todo } (i,j) \in A_{\mathcal{B}}, (s,t) \in X, \tag{3.34}$$

$$\sum_{\{i,j\}\in\delta(V_i)\cap\delta(V_j)} x_{ij} = z_{(i,j)} + z_{(j,i)} \quad \text{para todo } \{i,j\} \in E_{\mathcal{B}}, \tag{3.35}$$

$$\sum_{(i,w)\in\delta^-(\{w\})} f_{(i,w)}^{(r,s)} + \sum_{(i,w)\in\delta^-(\{w\})} f_{(i,w)}^{(s,r)} \leq 1 \quad \text{para todo } s \in V_{\mathcal{B}} \setminus \{r\}, w \in V_{\mathcal{B}} \setminus \{r,s\}, \tag{3.36}$$

$$z_{(i,j)} + z_{(j,i)} \leq 1 \quad \text{para todo } \{i,j\} \in E_B, \tag{3.37}$$

$$z(\delta^-(\{r\})) = 1, \tag{3.38}$$

$$0 \leq y_u \leq 1 \quad \text{para todo } u \in V, \tag{3.39}$$

$$0 \leq x_{ij} \leq 1 \quad \text{para todo } \{i,j\} \in E, \tag{3.40}$$

$$0 \leq z_{(i,j)} \leq 1 \quad \text{para todo } (i,j) \in A_{\mathcal{B}}, \tag{3.41}$$

$$f_{(i,j)}^{(s,t)} \geq 0 \quad \text{para todo } (i,j) \in A_{\mathcal{B}}, (s,t) \in X. \tag{3.42}$$

Dessa forma, PSBMG pode ser formulado como

$$\min \left\{ \sum_{\{i,j\}\in E} w_{ij}x_{ij} : (x,y,z,f) \in \mathcal{P}_{\mathrm{f}} \cap \left(\mathbb{B}^m \times \mathbb{B}^n \times \mathbb{B}^{|A_{\mathcal{B}}|} \times (\mathbb{R}^{|A_{\mathcal{B}}|} \times \mathbb{R}^k)\right) \right\}. \tag{3.43}$$

As restrições (3.31) garantem que apenas um vértice seja escolhido para cada

cluster. As restrições (3.32) fazem o acoplamento entre as variáveis associadas aos vértices e arestas de G. O balanço dos fluxos em cada vértice intermediário para todo produto em X é garantido pelas restrições (3.33). As restrições (3.34) fazem o acoplamento entre as variáveis de fluxo e as variáveis associadas aos arcos de $\mathcal{D}_B$. As restrições (3.35) fazem o acoplamento entre as arestas em G escolhidas para fazer parte da solução ótima e os arcos selecionados em $\mathcal{D}_B$. As restrições (3.36) garantem que os pares de caminhos direcionados pelos quais passam cada produto em X sejam vértice-disjuntos. As restrições (3.37) obrigam que a orientação das arestas do *backbone* seja única. As restrições (3.38) forçam que exatamente um arco incida sobre o vértice raiz r em $\mathcal{D}_B$. Por fim, as restrições (3.39), (3.40), (3.41) e (3.42) são as desigualdades triviais.

A formulação FLOW tem ao todo $n+m+2(k+|A_{\mathcal{B}}|-1)$ variáveis de decisão, $O(k)$ restrições (3.31), $O(m)$ restrições (3.32), $O(k^2)$ restrições (3.33), $O(k|A_{\mathcal{B}}|)$ restrições (3.34), $O(|A_{\mathcal{B}}|)$ restrições (3.35), $O(k^2)$ restrições (3.36), $O(|A_{\mathcal{B}}|)$ restrições (3.37) e uma restrição (3.38).

Capítulo 4

Um Método de Solução Exata para o PSBMG

Neste capítulo, abordamos um algoritmo *Branch-and-cut* que desenvolvemos para solucionar o PSBMG. Tal algoritmo utiliza a formulação $\mathcal{P}_c^+$, apresentada no Capítulo 3, com exponencialmente muitas restrições baseadas em *cutsets* e emprega as heurísticas construtivas e buscas locais discutidas no Capítulo 5 a fim de gerar soluções iniciais para o problema e também para aprimorar as soluções inteiras encontradas ao longo da sua execução. Nosso objetivo é fornecer, com um menor esforço computacional, novos certificados de otimalidade ou, quando isso não for possível, melhores limites inferiores e superiores para instâncias do PSBMG cujo ótimo é desconhecido. Ao longo deste capítulo, comentamos sobre os procedimentos de separação que fazem parte do nosso algoritmo e detalhamos seus aspectos de implementação. Por fim, expomos e analisamos os resultados obtidos, comparando-os com os resultados alcançados para a versão não-generalizada do PSBMG pelo algoritmo BC proposto por Kerivin et al. [2004].

4.1 Algoritmo Branch-and-cut

Branch-and-cut (BC) [Padberg & Rinaldi, 1991; Lucena & Beasley, 1996; Junger et al., 1995] é um método muito usado e bem sucedido para resolver problemas de Otimização Combinatória matematicamente formulados via Programação Linear Inteira. Em geral, um BC incorpora algoritmos de planos de corte a procedimentos enumerativos inteligentes, e.g., *Branch-and-bound* (BB), para avaliar limites de PL na resolução dos subproblemas dos nós da árvore de enumeração. Nosso BC emprega algoritmos de planos de corte para aproximar o limite de PL associado à formulação $\mathcal{P}_c^+$. A

seguir, discorremos sobre os procedimentos de separação de desigualdades do algoritmo BC e, na sequência, tratamos dos seus respectivos detalhes de implementação.

4.1.1 Procedimentos de Separação de Desigualdades

Na implementação do nosso algoritmo BC, foi utilizado o *solver* CPLEX, versão 10.2, para gerenciar a árvore de enumeração e resolver os programas lineares nos seus nós. É importante esclarecer que todas as rotinas de pré-processamento, heurísticas e separação de desigualdades válidas já pré-implementadas no CPLEX, e.g., cortes baseados em arredondamento inteiro, cortes de Gomory, cortes da mochila, etc., foram desativadas. Somente as classes de desigualdades *cutset*, *node cutset*, de partição de nós e de F-partição, descritas na Seção 3.1, foram separadas pelo nosso algoritmo.

Dada uma instância $G = (V, E)$ para o PSBMG, o arcabouço do nosso algoritmo BC inicia resolvendo o seguinte programa linear

$$\min \left\{ \sum_{\{i,j\} \in E} x_{ij} w_{ij} : (x, y) \in \mathcal{P}_0 \right\}, \tag{4.1}$$

onde o poliedro $\mathcal{P}_0$ é definido pelas desigualdades (3.6)-(3.8) da formulação $\mathcal{P}_c^+$ em conjunto com as *cutsets*

$$x(\delta(V_i)) \geq 2 \quad \text{para todo } i \in \{1, \ldots, k\}. \tag{4.2}$$

O algoritmo BC iterativamente separa na raiz da árvore de enumeração e em todos seus demais nós, nesta ordem, as desigualdades *cutset* (3.4), *node cutset* (3.5), de partição de nós (3.14) e de F-partição (3.10), até que os procedimentos de separação não encontrem nenhuma delas violada. Todos os problemas de separação são resolvidos no chamado ***grafo suporte condensado***, definido a seguir. Dada a solução $(\overline{x}, \overline{y})$ para a relaxação linear do subproblema em um nó qualquer da árvore de enumeração, o grafo suporte condensado associado a $(\overline{x}, \overline{y})$, denotado por $\mathscr{C} = (V_{\mathscr{C}}, E_{\mathscr{C}})$, é tal que $V_{\mathscr{C}} = \{1, \ldots, k\}$, $E_{\mathscr{C}} = \{\{i, j\} : \delta_G(V_i) \cap \delta_G(V_j) \neq \emptyset\}$ e $w_{ij}^{\mathscr{C}} = \sum_{\{i,j\} \in \delta_G(V_i) \cap \delta_G(V_j)} \overline{x}_{ij}$ para toda aresta $\{i, j\} \in E_{\mathscr{C}}$.

A separação das desigualdades *cutset* é feita via o algoritmo de fluxo máximo *preflow push-relabel* de Goldberg & Tarjan [1986] (GT). Em alto nível, procedemos da seguinte forma. Criamos uma cópia direcionada de $\mathscr{C}$, i.e., um digrafo $\mathcal{D}_{\mathscr{C}} = (V_{\mathscr{C}}, A_{\mathscr{C}})$, onde $A_{\mathscr{C}} = \{(i, j), (j, i) : \{i, j\} \in E_{\mathscr{C}}\}$, tal que o peso $w_{(i,j)}^{\mathcal{D}}$ do arco $(i, j) \in A_{\mathscr{C}}$ é dado por $w_{(i,j)}^{\mathcal{D}} = w_{ij}^{\mathscr{C}}$, sendo $w_{ij}^{\mathscr{C}}$ o peso da aresta $\{i, j\} \in E_{\mathscr{C}}$. Fixamos um vértice $u \in V_{\mathscr{C}}$ e, para todo $v \in V_{\mathscr{C}} \setminus \{u\}$, calculamos em $\mathcal{D}_{\mathscr{C}}$ o fluxo máximo de u para v. Se a capaci-

dade do corte associado a tal fluxo máximo for inferior a 2, encontramos assim uma *cutset* violada. Para instâncias densas, usamos a implementação tradicional $O(k^3)$ do algoritmo GT e, para instâncias esparsas, usamos a implementação $O\left(km_{\mathscr{C}} \log \frac{k^2}{m_{\mathscr{C}}}\right)$, onde $m_{\mathscr{C}} = |E_{\mathscr{C}}|$, com a estrutura de dados *dynamic trees* proposta por Sleator & Tarjan [1983]. Nos nossos experimentos, definimos que a instância $G = (V, E)$ para o PSBMG é esparsa se $\frac{2m}{n^2-n} < 0.4$ e densa caso contrário. Dessa maneira, a complexidade assintótica de tempo no pior caso do procedimento de separação das desigualdades *cutset* é $O(k^4)$ na implementação tradicional do algoritmo GT ou $O\left(k^2 m_{\mathscr{C}} \log \frac{k^2}{m_{\mathscr{C}}}\right)$ na implementação com *dynamic trees*.

Para a separação das desigualdades *node cutset*, utilizamos o algoritmo de Hao & Orlin [1994] (HO). Dado um digrafo com pesos reais positivos nos arcos, o algoritmo HO encontra, com a mesma complexidade assintótica do GT, o corte mínimo desse digrafo, i.e., o corte de menor capacidade dentre todos os cortes. A separação das desigualdades *node cutset* é feita como descrito a seguir. Dado $\mathcal{D}_{\mathscr{C}}$, para todo $u \in V_{\mathscr{C}}$, encontramos o corte mínimo no subdigrafo de $\mathcal{D}_{\mathscr{C}}$ induzido por $V_{\mathscr{C}} \setminus \{u\}$. Se a capacidade de tal corte for inferior a 1, encontramos uma *node cutset* violada. Similarmente à separação das desigualdades *cutset* com o algoritmo GT, usamos duas implementações do HO, a saber a implementação tradicional $O(k^3)$ para instâncias densas e, para instâncias esparsas, a implementação $O\left(km_{\mathscr{C}} \log \frac{k^2}{m_{\mathscr{C}}}\right)$ com *dynamic trees*. Assim, a complexidade assintótica de tempo no pior caso do procedimento de separação das desigualdades *node cutset* é $O(k^4)$ em instâncias densas e $O\left(k^2 m_{\mathscr{C}} \log \frac{k^2}{m_{\mathscr{C}}}\right)$ em instâncias esparsas.

A complexidade computacional da separação exata das desigualdades de F-partição é um problema aberto. Quando o conjunto F é fixo, as desigualdades de F-partição se reduzem às chamadas desigualdades de partição, que generalizam as *cutsets*. Nesse caso, é possível encontrar a desigualdade de F-partição mais violada em tempo polinomial usando um algoritmo proposto por Baiou et al. [2000]. Por não ser trivial o problema de se fixar a priori um conjunto "apropriado" de arestas F de G, no nosso algoritmo BC, implementamos uma heurística de propósito geral, descrita em [Kerivin et al., 2004], que busca encontrar, dada a solução $(\overline{x}, \overline{y})$ para a relaxação linear do subproblema em um nó qualquer da árvore de enumeração, o maior número possível de arestas $\{i, j\} \in E$ com $\overline{x}_{ij} = 1$ e utilizá-las para detectar desigualdades de F-partição violadas.

No intuito de esclarecer tal heurística de separação, vamos introduzir antes alguns conceitos necessários. Dado um grafo $H = (V_H, E_H)$ com pesos reais positivos nas arestas, a ***árvore de Gomory-Hu*** de H é uma árvore $\mathcal{T} = (V_H, E_{\mathcal{T}})$, também com pesos reais positivos nas arestas, tal que para toda aresta $\{i, j\} \in E_{\mathcal{T}}$, o corte mínimo que separa i de j em H é $\delta_H(S)$, onde S é qualquer uma costas de $\mathcal{T}$ resultantes da

remoção da aresta $\{i, j\}$ em $\mathcal{T}$. Mais ainda, dizemos que o custo do corte mínimo que separa i de j em H é precisamente o peso da aresta $\{i, j\}$ em $\mathcal{T}$. Além disso, para quaisquer vértices $u, v \in V_H$, tais que $\{u, v\} \notin E_{\mathcal{T}}$, o corte que separa u de v em H é $\delta_H(S)$, onde S é qualquer uma das costas de $\mathcal{T}$ resultantes da remoção da aresta de menor peso no caminho que conecta u a v em $\mathcal{T}$.

A heurística implementada no algoritmo BC para separar desigualdades de F-partição inicia obtendo a árvore de Gomory-Hu $\mathcal{T}_{\overline{\mathscr{C}}}$ do chamado grafo suporte condensado complementar $\overline{\mathscr{C}} = (V_{\overline{\mathscr{C}}}, E_{\overline{\mathscr{C}}})$. Sua única diferença em comparação com o grafo suporte condensado é a de que os pesos das suas arestas são computados da seguinte maneira. Dada a solução $(\overline{x}, \overline{y})$ para a relaxação linear do subproblema de um nó qualquer da árvore de enumeração, o peso $w_{ij}^{\overline{\mathscr{C}}}$ da aresta $\{i, j\} \in E_{\overline{\mathscr{C}}}$ é dado por $w_{ij}^{\overline{\mathscr{C}}} = \sum_{e \in \delta(V_i) \cap \delta(V_j)} (1 - \overline{x}_{ij})$. Na sequência, para cada aresta e de $\mathcal{T}_{\overline{\mathscr{C}}}$, tomamos S como qualquer uma das costas do corte resultante da remoção de e em $\mathcal{T}_{\overline{\mathscr{C}}}$ e construímos duas partições de $V_{\overline{\mathscr{C}}}$. A primeira delas é $\Pi_1 = \{S, \{\overline{v}_1\}, ..., \{\overline{v}_p\}\}$, onde $p = |V_{\overline{\mathscr{C}}} \setminus S|$ e $\{\overline{v}_1, \ldots, \overline{v}_p\} = V_{\overline{\mathscr{C}}} \setminus S$. A segunda é $\Pi_2 = \{V_{\overline{\mathscr{C}}} \setminus S, \{v_1\}, \ldots, \{v_{p'}\}\}$, onde $p' = |S|$ e $\{v_1, \ldots, v_{p'}\} = S$. Sejam $F_1, F_2 \subset \delta_G(S)$. O conjunto F_1 é definido como descrito a seguir. Listamos as arestas em $\delta_G(S)$ em ordem não crescente em relação dos valores das suas respectivas variáveis de decisão em $\overline{x}$ e escolhemos as $|F_1|$ maiores dentre elas, com $|F_1|$ ímpar. O objetivo é encontrar uma desigualdade de F-partição violada usando Π_1 e F_1 cujo lado direito, i.e., $|\Pi_1| - \lceil \frac{|F_1|}{2} \rceil$, assuma o maior valor possível. O cálculo do melhor valor de $|F_1|$ é feito executando uma busca binária em $\{1, 3 \ldots, 2q + 1\}$, onde $q = \min\{|\delta_G(S)|, p\}$. A busca por uma desigualdade de F-partição violada usando Π_2 e F_2 é conduzida analogamente. Para obter a árvore de Gomory-Hu do grafo suporte condensado complementar utilizamos o algoritmo $O(k^4)$ proposto por Gusfield [1990]. Por fim, a complexidade assintótica de tempo no pior caso da nossa heurística de separação das desigualdades de F-partição é de $O(k^4)$.

Barahona & Kerivin [2003] demonstraram que o problema da separação exata das desigualdades de partição de nós pode ser resolvido em $O(k^5)$ no pior caso. Utilizamos na separação das desigualdades de partição de nós um algoritmo $\left(2 - \frac{2}{k}\right)$-aproximativo proposto por Saran & Vazirani [1991] (SV) originalmente para o chamado *Problema do t-Corte Mínimo* (PtCM). Dado um grafo $H = (V_H, E_H)$ com pesos reais positivos nas arestas e um número inteiro $t \geq 2$, o PtCM consiste em encontrar um t-corte em H, i.e., uma t-partição $\{W_1, \ldots, W_t\}$ de V_H, cujo custo seja mínimo, i.e., tal que a soma dos pesos das arestas em $\delta(W_1, \ldots, W_t)$ seja mínima. Para o caso especial de partições com tamanho fixo, i.e., quando t é uma constante e não mais parte da instância do PtCM, o problema da separação das desigualdades de partição de nós pode ser resolvido pelo algoritmo de Goldschmidt & Hochbaum [1988] em $O(n_H^{t^2})$, onde $n_H = |V_H|$.

O algoritmo SV também inicia computando a árvore de Gomory-Hu $\mathcal{T}_H$ de H. Em seguida, são listadas em ordem não descrescente as arestas de $\mathcal{T}_H$ pelos seus pesos, lembrando que cada uma delas induz um corte em H. Suponhamos que $(e_1, e_2, \ldots, e_{m_H})$, onde $m_H = |E_H|$, seja tal ordenação e que $(\delta_H(S_1), \delta_H(S_2), \ldots, \delta_H(S_{m_H}))$ sejam os cortes em H associados às arestas de $\mathcal{T}_H$ quando ordenadas, i.e., para todo $i \in \{1, \ldots, m_H\}$, $\delta_H(S_i)$ é o corte induzido em H por qualquer uma das duas componentes conexas de $\mathcal{T}_H$ resultantes da remoção da aresta e_i em $\mathcal{T}_H$. Dado um número inteiro $t \geq 2$, o algoritmo SV obtém um t-corte em H de custo, no máximo, o dobro do custo do t-corte mínimo da seguinte maneira. Ele itera sobre a lista $(\delta_H(S_1), \ldots, \delta_H(S_{m_H})))$ e, até que se forme um t-corte, o algoritmo escolhe gulosamente o j-ésimo corte $\delta_H(S_j)$, para $j \in \{1, \ldots, t\}$, apenas se $\delta_H(S_j) \not\subset (\cup_{d=1}^{j-1} \delta_H(S_d))$. A complexidade assintótica de tempo no pior caso do algoritmo SV é $O(n_H^4)$. O procedimento heurístico de separação das desigualdades de partição de nós implementado no nosso algoritmo BC faz k chamadas ao algoritmo de Saran-Vazirani descrito anteriormente, de forma que, na i-ésima chamada, usa-se como grafo de entrada o subgrafo de $\mathscr{C}$ induzido por $V_{\mathscr{C}} \setminus \{i\}$. Portanto, a complexidade assintótica de tempo no pior caso da heurística de separação das desigualdades de partição de nós é $O(k^5)$. No nosso algoritmo BC, executamos tal heurística primeiramente para $t = 3$ e, somente quando nenhuma desigualdade violada é encontrada, executamos novamente uma última vez para $t = 4$. Nesse ponto, é crucial salientar que para o caso particular em que $t = 3$, é possível implementar a heurística de separação das desigualdades de partição de nós em $O(k^4)$, pois, nesse cenário, como demonstrado em [Saran & Vazirani, 1991], ela equivale a três execuções do algoritmo HO, que, como já afirmamos, tem complexidade assintótica de tempo no pior caso $O(k^3)$.

A fim de acelerar a separação das desigualdades, empregamos uma modificação dos testes de redução de Padberg & Rinaldi [1990] proposta por Chekuri et al. [1997]. Dada uma instância qualquer para o *Problema do Corte de Custo Mínimo*, os testes de Padberg-Rinaldi tentam encontrar arcos passíveis de serem contraídos nessa instância, i.e., arcos que garantidamente não farão parte do corte de custo mínimo que separa os vértices fonte e destino da instância em questão e que, portanto, podem ser contraídos, diminuindo assim o tamanho da instância.

4.1.2 Detalhes de Implementação

Nesta subseção, comentamos sobre os detalhes de implementação do nosso algoritmo BC. Partimos das estruturas de dados usadas para a representação de grafos e digrafos. Devido ao uso dos testes de Padberg-Rinaldi nos procedimentos de sepa-

ração de desigualdades, foi essencial o emprego de técnicas que viabilizassem de forma eficiente a operação de contração de arestas. Uma desvantagem do ponto de vista algorítmico em relação a tal operação é que ela pode gerar laços (*self-loops*) ou arestas paralelas no grafo resultante. Para lidar com isso, implementamos duas estratégias, a saber compressão compacta e compressão *set-union*.

A estratégia de compressão compacta trata as contrações de arestas de forma explícita, i.e., "fisicamente" remove vértices e arestas do grafo subjacente e mantém em todo instante sua topologia real. Para isso, utilizamos listas de adjacência para codificar grafos como digrafos, substituindo arestas por pares de arcos anti-paralelos. Seja $H = (V_H, E_H)$ um grafo qualquer e uma aresta $\{u, v\} \in E_H$, assumindo que $d(v) \leq d(u)$, onde $d(u)$ e $d(v)$ são, respectivamente, os graus dos vértices u e v em H antes da operação de contração. Para contrairmos $\{u, v\}$, substituímos cada aresta do tipo $\{v, i\} \in E_H$, para algum $i \in V_H$, por uma aresta $\{u, i\}$, removemos o vértice v de H, eliminamos laços e fundimos quaisquer arestas paralelas que possam ter sido criadas. Com o uso cuidadoso de arranjos de *bits* e de uma função *hash* simples que, para cada aresta, retorna em tempo constante e sem colisões os ponteiros para seus respectivos arcos anti-paralelos, conseguimos implementar a compressão compacta em $O(d(v))$.

A estratégia de compressão *set-union* trata as contrações de arestas de forma implícita. Em vez de literalmente eliminar um dos vértices extremos durante a contração de uma determinada aresta, ambas as extremidades dessa aresta são fundidas em um conjunto de vértices chamado de pseudo-vértice. Para isso, implementamos uma estrutura de dados do tipo *disjoint-set union* usando *florestas de disjoint-sets* com compressão de caminhos. Cada pseudo-vértice possui um vértice representante distinto e cada vértice possui um ponteiro que indica a qual pseudo-vértice ele pertence. Dado um grafo qualquer com n vértices, a estratégia de compressão *set-union* pode ser implementada em $O(\alpha(n))$, em que $\alpha(n)$ é a inversa da função de Ackermann. Por se tratar de estruturas de dados mais complexas e cuja discussão minuciosa pode se estender por demais, referimos o leitor para os trabalhos [Cormen et al., 1990; Chekuri et al., 1997].

No tocante ao algoritmo GT utilizado na separação das desigualdades *cutset*, nosso foco principal foi tentar reduzir o número de operações *push* e *relabel*, pois são justamente elas que dominam sua complexidade assintótica. Para diminuir o número de *pushes*, implementamos a política *highest label* para a escolha de vértices sobrecarregados usando *buckets* de listas encadeadas para agrupar os vértices de acordo com seus *labels*. Para diminuir o número de *relabelings*, implementamos as estratégias de *gap heuristic* e *global relabeling*. A primeira delas detecta vértices que podem ser con-

traídos na origem e, portanto, diminui o tamanho do grafo como um todo, acelerando o algoritmo. A segunda reajusta periodicamente os *labels* dos vértices possibilitando, com um número menor de *relabelings*, que mais vértices com excesso de fluxo possam ser descarregados. Para uma discussão mais detalhada dessas estratégias, citamos os trabalhos [Cherkassky, 1979; Cherkassky & Goldberg, 1994; Derigs & Meier, 1989]. Em relação ao algoritmo HO, utilizado na separação das *node cutsets*, implementamos a chamada heurística *excess detection* proposta por Chekuri et al. [1997] para melhorar seu desempenho. Tal heurística detecta vértices no grafo cujo excesso de fluxo ultrapassa o valor corrente do corte mínimo, contrai esses vértices na origem e satura os arcos que saem deles.

Buscando acelerar ainda mais nosso BC, seguimos duas idéias desenvolvidas por Koch & Martin [1998], a saber a estratégia de se usar *creep-flow* e *cortes aninhados* nos algoritmos GT e HO. Discorremos sobre cada uma delas a seguir. Em essência, na estratégia de *creep-flow*, dada a solução $(\overline{x}, \overline{y})$ da relaxação linear do subproblema em qualquer nó da árvore de enumeração, antes de chamar os algoritmos GT e HO, perturbamos por uma pequena constante ϵ as capacidades das arestas e no grafo G da instância do PSBMG para as quais $\overline{x}_e = 0$. Dessa maneira, garantimos que os cortes obtidos sejam arco-minimais, i.e., que sejam cortes de capacidade mínima e com a menor quantidade possível de arcos. Embora essa estratégia possa provocar um dispêndio maior de tempo de execução por parte dos algoritmos GT e HO, naturalmente porque mais arcos passam a ser levados em consideração, confirmamos em nossos experimentos que o uso de *creep-flow* foi benéfico, visto que o tempo gasto pelo CPLEX para reotimizar os programas lineares ao longo da execução do BC reduziu por um fator de três. Essa constatação é bem razoável, uma vez que cortes mais esparsos correspondem a linhas mais esparsas na matriz de restrições da relaxação linear dos subproblemas dos nós da árvore de enumeração. Fixamos $\epsilon = 10^{-5}$ nos nossos experimentos. Na estratégia de cortes aninhados, sempre que executamos os algoritmos GT e HO e obtemos um corte mínimo em $\mathcal{D}_{\mathscr{C}}$, retiramos de $\mathcal{D}_{\mathscr{C}}$ os arcos que formam tal corte mínimo e executamos novamente GT e HO. Repetimos esse processo enquanto desigualdades *cutsets* e *node cutsets* violadas forem encontradas.

Discutimos agora a política de inserção de cortes implementada no algoritmo BC, considerada fundamental para o controle do tamanho dos programas lineares nos nós da árvore de enumeração e para retardar o fenômeno de *tailing-off*. Na raiz da árvore de BB, simplesmente inserimos todos os cortes encontrados. Nos demais nós, inserimos apenas os cortes com violação mínima l e θ-ortogonais ao corte de maior violação. Dado um número real $0 < \theta < 1$, dizemos que um par de cortes normalizados (com a norma Euclidiana) de uma determinada classe de desigualdades é θ-***ortogonal*** se o

co-seno do ângulo formado entre seus respectivos vetores de incidência é, no máximo, θ. Após um estudo experimental de ajuste, escolhemos usar $\theta = 0.09$ nos nossos testes. Os valores dos limiares de violação fixados foram de 0.5 para as *cutsets* e *node cutsets*, 0.4 para as desigualdades de F-partição e 1 para as de partição de nós.

Com o objetivo de detectar e mitigar o efeito de *tailing-off*, nosso algoritmo BC aborta todos os procedimentos de separação de desigualdades e faz *branching* direto se, em cada nó da árvore de enumeração, não houver uma melhoria mínima de $p\%$ no valor do limite inferior após N iterações da fase de planos de corte. Fixamos $p = 0.01$ e $N = 8$, pois após vários experimentos de ajuste, observamos que a média do número de iterações do algoritmo de planos de corte em cada nó da árvore de enumeração durante toda a execução do BC era de aproximadamente 4. Surpreendentemente notamos que poucas vezes tal número ultrapassava o valor de 6 iterações. Acreditamos que esse fenômeno se justifica pelo fato de que as restrições de acoplamento entre as variáveis de decisão da formulação $\mathcal{P}_c^+$ foram mantidas no modelo de PL.

É importante salientar que as heurísticas e as buscas locais descritas no Capítulo 5 foram usadas nos nossos experimentos antes e durante a execução do algoritmo BC. Antes da sua execução, invocamos a heurística *cover-and-stitch* combinada com a heurística de refinamento *destroy-and-restore* e com os procedimentos de busca local *cluster-head exchange* e *edge swap* integrados em um arcabouço do tipo *Variable Neighborhood Descent* (VND). Descrevemos a seguir em mais detalhes como isso foi feito. Inicialmente, executamos a heurística *cover-and-stitch* e obtemos uma solução viável S para uma instância G do PSBMG. Em seguida, submetemos S a um arcabouço VND que une e aplica as buscas locais *cluster-head exchange* e *edge swap*, nessa ordem, até que nenhuma melhora seja feita na solução corrente e que uma nova solução aprimorada $\widetilde{S}$ seja retornada. Quando isso acontece, a heurística de refinamento *destroy-and-restore* é executada sobre $\widetilde{S}$. Se tal execução não for bem sucedida, usamos $\widetilde{S}$ como solução inteira inicial para o BC. Do contrário, repetimos o processo até que o arcabouço VND atinja um ótimo local e a heurística de refinamento *destroy-and-restore* falhe. Nesse caso, a solução resultante é fornecida ao algoritmo BC como solução viável de partida.

Também empregamos tais buscas locais e a heurística de refinamento durante a execução do BC com o intuito de regularmente aprimorar a melhor solução inteira corrente. Para isso, estabelecemos três critérios de aplicação. Suponhamos que S_i e g_i sejam, respectivamente, uma solução inteira encontrada pelo algoritmo BC em um instante de tempo i e o *gap* de dualidade na árvore de enumeração no momento em que S_i foi encontrada. Se para qualquer outra solução inteira S_j encontrada pelo BC em um instante de tempo j posterior a i, o *gap* de dualidade na árvore de enumeração no instante j não tiver diminuído pelo menos 10% em relação a g_i, então aplicamos

o VND a S_j. No segundo critério, se mais de 300 nós da árvore de enumeração já foram explorados e o *gap* de dualidade atual não descreceu em pelo menos 4% em reação ao seu valor no instante de tempo em que a melhor solução inteira corrente $\overline{S}$ foi obtida, aplicamos a heurística de refinamento *cover-and-stitch* sobre $\overline{S}$ uma única vez. No terceiro e último critério, se mais de 500 nós da árvore de enumeração já foram explorados e não houve nenhum aprimoramento da melhor solução inteira corrente $\overline{S}$, aplicamos sobre $\overline{S}$ a heurística de refinamento repetidas vezes enquanto ela não falhar e, em seguida, quando ela falhar, executamos o VND como descrito anteriormente até que ele atinja um ótimo local.

Por fim, cabe ressaltar que na ordem de separação das desigualdades mencionadas na subseção anterior, apenas separamos cortes de uma nova classe de desigualdades quando nenhum corte das classes de desigualdades precedentes foi encontrado. O principal motivo porque separamos *cutsets* e *node cutsets* antes das demais desigualdades reside no fato de que, nos nossos experimentos computacionais, pudemos observar que seus respectivos procedimentos de separação foram os mais rápidos em comparação com os demais e que, além disso, essas desigualdades também foram as que mais contribuíram para o limite de PL. Ao longo de toda execução do nosso algoritmo BC, priorizamos o *branching* nas variáveis de decisão da formulação $\mathcal{P}_c^+$ associadas aos vértices do grafo da instância e, em nenhum momento, usamos a estratégia de *strong branching*.

4.1.3 Experimentos Computacionais

Nesta subseção, discorremos primeiramente sobre as instâncias utilizadas em todos os experimentos computacionais conduzidos e também sobre as especificidades do ambiente computacional no qual os testes foram realizados. Em seguida, expomos e discutimos os resultados obtidos.

4.1.3.1 Instâncias de teste

As instâncias de teste usadas nos nossos experimentos computacionais foram obtidas do repositório TSPLIB [Reinelt, 1991], muito conhecido pela comunidade acadêmica de Otimização Combinatória por reunir centenas de instâncias de *benchmark* para o PCV e algumas de suas variações. Vale a pena dizer que as instâncias consideradas neste trabalho pertencem às classes de instâncias Euclidianas bidimensionais e de instâncias geográficas. Instâncias Euclidianas bidimensionais são definidas por um conjunto de pontos no plano Euclidiano usualmente representados por pares de coordenadas cartesianas. Os pesos das arestas dos grafos associados a essas instân-

cias são definidos precisamente como a distância Euclidiana que separa seus vértices extremos. Similarmente, as instâncias geográficas são definidas também por um conjunto de pontos, i.e., de localizações na superfície da Terra expressas por suas latitudes e longetudes. O peso de uma aresta dos grafos associados às instâncias geográficas é dado pela distância geodésica que liga suas extremidades. Para mais detalhes de como esse cálculo é definido, sugerimos ao leitor que consulte a documentação da TSPLIB [Reinelt, 1991]. Para os nossos experimentos computacionais, arredondamos os pesos de todas as instâncias usando a função `round` [BSD Library Functions Manual, 2012] do arquivo cabeçalho `math.h` da biblioteca padrão da linguagem de programação C. Adaptamos as instâncias da TSPLIB selecionadas para o PSBMG por meio de uma rotina de *clustering* geográfico proposta por Fischetti et al. [1997].

Na Tabela 4.1, listamos as 50 instâncias utilizadas e seus respectivos números de vértices n, de arestas m e de *clusters* k. Das 50 instâncias, somente 7 são geográficas, a saber europ47, spain47, brazil58, gr96africa, gr137america, bier127 e gr202europe. O número de vértices das instâncias varia entre 47 e 442 e o número de *clusters* entre 9 e 189. Tais instâncias já foram utilizadas em vários trabalhos da literatura, entre eles citamos [Hu et al., 2010a, 2008; Feremans et al., 2004; Pop et al., 2006]. Todos esses trabalhos abordaram versões generalizadas de outros problemas conhecidos em Otimização Combinatória, e.g., o *Problema da Árvore Geradora Mínima Generalizado* [Myung et al., 1995], o PSABMG e o PCVG [Applegate et al., 2006].

4.1.3.2 Ambiente Computacional

Os algoritmos descritos nesta dissertação foram implementados na linguagem de programação C++ e os experimentos computacionais foram executados em uma máquina com processador Intel R CoreTM i7-980 hexa-core, 3,33 GHz, 16 GB de memória RAM e 12 MB de memória *cache* L3 compartilhada. O compilador g++, versão 4.6.1, foi utilizado com a *flag* de otimização `-O3` ativada. Todas as medições de tempo de execução foram feitas com chamadas a rotinas da biblioteca LEMON [LEMON Graph Library, 2012], versão 1.2.3. Além disso, utilizamos também o *software* CPLEX, versão 10.2 [IBM ILOG CPLEX Optimizer, 2012], via a interface da biblioteca Concert Technology, versão 2.4 [ILOG Concert Technology, 2012], para solução de programas lineares e para o gerenciamento da árvore de enumeração do nosso algoritmo.

4.1.3.3 Discussão dos resultados

Nesta subseção, avaliamos experimentalmente o algoritmo BC que propomos para o PSBMG. Ressaltamos que, em todos os testes feitos, foi fixado um limite de tempo de

execução de três horas. Além disso, utilizamos a heurística construtiva *cover-and-stitch* combinada com os procedimentos de busca local discutidos no Capítulo 5 para fornecer ao algoritmo BC soluções iniciais de boa qualidade. Em linhas gerais, dividimos nossos experimentos em cinco categorias diferentes. Para fins de clareza, discorremos sobre cada uma delas separadamente.

Com o primeiro conjunto de testes, nosso objetivo foi mensurar o ganho obtido ao separar pelo menos uma das duas desigualdades válidas para o PSBMG introduzidas na Subseção 3.1.1. Com esse objetivo, isolamos dois cenários. No primeiro deles, chamado a seguir de $\mathcal{P}_c$ por um abuso de notação, separamos no algoritmo BC apenas as classes de desigualdades baseadas em *cutsets* que compõem a formulação $\mathcal{P}_c$ para o PSBMG. No segundo cenário, denotado daqui em diante, também com um abuso de notação, por $\mathcal{P}_c^+$, separamos não só as *cutsets* e *node cutsets*, como também as duas outras classes de desigualdades válidas, a saber as desigualdades de partição de nós e de F-partição, que compõem a formulação $\mathcal{P}_c^+$ para o PSBMG. Restringimo-nos a essas configurações, pois houve vários indícios nos testes realizados que os melhores resultados foram alcançados unanimemente no segundo cenário. Tal constatação está registrada na Figura 4.1, na qual mostramos a evolução do *gap* de dualidade ao longo do tempo de execução do BC usando a política de inserção de cortes θ-ortogonais e considerando todas as combinações possíveis de separação de desigualdades.

Antes de iniciar a discussão dos resultados, enfatizamos também que em todos os testes computacionais, como mencionado na Subseção 4.1.2, optamos pela política de inserção de cortes θ-ortogonais. Novamente, para ilustrar o motivo para essa decisão, apresentamos na Figura 4.2 outro gráfico que mostra a evolução do *gap* de dualidade durante a execução do algoritmo BC no cenário $\mathcal{P}_c^+$ considerando três políticas distintas de inserção de cortes, a saber a que insere absolutamente todos os cortes encontrados, apenas o corte mais violado de cada classe de desigualdades e os cortes θ-ortogonais.

Na Tabela 4.2, para os dois cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$, expomos, da esquerda para direita, seus respectivos limites de PL, o melhor limite inferior, a melhor solução inteira obtida e o total de nós na árvore de enumeração. Complementamos essa tabela com a Tabela 4.3, na qual podem ser encontrados os *gaps* de dualidade na raiz e os *gaps* de dualidade ao final da execução do BC para ambos os cenários aqui considerados. Convém esclarecer que, em todas as tabelas incluídas neste capítulo, o cálculo dos *gaps* foi realizado como descrito a seguir. Tomando como $\tilde{c}(G)$ o custo da melhor solução obtida, $w_G(\mathcal{P}_c)$ e $\underline{w}(G)$, respectivamente, o limite de PL avaliado para a formulação $\mathcal{P}_c$ e o melhor limite inferior para uma instância G, calculamos o *gap* na raiz no cenário $\mathcal{P}_c$ pela fórmula $\dfrac{\tilde{c}(G) - w_G(\mathcal{P}_c)}{w_G(\mathcal{P}_c)}$ e o *gap* após a execução do algoritmo como $\dfrac{\tilde{c}(G) - \underline{w}(G)}{\underline{w}(G)}$. Os *gaps* na

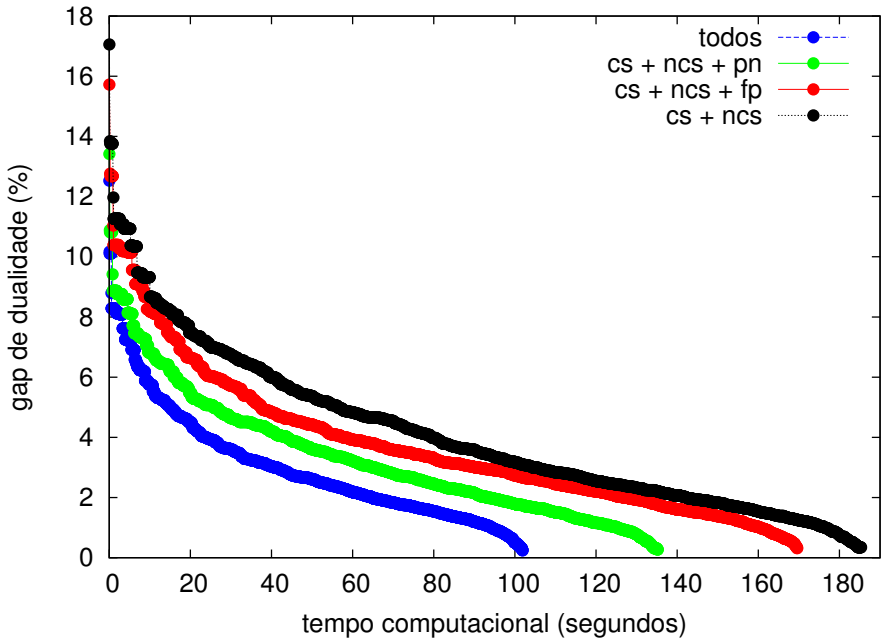

Figura 4.1. Evolução dos *gaps* de dualidade ao longo do tempo de execução (em segundos) do algoritmo BC para a instância gr96 considerando as quatro combinações possíveis de cenários de separação de cortes. Na legenda, "cs", "ncs", "pn" e "fp" significam, respectivamente, as desigualdades *cutsets*, *node cutsets*, de partição de nós e de F-partição.

raiz e após da execução do BC no cenário $\mathcal{P}_c^+$ são calculados analogamente.

Podemos observar nas Tabelas 4.2 e 4.3 que, no cenário $\mathcal{P}_c^+$, 30 das 50 instâncias de testes (ou 60% delas) foram resolvidas na otimalidade dentro do limite de tempo de execução. Isso representa quase o dobro do número de instâncias resolvidas na otimalidade no cenário $\mathcal{P}_c$, a saber 18 (36% das instâncias). No tocante ao número de vértices, a menor instância resolvida na otimalidade em ambos os cenários foi europ47 e as maiores foram, respectivamente, gr137america e gr202europe nos cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$. O *gap* de dualidade médio das instâncias para as quais o algoritmo BC não conseguiu provar a otimalidade da melhor solução inteira encontrada foi de aproximadamente 9% no cenário $\mathcal{P}_c^+$ e de 13% no cenário $\mathcal{P}_c$. Os valores médios relativamente elevados dos *gaps* de dualidade na raiz sugerem que a formulação $\mathcal{P}_c^+$ pode ser fortalecidas consideravelmente.

Na Tabela 4.2, o ganho médio no *gap* de dualidade na raiz devido à separação das desigualdades de partição de nós e de F-partição foi de aproximadamente 6%. Além disso, observamos que, em relação ao cenário $\mathcal{P}_c$, $\mathcal{P}_c^+$ proporcionou um acréscimo percentual médio do melhor limite inferior de cerca de 4%. Mais ainda, tal acréscimo chegou a 8%, considerando apenas as 12 instâncias que foram resolvidas na otimalidade

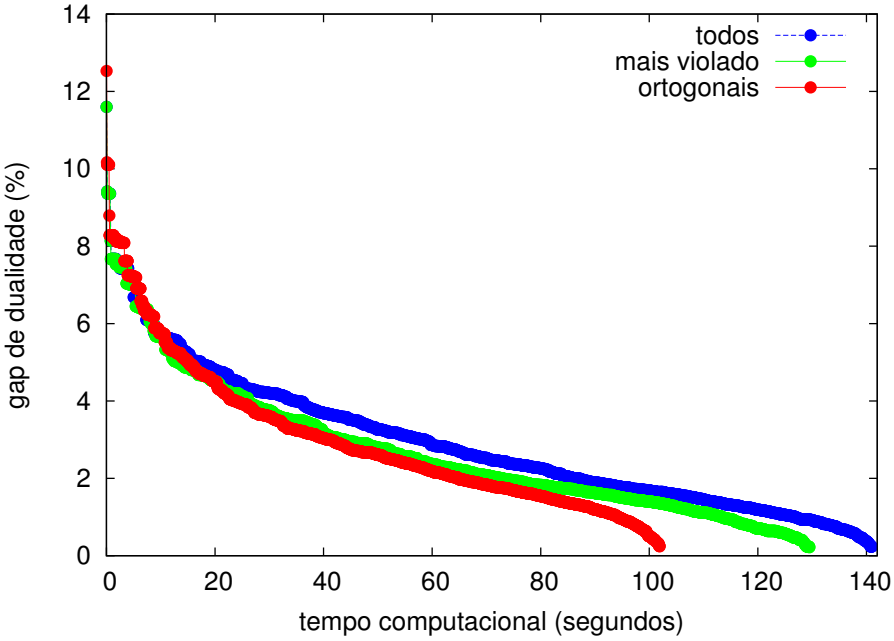

Figura 4.2. Evolução dos *gaps* de dualidade ao longo do tempo de execução (em segundos) do algoritmo BC para a instância gr96 no cenário $\mathcal{P}_c^+$ considerando as três combinações possíveis de políticas de inserção de cortes, a saber inserir todos os cortes violados encontrados, inserir apenas o mais violado e inserir cortes θ-ortogonais.

somente no cenário $\mathcal{P}_c^+$. O aprimoramento percentual médio no custo da melhor solução inteira no cenário $\mathcal{P}_c^+$ em comparação com o $\mathcal{P}_c$ foi menor do que 1%. Por fim, no que diz respeito ao número de nós explorados na árvore de enumeração, apontamos que, no cenário $\mathcal{P}_c^+$, nosso algoritmo resolveu, em média, 13% menos nós.

Na segunda categoria de testes, discutimos, nos dois cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$, o tempo de execução demandado pelo algoritmo BC para avaliar o limite de PL, seu tempo total de execução, o tempo de execução para obter a melhor solução inteira e o tempo total de execução utilizado na separação das classes de desigualdades envolvidas. Todas as medidas de tempo relatadas nesta dissertação foram dadas em segundos. Nas colunas da Tabela 4.4, listamos, da esquerda para direita, tais medidas nessa mesma ordem. A sigla LTE significa "Limite de Tempo Excedido". Ela indica que o algoritmo BC não conseguiu resolver uma determinada instância na otimalidade dentro do limite de tempo de execução de três horas e que, portanto, sua execução precisou ser abortada. Na Tabela 4.5, para cada um dos cenários em análise, mostramos as frações percentuais do tempo utilizado pelo algoritmo BC na avaliação do limite de PL e na separação das desigualdades, tomando como base os tempos totais de execução mostrados na Tabela 4.4.

Começamos ressaltando o fato de que, para as 30 instâncias resolvidas na otimalidade no cenário $\mathcal{P}_c^+$, houve um ganho médio em tempo de execução de 50% em relação aos tempos de execução despendidos pelo BC no cenário $\mathcal{P}_c$. Considerando apenas as 18 instâncias resolvidas na otimalidade em ambos os cenários, notamos que $\mathcal{P}_c^+$ superou $\mathcal{P}_c$ no tempo por uma margem de 17% na média. Além disso, em relação ao tempo necessário para avaliar o limite de PL, o cenário $\mathcal{P}_c^+$ foi, na média, menos de 1% mais lento. É importante registrar que para uma porção substancial das instâncias, principalmente aquelas com pelo menos 100 vértices e 30 *clusters*, a melhor solução inteira na árvore de enumeração foi encontrada pelo algoritmo BC depois de passados mais de três quartos do limite de tempo de execução, independentemente do cenário considerado.

Na Tabela 4.5, notamos que a quantidade de tempo utilizada pelos procedimentos de separação das desigualdades do nosso algoritmo e o tempo gasto para avaliar o limite de PL não foram proibitivos quando comparados com o tempo total de execução. Isso nos leva a concluir que a fonte principal de esforço computacional do algoritmo BC foi a solução dos programas lineares associados aos subproblemas dos nós da árvore de enumeração. Tal observação novamente reflete o fato de que a formulação $\mathcal{P}_c^+$ produz limites inferiores fracos, o que leva o BC a fazer mais *branching* e a, consequentemente, gastar uma fração considerável do tempo de execução resolvendo programas lineares.

Na terceira categoria de testes, conduzimos um estudo experimental comparativo do desempenho do nosso algoritmo BC quando executado em instâncias para o PSBM, i.e., para a versão não-generalizada do PSBMG. Com esse objetivo, tomamos como linha de base de comparação um algoritmo BC proposto por Kerivin et al. [2004] para o PSBM. Nossa meta foi avaliar que a formulação $\mathcal{P}_c^+$ apresentada para o PSBMG neste trabalho fornecia resultados equiparáveis aos resultados mostrados em [Kerivin et al., 2004], pelo menos em termos dos *gaps* de dualidade obtidos para as instâncias de teste consideradas. Dessa maneira, na Tabela 4.6, explicitamos, da esquerda para direita, os *gaps* de dualidade na raiz e os tempos de execução para avaliar o limite de PL com o algoritmo BC proposto por Kerivin et al. [2004] para o PSBM e com o nosso algoritmo. Esclarecemos que as entradas "–" na coluna dos *gaps* de dualidade relativos ao nosso algoritmo simbolizam que não foi possível determinar o valor exato do *gap* devido ao estouro do limite de tempo de execução.

Notamos primeiramente que os *gaps* obtidos pelo nosso algoritmo BC são ligeiramente maiores que os correspondentes *gaps* de dualidade apresentados em [Kerivin et al., 2004], embora ambos tenham a mesma ordem de grandeza. Acreditamos que tal constatação está parcialmente relacionada ao fato de que os autores daquele trabalho utilizaram o algoritmo proposto por Barahona & Kerivin [2003], mencionado na Sub-

seção 4.1.2, na separação das desigualdades de partição de nós. Por se tratar de um algoritmo de separação exata, espera-se que ele consiga encontrar cortes de qualidade superior aos cortes detectados pela nossa heurística de separação.

Em relação aos tempos de execução para avaliar o limite de PL, o algoritmo BC de Kerivin et al. [2004] dominou o nosso por um fator médio de aproximadamente 2.5. Ainda que os valores dos tempos de execução do nosso algoritmo BC, relatados na Tabela 4.6, estejam menores em magnitude, tivemos que contabilizar a discrepância entre os ambientes computacionais dos dois trabalhos. Em [Kerivin et al., 2004], os autores utilizaram um Pentium 3, 450 MHz com 250 MB de memória RAM, i.e., um ambiente computacional cerca de oito vezes inferior em desempenho ao nosso, conforme nossas estimativas que relativizam as frequências dos relógios dos processadores e as capacidades das memórias RAM dos dois ambientes computacionais. Mesmo com tamanha diferença, o algoritmo BC de Kerivin et al. [2004] foi capaz de suplantar nosso algoritmo no quesito tempo de execução devido ao uso de sete testes de redução projetados pelos autores e usados na aceleração dos procedimentos de separação de desigualdades do algoritmo por eles proposto.

Na quarta categoria de experimentos, investigamos de que maneira a esparsidade do conjunto de arestas das instâncias influenciou o desempenho do algoritmo BC. Para isso, geramos instâncias esparsas a partir das instâncias já selecionadas, procedendo da seguinte maneira. Dada uma instância qualquer $G = (V, E)$ para o PSBMG, fixamos em G as arestas que formam a melhor solução inteira $S = (V_S, E_S)$ encontrada para essa instância pelo nosso algoritmo BC nos dois cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$. Em seguida, construímos uma nova instância esparsa a partir de G retirando, aleatoriamente e com distribuição de probabilidade uniforme, $(100 - d)\%$ das arestas do conjunto $E \setminus E_S$, para todo $d \in \{25, 50, 75, 100\}$. Os resultados da análise do efeito da esparsidade na execução do nosso algoritmo BC estão reunidos na Tabela 4.7. Nela mostramos, para cada nível de densidade de arestas, os *gaps* de dualidade após a execução do BC e o seu tempo total de execução no cenário $\mathcal{P}_c^+$.

Na Tabela 4.7, observamos que, além das 30 instâncias resolvidas na otimalidade no cenário de densidade total, nenhuma instância de teste a mais foi resolvida na otimalidade para os níveis de densidade de 75% e 50%. No entanto, para 25% de densidade de arestas, 8 novas instâncias foram resolvidas na otimalidade, a maior delas, em número de vértices, sendo pr264 e a menor kroc100. Além disso, observamos que 12 instâncias permaneceram com *gap* de dualidade diferente de zero em todos os casos. Para tais instâncias, em cada diminuição de 25% no número de arestas, o ganho médio no *gap* de dualidade foi de aproximadamente 4%. No tocante às instâncias que foram resolvidas na otimalidade em todos os casos, notamos que nosso algoritmo BC

ficou, na média, 21% mais rápido à medida elas foram esparsificadas de 25% em 25%.

Na quinta categoria de testes, procuramos analisar o impacto causado na solução do problema pela *clusterização* ou aglomeração dos vértices das instâncias. Com esse objetivo, projetamos grupos de experimentos para estudar o desempenho do nosso algoritmo BC quando executado em uma mesma instância com vértices submetidos a quatro níveis diferentes de aglomeração. Na sequência, relatamos como foram geradas as instâncias para essa categoria de testes. Sejam $G = (V, E)$ uma instância para o PSBMG e Q uma k-partição de V. Para todo $\alpha \in \{0, 25, 50, 75\}$, geramos uma nova instância $G' = (V, E')$ a partir de G, com uma k'-partição Q' de V, onde $k' = \left\lceil \frac{(100-\alpha)}{100} n \right\rceil$, da seguinte maneira. Denotando por q e r, respectivamente, o quociente e o resto da divisão inteira de n por k', posicionamos $q+1$ vértices de V em cada um dos primeiros r *clusters* de Q' e q vértices de V em cada um dos $k' - r$ *clusters* restantes de Q'. A ordem com que escolhemos tais vértices segue a mesma ordem na qual eles se encontram listados nos arquivos de texto das instâncias da TSPLIB.

Finalmente, na Tabela 4.8, explicitamos os *gaps* de dualidade após a execução do nosso algoritmo BC e seu tempo total de execução no cenário $\mathcal{P}_c^+$ para os quatro níveis de aglomeração descritos. Em uma primeira análise, para o caso de aglomeração nula, i.e., em que cada *cluster* contém exatamente um vértice ou, em outras palavras, quando o PSBMG se degenera no PSBM, todas as instâncias de teste listadas são resolvidas na otimalidade, com exceção das instâncias pr299, pr439 e pcb442. Para cada transição do nível de aglomeração em que o número de vértices por *cluster* das instâncias aumenta em 25%, 9 novas instâncias, na média, deixam de ser resolvidas na otimalidade pelo BC dentro do limite de tempo de execução imposto. No entanto, em todos os níveis de aglomeração, o algoritmo BC consegue resolver na otimalidade 20 instâncias de teste, a menor delas sendo europ47 e a maior pr107. Para essas instâncias, observamos que, na média, o tempo de execução total do BC praticamente dobra sempre que o nível de aglomeração dos vértices cresce em 25%.

Tabela 4.1. Nomes, número de vértices (n), de arestas (m) e de *clusters* (k) das instâncias da TSPLIB utilizadas nos experimentos computacionais.

Instâncias			
Nomes	n	m	k
europ47	47	1042	27
spain47	47	985	15
att48	48	1010	12
hk48	48	995	10
gr48	48	1017	9
eil51	51	1158	11
brazil58	58	1464	12
st70	70	2248	18
eil76	76	2660	15
pr76	76	2661	17
gr96	96	4292	21
gr96africa	96	4463	52
rat99	99	4609	20
kroc100	100	4714	23
kroa100	100	4727	24
krod100	100	4716	70
rd100	100	4703	22
kroe100	100	4702	25
krob100	100	4716	27
eil101	101	4776	21
lin105	105	5130	23
pr107	107	5441	22
gr120	120	6820	31
pr124	124	7315	25
bier127	127	7191	26
pr136	136	8879	28
gr137	137	8892	30
gr137america	137	8251	35
pr144	144	9952	29
kroa150	150	10809	35
krob150	150	10807	44
pr152	152	11080	31
u159	159	12031	32
rat195	195	18478	39
d198	198	18841	58
krob200	200	19430	40
kroa200	200	19409	47
gr202	202	19532	41
gr202europe	202	19018	34
ts225	225	24650	105
pr226	226	24626	56
gil262	262	33521	63
pr264	264	34028	93
pr299	299	43786	116
lin318	318	49363	167
rd400	400	78754	110
fl417	417	84841	134
gr431	431	88666	87
pr439	439	94585	98
pcb442	442	96360	109
pr439	439	94585	185
pcb442	442	96360	189

Tabela 4.2. Limite de PL, melhor limite inferior, melhor solução inteira e número de nós na árvore de enumeração obtidos pelo algoritmo BC nos cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$.

Instâncias	Lim. Prog. Linear		Melhor lim. inf.		Melhor sol. int.		Núm. nós	
Nomes	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$
europ47	16559.4	17325.6	18246	18246	18246	18246	254	212
spain47	2810.8	3003.3	3271	3271	3271	3271	592	528
att48	13124.2	13728.3	17669	17669	17669	17669	1262	1080
hk48	4742.2	5057.6	6493	6493	6493	6493	4781	4202
gr48	1387	1479.9	1860	1860	1860	1860	376	315
eil51	121.3	127.2	171	171	171	171	1917	1700
brazil58	12651	13521.7	15430	15430	15430	15430	1069	956
st70	227.2	241.7	304	304	304	304	3711	3249
eil76	173.4	180.6	237	237	237	237	25719	22711
pr76	51703.2	54460	63106	63106	63106	63106	10676	9304
gr96	221.2	230.8	261	261	261	261	5372	4543
gr96africa	329.5	349.6	362	362	362	362	3924	3299
rat99	335.8	351.7	476.1	507	511	507	11285	10011
kroc100	6723.7	7111.5	9328.9	9606.6	9806	9784	11327	10032
kroa100	6663	6946.5	9620.1	9964	9968	9964	12208	10282
krod100	6854.3	7304.1	9628	9628	9628	9628	13980	12306
rd100	2522.3	2679.7	3384	3384	3384	3384	34313	29420
kroe100	7057.4	7393.5	9995	9995	9995	9995	12128	10115
krob100	7126.3	7619	9437	9437	9437	9437	8244	6856
eil101	197.1	209	248.6	261	261	261	93929	78923
lin105	6338.8	6666.8	8123.3	8219	8219	8219	12644	10758
pr107	25191	26619.2	27526.7	27659	27659	27659	12440	10653
gr120	2218.4	2364	2694.6	2818	2829	2818	54731	49001
pr124	29408.8	31447.1	34138.9	36707	36761	36707	49019	43757
bier127	65993.5	69627.9	72464	72464	72464	72464	22770	20491
pr136	27606.1	29226.7	34660.6	38695.4	41993	41157	24907	20742
gr137	351.8	370.6	395.5	416.2	445	440	26382	23239
gr137america	231.9	241.5	280	280	280	280	7350	6321
pr144	42340.3	44255.6	45049	45321.7	45903	45830	31506	26261
kroa150	8338.6	8864.8	9986.9	11124.8	11697	11520	18277	15680
krob150	8775.2	9309.3	10098.8	11175	11199	11175	20879	18400
pr152	45119.8	47606.2	48524.6	51718	51858	51718	21189	17671
u159	16444.9	17312.1	19282.6	22403.3	23002	22976	14962	13460
rat195	599.4	631.9	656.2	702.1	851	813	6784	5766
d198	9484.8	10118.8	9858.3	10589	10702	10589	4527	3953
krob200	9362.4	9847.2	10258	11365.7	13344	13045	5374	4769
kroa200	9716.6	10212.3	10805.8	12886.1	13689	13570	5450	4848
gr202	293.9	306.9	301.2	318	324	318	3615	3001
gr202europe	154.9	164.1	169.5	192	193	192	3836	3215
ts225	40449.3	42895.9	43378.5	47241.9	68850	68122	2588	2272
pr226	56505.7	59033.2	58753.6	60178.5	64269	64111	3747	3219
gil262	766.3	806	796.5	807	1038	983	1208	1086
pr264	25864.3	27614.6	26418.3	28544.1	30269	30121	1515	1283
pr299	15906.1	16567.1	16605.7	16989	23417	22709	794	662
lin318	17137.3	18213.5	18462.5	19538.5	21589	20795	525	459
rd400	4799.6	5094	5115	6124.6	6801	6765	83279	80172
fl417	7999.1	8444.3	8466.6	9002.3	10147	9987	178209	155126
gr431	789.1	899.2	954.7	1004.8	1291	1273	193651	175028
pr439	48456.9	51639.8	51808.4	53408.1	60815	59970	162987	135086
pcb442	14649.7	15512.8	15636.9	18098.4	22321	19957	175034	155123

Tabela 4.3. *Gaps* na raiz e *gaps* ao final da execução do algoritmo BC nos cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$.

Instâncias	*Gaps* raiz		*Gaps* final exec.	
Nomes	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$
europ47	9.2	5	0	0
spain47	14.1	8.2	0	0
att48	25.7	22.3	0	0
hk48	27	22.1	0	0
gr48	25.4	20.4	0	0
eil51	29.1	25.6	0	0
brazil58	18	12.4	0	0
st70	25.3	20.5	0	0
eil76	26.8	23.8	0	0
pr76	18.1	13.7	0	0
gr96	19.2	11.6	0	0
gr96africa	9	3.4	0	0
rat99	33.8	30.6	6.1	0
kroc100	31.3	27.3	4.7	1.8
kroa100	33.1	30.3	3.5	0
krod100	28.8	24.1	0	0
rd100	25.5	20.8	0	0
kroe100	29.4	26	0	0
krob100	24.5	19.3	0	0
eil101	24.5	19.9	4.8	0
lin105	22.9	18.9	1.2	0
pr107	8.9	3.8	0.5	0
gr120	21.3	16.1	4.4	0
pr124	19.9	14.3	7	0
bier127	8.9	3.9	0	0
pr136	32.9	29	15.8	6
gr137	20	15.8	10.1	5.4
gr137america	17.2	13.8	0	0
pr144	7.6	3.4	1.7	1.1
kroa150	27.6	23	13.3	3.4
krob150	21.5	16.7	9.6	0
pr152	12.8	8	6.2	0
u159	28.4	24.7	16.1	2.5
rat195	26.3	22.3	19.3	13.6
d198	10.4	4.4	6.9	0
krob200	28.2	24.5	21.4	12.9
kroa200	28.4	24.7	20.4	5
gr202	7.6	3.5	5.3	0
gr202europe	19.3	14.5	11.7	0
ts225	40.6	37	36.3	30.7
pr226	11.9	7.9	8.4	6.1
gil262	22	18	19	17.9
pr264	14.1	8.3	12.3	5.2
pr299	30	27	26.9	25.2
lin318	17.6	12.4	11.2	6
rd400	29.1	24.7	24.4	9.5
fl417	19.9	15.4	15.2	9.9
gr431	38	29.4	25	21.1
pr439	19.2	13.9	13.6	10.9
pcb442	26.6	22.3	21.6	9.3
Médias	22.3	17.8	8.1	4.1

Tabela 4.4. Tempo de execução (em segundos) para avaliar o limite de PL, tempo total de execução do algoritmo BC, tempo de execução para obter a melhor solução inteira e tempo de execução agregado usado na separação das desigualdades nos cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$.

Instâncias	Lim. Prog. Linear		Tempo total exec.		Melhor sol. int.		Separ. desig.	
Nomes	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$
europ47	0.2	0.4	5.2	4.4	4.6	3.9	0.5	0.5
spain47	0	0.1	3	2.7	2.6	1.8	0.3	0.4
att48	0.1	0.1	3.3	3	2.8	1.6	0.4	0.5
hk48	0.1	0.1	32.7	26.9	29	23	3	4.3
gr48	0	0	2.2	1.8	1.9	0.9	0.2	0.2
eil51	0.1	0	16.6	13.8	14.2	12	2.1	1.5
brazil58	0.1	0.2	14.7	11.8	12.4	10	1.8	1.6
st70	0.2	0.2	77.7	64.9	68.2	55	7.7	7.7
eil76	0.2	0.2	218.4	179.9	192	153	25.2	19.6
pr76	0.3	0.3	315.7	273.2	277.9	226	37.8	36.3
gr96	0.5	0.7	190.8	124.3	130.5	108	18.2	18.2
gr96africa	2.5	5	387.3	354.7	336.2	296	45.3	39.6
rat99	1.1	1.2	2770.1	2295.2	2312.2	1944	258.2	327.4
kroc100	0.9	1.1	LTE	LTE	8122	6948	1153	1033.9
kroa100	0.7	0.8	LTE	8486.9	8300.6	6975	996.4	998.2
krod100	0.7	0.9	10615.6	10147.2	9386.3	8434	1024.9	1471.2
rd100	1.1	1.2	2663.9	2205.1	2345	1887	341.9	256.3
kroe100	1	0.8	8420.8	7224.6	7384	6340	858.1	966.1
krob100	1	1.1	168.5	145.6	148.4	127	19.4	22.4
eil101	0.7	0.8	LTE	1312.7	1312.2	1138	161.5	210
lin105	0.7	0.9	LTE	8800.5	8648.4	7451	990.6	1153.1
pr107	1.8	2.4	LTE	6817.5	6572.2	5846	839.1	902.9
gr120	2.1	4	LTE	7506.6	7747.4	6549	851.3	1010
pr124	2.8	3.9	LTE	4183.7	4191.7	3511	547.8	437
bier127	1.7	2.7	LTE	3924.3	3909.5	3371	427.9	399.4
pr136	3.1	4.3	LTE	LTE	10573.6	8423	1326.6	1454.6
gr137	3.3	3.9	LTE	LTE	5784.4	4689	722.3	726.2
gr137america	3	4.2	44.5	33.6	37.4	30	5.3	4.9
pr144	5	6.4	LTE	LTE	9303.6	7461	1395.2	943
kroa150	4.3	6.2	LTE	LTE	3997.5	3542	579.9	635.8
krob150	4.2	4.5	LTE	2661.8	2738.1	2283	340.3	300.1
pr152	6.6	8.7	LTE	8511.8	8490.1	7153	1096.7	1006.7
u159	6.1	7.3	LTE	LTE	9972.7	8171	1017.1	1303.2
rat195	15.5	13.4	LTE	LTE	4143.3	3322	454.2	565.9
d198	32.9	33	LTE	3122.2	3167.2	2630	361.8	324.8
krob200	22.7	27.7	LTE	LTE	2491.7	2123	303.2	374
kroa200	12.8	15.8	LTE	LTE	8055	6858	1171.1	1394.2
gr202	23.7	30.3	LTE	2158.4	2134.8	1836	285.5	309.9
gr202europe	19.5	16.4	LTE	9196.2	8844.4	7825	1295	1479.7
ts225	20.6	27.7	LTE	LTE	7377.8	6514	1031.8	821.9
pr226	56.4	92.6	LTE	LTE	9507.1	8138	1078.4	1185.9
gil262	69	106.1	LTE	LTE	2817.6	2393	421.2	457.8
pr264	163.8	203.8	LTE	LTE	5705.7	4643	719.4	901.7
pr299	374.9	512	LTE	LTE	6316.1	5235	842.9	649.4
lin318	313	516.3	LTE	LTE	8992.4	8058	1069.7	1549.9
rd400	636.4	702.1	LTE	LTE	9002.1	8993.2	1294.4	1371.3
fl417	703.9	952.5	LTE	LTE	9345.1	9112.2	1304.5	1670.3
gr431	972.9	1103.3	LTE	LTE	9910.3	9336.5	1502.6	1938.2
pr439	512.2	748.1	LTE	LTE	9244.2	8999.2	2034.8	2204.1
pcb442	664.7	786.1	LTE	LTE	10022.1	9978.3	2147	2560.3
pr439	639.7	840.3	LTE	LTE	9678.8	9441.2	2203.4	2576.3
pcb442	664.7	786.1	LTE	LTE	9546.3	9002.6	2299.1	2890.1

Tabela 4.5. Frações percentuais dos tempos de execução, em relação ao tempo total de execução relatado na Tabela 4.4, que o algoritmo BC usa para avaliar o limite de PL e para separar as desigualdades nos cenários $\mathcal{P}_c$ e $\mathcal{P}_c^+$.

Instâncias	Lim. Prog. Linear (%)		Separ. desig. (%)	
Nomes	$\mathcal{P}_c$	$\mathcal{P}_c^+$	$\mathcal{P}_c$	$\mathcal{P}_c^+$
europ47	3.85	9.09	9.62	11.36
spain47	0	3.7	10	14.81
att48	3.03	3.33	12.12	16.67
hk48	0.31	0.37	9.17	15.99
gr48	0	0	9.09	11.11
eil51	0.6	0	12.65	10.87
brazil58	0.68	1.69	12.24	13.56
st70	0.26	0.31	9.91	11.86
eil76	0.09	0.11	11.54	10.89
pr76	0.1	0.11	11.97	13.29
gr96	0.34	0.56	12.23	14.64
gr96africa	0.65	1.41	11.7	11.16
rat99	0.04	0.05	9.32	14.26
kroc100	0.01	0.01	10.68	9.57
kroa100	0.01	0.01	9.23	11.76
krod100	0.01	0.01	9.65	14.5
rd100	0.04	0.05	12.83	11.62
kroe100	0.01	0.01	10.19	13.37
krob100	0.59	0.76	11.51	15.38
eil101	0.01	0.06	1.5	16
lin105	0.01	0.01	9.17	13.1
pr107	0.02	0.04	7.77	13.24
gr120	0.02	0.05	7.88	13.45
pr124	0.03	0.09	5.07	10.45
bier127	0.02	0.07	3.96	10.18
pr136	0.03	0.04	12.28	13.47
gr137	0.03	0.04	6.69	6.72
gr137america	6.74	12.5	11.91	14.58
pr144	0.05	0.06	12.92	8.73
kroa150	0.04	0.06	5.37	5.89
krob150	0.04	0.17	3.15	11.27
pr152	0.06	0.1	10.15	11.83
u159	0.06	0.07	9.42	12.07
rat195	0.14	0.12	4.21	5.24
d198	0.3	1.06	3.35	10.4
krob200	0.21	0.26	2.81	3.46
kroa200	0.12	0.15	10.84	12.91
gr202	0.22	1.4	2.64	14.36
gr202europe	0.18	0.18	11.99	16.09
ts225	0.19	0.26	9.55	7.61
pr226	0.52	0.86	9.99	10.98
gil262	0.64	0.98	3.9	4.24
pr264	1.52	1.89	6.66	8.35
pr299	3.47	4.74	7.8	6.01
lin318	2.9	4.78	9.9	14.35
rd400	5.89	6.5	11.99	12.7
fl417	6.52	8.82	12.08	15.47
gr431	9.01	10.22	13.91	17.95
pr439	4.74	6.93	18.84	20.41
pcb442	6.15	7.28	19.88	23.71
pr439	5.92	7.78	20.4	23.85
pcb442	6.15	7.28	21.29	26.76
Médias	1.4	2.1	10	12.7

Tabela 4.6. *Gaps* na raiz e os tempos de execução (em segundos) do nosso algoritmo BC para avaliar o limite de PL com a formulação $\mathcal{P}_c^+$ e do algoritmo BC proposto por Kerivin et al. [2004] para o PSBM. Para uma comparação justa dos tempos de execução, em decorrência das diferenças dos ambientes computacionais de ambos os trabalhos, cabe ressaltar que os nossos resultados precisam ser multiplicados por um fator de 8.

Instâncias	*Gaps* raiz		Tempo exec. lim. Prog. Linear	
Nomes	[Kerivin et al., 2004]	Este trabalho	[Kerivin et al., 2004]	Este trabalho
eil51	0	0.01	1.89	0.51
A50	0.06	0.11	0.8	0.42
st70	0.22	0.89	1.98	0.56
eil76	0.19	0.51	0.29	0.23
pr76	0.24	0.33	3.06	1.34
rat99	0.13	0.15	2.1	0.78
kroa100	0.1	0.17	9.47	2.16
krob100	0.21	0.22	20.51	4.89
kroc100	0.19	0.25	7.49	2.03
krod100	0.15	0.17	2.55	0.35
kroe100	0.03	0.09	4.69	1.34
rd100	0	0.01	1.16	0.4
lin105	0	0	0.78	0.12
pr107	0	0	0.55	0.01
pr124	1.18	1.2	24.87	5.98
bier127	0.18	0.43	52.1	12.54
ch130	0.15	0.15	180.75	46.71
pr144	0.17	0.18	20.33	5.3
ch150	0.18	0.18	179.48	45.78
pr152	0.45	0.56	1142.27	229.86
rat195	0.36	0.4	2923.05	584.22
d198	0.26	0.28	3034.25	749.01
A200	0.1	0.11	165.31	45.5
krob200	0.14	0.16	215.4	53.1
ts225	0	0.3	6.91	2.57
tsp225	0	–	19898.87	LTE
pr264	0.22	0.22	606.38	150.23
pr299	0.004	0.023	518.72	169.86
lin318	0.03	0.1	4204.63	702.46
u574	0.04	–	199424.78	LTE
Médias	0.2	0.3		

Tabela 4.7. *Gaps* ao final da execução e tempo total de execução (em segundos) do algoritmo BC com a formulação $\mathcal{P}_c^+$ considerando diferentes níveis de densidade de arestas nas instâncias de teste.

Instâncias	*Gaps* final exec.				Tempo total exec.			
	Níveis de dens. arestas				Níveis de dens. arestas			
Nomes	25%	50%	75%	100%	25%	50%	75%	100%
europ47	0	0	0	0	1.3	2.4	3.5	4.4
spain47	0	0	0	0	0.7	1.4	2.1	2.7
att48	0	0	0	0	0.8	1.6	2.4	3
hk48	0	0	0	0	7.3	14.6	20.8	26.9
gr48	0	0	0	0	0.5	1	1.5	1.8
eil51	0	0	0	0	4.2	7.4	10.7	13.8
brazil58	0	0	0	0	3.5	6.3	9.5	11.8
st70	0	0	0	0	19.4	34.6	51.3	64.9
eil76	0	0	0	0	54.1	96.6	140.1	179.9
pr76	0	0	0	0	77.7	141.1	221.5	273.2
gr96	0	0	0	0	36	65.6	99.6	124.3
gr96africa	0	0	0	0	106.3	182.7	284.9	354.7
rat99	0	0	0	0	642.2	1247	1776.8	2295.2
kroc100	0	0.8	1.2	1.8	8932.1	LTE	LTE	LTE
kroa100	0	0	0	0	2300.5	4624.8	6667.7	8486.9
krod100	0	0	0	0	2773.6	5201	8131.5	10147.2
rd100	0	0	0	0	678.5	1189.5	1708.8	2205.1
kroe100	0	0	0	0	2226.8	3893.8	5771.7	7224.6
krob100	0	0	0	0	41.8	78.5	116.6	145.6
eil101	0	0	0	0	395	681	1060.2	1312.7
lin105	0	0	0	0	2660.3	4518.1	7039.8	8800.5
pr107	0	0	0	0	1882.8	3554.8	5546.3	6817.5
gr120	0	0	0	0	2221.1	3843.7	6067.1	7506.6
pr124	0	0	0	0	1285.3	2285.7	3417.5	4183.7
bier127	0	0	0	0	1206.4	2127.2	3036.2	3924.3
pr136	0	2.6	4.4	6	9456.7	LTE	LTE	LTE
gr137	0	2.3	3.8	5.4	10003.1	LTE	LTE	LTE
gr137america	0	0	0	0	10.1	18.4	26.4	33.6
pr144	0	0.4	0.8	1.1	9789.3	LTE	LTE	LTE
kroa150	0	1.6	2.4	3.4	9435.7	LTE	LTE	LTE
krob150	0	0	0	0	758.6	1437.1	2154.3	2661.8
pr152	0	0	0	0	2568.5	4433.6	6967.7	8511.8
u159	0	1	1.8	2.5	9856.7	LTE	LTE	LTE
rat195	2.9	5.6	9.5	13.6	LTE	LTE	LTE	LTE
d198	0	0	0	0	941.2	1700.1	2497.7	3122.2
krob200	2.6	5.3	9.2	12.9	LTE	LTE	LTE	LTE
kroa200	0	2.2	3.6	5	9932.1	LTE	LTE	LTE
gr202	0	0	0	0	594.5	1165.2	1727.9	2158.4
gr202europe	0	0	0	0	2756.5	4867.3	7450.2	9196.2
ts225	5	13.1	22.1	30.7	LTE	LTE	LTE	LTE
pr226	1.1	2.8	4.3	6.1	LTE	LTE	LTE	LTE
gil262	3.5	8.2	12.9	17.9	LTE	LTE	LTE	LTE
pr264	0	2.1	3.6	5.2	9380.2	LTE	LTE	LTE
pr299	4.3	11.2	18.4	25.2	LTE	LTE	LTE	LTE
lin318	1.8	3.3	5.3	6	LTE	LTE	LTE	LTE
rd400	2.2	4	6.7	9.5	LTE	LTE	LTE	LTE
fl417	2.5	4.5	7.2	9.9	LTE	LTE	LTE	LTE
gr431	4.4	9.4	15.3	21.1	LTE	LTE	LTE	LTE
pr439	2.5	4.7	7.9	10.9	LTE	LTE	LTE	LTE
pcb442	1.7	4	6.7	9.3	LTE	LTE	LTE	LTE
Médias	1.7	1.8	2.9	4.1				

Tabela 4.8. *Gaps* ao final da execução e tempo total de execução (em segundos) do algoritmo BC com a formulação $\mathcal{P}_c^+$ considerando diferentes níveis de aglomeração dos vértices nas instâncias de teste.

Instâncias	*Gaps* na árv. BB				Tempo total exec.			
	Níveis de aglomeração				Níveis de aglomeração			
Nomes	75%	50%	25%	0%	75%	50%	25%	0%
europ47	0	0	0	0	31.2	12.8	8.9	1.2
spain47	0	0	0	0	44.3	20.7	17.7	1.1
att48	0	0	0	0	60	26.9	21.7	1.4
hk48	0	0	0	0	55.8	23.7	20	1.7
gr48	0	0	0	0	50	24.9	19.4	0.9
eil51	0	0	0	0	57.6	28.9	18.4	1.5
brazil58	0	0	0	0	68.4	31.7	25	1.7
st70	0	0	0	0	73.2	36.2	28.8	1.2
eil76	0	0	0	0	30.3	14.6	12.8	2.2
pr76	0	0	0	0	175.3	81.1	46.7	2.1
gr96	0	0	0	0	98.5	47.6	43.9	3.2
gr96africa	0	0	0	0	79.4	36.7	28.7	3.7
rat99	0	0	0	0	76.2	35.6	30.5	3.7
kroc100	0	0	0	0	311.7	136.7	132.5	5.6
kroa100	0	0	0	0	560.6	272.8	222.3	5.7
krod100	0	0	0	0	228.6	103.9	102	5.2
rd100	0	0	0	0	70	36.3	25	5.9
kroe100	0	0	0	0	138.4	62.6	40.1	6.1
krob100	0	0	0	0	32.4	16.7	9.8	7.7
eil101	30.8	16.3	13.8	0	LTE	LTE	LTE	8.0
lin105	0	0	0	0	21.1	11.4	8.2	4.3
pr107	0	0	0	0	17.8	8.6	5.7	6.2
gr120	31.3	27.5	12.4	0	LTE	LTE	LTE	9.5
pr124	0	0	0	0	1495.7	737.6	578.3	7.8
bier127	41.2	0	0	0	LTE	671.2	414.8	24.3
pr136	32.9	19.9	0	0	LTE	LTE	473.3	32.7
gr137	30.3	19.7	0	0	LTE	LTE	366.3	42.9
gr137america	7.5	0	0	0	LTE	1192.5	205.5	58.1
pr144	37.9	23.9	0	0	LTE	LTE	10480.2	65.1
kroa150	17.4	9.8	0	0	LTE	LTE	9072.3	450.7
krob150	24.2	14	0	0	LTE	LTE	10437.3	293
pr152	16.5	10.6	7.1	0	LTE	LTE	LTE	332.3
u159	12	6.8	5.9	0	LTE	LTE	LTE	426.1
rat195	7.8	4.1	3	0	LTE	LTE	LTE	842.7
d198	22.4	13.8	0	0	LTE	LTE	2819.7	910.9
krob200	15.3	9.3	6.3	0	LTE	LTE	LTE	150.2
kroa200	11.9	0	0	0	LTE	9505.7	8247	821.7
gr202	12.2	7.6	0	0	LTE	LTE	109.5	501.2
gr202europe	42.9	25.5	16.9	0	LTE	LTE	LTE	70.2
ts225	45.8	28.8	19.6	0	LTE	LTE	LTE	948.5
pr226	10.5	6.1	4.1	0	LTE	LTE	LTE	741.2
gil262	0.9	0.7	0.5	0	LTE	LTE	LTE	600.2
pr264	4.7	2.9	2	0	LTE	LTE	LTE	223
pr299	17.9	8.7	6.4	–	LTE	LTE	LTE	LTE
lin318	27.8	10.3	5.6	0	LTE	LTE	LTE	1011.2
rd400	43.1	24.7	20.7	0	LTE	LTE	LTE	2075.1
fl417	35.2	19.7	14.3	0	LTE	LTE	LTE	2218.0
gr431	36.5	22.4	18.5	0	LTE	LTE	LTE	2091.5
pr439	37.8	21.3	19.3	–	LTE	LTE	LTE	LTE
pcb442	40.2	26.7	20.3	–	LTE	LTE	LTE	LTE
Médias	14.2	7.9	4.2	–				

Tabela 4.9. Número de cortes encontrados das classes de desigualdades separadas pelo algoritmo BC considerando dois cenários de execução distintos $\mathcal{P}_c$ e $\mathcal{P}_c^+$.

Instâncias	$\mathcal{P}_c$		$\mathcal{P}_c^+$			
Nomes	*cutsets*	*node cutsets*	*cutsets*	*node cutsets*	partição de nós	F-partição
europ47	27	27	23	21	24	20
spain47	15	15	12	11	7	7
att48	10	10	8	7	25	7
hk48	10	10	8	7	17	2
gr48	10	10	8	7	13	6
eil51	11	11	9	8	1	4
brazil58	12	12	10	9	29	8
st70	14	14	12	10	46	9
eil76	16	16	14	12	22	18
pr76	16	16	13	12	51	10
gr96	20	20	17	14	63	10
gr96africa	50	50	42	39	206	50
rat99	20	20	17	15	180	13
kroc100	20	20	17	15	374	7
kroa100	20	20	17	14	112	11
krod100	20	20	17	14	130	23
rd100	20	20	17	15	123	10
kroe100	20	20	17	14	105	11
krob100	20	20	17	15	52	9
eil101	21	21	17	16	438	32
lin105	21	21	17	15	97	12
pr107	22	22	18	17	176	10
gr120	24	24	20	18	542	16
pr124	25	25	21	18	282	9
bier127	26	26	21	20	320	23
pr136	28	28	22	21	583	15
gr137	28	28	22	21	100	39
gr137america	35	35	31	26	361	40
pr144	29	29	23	22	290	11
kroa150	30	30	26	23	319	29
krob150	30	30	26	23	548	13
pr152	31	31	27	24	585	16
u159	32	32	28	24	671	18
rat195	39	39	33	29	115	77
d198	40	40	34	29	661	26
krob200	40	40	34	29	415	30
kroa200	40	40	35	30	381	29
gr202	41	41	36	30	824	64
gr202europe	34	34	30	26	464	52
ts225	45	45	37	34	534	58
pr226	46	46	40	35	717	20
gil262	53	53	47	41	472	67
pr264	53	53	43	39	977	25
pr299	60	60	50	45	526	50
lin318	64	64	55	48	534	44
rd400	317	759	282	623	339	32
fl417	648	1303	552	1054	773	45
gr431	428	693	401	620	882	30
pr439	355	1043	320	862	855	26
pcb442	560	711	499	584	245	22

Capítulo 5

Métodos de Solução Heurística para o PSBMG

Métodos heurísticos, em geral, são projetados com o objetivo de obter rapidamente soluções viáveis de boa qualidade para problemas computacionalmente intratáveis e, por isso, desempenham um papel importante em Otimização Combinatória. Em particular, no contexto de famílias de algoritmos exatos que usam mecanismos de enumeração implícita, e.g., algoritmos *Branch-and-cut*, soluções heurísticas vêm a ser úteis na poda de ramos sub-ótimos da chamada árvore de enumeração de soluções. Neste capítulo, discutimos três heurísticas e dois procedimentos de busca local para o PSBMG. Duas das três heurísticas abordadas, *2-tree* e *cover-and-stitch*, são ditas construtivas e a terceira delas, denominada *destroy-and-restore*, é o que chamamos de uma de heurística de refinamento. Apresentamos também duas buscas locais para o PSBMG, a saber *cluster-head exchange* e *edge swap*. Para todos os métodos de solução heurística propostos a seguir, tratamos brevemente dos seus respectivos detalhes de implementação. Por fim, encerramos o capítulo apresentando e analisando os resultados obtidos nos experimentos computacionais realizados.

5.1 Heurísticas Construtivas

Dado um problema de Otimização Combinatória, uma ***heurística construtiva*** é, via de regra, um algoritmo eficiente capaz de encontrar uma solução viável, não necessariamente ótima, partindo unicamente de uma instância para esse problema. A seguir, introduzimos nas Subseções 5.1.1 e 5.1.2 duas heurísticas para o PSBMG, chamadas de *2-tree* e *cover-and-stitch*.

5.1.1 Heurística *2-tree*

A heurística *2-tree* (2T) que apresentamos é uma adaptação de uma heurística proposta para o PSBM por Monma & Shallcross [1989]. Dada uma instância I para o PSBMG, definimos uma ***árvore geradora generalizada*** (AGG) de I como um subgrafo conexo acíclico contendo exatamente um vértice de cada *cluster* de I. Intuitivamente, a 2T obtém uma solução viável para o PSBMG em I combinando uma AGG T de I com uma árvore geradora mínima do subgrafo de I induzido pelas folhas de T. Implementamos a heurística 2T usando uma modificação do conhecido algoritmo de Prim [1957]. Abaixo mostramos seu pseudo-código. A heurística 2T recebe como entrada uma instância para o PSBMG, i.e., um grafo $G = (V, E)$, com $n = |V|$, uma k-partição $Q = \{V_1, \ldots, V_k\}$ de V, onde $k = |Q| > 2$, e uma função $\gamma : V \to \{1, \ldots, k\}$, tal que para todo vértice $v \in V$, $\gamma(v)$ indica o índice do *cluster* em G ao qual o vértice v pertence. A heurística retorna uma solução (não necessariamente viável) S para o PSBMG em G.

2-TREE$(G = (V, E), Q, \gamma)$
1 $n_S = |Q|$
2 Escolha a aresta $\{u, v\}$ de menor peso em E
3 $E_S = \{\{u, v\}\}$
4 $V_S = \{u, v\}$
5 $i = \gamma(u)$
6 $j = \gamma(v)$
7 $X = (V_i \setminus \{u\}) \cup (V_j \setminus \{v\})$ // Todas as arestas em $\delta_G(X)$ serão excluídas
8 **for** $h = 2$ **to** $n_S - 1$ // Laço no qual é contruída uma AGG de G
9 Escolha a aresta $\{p, q\}$ de menor peso em $\delta_G(V_S) \setminus \delta_G(X)$
10 Tome z tal que $z \in \{p, q\} \setminus V_S$
11 $w = \gamma(z)$
12 $X = X \cup (V_w \setminus \{z\})$
13 $V_S = V_S \cup \{z\}$
14 $E_S = E_S \cup \{\{p, q\}\}$
15 Construa o conjunto de vértices folhas V_F da AGG (V_S, E_S) de G
16 Contrua E_F, i.e., o conjunto de arestas de E com ambas as extremidades em V_F
17 $T =$ PRIM$((V_F, E_F))$ // T é a árvore geradora mínima do subgrafo (V_F, E_F) de G
18 $E_S = E_S \cup E(T)$ // S é complementado pelas arestas que formam T
19 $S = (V_S, E_S)$
20 **return** S

Figura 5.1. Pseudo-código da heurística construtiva *2-tree* para o PSBMG

Explicamos agora brevemente o pseudo-código da heurística 2T apresentado acima. Nas linhas 1-7 inicializamos o número de vértices n_S da solução em construção

S, seu conjunto de arestas E_S, de vértices V_S e o conjunto X de vértices excluídos. Os conjuntos E_S e V_S recebem, respectivamente, a aresta $\{u, v\}$ de menor custo em G (empates são resolvidos arbitrariamente) e seus vértices extremos u e v. O conjunto X é usado para evitar que a heurística 2T encontre soluções contendo mais de um vértice por *cluster*. No laço das linhas 8-14, construímos uma AGG de G via uma modificação imediata do algoritmo de Prim. A cada iteração desse laço, uma nova aresta é adicionada de forma gulosa à AGG em questão. Nas linhas 15 e 16, construímos o subgrafo (V_F, E_F) induzido pelas folhas da AGG obtida no laço das linhas 8-14. Por fim, usamos o algoritmo de Prim tradicional na linha 16 para encontrar uma árvore geradora mínima T de (V_F, E_F) e complementamos a solução em construção S com o conjunto de arestas $E(T)$ de T. A complexidade assintótica de tempo no pior caso da nossa implementação da heurística *2-tree* é $O(n^2)$. Para isso, simplesmente usamos listas de adjacência para representação de grafos e fizemos buscas lineares nos passos 2 e 9. O mesmo se aplica ao algoritmo de Prim chamado na linha 17, que executa em $O(k^2)$ no pior caso.

A seguir, provamos que a heurística 2T sempre encontra uma solução viável para o PSBMG quando o grafo de entrada G é completo, a menos de arestas *intra-clusters*, i.e., arestas cujas extremidades sejam vértices pertencentes a um mesmo *cluster*. Para isso, claramente, é suficiente provar que a solução S encontrada pela heurística 2T é um subgrafo biconexo contendo exatamente um vértice de cada *cluster* de G.

Proposiçãrafo 4. *Dada uma instância para o PSBMG em que o grafo de entrada G é completo, a solução S encontrada pela heurística 2-tree é viável para G.*

Prova. A princípio, certamente podemos afirmar que para todo cluster C de G, $|C \cap V_S| = 1$, i.e., S não contém mais de um vértice por *cluster*. O que nos garante isso é precisamente o conjunto de vértices excluídos X que é atualizado em cada iteração do laço 8-14. Resta provar, portanto, que S é biconexo. Para isso, recorremos à definição de grafo biconexo e mostramos que para quaisquer vértices distintos $u, v \in V_S$, existem dois caminhos vértice-disjuntos ligando u a v em S. Vamos denotar por $\mathcal{T}_S$ a AGG de G construída no laço das linhas 8-14 enraizada em um vértice arbitrário r de S com $d_{\mathcal{T}_S}(r) > 1$. Observamos que r precisa existir, pois toda árvore com $k > 2$ vértices tem, no mínimo, um vértice que não é folha. Dividimos a prova em dois casos.

Caso 1. Suponhamos que dados quaisquer vértices distintos u e v em $\mathcal{T}_S$, as subárvores de $\mathcal{T}_S$ enraizadas em u e em v sejam vértice-disjuntas. Seja b o primeiro ancestral comum de u e v em $\mathcal{T}_S$, i.e., o ancestral comum de u e v mais distante de r. Sendo assim, certamente, existe um caminho entre u e v em $\mathcal{T}_S$, portanto em

S, passando por b. Mostramos agora que existe um segundo caminho em S, vértice-disjunto com o anterior, que liga u a v. Tal caminho é formado por três sub-caminhos, a saber o caminho que liga u a qualquer vértice folha f da sub-árvore de $\mathcal{T}_S$ enraizada em u, o caminho que liga v a qualquer vértice folha l da sub-árvore de $\mathcal{T}_S$ enraizada em v e, por fim, o caminho que liga f a l em T, portanto em S, onde T é a árvore geradora mínima encontrada pelo algoritmo de Prim na linha 17 do pseudo-código da heurística 2T.

Caso 2. Suponhamos que dados quaisquer vértices distintos u e v em $\mathcal{T}_S$, as sub-árvores de $\mathcal{T}_S$ enraizadas em u e em v não sejam vértice-disjuntas. Sendo assim, certamente, existe um caminho entre u e v em $\mathcal{T}_S$, portanto em S. Mostramos que existe um segundo caminho em S, vértice-disjunto ao anterior, que liga u a v da seguinte maneira. Se as sub-árvores de $\mathcal{T}_S$ enraizadas por u e v não são vértice-disjuntas, então ou u é descendente de v em $\mathcal{T}_S$ ou o contrário. Sem perda de generalidade, suponhamos que u seja descendente de v em $\mathcal{T}_S$. Formamos o segundo caminho entre u e v em S pela união de quatro sub-caminhos, a saber o caminho que liga u a qualquer vértice folha f da sub-árvore de $\mathcal{T}_S$ enraizada em u, o caminho que liga v a r em $\mathcal{T}_S$, o caminho que liga r a qualquer vértice folha l que não esteja na sub-árvore de $\mathcal{T}_S$ enraizada em v (que necessariamente existe dada a hipótese do caso 2 e o fato de que $d_{\mathcal{T}_S}(r) > 1$) e o caminho que liga f a l em T, portanto em S, onde T é a árvore geradora mínima encontrada pelo algoritmo de Prim na linha 17 do pseudo-código da heurística 2T.

□

Um último esclarecimento a ser feito em relação à heurística 2T é sobre como ela procede quando o grafo de entrada G da instância para o PSBMG não é completo. Nesse caso, não há a garantia de que a solução S encontrada seja viável. Sendo assim, quando G não é completo, para todo par de vértices distintos $u, v \in V$, se $\{u, v\} \notin E$, adicionamos a G uma aresta artificial $\{u, v\}$ com peso $w^G_{uv} = \infty$. Se ao final da sua execução, a solução S possuir alguma aresta artificial, a heurística 2T retorna o grafo nulo $(\emptyset, \emptyset)$, sinalizando que foi incapaz de encontrar uma solução viável para G.

5.1.2 Heurística *cover-and-stitch*

Nesta subseção, discutimos uma segunda heurística construtiva para o PSBMG, chamada de *cover-and-stitch* (CS). Tal heurística obtém uma solução viável para uma dada instância G do PSBMG em três fases. Julgamos mais apropriado não incluir um pseudo-código para a heurística CS. Em vez disso, explicamos suas fases a princípio resumidamente e, em seguida, discutimos separadamente as particularidades de cada

uma delas. Para uma melhor compreensão, recomendamos ao leitor que, se preciso, consulte as definições dos termos *base* e *moldura* dadas na Seção 2.4.

Na sua primeira fase, chamada de *emolduramento*, a heurística CS procura obter uma moldura biconexa de custo mínimo M de G, i.e., fixamos uma base B em G, de tal forma que o subgrafo de G induzido por B seja biconexo e tenha o menor custo possível. Na segunda fase, chamada de *cobertura*, resolvemos em M o chamado *Problema do 2-Emparelhamento Perfeito Mínimo* ou *Problema da Cobertura por Ciclos Mínima* (PCCM), i.e., buscamos encontrar um subgrafo de custo mínimo $\widetilde{M}$ de M, tal que, para todo vértice u de M, exatamente duas arestas incidam sobre u. Se $\widetilde{M}$ for um ciclo Hamiltoniano, então a heurística CS finaliza sua execução, dado que $\widetilde{M}$ já é uma solução viável para G. Caso contrário, suponhamos que $\widetilde{M}$ seja uma p-cobertura por ciclos de M, i.e., que $\widetilde{M}$ seja formado por p ciclos vértice-disjuntos, para algum inteiro $p > 1$. Construímos o chamado ***multigrafo de ciclos*** $\mathcal{C} = (V_{\mathcal{C}}, E_{\mathcal{C}})$ de $\widetilde{M}$ da seguinte maneira. Cria-se um vértice em $\mathcal{C}$ para cada ciclo de $\widetilde{M}$ e a toda aresta $\{u, t\}$ de M, sendo u e t vértices pertences a ciclos diferentes C_u e C_t de $\widetilde{M}$, é associada uma aresta $\{v_{C_u}, v_{C_t}\}$ em $\mathcal{C}$, onde v_{C_u} e v_{C_t} são, respectivamente, os vértices de $\mathcal{C}$ relativos a C_u e C_t. Mais ainda, para qualquer aresta de $\mathcal{C}$, seu peso em $\mathcal{C}$ é igual ao seu peso em G. Por fim, a heurística CS começa sua terceira e última fase, denominada de *costura*, na qual os ciclos de $\widetilde{M}$ são unidos (ou "costurados") pelas as arestas que formam um ciclo Hamiltoniano passando pelos vértices de $\mathcal{C}$.

Para escolher uma moldura biconexa de custo mínimo M de G, a heurística CS inicia resolvendo um problema de PI definido da seguinte maneira. Sejam $x \in \mathbb{B}^n$ e $y \in \mathbb{B}^m$ variáveis de decisão associadas às arestas e aos vértices de G, conforme definidas na Seção 3.1. Consideremos $P_{\text{emol}} \subset \mathbb{R}^{n+m}$ o poliedro definido pelas desigualdades

$$y(V_i) = 1 \quad \text{para todo } i \in \{1, \ldots, k\}, \tag{5.1}$$

$$x(\delta(\{u\})) = y_u d(u) \quad \text{para todo } u \in V, \tag{5.2}$$

$$x(\delta(W)) \geq 2 \quad \text{para todo } W \subset Q, W \neq \emptyset, \tag{5.3}$$

$$x(\delta(W) \setminus \delta(V_i)) \geq 1 \quad \text{para todo } i \in \{1, \ldots, k\}, W \subset Q \setminus \{V_i\}, W \neq \emptyset, \tag{5.4}$$

$$0 \leq y_u \leq 1 \quad \text{para todo } u \in V, \tag{5.5}$$

$$0 \leq x_{ij} \leq 1 \quad \text{para todo } \{i, j\} \in E. \tag{5.6}$$

O problema de PI da fase de emolduramento resolvido pela heurística CS pode ser

expresso por

$$\min \left\{ \sum_{\{i,j\} \in E} w_{ij}^G x_{ij} : (x, y) \in P_{\text{emol}} \cap (\mathbb{B}^n \times \mathbb{B}^m) \right\}. \tag{5.7}$$

Para instâncias completas do PSBMG, usamos apenas as restrições (5.1), (5.2), (5.5) e (5.6) para a definição de P_{emol}. Na nossa implementação da heurística CS, o *solver* CPLEX foi empregado para resolver o problema de emolduramento.

Sejam (x^*, y^*) a solução ótima para (5.7) e $M = (V_M, E_M)$ a moldura biconexa de custo mínimo induzida por (x^*, y^*), i.e., M é tal que $V_M = \{v \in V : y_v^* = 1\}$ e $E_M = \{\{i, j\} \in E : x_{ij}^* = 1\}$. Na fase de cobertura, a heurística CS resolve o PCCM em M, i.e., procuramos encontrar uma cobertura por ciclos de custo mínimo de M. Sejam $z \in \mathbb{B}^{|E_M|}$ variáveis de decisão associadas às arestas de M. Se S é uma solução viável para o PCCM em M, para toda aresta $\{i, j\} \in E_M$, definimos

$$z_{ij} = \begin{cases} 1 & \text{se } \{i, j\} \in E_M \\ 0 & \text{caso contrário.} \end{cases} \tag{5.8}$$

Seja $P_{\text{cob}} \subset \mathbb{R}^{|E_M|}$ o poliedro definido pelas desigualdades

$$z(\delta(\{u\})) = 2 \quad \text{para todo } u \in V_M, \tag{5.9}$$

$$z(\delta(W) \setminus F) - z(F) \geq 1 - |F| \quad \text{para todo } W \subset V_M, F \subseteq \delta(W), |F| \text{ é ímpar}, \tag{5.10}$$

$$0 \leq z_{ij} \leq 1 \quad \text{para todo } \{i, j\} \in E. \tag{5.11}$$

O PCCM em M é dado por

$$\min \left\{ \sum_{\{i,j\} \in E_M} w_{ij}^G z_{ij} : z \in P_{\text{cob}} \cap \mathbb{B}^{|E_M|} \right\}. \tag{5.12}$$

A envoltória convexa dos pontos pertencentes ao conjunto $P_{\text{cob}} \cap \mathbb{B}^{|E_M|}$, denotada por $conv(P_{\text{cob}} \cap \mathbb{B}^{|E_M|})$, é chamada de *Politopo dos 2-Emparelhamentos Perfeitos* de M. As restrições (5.10) são as conhecidas desigualdades *Blossom*. Edmonds [1965a] e Pulleyblank [1973] demonstraram que os vértices do poliedro P_{cob} possuem somente coordenadas 0 ou 1, i.e., que as restrições que definem P_{cob} são suficientes para induzir $conv(P_{\text{cob}} \cap \mathbb{B}^{|E_M|})$. Sendo assim, para resolver (5.12), i.e., para solucionar a fase de cobertura da heurística CS, implementamos um algoritmo de planos de corte que separa de forma exata as desigualdades (5.10) em tempo polinomial usando o algoritmo de Letchford et al. [2004] (LRT). Embora Edmonds [1965a] tenha descrito um algoritmo

exato puramente combinatório para o PCCM, i.e., um algoritmo que não faz chamadas a nenhum tipo de método numérico para solução de programas lineares e que explora somente a natureza combinatória do problema, é um consenso na literatura, conforme apontado por Grötschel & Holland [1987], que uma implementação eficiente de tal algoritmo não é trivial. Por essa razão, escolhemos implementar um algoritmo de planos de corte que usa o LRT para separar desigualdades *Blossom*.

Seja $\widehat{P} \subset P_{\text{cob}}$ o poliedro definido apenas pelas desigualdades (5.9) e (5.11). O algoritmo de planos de corte que soluciona o problema de cobertura da heurística CS inicia calculando a solução ótima do seguinte problema de PL

$$\min \left\{ \sum_{\{i,j\} \in E_M} z_{ij} w_{ij}^G : z \in \widehat{P} \right\}. \tag{5.13}$$

Em seguida, iterativamente separamos desigualdades *Blossom* (5.10) até que nenhuma delas esteja violada. Para isso, como dito anteriormente, usamos o algoritmo LRT. Em suma, dada a solução $\overline{z}$ obtida em uma determinada iteração do algoritmo de planos de corte para resolver (5.13), LRT computa uma árvore de Gomory-Hu T do grafo suporte associado a $\overline{z}$ e verifica se é possível encontrar, para todos os cortes em M induzidos pelas arestas de T, conjuntos $W \subset V_M$ e $F \subseteq \delta(W)$, com $|F|$ ímpar, que definam uma *Blossom* violada em M, onde W é um subconjunto de qualquer uma das costas de um corte em T. Ressaltamos que todos os cortes encontrados ao longo da execução do algoritmo são inseridos no modelo. A complexidade assintótica de tempo no pior caso da nossa implementação do algoritmo LRT é $O(k^4)$, que é, por sua vez, essencialmente dominada pelo algoritmo de GU usado internamente para calcular T.

Uma vez terminada a fase de cobertura, obtemos uma cobertura por ciclos mínima $\widetilde{M} = (V_M, E_{\widetilde{M}})$ de M. Se $\widetilde{M}$ for um ciclo Hamiltoniano nos vértices de M, então chegamos ao final da execução da heurística CS, visto que $\widetilde{M}$ já é considerada viável para o PSBMG. No entanto, se $\widetilde{M}$ for uma p-cobertura por ciclos de M, para algum inteiro $p > 1$, entramos na última fase da heurística CS, a saber a fase de costura. Nela, computamos primeiramente o multigrafo de ciclos $\mathcal{C}$ associado a $\widetilde{M}$, como definido anteriormente. Uma ressalva importante a ser feita neste momento é que durante a construção de $\mathcal{C}$, criamos um mapeamento $\phi : E_{\mathcal{C}} \to E_{\widetilde{M}}$ para facilitar, dada uma aresta $\{q, w\} \in E_{\mathcal{C}}$, a "recuperação" a aresta $\{i, j\} \in E_M$ que originou $\{q, w\}$ em $\mathcal{C}$, i.e., do par de vértices $i, j \in V_M$, chamados a seguir de ***vértices de contato***.

Na fase de costura da heurística CS, utilizamos o algoritmo $\frac{3}{2}$-aproximativo de Christofides [1976] (CF) para encontrar um ciclo Hamiltoniano em $\mathcal{C}$. Na nossa implementação desse algoritmo, usamos uma implementação $O(k^2)$ do algoritmo de Prim

com o objetivo de obter uma árvore geradora mínima T de $\mathcal{C}$ e, em seguida, executamos o algoritmo de Micali & Vazirani [1980] em $O(k^3)$, no pior caso, para computar um emparelhamento perfeito de custo mínimo do subgrafo de $\mathcal{C}$ induzido pelos vértices de grau ímpar em T. Dessa forma, a complexidade assintótica de tempo no pior caso da nossa implementação do algoritmo CF é $O(k^3)$.

Seja $(V_{\mathcal{C}}, E^*)$ a solução obtida na fase de costura da heurística CS via o algoritmo CF. Para unirmos (ou "costurarmos") os ciclos que compõem $\widetilde{M}$, simplesmente unimos as arestas no conjunto $E_\phi = \{\phi(e) \in E_M : e \in E^*\}$ àquelas pertencentes a $E_{\widetilde{M}}$. Na sequência, provamos que a solução $S = (V_M, E_S)$ encontrada pela heurística CS é viável para G, se G for uma instância completa e se S não possuir ***vértices críticos***, i.e., vértices $i \in V_M$, com $|\delta_S(\{i\}) \cap E_\phi| = 2$. (O motivo dessa restrição consiste na observação de que todo vértice crítico é também um ponto de articulação.)

Proposiçãrafo 5. *Dada uma instância G completa para o PSBMG, a solução S encontrada pela heurística CS é viável para G se S não possuir vértices críticos.*

Prova. Suponhamos que G seja uma instância completa e que S não tenha vértices críticos. Mostramos que S é uma solução viável para o PSBMG em G provando que S é um subgrafo biconexo de G contendo exatamente um vértice de cada *cluster*. Certamente, podemos afirmar que S contém somente um vértice de cada *cluster* de G, pois S é um subgrafo de uma moldura de G obtida na fase de emolduramento da heurística CS. Resta provar, portanto, que S é biconexo. Para isso, novamente recorremos à definição de grafo biconexo e mostramos que, para quaisquer dois vértices distintos $u, t \in V_S$, é possível encontrar dois caminhos vértice-disjuntos em S conectando u a t. Sejam $\widetilde{M}$ a solução obtida na fase de cobertura da heurística CS e C_u e C_t, respectivamente, os ciclos de $\widetilde{M}$ que contêm u e v. Além disso, sejam também v_{C_u} e v_{C_t} os vértices no multigrafo de ciclos $\mathcal{C}$ relativos aos ciclos C_u e C_t. Dividimos a prova em dois casos.

Caso 1. Suponhamos que C_u e C_t coincidam, i.e., que os vértices u e t estejam em um mesmo ciclo de $\widetilde{M}$. Dessa forma, certamente existem dois caminhos vértice-disjuntos entre u e t, uma vez que todo ciclo é um grafo biconexo.

Caso 2. Suponhamos que C_u e C_t não coincidam, i.e., que os vértices u e t estejam em ciclos vértice-disjuntos de $\widetilde{M}$. Vimos que, na fase de cobertura, obtemos um ciclo Hamiltoniano passando pelos vértices de $\mathcal{C}$. Novamente, como todo ciclo é um grafo biconexo, é imediato constatar que necessariamente precisam existir pelo menos dois caminhos vértice-disjuntos ligando os vértices v_{C_u} e v_{C_t} em $\mathcal{C}$. Logo, como por hipótese

S não possui vértices críticos, também deve ser possível partir de u e chegar a t por dois caminhos vértice-disjuntos em S.

□

Uma observação final a ser dada em relação à heurística CS diz respeito à maneira como ela trata vértices críticos que podem aparecer nas soluções por ela encontradas. Se existir algum vértice crítico em S, a heurística CS simplesmente retira de $\mathcal{C}$ as arestas usadas na fase de costura que levaram à construção de S e, em seguida, executa a fase de costura novamente. Portanto, novas arestas são usadas para costurar os ciclos de $\widetilde{M}$. Enquanto existirem vértices críticos, repetimos esse procedimento. O número de repetições é de, no máximo, três. Se após três tentativas fracassadas, a heurística CS não conseguir obter uma solução sem vértices críticos, ela retorna o grafo nulo $(\emptyset, \emptyset)$. Por fim, se a instância G para o PSBMG não for completa, procedemos da seguinte maneira. Resolvemos a fase de emolduramento normalmente e inserimos arestas artificiais de custo infinito na moldura biconexa mínima M encontrada, analogamente como descrito na Subseção 4.1.1 para a heurística *2-tree*. Se existirem tais arestas artificiais na solução encontrada ao final da fase de cobertura, a heurística CS aborta sua execução e retorna o grafo nulo $(\emptyset, \emptyset)$.

5.2 Heurística de Refinamento *destroy-and-restore*

Uma ***heurística de refinamento*** para um dado problema de Otimização Combinatória é um algoritmo eficiente que recebe como entrada uma solução viável para uma instância desse problema e busca aprimorá-la. Nesta subseção, propomos uma heurística de refinamento para o PSBMG, chamada de *destroy-and-restore* (DR). Discutimos a seguir como tal heurística tenta melhorar uma determinada solução viável do problema e comentamos também sobre seus detalhes de implementação. Similar ao que fizemos na subseção anterior, para fins de clareza, preferimos não incluir o pseudo-código da heurística DR nesta subseção, mas apenas explicá-la em linguagem corrente.

A heurística DR, em alto nível, decompõe o refinamento de uma dada solução viável para o PSBMG em dois sub-problemas ou fases, a saber um sub-problema de *destruição* e outro de *restauração*. Seja $S = (V_S, E_S)$ uma solução viável para uma instância G do PSBMG e $M = (V_S, E_M)$ a moldura em G que contém S, i.e., o subgrafo de G induzido por V_S. Na fase de destruição, como o nome já sugere, nosso objetivo é remover arestas do conjunto E_S e, assim, tornar S inviável. Em seguida, na fase de restauração, tentamos recuperar a viabilidade de S com o mínimo esforço

possível, i.e., remontando seus "pedaços", criados na fase anterior, com um conjunto de novas arestas presentes em M de baixo custo agregado. Na sequência, tratamos individualmente de cada uma das duas fases que compõem a heurística DR.

Na fase de destruição, procuramos encontrar um subconjunto $W \subset V_S$, tal que o corte $\delta_S(W)$ seja suficiente para desconectar S. Mais do que isso, o desejável seria que conseguíssimos determinar W de maneira que $\delta_S(W)$ tivesse o máximo custo possível, i.e., que a soma dos pesos das arestas em $\delta_S(W)$ fosse máxima. A motivação para procedermos assim decorre do fato de que se $\delta_S(W)$ realmente fosse um corte máximo separando S em duas ou mais componentes conexas, então dificilmente conseguiríamos deteriorar a qualidade de S na fase de restauração, uma vez que qualquer que seja o conjunto E_{res} de arestas escolhido para viabilizar S, o custo total de E_{res} estaria limitado superiormente pelo custo de $\delta_S(W)$. Sendo assim, na fase de destruição da heurística DR, buscamos resolver em S o conhecido *Problema do Corte Máximo* (PCM). Por se tratar de um problema que é NP-difícil na sua versão de otimização [Karp, 1972], recorremos a uma estratégia de solução aproximada. Para isso, empregamos o algoritmo $0,878$-aproximativo probabilístico de Goemans & Williamson [1995b] (GW).

Em essência, o algoritmo GW obtém uma solução aproximada para uma instância qualquer do PCM explorando a solução de uma relaxação de Programação Não-Linear (mais especificamente, de Programação Semi-definida) do PCM. Consideremos a seguinte formulação de PI Quadrática para o PCM. Seja $H = (V_H, E_H)$, onde $n_H = |V_H|$, um grafo com pesos reais não-negativos nas arestas. Definimos variáveis de decisão $y \in \{1, -1\}^{n_H}$ associadas aos vértices de H. Tomemos H como uma instância para o PCM e $W \subset V_H$ uma solução viável para o PCM em H, i.e., o corte $\delta_H(W)$ que W induz em H. Para todo vértice $i \in V_H$, definimos

$$y_i = \begin{cases} 1 & \text{se } i \in W, \\ -1 & \text{caso contrário.} \end{cases} \tag{5.14}$$

Dessa forma, o PCM pode ser expresso por

$$\max \left\{ \frac{1}{2} \sum_{\{i,j\} \in E_H} w_{ij}^H (1 - y_i y_j) : y \in \{-1, 1\}^{n_H} \right\}. \tag{5.15}$$

Na sequência, definimos a seguinte relaxação para (5.15). Para todo $i \in \{1, \ldots, n_H\}$, associamos um vetor $v_i \in [-1, 1]^{n_H}$ à variável de decisão y_i, i.e, associamos a y_i um ponto na esfera n_H-dimensional de raio unitário centrada na origem, denotada a seguir

por $\mathcal{S}_{n_H}$. Uma relaxação de Programação Semi-definida para o PCM é dada por

$$\max \left\{ \frac{1}{2} \sum_{\{i,j\} \in E_H} w_{ij}^H (1 - \upsilon_i \cdot \upsilon_j) : \upsilon_i \in \mathcal{S}_{n_H} \text{ para todo } i \in \{1, \ldots, n_H\} \right\}, \tag{5.16}$$

onde $\upsilon_i \cdot \upsilon_j$, para quaisquer $i, j \in \{1, \ldots, n_H\}$ distintos, denota o produto interno entre os vetores υ_i e υ_j.

Sejam $\upsilon_1^*, \ldots, \upsilon_{n_H}^*$ vetores ótimos para (5.16), r um ponto em $\mathcal{S}_{n_H}$ escolhido aleatoriamente segundo uma distribuição de probabilidade contínua uniforme definida sobre $\mathcal{S}_{n_H}$ e $W^* = \{i \in V_H : \upsilon_i^* \cdot r \geq 0\}$. Goemans & Williamson [1995b] demonstraram que

$$\mathbb{E}[W^*] = \sum_{\{i,j\} \in E_H} w_{ij}^H \frac{\arccos(\upsilon_i, \upsilon_j)}{\pi}, \tag{5.17}$$

onde $\arccos(\upsilon_i, \upsilon_j)$, para quaisquer $i, j \in \{1, \ldots, n_H\}$ distintos, denota o arco do coseno do ângulo formado entre os vetores υ_i e υ_j e $\mathbb{E}[W^*]$ denota o custo esperado do corte $\delta_H(W^*)$ induzido por W^* em H. Mais do que isso, os autores provaram que

$$\mathbb{E}[W^*] \geq \alpha \left[\frac{1}{2} \sum_{\{i,j\} \in E_H} w_{ij}^H (1 - \upsilon_i \cdot \upsilon_j) \right], \tag{5.18}$$

onde $\alpha = \min \left\{ \frac{2\beta}{\pi(1-\cos\beta)} : 0 \leq \beta \leq \pi \right\} \approx 0,878$, o que implica um algoritmo de aproximação probabilístico para o PCM em H com garantia de desempenho $0,878$. Intuitivamente, de posse da solução ótima $\upsilon_1^*, \ldots, \upsilon_{n_H}^*$ para (5.16) e de $r \in \mathcal{S}_{n_H}$, o algoritmo GW escolhe o corte em H induzido pelos vértices cujos vetores ótimos associados estejam situados "acima" do hiperplano aleatório com normal r que passa pela origem do espaço real n_H-dimensional.

Na nossa implementação do algoritmo GW, utilizamos o *solver* de Programação Semi-definida SDPA [2012] para resolver (5.16). Para gerar $r \in \mathcal{S}_{n_H}$, simplesmente sorteamos independentemente n_H números aleatórios usando a distribuição de probabilidade contínua normal padrão e, em seguida, normalizamos o vetor assim obtido. Para isso, usamos o gerador de números pseudo-aleatórios da GNU *Scientific Library* [Galassi, 2009].

Após finalizada a fase de destruição, obtemos com o algoritmo GW um corte $\delta_S(W^*)$ em S induzido pelo subconjunto $W^* \subset V_S$. Seja $\overline{W}^* = V_S \setminus W^*$. Para viabilizar S novamente, iniciamos a fase de restauração da heurística DR procedendo da seguinte maneira. Com o algoritmo de Prim, calculamos as árvores geradoras mínimas

$T_{W^*} = (V_{W^*}, E_{W^*})$ e $T_{\overline{W}^*} = (V_{\overline{W}^*}, E_{\overline{W}^*})$ dos subgrafos de M induzidos, respectivamente, por W^* e $\overline{W}^*$. Unimos T_{W^*} a $T_{\overline{W}^*}$ tomando a aresta $\widehat{e}$ de menor peso em $\delta_M(W^*)$ e, assim, construímos a árvore $Z = (V_S, E_Z)$, onde $E_Z = E_{W^*} \cup E_{\overline{W}^*} \cup \{\widehat{e}\}$. Na sequência, utilizamos o algoritmo $\frac{3}{8}$-aproximativo de Khuller & Thurimella [1992] (KT) para resolvermos em M o chamado *Problema da Biconexão Mínima* (PBM), que consiste em encontrar o subconjunto de arestas $E' \subset E_M \setminus E_Z$ de mínimo custo total, tal que o grafo $(V_S, E_Z \cup E')$ seja biconexo.

Por brevidade, julgamos melhor não comentar em detalhes o algoritmo KT nesta subseção. Para discutirmos em resumo como ele resolve de forma aproximada o PBM, necessitamos de alguns conceitos. Dado um digrafo $D = (V_D, A_D)$ com pesos reais não-negativos associados aos arcos e um vértice $b \in V_D$, chamamos de ***arborescência*** de D enraizada em b qualquer subdigrafo de D no qual exista precisamente um caminho direcionado conectando b a qualquer vértice $u \in V_D \setminus \{b\}$. Uma ***arborescência de busca em profundidade*** (ABP) de D enraizada em b é qualquer arborescência criada pela execução de uma busca em profundidade (*depth-first search*) em D partindo de b. Uma ***ramificação ótima*** (*optimum branching*) em D é uma arborescência de custo mínimo de D, i.e., uma arborescência de D cuja soma dos pesos dos seus arcos é mínima.

Em suma, o algoritmo KT transforma a árvore Z, criada no começo da fase de restauração, em um grafo biconexo encontrando uma ramificação ótima $\mathcal{R}$ de uma ABP apropriadamente criada a partir de Z e M. Por fim, usam-se os arcos de $\mathcal{R}$ para induzir um subconjunto de arestas $E' \subset E_M \setminus E_Z$ de baixo custo agregado, tal que o grafo $(V_S, E_Z \cup E')$ seja biconexo. Para computar $\mathcal{R}$, usamos uma implementação $O(k^2)$ de Tarjan [1977] do algoritmo proposto por Edmonds [1967].

5.3 Procedimentos de Busca Local

Buscas locais (BL) [Hoos & Stutzle, 2005; Aarts & Lenstra, 2003] integram uma coleção de técnicas heurísticas empregadas na resolução de problemas computacionais desafiadores, muitos deles enquadrados no campo da Otimização Combinatória. Sejam uma instância I de um desses problemas e S uma solução viável para I. Dizemos que uma ***estrutura de vizinhança*** para tal problema, definida em relação a S, é um mapeamento que associa a S um subconjunto do espaço de viabilidade de I formado por soluções que diferem de S por algum determinado critério. Sendo assim, uma ***busca local*** é uma estratégia que prescreve como movimentar dentro de uma dada estrutura de vizinhança centrada em S visando a otimizar localmente uma função objetivo.

Nesta seção, introduzimos procedimentos de busca local baseados em duas estruturas de vizinhança para o PSBMG, a saber *cluster-head exchange* e *edge swap*. Fazemos uma breve discussão também dos seus detalhes de implementação. Nos experimentos computacionais realizados, combinamos essas buscas locais com o objetivo de aprimorar as soluções geradas pelas heurísticas construtivas abordadas nas Subseções 5.1.1 e 5.1.2.

5.3.1 Vizinhança *cluster-head exchange*

Para uma melhor compreensão desta subseção, recomendamos ao leitor, se preciso, que recorde o conceito de *backbone* descrito na Seção 3.3.

Dada uma instância G do PSBMG e uma solução viável $S = (V_S, E_S)$ para G, a vizinhança ***cluster-head exchange*** (CHE) de S é o subconjunto do espaço de viabilidade de G formado pelas soluções que possuem o mesmo *backbone* de S, mas que diferem de S em exatamente um *cluster-head*. O principal objetivo da vizinhança CHE para o PSBMG é, uma vez fixo o *backbone* de S, procurar otimizar a escolha dos vértices eleitos de cada *cluster* de G.

Formalmente, sendo $\mathcal{V}(G)$ a espaço de viabilidade do PSBMG para G, i.e., o conjunto de todas as soluções viáveis para a instância G, a vizinhança *cluster-head exchange* de S, denotada por $\text{CHE}_G(S)$, é definida como $\text{CHE}_G(S) = \{\overline{S} = (V_{\overline{S}}, E_{\overline{S}}) \in \mathcal{V}(G) : \mathcal{B}_G(\overline{S}) = \mathcal{B}_G(S), |V_{\overline{S}} \setminus V_S| = 1\}$. Para a vizinhança CHE, implementamos a chamada ***busca local melhor aprimorante***, na qual, dada a solução viável de partida S e a vizinhança CHE de S, movemo-nos para a solução aprimorante $S^* \in \mathcal{V}(G)$ de menor custo dentre todas as soluções em $\text{CHE}_G(S)$, i.e., movemos de S para S^*, tal que $c(S^*) = \min\{c(\overline{S}) : \overline{S} \in \text{CHE}_G(S)\}$ e $c(S^*) < c(S)$. Se não existir tal solução aprimorante, dizemos que chegamos a um ***ótimo local*** da vizinhança CHE para a solução de partida S.

O tamanho da vizinhança CHE de S é $\Theta(n-k)$. Usando uma tabela *hash* simples e sem colisões, conseguimos avaliar em tempo constante o custo de uma solução na vizinhança CHE de S. Logo, a complexidade assintótica de tempo de um movimento da busca local melhor aprimorante na vizinhança CHE de S é $O(n-k)$ no pior caso.

5.3.2 Vizinhança *edge swap*

Dadas uma instância G para o PSBMG, uma solução viável $S = (V_S, E_S)$ para G e a moldura $M = (V_S, E_M)$ que contém S, a vizinhança ***edge-swap*** (ES) de S, denotada por $\text{ES}_G(S)$, é o subconjunto do espaço de viabilidade de G formado pelas

soluções que podem ser obtidas adicionando a S duas arestas $p, q \in E_M \setminus E_S$ e retirando de S, em ordem não crescente de peso, a quantidade máxima possível das arestas em E_S que se tornaram redundantes devido à inserção de p e q. Em suma, a vizinhança *edge swap* para o PSBMG procura complementar a CHE. Ao passo que na vizinhança CHE mantemos o *backbone* de S e variamos os *cluster-heads*, na vizinhança ES mantemos os *cluster-heads* e variamos o *backbone*, i.e., buscamos otimizar a maneira com que os *cluster-heads* estão interligados. Na nossa implementação da vizinhança ES, utilizamos a chamada ***busca local primeira aprimorante***. Dada a solução viável de partida S, movemo-nos para a primeira solução aprimorante $S^* \in \mathcal{V}(S)$ na vizinhança ES de S. Na sequência, formalizamos o significado de "primeira".

Antes de avaliar qualquer solução em $\mathrm{ES}_G(S)$, armazenamos em um *min-heap de Fibonacci* $\mathcal{H}$ [Cormen et al., 1990], com um custo de $O(k^2)$ no pior caso, todos os pares não ordenados de arestas em $E_M \setminus E_S$. Utilizamos o soma dos pesos dos pares das arestas como critério de ordenação dos nós de $\mathcal{H}$. Em seguida, procedemos da seguinte maneira. Retiramos em $O(1)$ o par de arestas $\{p, q\}$ correspondente à raiz de $\mathcal{H}$. Inserimos tais arestas em S e, com um algoritmo $O(k^2)$ proposto por Hopcroft & Tarjan [1973], removemos de $E_S \setminus \{p, q\}$, em ordem não crescente de peso, o subconjunto de arestas $R \in E_S \setminus \{p, q\}$ que se tornaram redundantes devido à adição de p e q a S. Claramente, tal movimento será aprimorante somente se $w_p + w_q < \sum_{r \in R} w_r$. Se essa condição for satisfeita, executamos tal movimento, i.e., inserimos em $\mathcal{H}$, com um custo $O(k^2)$ no pior caso, todos os pares não ordenados de arestas pertencentes a R e repetimos o processo até atingir um ótimo local de ES.

O tamanho da vizinhança ES de S é $O(k^2)$. O custo para avaliar cada solução pertencente a $\mathrm{ES}_G(S)$ é $O(k^2)$ no pior caso. Logo, a complexidade assintótica de tempo, no pior caso, de um movimento da busca local primeira aprimorante na vizinhança ES de S é $O(k^4)$.

5.4 Experimentos Computacionais

Nesta seção, discutimos os resultados alcançados pelas heurísticas e buscas locais apresentadas neste trabalho. Observamos que os métodos heurísticos só foram usados em conjunção com o algoritmo BC para instâncias com pelo menos 76 vértices e 15 *clusters*, pois o efeito da aplicação dos mesmos não foi discernível em instâncias com um número inferior de vértices ou de *clusters*. Similarmente ao que fizemos no Capítulo 4, para uma discussão mais clara dos resultados, dividimos nossos testes em categorias. Na sequência, comentamos sobre cada uma delas separadamente.

Para a primeira categoria de testes, realizamos um estudo empírico comparativo dos desempenhos isolados das heurísticas construtivas 2T e CS. Em seguida, procuramos mensurar as melhorias proporcionadas pela heurística de refinamento e pelas buscas locais nos custos das soluções obtidas pelas heurísticas 2T e CS. Finalmente, estabelecemos uma comparação entre as soluções heurísticas fornecidas ao algoritmo BC e a primeira solução inteira encontrada na árvore de enumeração do BC com a formulação $\mathcal{P}_c^+$ e quando executado sem nenhuma solução inicial. Neste último cenário, é relevante registrar que o nosso algoritmo BC, em todo instante, empregou uma estratégia de caminhamento em largura pela árvore de enumeração, que, normalmente, costuma resultar em primeiras soluções inteiras de melhor qualidade. Na Tabela 5.1, explicitamos os custos das soluções encontradas pelos métodos heurísticos nos três cenários mencionados anteriormente e também o custo da primeira solução inteira do BC. Na Tabela 5.2, encontram-se os respectivos tempos de execução despendidos pelos métodos heurísticos e pelo BC para obter tais soluções.

No tocante às heurísticas construtivas, notamos que os custos das soluções encontradas pela CS foram, na média, 15% melhores que os das soluções obtidas pela 2T, embora a CS tenha executado em um tempo quase 9 vezes maior que o da heurística 2T. Salientamos que, para todas as instâncias de teste consideradas, a CS dominou a 2T em relação à qualidade das soluções obtidas. Além disso, notamos que a heurística de refinamento DR aprimorou em aproximadamente 15%, na média, as soluções fornecidas pela 2T e menos de 1% no caso da CS. Em relação às buscas locais combinadas no arcabouço VND, observamos que elas possibilitaram um ganho médio de 16% e 8%, respectivamente, nos custos das soluções fornecidas pela 2T e CS. Por fim, a heurística CS aliada ao VND obteve, com um quarto do tempo do BC, soluções com custos, na média, 8% menores em relação ao custo da primeira solução inteira obtida pelo BC.

Na Tabela 5.3, explicitamos, para cada instância, o tempo total de execução do BC, o tempo de execução demandado pelas intervenções das buscas locais durante a execução do BC e a fração percentual de tais intervenções em relação ao tempo total de execução do BC. Em resumo, podemos notar que a atuação das buscas locais ao longo da execução do BC é moderada para a maioria das instâncias, com exceção de 8 delas para quais tal valor ultrapassa a margem dos 10%. Na Tabela 5.4, para todas as instâncias resolvidas na otimalidade pelo nosso algoritmo BC, listamos os *gaps* das soluções encontradas pela heurística CS acoplada às buscas locais. Observamos que todos os valores de *gaps* mostrados na tabela estão abaixo de 5% e que, inclusive, para uma das instâncias, a saber gr137america, nosso método heurístico conseguiu obter a solução ótima.

Na segunda categoria de testes, conduzimos uma análise entre a qualidade dos resultados obtidos com a heurística CS aliada ao VND e os resultados alcançados pelo algoritmo memético de Hu & Raidl [2009]. Na Tabela 5.5, mostramos os tempos de execução demandandos por cada abordagem de solução e os respectivos custos das soluções por elas obtidas. Mencionamos que em [Hu & Raidl, 2009], os autores impuseram dois critérios de parada para os seus experimentos computacionais, um deles, no qual obtiveram os melhores resultados, baseou-se em um limite de tempo de execução grande o suficiente para garantir a convergência do algoritmo memético. É precisamente os resultados relativos a esse critério de parada que relatamos na Tabela 5.5. Observamos também que o ambiente computacional utilizado por Hu & Raidl [2009] foi um Intel Core 2 Quad, 2.4 GHz com 4 GB de memória RAM, i.e., um ambiente computacional cerca de três vezes inferior em desempenho ao nosso. Ainda levando esse fator em consideração, podemos notar que a heurística CS integrada com o VND conseguiu encontrar, com 40% a menos de tempo de execução, melhores soluções que o algoritmo memético de Hu & Raidl [2009], com exceção de quatro instâncias, a saber gr137, kroa150, gr202 e rd400, sendo que, para esta última, ambas as abordagens de solução encontraram soluções de custo idêntico.

Na terceira e última categoria de testes, procuramos aferir quantas vezes as buscas locais chamadas ao longo da execução do BC efetivamente tiveram algum efeito. Além disso, quisemos determinar a autoria da melhor solução obtida após a execução do BC para as instâncias de teste consideradas, i.e., qual foi o método, o BC ou as buscas locais, que encontrou tal solução. Sendo assim, relatamos na Tabela 5.6 o número total de execuções das buscas locais, o número de execuções positivas, i.e., execuções que aprimoraram a melhor solução inteira corrente, a fração percentual das execuções positivas em relação ao número total de execuções e a autoria da solução. Observamos na Tabela 5.6 que, na média, a aplicação das buscas locais foi positiva aproximadamente metade do número total de vezes em que elas foram chamadas. Para 23 instâncias (ou 54% delas), a autoria da melhor solução encontrada foi das buscas locais; para 6 instâncias (o que representa 14%), a autoria foi devida ao nosso algoritmo BC e, por fim, para as 13 instâncias restantes (32% do total) houve empate entre o BC e as buscas locais.

Tabela 5.1. Custos das soluções encontradas pelas heurísticas construtivas *2-tree* (2T) e *cover-and-stitch* (CS) quando usadas isoladamente, quando combinadas com a heurística de refinamento *destroy-and-restore* (DR) e com as buscas locais (VND). Comparamos também os resultados com o custo da primeira solução inteira encontrada pelo algoritmo BC com a formulação $\mathcal{P}_c^+$ ao longo da árvore de enumeração quando não é fornecida a ele nenhuma solução inicial.

Instâncias	Heur. const.		Heur. refinamento		Buscas locais		Prim. sol.
Nomes	2T	CS	2T+DR	CS+DR	2T+VND	CS+VND	int. BC
eil76	278	247	247	245	246	244	248
pr76	73835	65401	65451	65050	65277	64806	65576
gr96	312	269	269	268	268	266	270
gr96africa	445	390	390	385	388	379	392
rat99	631	547	548	542	544	533	550
kroc100	11979	10402	10411	10314	10372	10168	10456
kroa100	12361	10593	10593	10514	10549	10366	10635
krod100	11595	10117	10118	10058	10077	9940	10148
rd100	4040	3570	3572	3549	3556	3510	3584
kroe100	12241	10457	10464	10377	10433	10310	10485
krob100	11421	10082	10082	9992	10039	9841	10129
eil101	321	281	281	278	280	274	283
lin105	9952	8749	8746	8677	8721	8589	8780
pr107	32135	27763	27765	27749	27755	27736	27770
gr120	3376	2930	2931	2917	2923	2891	2938
pr124	43560	37897	37936	37723	37823	37540	37994
bier127	85918	75605	75674	75254	75415	74521	75836
pr136	50612	43959	43950	43531	43736	43155	44143
gr137	503	446	446	445	446	441	447
gr137america	315	280	280	280	280	280	280
pr144	51850	46250	46264	46189	46228	46100	46279
kroa150	13511	11955	11971	11906	11929	11619	11986
krob150	13143	11465	11474	11432	11456	11380	11490
pr152	59937	53241	53290	53034	53121	52852	53349
u159	27213	24078	24103	23942	24007	23652	24146
rat195	1113	953	954	935	940	913	961
d198	12363	10767	10770	10739	10757	10702	10779
krob200	15850	13625	13647	13538	13590	13250	13667
kroa200	15760	13821	13828	13789	13800	13741	13838
gr202	377	322	322	321	322	321	323
gr202europe	238	206	207	205	206	202	208
ts225	81301	69830	69810	69580	69731	69207	69932
pr226	75500	64688	64701	64611	64653	64509	64738
gil262	1220	1069	1067	1056	1063	1042	1074
pr264	35415	31111	31126	30995	31043	30828	31194
pr299	28535	24516	24583	24311	24460	22752	24668
lin318	24592	21153	21161	21108	21136	21024	21181
rd400	10723	9104	9149	8743	8966	6765	9259
fl417	13070	11305	11336	11092	11208	10034	11392
gr431	5232	4302	4439	3971	4118	1290	4539
pr439	73426	63355	63388	62840	63093	60240	63590
pcb442	27864	24153	24341	23559	24007	22170	24503

Tabela 5.2. Tempos de execução (em segundos) das heurísticas construtivas *2-tree* (2T) e *cover-and-stitch* (CS) para a encontrar a solução inicial quando usadas isoladamente, quando combinadas com a heurística de refinamento *destroy-and-restore* (DR) e com as buscas locais (VND). Comparamos também os resultados com o tempo de execução gasto pelo algoritmo BC com a formulação $\mathcal{P}_c^+$ para encontrar a primeira solução inteira ao longo da árvore de enumeração, usando uma estratégia de caminhamento em largura, quando não é fornecida a ele nenhuma solução inicial.

Instâncias	Heur. const.		Heur. refinamento		Buscas locais		Prim. sol.
Nomes	2T	CS	2T+DR	CS+DR	2T+VND	CS+VND	int. BC
eil76	0.03	0.27	0.28	0.59	0.88	1.01	4
pr76	0.02	0.2	0.2	0.42	0.57	0.72	3.1
gr96	0.02	0.28	0.32	0.61	0.82	0.92	4.2
gr96africa	0.27	1.96	3.79	6.1	8.6	9.6	39.3
rat99	0.07	0.47	0.83	1.38	2	2.35	9.6
kroc100	0.05	0.55	0.7	1.13	1.77	2.07	8.2
kroa100	0.06	0.36	0.54	1.51	1.86	1.87	8.7
krod100	0.06	0.57	0.8	1.44	1.79	2.16	10
rd100	0.06	0.51	0.76	1.47	1.76	1.92	8.7
kroe100	0.03	0.27	0.61	0.98	1.41	1.76	6.8
krob100	0.05	0.54	0.65	1.28	1.57	1.72	8.2
eil101	0.08	0.84	0.82	2.21	2.61	3.14	13.5
lin105	0.05	0.54	0.54	1.43	1.53	2.31	8.9
pr107	0.05	0.45	0.76	1.55	1.78	2.16	8.8
gr120	0.09	0.76	1.1	2.71	3.2	4.5	16.9
pr124	0.13	1.15	1.66	3.44	3.67	4.68	19.2
bier127	0.17	1.76	2.39	3.85	4.88	6.35	26.6
pr136	0.26	3.31	4.49	7.97	11.25	12.43	51.8
gr137	0.24	2.78	3.73	5.94	8.4	9.96	40.9
gr137america	0.26	2.31	2.33	6.63	7.06	8.75	37.5
pr144	0.14	1.28	2.5	3.79	5.82	6.72	26.8
kroa150	0.38	2.95	3.7	8.57	10.06	14.72	58.5
krob150	0.3	2.52	3.23	7.87	9.99	11.7	47.1
pr152	0.32	2.62	4.23	9.47	11.15	15.74	62.5
u159	0.61	6.05	11.57	17.23	22.65	25.06	118.8
rat195	1.84	21.14	22.61	42.56	64.91	77.2	308.6
d198	2.68	24.74	26.03	60.4	75.73	97.14	405.7
krob200	0.78	6.41	7.61	16.6	21.92	29.85	114.9
kroa200	1.55	14.08	18.88	34.59	54.91	58.78	264.7
gr202	2.93	26.51	36.64	91.69	110.77	112.98	530.6
gr202europe	2.71	20.36	27.5	66.58	79.42	104.69	422.1
ts225	1.81	15.07	27.1	54.23	61.61	75	337.9
pr226	1.78	18.36	34.74	62.81	73.9	84.18	354.9
gil262	1.8	18.83	29.9	48.21	56.62	67.9	308.3
pr264	10.45	90.68	111.35	265.57	291.42	328.73	1504.2
pr299	2.22	21.79	34.4	72.01	83.03	95.68	400.6
lin318	1.53	14	21.76	44.39	66.49	79.94	303.4
rd400	2.55	33.2	34.86	76.8	96.96	137	508.2
fl417	4.32	42.96	55.76	101.76	123.37	155.77	702.4
gr431	6.06	58.82	92.4	158.39	172.46	213.97	1002.3
pr439	6.94	64.31	122.37	223.02	236.13	322.44	1245.2
pcb442	7.93	69.64	86.69	206.61	244.34	346.47	1353.6

Tabela 5.3. Tempo total de execução (em segundos) do algoritmo BC com a formulação $\mathcal{P}_c^+$, tempo total de execução do procedimentos de busca local (VND) durante a execução do algoritmo BC e sua fração percentual em relação ao tempo total de execução do BC.

Instâncias Nomes	Tempo total exec. BC	Tempo VND	Tempo VND (%)
eil76	179.9	8.44	4.7
pr76	273.2	15.04	5.5
gr96	124.3	8.36	6.7
gr96africa	354.7	43.45	12.2
rat99	2295.2	24.8	1.1
kroc100	LTE	17.52	0.2
kroa100	8486.9	21.73	0.3
krod100	10147.2	34.53	0.3
rd100	2205.1	15.94	0.7
kroe100	7224.6	32.39	0.4
krob100	145.6	5.88	4
eil101	1312.7	78.51	6
lin105	8800.5	42.26	0.5
pr107	6817.5	32.95	0.5
gr120	7506.6	96.85	1.3
pr124	4183.7	46.01	1.1
bier127	3924.3	170.84	4.4
pr136	LTE	202.32	1.9
gr137	LTE	281.81	2.6
gr137america	33.6	9.6	28.6
pr144	LTE	87.3	0.8
kroa150	LTE	280.74	2.6
krob150	2661.8	52.74	2
pr152	8511.8	92.01	1.1
u159	LTE	222.21	2.1
rat195	LTE	890.5	8.2
d198	3122.2	964.14	30.9
krob200	LTE	338.34	3.1
kroa200	LTE	400.6	3.7
gr202	2158.4	388.65	18
gr202europe	9196.2	415.31	4.5
ts225	LTE	74.67	0.7
pr226	LTE	95.54	0.9
gil262	LTE	90.57	0.8
pr264	LTE	452.5	4.2
pr299	LTE	46.54	0.4
lin318	LTE	22.78	0.2
rd400	LTE	1732.5	16
fl417	LTE	1543.2	14.3
gr431	LTE	1674.3	15.5
pr439	LTE	1286.3	11.9
pcb442	LTE	1342.6	12.4
Média			5.7

Tabela 5.4. Custos das melhores soluções inteiras obtidas pelo algoritmo BC com a formulação $\mathcal{P}_c^+$ e custos das melhores soluções que foram obtidas pela heurística *cover-and-stitch* (CS) integrada com as buscas locais (VND). Mostramos também os *gaps* dessas soluções heurísticas calculados em relação aos custos das soluções ótimas obtidas pelo o algoritmo BC.

Instâncias Nomes	Custo sol. ótima (BC)	Melhor solução heur. (CS+VND)	*Gap* melhor sol. heur. (CS+VND)
eil76	237	244	2.87
pr76	63106	64806	2.62
gr96	261	266	1.88
gr96africa	362	379	4.49
rat99	507	533	4.88
kroa100	9964	10366	3.88
krod100	9628	9940	3.14
rd100	3384	3510	3.59
kroe100	9995	10310	3.06
krob100	9437	9841	4.11
eil101	261	274	4.74
lin105	8219	8589	4.31
pr107	27659	27736	0.28
gr120	2818	2891	2.53
pr124	36707	37540	2.22
bier127	72464	74521	2.76
gr137america	280	280	0
krob150	11175	11380	1.8
pr152	51718	52852	2.15
d198	10589	10702	1.06
gr202	318	321	0.93
gr202europe	192	202	4.95
Média			2.8

Tabela 5.5. Tempos de execução (em segundos) e custos das soluções encontradas pela combinação da heurística construtiva *cover-and-stitch* (CS) com as buscas locais (VND) e das soluções encontradas pelo algoritmo memético proposto por Hu & Raidl [2009]. Para uma comparação justa dos resultados

Instâncias	Custo sol. heur.		Tempo exec.	
Nomes	[Hu & Raidl, 2009]	CS+VND	[Hu & Raidl, 2009]	CS+VND
gr137	440	441	150	9.96
kroa150	11532	11619	150	14.72
d198	10781	10702	300	97.14
krob200	13336	13250	300	29.85
gr202	318	321	300	112.98
ts225	69932	69207	300	75
pr226	67048	64509	300	84.18
gil262	1083	1042	300	67.9
pr264	31220	30828	300	328.73
pr299	22793	22752	450	95.68
lin318	21181	21024	450	79.94
rd400	6765	6765	600	137
fl417	10147	10034	600	155.77
gr431	1291	1290	600	213.97
pr439	60815	60240	600	322.44
pcb442	22321	22170	600	346.47

Tabela 5.6. Total de execuções das buscas locais durante a execução do algoritmo BC com a formulação $\mathcal{P}_c^+$, número de execuções positivas, i.e., execuções que efetivamente aprimoraram a melhor solução inteira corrente, as frações percentuais das execuções positivas em relação ao total de execuções e a autoria da melhor solução inteira relatada na Tabela 4.2, i.e., o método que obteve a melhor solução inteira conhecida.

Instâncias Nomes	Total execuções	Execuções positivas	Execuções positivas (%)	Autoria solução
eil76	91	52	57.1	VND
pr76	168	85	50.6	VND
gr96	81	45	55.6	VND
gr96africa	38	20	52.6	BC
rat99	85	41	48.2	BC
kroc100	75	41	54.7	VND
kroa100	174	100	57.5	Empate
krod100	170	88	51.8	VND
rd100	124	70	56.5	BC
kroe100	128	77	60.2	BC
krob100	38	21	55.3	BC
eil101	311	150	48.2	VND
lin105	136	82	60.3	Empate
pr107	160	82	51.3	VND
gr120	166	97	58.4	Empate
pr124	125	68	54.4	Empate
bier127	227	127	55.9	VND
pr136	150	85	56.7	VND
gr137	209	102	48.8	VND
gr137america	49	30	61.2	VND
pr144	83	47	56.6	VND
kroa150	129	76	58.9	Empate
krob150	65	34	52.3	Empate
pr152	63	37	58.7	VND
u159	81	39	48.1	Empate
rat195	89	45	50.6	VND
d198	80	47	58.8	Empate
krob200	77	44	57.1	VND
kroa200	63	31	49.2	Empate
gr202	51	31	60.8	Empate
gr202europe	47	26	55.3	VND
ts225	11	5	45.5	VND
pr226	21	10	47.6	VND
gil262	19	9	47.4	VND
pr264	24	13	54.2	BC
pr299	9	4	44.4	VND
lin318	2	1	50	VND
rd400	336	172	51.2	Empate
fl417	229	131	57.2	VND
gr431	475	241	50.7	Empate
pr439	426	232	54.5	VND
pcb442	635	384	60.5	Empate
Média			53.9	

Capítulo 6

Conclusões e Trabalhos Futuros

Nesta dissertação, abordamos o *Problema do Subgrafo Biconexo Mínimo Generalizado* (PSBMG). Propomos três formulações de Programação Inteira para o PSBMG. Duas delas, uma direcionada e a outra não-direcionada, se baseiam em um número exponencial de restrições *cutsets* e em resultados de Teoria de Grafos que garantem sua validade. A terceira formulação usa um número polinomial de restrições de fluxos multi-produto para assegurar a biconexidade das soluções viáveis. Apresentamos também duas classes de desigualdades válidas para o politopo das soluções inteiras do PSBMG, a saber as desigualdades de partição de nós e de F-partição, adaptadas de outras desigualdades válidas para o PSBM, i.e., para a versão não-generalizada do PSBMG. Introduzimos uma heurística de separação para as desigualdades de partição de nós baseada em um algoritmo de aproximação proposto por Saran & Vazirani [1991] originalmente para o *Problema do t-Corte Mínimo*.

Discutimos neste trabalho duas abordagens de solução para o PSBMG. A primeira delas trata-se de um algoritmo exato *Branch-and-cut* que utiliza uma das três formulações de Programação Inteira propostas. Realizamos diversos experimentos computacionais com 50 instâncias obtidas do repositório TSPLIB [Reinelt, 1991] com o intuito de avaliar o impacto da separação das desigualdades de partição de nós e de F-partição no desempenho desse algoritmo. Em linhas gerais, a separação de tais desigualdades possibilitou ganhos médios de, respectivamente, 6% e 4% do *gap* de dualidade na raiz e do melhor limite inferior obtidos. No que diz respeito ao número de nós explorados na árvore de enumeração, apontamos que nosso algoritmo *Branch-and-cut*, novamente devido à separação dessas desigualdades, resolveu, em média, 13% menos nós. Das 50 instâncias utilizadas, fornecemos 30 novos certificados de otimalidade. Além disso, provamos melhores limites inferiores e superiores para todas as instâncias de teste consideradas em outros trabalhos na literatura que abordaram o PSBMG. Observamos

uma dificuldade por parte do nosso algoritmo *Branch-and-cut* em resolver na otimalidade, dentre as instâncias de teste selecionadas, aquelas com mais de 210 vértices e 60 *clusters*. De uma forma mais geral, o algoritmo obteve *gaps* de dualidade na margem de 15% para instâncias com pelo menos 100 vértices e com um nível de aglomeração de vértices superior a 50%.

A segunda abordagem de solução para o PSBMG discutida nesta dissertação baseou-se em heurísticas. Propomos duas heurísticas construtivas, uma heurística de refinamento e duas rotinas de busca local integradas em um arcabouço do tipo *Variable Neighborhood Descent*. Com a combinação desses métodos, considerando as 30 instâncias resolvidas na otimalidade pelo algoritmo *Branch-and-cut*, encontramos soluções viáveis para o PSBMG com 3% de *gap* médio. Mais do que isso, concluímos que nossas heurísticas superaram, em qualidade e em tempo de execução, a primeira solução inteira encontrada pelo *Branch-and-cut* ao longo da árvore de enumeração quando este é executado desprovido de uma solução inicial. Comparamos nossos resultados com os resultados mostrados em [Hu & Raidl, 2009]. Para as 16 instâncias da TSPLIB consideradas por Hu & Raidl [2009], as heurísticas aqui propostas obtiveram melhores resultados para 12 instâncias, usando, na média, 60% do tempo total de execução do algoritmo memético introduzido por esses autores. Por fim, concluímos que, quando utilizadas conjuntamente com o *Branch-and-cut* durante sua execução, nossas buscas locais aprimoraram a melhor solução inteira corrente aproximadamente metade das vezes em que elas foram chamadas e que, para 54% das instâncias analisadas, a autoria da melhor solução inteira relatada neste trabalho foi das buscas locais.

Como direções para trabalhos futuros, em primeiro lugar, julgamos importante que seja feito um estudo poliédrico para o PSBMG. Embora estudos assim já tenham sido conduzidos para o PSBM, encontramos dificuldades técnicas em adaptá-los ou em transpô-los para o PSBMG. Acreditamos que isso se deve precisamente à natureza generalizada do problema, que torna sua estrutura facial substancialmente mais complexa. Uma possível saída que vislumbramos para esse desafio é consultar a (infelizmente escarça) literatura de versões generalizadas de problemas de Otimização Combinatória em busca de argumentos de prova que possam se adequar ao nosso caso. Além disso, consideramos interessante também um futuro estudo de reformulações para o PSBMG baseadas na técnica de geração de colunas, bem como a implementação e a experimentação com heurísticas (ou métodos exatos) fundamentadas em Programação Matemática, e.g., *Local Branching* [Fischetti & Lodi, 2003] e *Feasibility Pump* [Fischetti et al., 2005].

Uma deficiência a suprir deste trabalho foi a ausência de um estudo comparativo teórico e experimental das formulações de Programação Inteira aqui discutidas. No

lado experimental, escolhemos dar prioridade à formulação não-direcionada com exponencialmente muitas restrições baseadas em *cutsets* pelo motivo de que vários testes preliminares a favoreceram em termos de qualidade dos resultados. Observamos empiricamente que todas as três formulações forneceram valores comparáveis de limites de Programação Linear. Admitimos desconhecer ainda a razão pela qual a formulação direcionada baseada em *directed cutsets* não dominou as demais, considerando que tal formulação é a mais forte do estado da arte do PSBM.

No tocante ao algoritmo *Branch-and-cut* proposto nesta dissertação, acreditamos que também há oportunidades de melhorias. A princípio, contemplamos as idéias de projetar uma nova e melhor heurística de separação das desigualdades de F-partição e de implementar o algoritmo de Barahona & Kerivin [2003] para realizar a separação exata das desigualdades de partição de nós. Mais do que isso, acreditamos que nosso algoritmo careça e que possa se beneficiar de mecanismos mais sofisticados para gerenciar o *pool* interno de cortes encontrados ao longo da sua execução. Em relação aos experimentos computacionais, julgamos como uma iniciativa positiva o planejamento de testes que utilizem instâncias adaptadas retiradas da TSNDPLib [TSNDPLib, 2008]. Esclarecemos que isso não foi possível nesta dissertação, visto que a documentação da TSNDPLib não fornece instruções claras a respeito do formato dos arquivos de dados das suas instâncias. Contactamos os mantenedores do repositório a fim de resolver tal impasse, no entanto não obtivemos uma resposta em tempo hábil o suficiente para que conseguíssemos executar um número maior de experimentos computacionais.

Referências Bibliográficas

Aarts, E. & Lenstra, J. K. (2003). *Local search in combinatorial optimization*. Princeton University Press.

Applegate, D. L.; Bixby, R. E.; Chvátal, V. & Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Arora, S. & Barak, B. (2007). *Computational Complexity: A Modern Approach*. Cambridge University Press.

Baiou, M.; Barahona, F. & Mahjoub, A. R. (2000). Separation of partition inequalities. *Mathematics of Operations Research*, 25:243--254.

Barahona, F. & Kerivin, H. (2003). Separation of partition inequalities with terminals. Relatório técnico RC22984, IBM Research Institute.

Barahona, F. & Mahjoub, A. R. (1995). On two-connected subgraph polytopes. *Discrete Mathematics*, pp. 19--34.

Bollobás, B. (1998). *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer-Verlag.

BSD Library Functions Manual (2012). `http://www.manpagez.com/man/3/round/`. *Visitado em 8 de fevereiro de 2012*.

Chekuri, C. S.; Goldberg, A. V.; Karger, D. R.; Levine, M. S. & Stein, C. (1997). Experimental study of minimum cut algorithms. Em *Proceedings of the eighth annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 324--333.

Cherkassky, B. V. (1979). *A fast algorithm for computing maximum flow in a network*. Collected papers in combinatorial methods for flow problems. Institute for Systems Studies of Moscow.

Cherkassky, B. V. & Goldberg, A. V. (1994). On implementing push-relabel method for the maximum flow problem. Relatório técnico STAN-CS-94-1523, Department of Computer Science, Stanford University.

Chimani, M.; Kandyba, M.; Ljubic, I. & Mutzel, P. (2010). Orientation-based models for {0,1,2}-survivable network design: theory and practice. *Mathematical Programming*, 124(1-2):413--439.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Relatório técnico 388, Graduate School of Industrial Administration, CMU.

Clarke, L. & Anandalingam, G. (1995). A bootstrap heuristic for designing minimum cost survivable networks. *Computers and Operations Research*, 22(9):921--934.

Cormen, T. H.; Leiserson, C. E. & Rivest, R. L. (1990). *Introduction to algorithms*. MIT Press.

Derigs, U. & Meier, W. (1989). Implementing goldberg's max-flow algorithm - a computational investigation. *Methods and Models of Operations Research*, 33:383--403.

Diestel, R. (2005). *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag.

Dirac, G. A. (1967). Minimally 2-connected graphs. *J. Reine Angew.*, pp. 204--216.

Edmonds, J. (1965a). Maximum matching and a polyhedron with 0-1 vertices. *Journal of Research of the National Bureau of Standards*, 69B:125--130.

Edmonds, J. (1965b). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449--467.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

Eswaran, K. & Tarjan, R. (1976). Augmentation problems. *SIAM Journal on Computing*, 5:653--665.

Feremans, C.; Labbé, M. & Laporte, G. (2004). The generalized minimum spanning tree problem: Polyhedral analysis and branch-and-cut algorithm. *Networks*, 43:71--86.

Fischetti, M.; Glover, F. & Lodi, A. (2005). The feasibility pump. *Mathematical Programming*, 104:91--104.

Fischetti, M. & Lodi, A. (2003). Local branching. *Mathematical Programming*, 98:23--47.

Fischetti, M.; Salazar, J. J. & Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45:378–394.

Fleischer, L. (2001). A 2-approximation for minimum cost $\{0, 1, 2\}$-vertex connectivity. Em *Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pp. 115--129.

Frank, A. & Tardos, E. (1989). An application of submodular flows. *Linear Algebra and its Applications*, 114/115:320--348.

Frederickson, G. N. & JaJa, J. (1981). Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270--283.

Frederickson, G. N. & JaJa, J. (1982). On the relationship between the biconnectivity augmentation and travelling salesman problems. *Theoretical Computer Science*, pp. 189--201.

Galassi, M. (2009). *GNU Scientific Library Reference Manual*. RRP.

Garg, N.; Vempala, S. & Singla, A. (1993). Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. Em *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pp. 103--111.

Gendreau, M. & Potvin, J., editores (2010). *International Series in Operations Research and Management Science - Handbook of Metaheuristics*, volume 146. Springer.

Goemans, M. X. & Williamson, D. P. (1995a). A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296--317.

Goemans, M. X. & Williamson, D. P. (1995b). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42.

Goldberg, A. V. & Tarjan, R. E. (1986). A new approach to the maximum flow problem. Em *Proceedings of the eighteenth annual ACM Symposium on Theory of Computing*, pp. 136--146.

Goldschmidt, O. & Hochbaum, D. S. (1988). Polynomial algorithm for the k-cut problem. Em *Proceedings of the twenty-ninth annual IEEE Symposium on the Foundations of Computer Science*, pp. 444–451.

Grötschel, M. & Holland, O. (1987). A cutting plane algorithm for minimum perfect 2-matchings. *Computing*, 39:327--344.

Grötschel, M.; Monma, C. & Stoer, M. (1992a). Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40:309--330.

Grötschel, M.; Monma, C. & Stoer, M. (1992b). Integer polyhedra arising in the design of communication networks with low-connectivity constraints. *Operations Research*, 40:474--504.

Grötschel, M.; Stoer, M. & Monma, C. (1995). *Design of Survivable Networks - Models, methods and applications*. Elsevier.

Gusfield, D. (1990). Very simple method for all pairs network flow analysis. *SIAM Journal on Computing*, 19:143--155.

Hao, J. & Orlin, J. L. (1994). A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17:424--446.

Hoos, H. H. & Stutzle, T. (2005). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann.

Hopcroft, J. & Tarjan, R. (1973). Efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372--378.

Hu, B.; Leitner, M. & Raidl, G. (2008). Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14:473--499.

Hu, B.; Leitner, M. & Raidl, G. (2010a). The generalized minimum edge biconnected network problem: Efficient neighborhood structures for variable neighborhood search. *Networks*, 55:257–275.

Hu, B.; Leitner, M. & Raidl, G. (2010b). The generalized minimum edge-biconnected network problem: Efficient neighborhood structures for variable neighborhood search. *Networks*, 55:256--275.

Hu, B. & Raidl, G. (2009). A memetic algorithm for the Generalized Minimum Vertex-biconnected Network Problem. Em *Proceedings of the ninth International Conference on Hybrid Intelligent Systems*, pp. 63--68.

IBM ILOG CPLEX Optimizer (2012). `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`. *Visitado em 15 de janeiro de 2012.*

ILOG Concert Technology (2012). `http://www-01.ibm.com/software/`. *Visitado em 8 de fereveiro de 2012.*

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

Johnson, D. S.; Minkoff, M. & Phillips, S. (2000). The prize-collecting steiner tree problem: Theory and practice. Em *Proceedings of the eleventh ACM-SIAM Symposium on Discrete Algorithms*, pp. 760--769.

Junger, M.; Reinelt, G. & Thienel, S. (1995). *Practical problem solving with cutting plane algorithms in combinatorial optimization*. Combinatorial Optimization, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. ACM.

Karp, R. M. (1972). *Reducibility among combinatorial problems*. Complexity of Computer Computations. New York: Plenum.

Kerivin, H. & Mahjoub, A. R. (2005). Design of Survivable Networks: A survey. *Networks*, 46(1):1--21.

Kerivin, H.; Mahjoub, A. R. & Nocq, C. (2004). *(1,2)-survivable networks: Facets and branch-and-cut*. MPS-SIAM Series in Optimization. SIAM.

Khuller, S. & Raghavachari, B. (1995). Improved approximation algorithms for uniform connectivity problems. Em *Proceedings of the twenty-seventh annual ACM Symposium on Theory of Computing*, STOC '95, pp. 1--10.

Khuller, S. & Thurimella, R. (1992). Approximation algorithms for graph augmentation. Em *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pp. 330–341. Springer Berlin / Heidelberg.

Khuller, S. & Vishkin, U. (1992). Biconnectivity approximations and graph carvings. Em *Proceedings of the twenty-fourth annual ACM Symposium on Theory of Computing*, STOC '92, pp. 759--770.

Koch, T. & Martin, A. (1998). Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207--232.

LEMON Graph Library (2012). `http://lemon.cs.elte.hu/trac/lemon/wiki/Documentation`. *Visitado em 8 de fereveiro de 2012.*

Letchford, A.; Reinelt, G. & Theis, D. (2004). A faster exact separation algorithm for blossom inequalities. Em *Integer Programming and Combinatorial Optimization*, volume 3064 of *Lecture Notes in Computer Science*, pp. 19--52. Springer Berlin / Heidelberg.

Lucena, A. & Beasley, J. (1996). *Branch-and-cut algorithms*. Oxford University Press.

Magnanti, T. L. & Raghavan, S. (2005). Strong formulations for network design problems with connectivity requirements. *Networks*, 45(2):61--79.

Mahjoub, A. R. (1994). Two-edge connected spanning subgraphs and polyhedra. *Mathematical Programming*, pp. 199--208.

Mahjoub, A. R. & Nocq, C. (1999). On the linear relaxation of the 2-node connected subgraph polytope. *Discrete Applied Mathematics*, pp. 389--416.

Menger, K. (1927). Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96--115.

Micali, S. & Vazirani, V. V. (1980). An o(v$\hat{1}$/2e) algorithm for finding maximum matching in general graphs. Em *Proceedings of the twenty-first annual Symposium on he Foundations of Computer Science*, pp. 17--27.

Monma, C. & Shallcross, D. F. (1989). Methods for designing communication networks with certain two-connected survivability constraints. *Operations Research*, 37(4):531--541.

Monma, C. L.; Munson, B. S. & Pulleyblank, W. R. (1990). Minimum-weight two-connected spanning networks. *Mathematical Programming*, pp. 153--171.

Myung, Y.; Lee, C. & Tcha, D. (1995). On the generalized minimum spanning tree problem. *Networks*, 26(4):231--241.

Nash-Williams, C. S. J. A. (1960). On orientations, connectivity, and odd vertex pairings in finite graphs. *Canadian Journal of Mathematics*, pp. 555--567.

Padberg, M. & Rinaldi, G. (1990). An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47:19--36.

Padberg, M. & Rinaldi, G. (1991). A Branch-and-Cut algorithm for resolution of large scale of Symmetric Traveling Salesman Problem. *SIAM Review*, pp. 60--100.

Pagacz, A.; Hu, B. & Raidl, G. (2010). A memetic algorithm with population management for the Generalized Minimum Vertex-biconnected Network Problem. Em *Proceedings of the second International Conference on Intelligent Networking and Collaborative Systems*, pp. 356--361.

Pop, P. C.; Kern, W. & Still, G. (2006). A new relaxation method for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 170(3):900--908.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389--1401.

Pulleyblank, W. R. (1973). *Faces of matching polyhedra.* Tese de doutorado, University of Waterloo.

Ravi, R. & Williamson, D. P. (1995). An approximation algorithm for minimum-cost vertex-connectivity problems. Em *Proceedings of the sixth annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 332--341.

Reinelt, G. (1991). TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing*, 4(3):376–384.

Saran, H. & Vazirani, V. V. (1991). Finding k-cuts within twice the optimal. Em *Proceedings of the thirty-second annual IEEE Symposium on the Foundations of Computer Science*, pp. 743--751.

Savelsbergh, M. W. P. & Nemhauser, G. (1998). Functional description of MINTO, a Mixed INTeger Optimizer (Version 3.0). Relatório técnico COC-91-03D, Georgia Institute of Technology.

SDPA (2012). `http://sdpa.sourceforge.net/`. *Visitado em 2 de fevereiro de 2012.*

Sleator, D. D. & Tarjan, R. E. (1983). A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362--391.

Steiglitz, K.; Kleitman, D. & Weiner, P. (1969). The design of minimum-cost survivable networks. *IEEE Transactions on Circuit Theory*, 16(4):455--460.

Stoer, M. (1992). *Design of Survivable Networks.* Springer-Verlag.

Tarjan, R. E. (1977). Finding optimum branching. *Networks*, 7:25–35.

TSNDPLib (2008). Tsndplib: Collection of benchmark instances for the topological {0, 1, 2}-survivable network design problem. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

Vempala, S. & Vetta, A. (2000). Factor 4/3 approximations for minimum 2-connected subgraphs. Em *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '00, pp. 262--273.

West, D. B. (2001). *Introduction to Graph Theory*. Prentice Hall.