ALGORITMOS BRANCH-AND-CUT-AND-PRICE PARA O PROBLEMA DA ÁRVORE GERADORA DE CUSTO MÍNIMO COM RESTRIÇÃO DE GRAU

LUÍS HENRIQUE COSTA BICALHO

ALGORITMOS BRANCH-AND-CUT-AND-PRICE PARA O PROBLEMA DA ÁRVORE GERADORA DE CUSTO MÍNIMO COM RESTRIÇÃO DE GRAU

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

10 de abril de 2014

Bicalho, Luís Henrique Costa

B583a Algoritmos Branch-and-cut-and-price para o Problema da Árvore Geradora de Custo Mínimo com restrição de Grau / Luís Henrique Costa Bicalho. — Belo Horizonte, 2014
xviii, 55 f. ; 29cm

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

1. Computação – Teses. 2. Análise Numérica. 3. Otimização combinatória. 4. Árvores (Teoria dos Grafos). I. Orientador. II. Título.

CDU 519.6*61(043)

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

FOLHA DE APROVAÇÃO

Algoritmos branch-and-cut-and-price para o problema da árvore geradora de custo mínimo com restrição de grau

LUIS HENRIQUE COSTA BICALHO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

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

PROF. ABÍLIO PEREIRA DE LUCENA FILHO
Faculdade de Economia e Administração - UFRJ

PROF. GERALDO ROBSON MATEUS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 10 de abril de 2014.

Agradecimentos

Ao meu orientador, prof. Alexandre Salles da Cunha, agradeço pela minha formação, pelos ensinamentos, por sempre procurar me motivar e por ter me dado a honra de trabalhar com este tema que lhe é estimado e também fez parte de sua formação.

Aos meus pais, Cristina e Geraldo, e aos meus irmãos, Patrícia e Thiago, obrigado por acreditarem em mim e me apoiarem sempre.

Ao prof. Robson, obrigado pelos grandes ensinamentos em Pesquisa Operacional.

Aos companheiros do LaPO, em especial: Cristopher Moreira, Vitor Andrade, Vinicius Morais, Ramon Lopes, Amadeu Almeida, Franklin Almeida, Dilson Guimarães, Afonso Sampaio, Phillippe Samer e Dilson Pereira. Agradeço a vocês pelos bons momentos de laboratório e pelas ajudas nos momentos de dificuldade.

Aos amigos Alex Guimarães e Vitor Sales, agradeço pelas boas conversas e risadas na hora do cafezinho.

Agradeço a todos que contribuíram direta ou indiretamente para eu chegar onde estou hoje. Obrigado!

Resumo

Dados um grafo não direcionado $G = (V, E)$, valorado em suas arestas, e inteiros positivos $\{d_v : v \in V\}$ associados aos vértices de G, o Problema da Árvore Geradora de Custo Mínimo com Restrição de Grau (PAGMRG) consiste em encontrar uma árvore geradora mínima de G tal que o grau de cada vértice v na árvore não exceda d_v. Nesta dissertação, investigamos três abordagens de solução exata para resolver o PAGMRG, baseadas em Geração de Colunas (GC). Os procedimentos de GC aqui utilizados consistem em precificar dinamicamente as variáveis associadas às arestas de E em um modelo não direcionado para o problema. Os três algoritmos se diferenciam pela frequência em que a GC é utilizada. Para um deles, GC é aplicada somente no nó raiz de um algoritmo *Branch-and-cut*, enquanto em um outro, GC é aplicada ao longo de toda a árvore de busca. Um terceiro algoritmo somente aplica GC quando o número de variáveis de decisão em um subproblema satisfaz certas condições. As três abordagens de solução foram comparadas entre si e com os melhores métodos da literatura. Também introduzimos um novo conjunto de instâncias, mais difíceis de serem resolvidas. Nossos testes computacionais indicam que um dos métodos propostos é o melhor para resolver instâncias de médio porte com custos Euclidianos. Para instâncias com custos aleatórios, uma abordagem da literatura baseada em Programação por Restrições produziu resultados muito bons. Aquele algoritmo, entretanto, não é robusto, já que muitas vezes não é capaz de resolver instâncias com custos Euclidianos facilmente resolvidas aqui. Para instâncias com $|V| = 2000$, um algoritmo híbrido *Relax-and-cut*/*Branch-and-cut* na literatura permanece como a melhor escolha.

Palavras-chave: Árvore Geradora Mínima com Restrição de Grau, Branch-and-cut, Branch-and-cut-and-price, Geração de Colunas, Otimização Combinatória.

Abstract

Given an edge weighted undirected graph $G = (V, E)$ where positive integers $\{d_v : v \in V\}$ are assigned to its vertices, the Degree-Constrained Minimum Spanning Tree Problem (DCMST) consists in finding a minimum cost spanning tree of G such that the degree of each vertex v in the tree does not exceed d_v. In this dissertation, we investigate three exact solution approaches for solving DCMST, based on Column Generation (CG). The CG procedure used here consists in dynamically pricing variables assigned to the edges of E in an undirected model for the problem. The three algorithms differ by the frequency that CG is applied. In one of them, CG is applied only at the root node of a Branch-and-cut algorithm, while in another, it is applied throughout the entire search tree. A third algorithm only applies CG when the number of decision variables in a subproblem satisfies certain conditions. The three solution approaches were compared among themselves and to the best existing methods in the literature. We also introduced a new set of hard to solve instances. Our computational results indicate that one of the proposed methods is the best for medium size Euclidean cost instances. For instances with random costs, an approach in the literature that is based on Constraint Programming produced very good results. That algorithm, however, is not robust, since quite often it is unable to solve Euclidean cost instances easily solved here. For instances with $|V| = 2000$, a hybrid Relax-and-cut/Branch-and-cut in the literature remains as the best choice.

Keywords: Degree constrained minimum spanning tree, Branch-and-cut, Branch-and-cut-and-price, Column Generation, Combinatorial Optimization.

Lista de Tabelas

Lista de Abreviaturas e Siglas

BB	*Branch-and-bound*
BC	*Branch-and-cut*
BCP	*Branch-and-cut-and-price*
BCP-BC	Algoritmo Híbrido *Branch-and-cut-and-price* e *Branch-and-cut* para o PAGMRG
BL	Busca Local
GC	Geração de Colunas
GCBC	Algoritmo Híbrido que utiliza Geração de Colunas em uma etapa anterior à um *Branch-and-cut* para o PAGMRG
DB	Desigualdade Blossom
dE-1C	d-Emparelhamento 1-Capacitado
NDRC	*Non-delayed Relax-and-cut*
NDRC/BC	Algoritmo Híbrido *Non-delayed Relax-and-cut*/*Branch-and-cut* proposto em Lucena et al. [2011]
PAGM	Problema da Árvore Geradora de Custo Mínimo
PAGMRG	Problema da Árvore Geradora de Custo Mínimo com Restrição de Grau
PLM	Problema Linear Mestre
PLMR	Problema Linear Mestre Restrito
SEC	*Subtour Elimination Constraint*

VNS *Variable Neighborhood Search*

VNSL+ASG Algoritmo VNS-Lagrangeano em conjunto com um método de Árvore de Subgradiente proposto em Freitas [2011]

Sumário

Capítulo 1

Introdução

Estudos envolvendo problemas de otimização que buscam encontrar árvores ótimas em grafos são de grande interesse técnico e econômico. Isso se deve ao fato de que estas estruturas são capazes de modelar problemas que surgem, por exemplo, em redes de telecomunicações, em redes de transporte e em circuitos elétricos [Magnanti & Wolsey, 1995].

O problema de encontrar uma Árvore Geradora de Custo Mínimo (PAGM) em um grafo não direcionado, valorado em suas arestas, é um dos problemas mais conhecidos em Teoria dos Grafos. A solução do PAGM pode ser obtida em tempo polinomial através de um algoritmo guloso [Kruskal, 1956] [Prim, 1957].

No entanto, em muitas aplicações, faz-se necessária a adição de algumas restrições à topologia de árvores geradoras, o que pode fazer com que o novo problema de otimização se torne difícil de ser resolvido de forma eficiente. Em particular, nesta dissertação estamos interessados no estudo de um destes problemas, o Problema da Árvore Geradora de Custo Mínimo com Restrição de Grau (PAGMRG).

Para definir o problema, considere que $G = (V, E)$ denote um grafo não direcionado, conexo, no qual V é o conjunto de vértices e E o conjunto de arestas. Assuma que para cada aresta $e \in E$ seja associado um custo $c_e \in \mathbb{R}$. Considere $n = |V|$ e $m = |E|$. No PAGMRG, o grau de cada vértice $v \in V$ na árvore geradora é limitado superiormente por um valor inteiro d_v, tal que $1 \leq d_v \leq n - 1$. O objetivo deste problema é encontrar uma árvore geradora de mínimo custo em que o limite de grau não seja excedido em nenhum vértice de G.

Um exemplo de aplicação do PAGMRG ocorre em redes de comunicação [Gavish, 1982], onde os vértices representam os centros de comunicação e as arestas correspondem às possíveis conexões entre esses centros. As restrições de grau em um vértice servem para modelar a capacidade máxima de tráfego ou a quantidade máxima de

conexões nos terminais.

Esta dissertação tem como objetivo avaliar computacionalmente algoritmos exatos do tipo *Branch-and-cut-and-price* para a resolução exata do PAGMRG. Embora a formulação no qual se baseiam os algoritmos aqui propostos seja a mesma formulação empregada pelos melhores algoritmos da literatura [Cunha & Lucena, 2007; Lucena et al., 2011], reduções importantes no tempo de CPU são obtidas ao trabalharmos com as arestas do grafo de definição do problema em um esquema de Geração de Colunas. Estudamos também a utilização de Geração de Colunas em uma etapa de preprocessamento para o PAGMRG, e comparamos com uso de algoritmos *Relax-and-cut* para o mesmo propósito.

As principais contribuições deste trabalho, na ordem em que aparecem no texto, são:

- O estudo de três algoritmos *Branch-and-cut-and-price* para o PAGMRG, que se diferenciam pelos nós da árvore de enumeração em que se aplica o procedimento de Geração de Colunas. Um dos algoritmos implementa a Geração de Colunas apenas na raiz da árvore de enumeração, enquanto um segundo o faz em toda a árvore. O terceiro algoritmo utiliza a Geração de Colunas em alguns nós, dependendo do número de variáveis envolvidas na formulação daquele nó.

- A proposição de um novo conjunto de instâncias, mais difíceis, para o PAGMRG.

- Melhoramentos nos tempos de execução de muitas das instâncias anteriormente resolvidas.

O restante desta dissertação está organizado da seguinte forma. No Capítulo 2, examinamos os principais trabalhos relacionados ao PAGMRG, em conjunto com as formulações propostas para o problema até então. No Capítulo 3, apresentamos alguns métodos da literatura para tratar o problema e descrevemos os três métodos *Branch-and-cut-and-price* propostos nesta dissertação. No Capítulo 4, propomos um novo conjunto de instâncias para PAGMRG. Além disso, nesse mesmo capítulo, as abordagens sugeridas neste trabalho são comparadas entre si e com abordagens exatas da literatura, envolvendo tanto Programação Linear Inteira [Lucena et al., 2011; Andrade & Freitas, 2013] quanto Programação por Restrições [Fages et al., 2013]. Por fim, no Capítulo 5, discutimos as contribuições e as limitações deste trabalho e, finalmente, conclusões e direções para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Antes de discutir os principais trabalhos relacionados ao PAGMRG, vamos apresentar um caso especial do problema, denominado k-PAGMRG. Essa variante foi estudada por Narula & Ho [1980], Volgenant [1989] e Caccetta & Hill [2001]. O k-PAGMRG ocorre quando o mesmo limite k é imposto a todos os vértices de G, ou seja, $d_v = k$ para todo $v \in V$. Para o caso em que $k = n - 1$, o problema se reduz ao PAGM. Se $k = 2$, o problema equivale a encontrar um Caminho Hamiltoniano de custo mínimo, provadamente $\mathcal{NP}$-Difícil [Garey & Johnson, 1979].

Papadimitriou & Vazirani [1984] provaram que o k-PAGMRG continua $\mathcal{NP}$-Difícil se $3 \leq k \leq n - 2$ e se os custos associados às arestas forem arbitrários, já que é possível reduzir, em tempo polinomial, uma instância do problema do Caminho Hamiltoniano de custo mínimo em uma instância do k-PAGMRG. Além disso, Papadimitriou & Vazirani [1984] provaram que, se $k = 3$, se os vértices de V correspondem a pontos no plano Euclidiano e se o custo de cada aresta em E for a distância Euclidiana entre os respectivos vértices, o problema também é $\mathcal{NP}$-Difícil. Os autores também provaram que, se $k \geq 5$ e se os vértices possuem coordenadas inteiras no espaço Euclidiano de dimensão qualquer, o k-PAGMRG com custos Euclidianos é resolvido de forma polinomial.

O PAGMRG tem sido estudado há bastante tempo na literatura. Obruča [1968] foi um dos primeiros trabalhos a mencionar o PAGMRG. No entanto, o algoritmo *Branch-and-bound* (BB) proposto por Narula & Ho [1980] é tido como o primeiro método exato proposto para solucionar o problema. Além do algoritmo exato, os autores apresentaram duas heurísticas para o PAGMRG. Na primeira delas, partindo de uma solução viável para o PAGMRG, obtida através de uma adaptação no algoritmo de Prim [Prim, 1957], tenta-se obter melhores soluções através de um método de troca de arestas. Na outra heurística, partindo de uma AGM para o grafo que define o

problema, busca-se satisfazer as restrições de grau através de trocas de arestas da árvore. Savelsbergh & Volgenant [1985] melhoraram os resultados obtidos por Narula & Ho [1980] ao propor um BB que emprega regras de *branching* mais eficazes. Além disso, Savelsbergh & Volgenant [1985] introduziram um método que utiliza troca de arestas para caracterizar arestas garantidamente subótimas.

A seguir, apresentamos a Formulação Linear Inteira para o PAGMRG, proposta por Narula & Ho [1980] (também adotada por Savelsbergh & Volgenant [1985]), utilizada pela maioria dos trabalhos que abordam o problema [Gavish, 1982; Volgenant, 1989; Caccetta & Hill, 2001; Andrade et al., 2006; Freitas, 2011].

Seja $\{x_e : e \in E\}$ o conjunto de variáveis binárias associadas às arestas de G. Quando $x_e = 1$, a aresta e está presente na árvore geradora ótima e $x_e = 0$, quando não está presente. Assuma que $E(S)$ represente o conjunto das arestas com ambas as extremidades em $S \subset V$, e que $\delta(S)$ seja o conjunto das arestas que possuem uma única extremidade em S. Quando $S = \{i\}$, $i \in V$, denotamos $\delta(\{i\})$ por $\delta(i)$. Finalmente, considere $x(M) = \sum_{e \in M} x_e$ como sendo a soma das variáveis de decisão associadas às arestas de um subconjunto $M \subseteq E$. A formulação introduzida por Narula & Ho [1980] é dada por

$$\min \left\{ \sum_{e \in E} c_e x_e : x \in P_1 \cap \mathbb{B}^m \right\}, \tag{2.1}$$

onde P_1 é o politopo definido por

$$x(E) = |V| - 1, \tag{2.2}$$

$$x(E(S)) \leq |S| - 1, \ \forall S \subset V, \ S \neq \emptyset, \tag{2.3}$$

$$x(\delta(i)) \leq d_i, \ \forall i \in V, \tag{2.4}$$

$$x_e \geq 0, \ \forall e \in E. \tag{2.5}$$

Note que as desigualdades (2.3), conhecidas como Restrições de Eliminação de Subciclo (*Subtour Elimination Constraints*, SECs) [Dantzig et al., 1954], geradas para um conjunto S com dois elementos, por exemplo, $S = \{i, j\}$, para $i, j \in V$, implicam que $x_{ij} \leq 1$. Desigualdades SECs impedem a ocorrência de ciclos na solução. As desigualdades (2.2), (2.3) e (2.5) descrevem a envoltória convexa do politopo das Árvores Geradoras de G [Magnanti & Wolsey, 1995]. Por fim, as desigualdades *d-Emparelhamento* (2.4) garantem que o grau máximo de cada vértice não seja excedido na árvore que procuramos.

Gavish [1982] foi o primeiro a propor a dualização das restrições (2.4) em uma Relaxação Lagrangeana baseada na Formulação P_1. Outros trabalhos, por exemplo, Caccetta & Hill [2001] e Andrade et al. [2006], também exploraram esse método. Volgenant [1989] também propôs uma Relaxação Lagrangeana para P_1, dualizando as restrições (2.4). Naquela referência, a Relaxação Lagrangeana foi utilizada como mecanismo para obtenção de limites inferiores em um algoritmo BB. Na raiz da árvore de enumeração, os multiplicadores de Lagrange ótimos são aproximados por um método *dual ascent*. Além disso, Volgenant [1989] introduziu no BB um teste de fixação de variáveis para caracterizar arestas que comprovadamente não pertencem à qualquer solução ótima do PAGMRG. O algoritmo sugerido por Volgenant foi testado no contexto do k-PAGMRG, com $k \in \{3, 4, 5\}$, em grafos com no máximo 150 vértices, e arestas com custos Euclidianos ou aleatórios. O autor concluiu que, para o k-PAGMRG, as instâncias que possuem custos Euclidianos são mais fáceis de serem resolvidas à otimalidade.

Caccetta & Hill [2001] propuseram um algoritmo *Branch-and-cut* (BC), utilizando como preprocessamento um método de Relaxação Lagrangeana semelhante àquele proposto em Volgenant [1989], para o k-PAGMRG. O BC proposto por esses autores utiliza heurísticas para separar SECs e desigualdades *Cutsets*. O algoritmo de Caccetta & Hill [2001] foi testado em um conjunto de 3150 instâncias, com número de vértices variando de 100 a 800, distâncias não Euclidianas e $k \in \{3, 4, 5, 6\}$.

Andrade et al. [2006] apresentaram uma heurística gulosa *multi-start*, baseada no Algoritmo de Kruskal, que utiliza informações duais de uma Relaxação Lagrangeana de P_1 (dualizando-se também as restrições (2.4)) para alterar os custos de entrada do algoritmo. Além disso, foi proposto um mecanismo de *prevenção de inviabilidade* que garantidamente evita a obtenção de árvores inviáveis para o caso em que a instância seja definida por um grafo completo. Esse método foi testado em instâncias com até 900 vértices e com restrições de grau variando entre 1, 2, 3 e 4, de maneira não uniforme.

Em Cunha & Lucena [2007] foi introduzida uma nova formulação para o PAGMRG. Os autores se atentaram para o fato de que o conjunto de soluções viáveis para o PAGMRG está contido na interseção do politopo do *d-Emparelhamento 1-Capacitado* (dE-1C) com o politopo do PAGM. O politopo do dE-1C pode ser completamente descrito através de restrições lineares, dadas por (2.4)-(2.5), pelas restrições de capacidade $\{x_e \leq 1 : e \in E\}$, e pelas Desigualdades Blossom [Edmonds, 1965; Edmonds & Johnson, 1970]:

$$x(E(H)) + x(T) \leq \left\lfloor \frac{1}{2}(d(H) + |T|) \right\rfloor, \ \forall H \subseteq V, \ \forall T \subseteq \delta(H), \tag{2.6}$$

onde $d(H) = \sum_{i \in H} d_i$ e $(d(H) + |T|)$ é ímpar.

Em decorrência disso, Cunha & Lucena [2007] incorporaram à P_1 as Desigualdades Blossom (DBs), válidas para o dE-1C [Edmonds & Johnson, 1970]. A formulação proposta por Cunha & Lucena [2007] é, então, dada por

$$\min \left\{ \sum_{e \in E} c_e x_e : x \in P_2 \cap \mathbb{B}^m \right\}, \tag{2.7}$$

onde o poliedro P_2 é definido por

$$P_2 = \{x \in \mathbb{R}^m : x \text{ satisfaz (2.2)-(2.6)}\}. \tag{2.8}$$

Observe que o poliedro P_2 é a interseção de dois poliedros inteiros: o poliedro da AGM [Edmonds, 1971] e o poliedro do dE-1C [Edmonds, 1965]. Cunha [2006] mostrou que P_2 pode, eventualmente, possuir vértices fracionários. Assumindo que $X = \{x \in \mathbb{B}^m : x \in P_1\}$ denota o conjunto de vetores que representam as soluções viáveis para o PAGMRG, foram ilustrados casos onde $conv(X) \subset P_2 \subset P_1$ Cunha [2006].

Além das DBs, outras desigualdades válidas para o PAGMRG foram estudadas e avaliadas em Cunha [2006]. As desigualdades em questão, *Combs* e *Clique Trees*, foram generalizadas a partir do Problema do Caixeiro Viajante. No entanto, essas desigualdades não foram consideradas naquele trabalho (e pelo mesmo motivo não foram consideradas aqui) devido à dificuldade de sua separação.

Para resolver a Relaxação Linear de (2.7), Cunha & Lucena [2007] propuseram um algoritmo *Non-delayed Relax-and-cut* (NDRC), no qual as desigualdades (2.4) e (2.6) são dualizadas. Mais detalhes deste algoritmo serão dados na Seção 3.1.2.

Lucena et al. [2011] propuseram um algoritmo BC que utiliza como etapa de preprocessamento o procedimento NDRC introduzido em Cunha & Lucena [2007], denominado por NDRC/BC. O BC sugerido por Lucena et al. [2011] separa desigualdades SECs e *Cutsets* de forma exata, ao passo que DBs são separadas de forma heurística. O NDRC/BC foi testado em várias instâncias do PAGMRG presentes na literatura e comparado com um BC puro. Os autores conseguiram alcançar a solução ótima em todas as instâncias testadas e o método híbrido NDRC/BC produziu melhores resultados se comparado a um BC que não se beneficia de desigualdades SECs e DBs identificadas no NDRC.

Na dissertação de mestrado de Freitas [2011] foi apresentada uma heurística *Variable Neighborhood Search* (VNS) Lagrangeana em conjunto com um método de enumeração implícita, denominado Árvore de Subgradiente, para o PAGMRG. Apesar de

já ter sido proposta uma formulação mais forte para o problema, Freitas [2011] utilizou uma Relaxação Lagrangena baseada na Formulação P_1. Segundo os autores, o método proposto foi capaz de melhorar os limites inferiores ou provar a otimalidade das instâncias sugeridas por Andrade et al. [2006] e das instâncias introduzidas por Cunha & Lucena [2007]. No entanto, o algoritmo NDRC/BC proposto por Lucena et al. [2011] já havia encontrado o ótimo de todas as instâncias propostas por Andrade et al. [2006] e Cunha & Lucena [2007], fato que passou despercebido em Freitas [2011] e Andrade & Freitas [2013].

Em Fages et al. [2013] foi proposto um modelo de Programação por Restrições que utiliza uma Relaxação Lagrangeana na etapa de propagação. As restrições são geradas a partir de propriedades estruturais inerentes ao PAGMRG. Os autores conseguiram resolver as instâncias Euclidianas propostas em Andrade et al. [2006] e as instâncias aleatórias propostas por Cunha & Lucena [2007], em tempos de CPU consideravelmente baixos, se comparados àqueles obtidos por Lucena et al. [2011], mesmo levando em consideração as diferenças entre as máquinas empregadas nos experimentos computacionais. Entretanto, como veremos no Capítulo 4, o mesmo ganho computacional não foi observado para as instâncias Euclidianas, reconhecidamente mais difíceis, introduzidas por Cunha & Lucena [2007].

Algumas metaheurísticas também foram propostas para o PAGMRG. Zhou & Gen [1997] propuseram um Algoritmo Genético para o problema. Uma abordagem evolucionária que utiliza uma heurística baseada no Algoritmo de Prim [Prim, 1957] foi proposta por Knowles & Corne [2000]. Já em Ribeiro & Souza [2002] foi sugerido um algoritmo VNS que utiliza como busca local um método *Variable Neighborhood Descent*, ao invés de explorar uma única vizinhança. Baseado no algoritmo VNS proposto por Ribeiro & Souza [2002], de Souza & Martins [2008] propuseram um algoritmo *Skewed* VNS que utiliza um Algoritmo de Segunda Ordem na fase de perturbação, conseguindo, na maioria das instâncias, limites superiores iguais ou melhores aos obtidos por Ribeiro & Souza [2002].

Capítulo 3

Algoritmos *Branch-and-cut-and-price* para o PAGMRG

Neste capítulo, discutimos, inicialmente, algumas alternativas algorítmicas para avaliar o limite de Programação Linear da Formulação P_2, ao qual denotamos por PL(P_2), onde, dada uma formulação P para o PAGMRG, PL(P) = min $\{\sum_{e \in E} c_e x_e : x \in P\}$.

Logo após, os três algoritmos *Branch-and-cut-and-price* (BCP) para o PAGMRG são apresentados. No primeiro procedimento, a Geração de Colunas (GC) só é empregada na raiz da árvore de enumeração, com o objetivo de calcular PL(P_2). Através de informações duais obtidas no decorrer do procedimento, tentamos fixar arestas e obter soluções viáveis de boa qualidade. Nos algoritmos subsequentes, colunas e desigualdades válidas são geradas ao longo da árvore. No entanto, no terceiro algoritmo, a GC é realizada até que um determinado critério seja satisfeito, momento a partir do qual todas as arestas são explicitamente consideradas.

3.1 Alternativas para calcular PL(P_2)

3.1.1 Algoritmos de Planos de Corte

A Formulação P_2 possui duas classes de desigualdades válidas contendo exponencialmente muitas restrições: SECs (2.3) e DBs (2.6). Logo, mesmo para instâncias de pequenas dimensões, torna-se impraticável determinar Relaxações Lineares contendo explicitamente todas essas restrições. Para contornar esse problema, podemos utilizar algoritmos de Planos de Corte.

Para encontrar PL(P_2) através de um algoritmo de Planos de Corte, devemos, inicialmente, adotar uma *relaxação* de P_2 denotada por $\overline{R}$: $\min\{\sum_{e\in E} c_e x_e : x \in \overline{P}_2\}$, onde $\overline{P}_2 = \{x \in \mathbb{R}^m : x \text{ satisfaz } (2.2), (2.4) \text{ e } (2.5)\}$. Seja $\overline{x}$ a solução ótima de $\overline{R}$. Através de *algoritmos de separação*, tentamos encontrar SECs e/ou DBs que são violadas por $\overline{x}$. Se $\overline{x}$ não viola nenhuma das desigualdades SECs e DBs, então $\overline{x} \in P_2$ e portanto resolve a Relaxação Linear de (2.7). Caso contrário, adicionamos um conjunto de desigualdades violadas por $\overline{x}$ à relaxação $\overline{P}_2$, obtendo, então, uma nova solução ótima $\overline{x}$ que resolve $\overline{R}$. O processo acima se repete até que a solução corrente $\overline{x}$ não viole nenhuma das desigualdades pertencentes às classes (2.3) e (2.6), momento este em que PL(P_2) é avaliado. Se, ao final desse procedimento, $\overline{x}_e \in \mathbb{B}$, para todo $e \in E$, então $\overline{x}$ resolve o Problema (2.7).

Os algoritmos de separação para se obter desigualdades válidas SECs e DBs violadas são apresentados a seguir.

3.1.1.1 Algoritmos de Separação

Para decidir se $\overline{x} \in P_2$, devemos verificar se $\overline{x}$ satisfaz todas as desigualdades SECs (2.3) e DBs (2.6). Para separar o primeiro conjunto de desigualdades, utilizamos o método introduzido por Padberg & Wolsey [1983], com complexidade $O(n^4)$, descrito a seguir.

Definimos $D^* = (V^*, A^*)$ como sendo uma rede capacitada estabelecida a partir de $G = (V, E)$. Assuma $V^* = V$ e para cada $e = \{i, j\} \in E$, tal que $\overline{x}_e > 0$, cria-se dois arcos (i, j) e (j, i) que são adicionados à A^* com capacidade $h_{i,j} = h_{j,i} = \frac{1}{2}\overline{x}_e$. Assuma que $b_i = \overline{x}(\delta(i))$, $\forall i \in V$. Dois vértices auxiliares, rotulados por 0 e $n+1$, são adicionados à V^*, e novos arcos $(0, i)$ e $(i, n+1)$ são incluídos em A^* com capacidades $h_{0,i} = \max\{\frac{1}{2}b_i - 1,\, 0\}$ e $h_{i,n+1} = \max\{1 - \frac{1}{2}b_i,\, 0\}$, para todo $i \in V^*$.

Para determinar se alguma SEC (2.3) é violada por $\overline{x}$, Padberg & Wolsey [1983] mostraram que é necessário resolver $(n-2)$ problemas de fluxo máximo na rede dada por D^*. No entanto, para cada problema, realizamos pequenas modificações em D^*. Considerando o k-ésimo problema, para $k = 1, ..., n-2$, alteramos a capacidade $h_{0,k} = +\infty$ e, se $k \geq 2$, definimos $h_{i,n+1} = +\infty$, para $i = 1, ..., k-1$. Assim, evitamos gerar o mesmo subconjunto S nos diferentes problemas de fluxo máximo. No k-ésimo problema, obtemos, então, o corte mínimo (fluxo máximo) que separa 0 e $n+1$ em D^*, representado por $(S^k \cup \{0\}, V^* \setminus (S^k \cup \{0\}))$. Se $|S^k| - \overline{x}(E(S^k)) < 1$, então a SEC dada por $x(E(S^k)) \leq |S^k| - 1$ é violada pela solução $\overline{x}$.

As desigualdades DBs (2.6) podem ser separadas de forma exata através do algoritmo proposto por Letchford et al. [2004], cuja complexidade é $O(n^4)$. Tal algoritmo

consiste em determinar uma *cut-tree* [Gomory & Hu, 1961] em uma rede cujo número de vértices é $O(n)$. No entanto, a descrição deste algoritmo será omitida nesta seção, uma vez que, neste trabalho, DBs são separadas somente de forma heurística.

3.1.2 Relaxação Lagrangeana

Como visto no Capítulo 2, a maioria dos trabalhos que lidam com o PAGMRG de forma exata utilizam uma Relaxação Lagrangeana para a obtenção de limites duais. A Relaxação Lagrangeana consiste na decomposição de uma formulação para um determinado problema, de forma a explorar alguma estrutura *interessante* do problema em questão. Para isso, identificamos nessa formulação um conjunto de desigualdades *difíceis*, relaxamos tais desigualdades, e adicionamos à função objetivo o produto de cada uma delas por um *multiplicador de Lagrange* conveniente.

Podemos avaliar PL(P_2) através de uma Relaxação Lagrangeana definida, por exemplo, através da dualização das restrições (2.4) e (2.6). Assuma $\rho \in \mathbb{R}^n_-$ e $\lambda \in \mathbb{R}^q_-$, no qual q denota o número de restrições (2.6) em P_2, como sendo multiplicadores de Lagrange associados, respectivamente, às desigualdades (2.4) e (2.6). Limites inferiores válidos para a Formulação (2.7) podem ser obtidos através da resolução do Problema Lagrangeano L(ρ, λ), dado por:

$$\begin{aligned} \mathrm{L}(\rho,\lambda) = \min\Bigg\{ & \sum_{e\in E} c_e x_e + \sum_{i\in V} \rho_i (d_i - x(\delta(i))) \\ & + \sum_{H\subseteq V} \sum_{T\subseteq\delta(H)} \lambda_{H,T} \left(\left\lfloor \frac{1}{2}(d(H) + |T|) \right\rfloor - (x(H) + x(T)) \right) \\ & : x \text{ é uma AGM de } G \Bigg\}. \end{aligned} \tag{3.1}$$

O melhor limite inferior oferecido por (3.1) é obtido ao se resolver o problema Dual Lagrangeano definido por:

$$\mathrm{DL} = \max\{\mathrm{L}(\rho,\lambda) : \rho \leq 0, \lambda \leq 0\}. \tag{3.2}$$

Observe que, como q é uma função exponencial de $|V|$, o número de variáveis $\lambda_{H,T}$ do problema (3.2) também é exponencial. Essa característica faz com que o uso do Método de Subgradiente [Held & Karp, 1971], sem ser modificado para resolver DL, se torna pouco efetivo, pois, com o aumento do número de desigualdades dualizadas violadas, o passo no método se torna muito pequeno. Por este motivo, para encontrar

(ou aproximar) a solução ótima para o problema (3.2), Cunha & Lucena [2007] propuseram o uso de um algoritmo NDRC [Lucena, 2005]. Algoritmos do tipo *Relax-and-cut* [Lucena, 1992, 2005] são métodos que, por utilizar um esquema de dualização dinâmica, são capazes de lidar com um número exponencialmente grande de desigualdades candidatas à dualização Lagrangeana.

No algoritmo NDRC proposto em Cunha & Lucena [2007], definimos, inicialmente um subproblema Lagrangeano L', no qual as restrições de *d-Emparelhamento* (2.4) são dualizadas (associadas ao vetor de multiplicadores ρ) e as DBs (2.6) são relaxadas. Um problema Dual Lagrangeano DL' é, então, formulado. Utilizamos um Método de Subgradiente [Held & Karp, 1971] para aproximar DL'. Ao longo da execução do Método de Subgradiente, assim como no método de Planos de Corte, utilizamos um algoritmo de separação para tentar encontrar DBs violadas pela solução do subproblema L'. O algoritmo de separação, por separar um vetor $\overline{x} \in \mathbb{B}^m$, é mais simples e, diferentemente do algoritmo proposto por Letchford et al. [2004], é capaz de lidar com vetores que violam as restrições (2.4). Algumas (poucas) DBs violadas, se encontradas, são dualizadas à função objetivo de L', dando origem a um novo subproblema Lagrangeano L' e a um novo problema Dual Lagrangeano DL' associado. Esse procedimento se repete até que um critério de parada seja alcançado. A utilização de um Método de Subgradiente para resolver os problemas DL' ao longo da execução do *Relax-and-cut* se torna possível devido ao fato de que o número de DBs dualizadas dinamicamente ao longo do método é bastante inferior ao número total de DBs.

3.1.3 Geração de Colunas

A GC surgiu no contexto da Programação Linear através da Decomposição de Dantzig-Wolfe [Dantzig & Wolfe, 1960], utilizada para solucionar Programas Lineares de grande porte e/ou Programas Lineares com uma estrutura especial. Isso é feito através de uma reformulação do programa original, no qual este é descrito em termos dos raios extremos e vértices de um subproblema identificado na formulação original.

A Reformulação de Dantzig-Wolfe também pode ser aplicada em Programação Inteira. Os primeiros a utilizarem GC em Programação Inteira foram Gilmore & Gomory [1961, 1963], para o Problema de Padrões de Corte. Naquela referência, GC foi utilizada para obter a Relaxação Linear de uma reformulação para o problema, cujos subproblemas consistem em resolver o Problema da Mochila. Outros trabalhos também utilizaram GC para resolver problemas formulados por Programas Inteiros, por exemplo, Desrosiers et al. [1984] para um problema de Roteamento de Veículos, Desrochers & Soumis [1989] em um problema de Alocação de Tripulação, e de Carvalho

[2002] para o problema de Padrões de Corte. Detalhes sobre algoritmos de GC em Programação Inteira podem ser obtidos em Desaulniers et al. [2005].

A justificativa para empregarmos uma GC à Formulação P_2 é a seguinte. O número m de variáveis da Formulação P_2 é proporcional à $O(n^2)$. Para certos valores de n, mesmo uma formulação que envolve $O(n^2)$ variáveis pode ser tal que a avaliação de seus limites de Programação Linear seja computacionalmente cara.

Nesta dissertação, utilizamos GC combinada com algoritmos de Planos de Corte para avaliar PL(P_2). No entanto, o uso de GC difere da maioria dos trabalhos da literatura, uma vez que não aplicamos uma Reformulação de Dantzig-Wolfe à P_2. Além disso, diferentemente do que ocorre em muitos dos métodos de GC para Programas Inteiros, o cálculo do custo reduzido envolve a resolução de um Problema de Otimização que pode ser resolvido por inspeção. O método aqui proposto aplica GC diretamente à Formulação P_2. Sendo assim, considere o Problema Linear Mestre (PLM) dado por:

$$\min \left\{ \sum_{e \in E} c_e x_e : x \in P_2 \right\}. \tag{3.3}$$

Assuma que $\sigma \in \mathbb{R}$, $\gamma_S \in \mathbb{R}_-^{2^n-2}$, $\rho_i \in \mathbb{R}_-^n$ e $\lambda \in \mathbb{R}_-^q$ são variáveis duais associadas, respectivamente, às restrições (2.2)-(2.4) e (2.6). O programa dual do PLM é dado por:

$$\begin{aligned} \max \Bigg(& (|V|-1)\sigma + \sum_{S \subset V, S \neq \emptyset} (|S|-1)\gamma_S + \sum_{i \in V} d_i \rho_i \\ & + \sum_{H \subseteq V} \sum_{T \in \delta(H)} \left(\left\lfloor \frac{1}{2}(d(H) + |T|) \right\rfloor \right) \lambda_{H,T} \Bigg) \end{aligned} \tag{3.4}$$

Sujeito à:

$$c_e - \sigma - \sum_{S \subset V, S \neq \emptyset | e \in E(S)} \gamma_S - \sum_{i \in V | e \in \delta(i)} \rho_i - \sum_{H \subseteq V} \sum_{\substack{T \subseteq \delta(H) | \\ e \in E(H) \text{ ou } e \in T}} \lambda_{H,T} \geq 0, \forall e \in E, \tag{3.5}$$

$$\sigma \in \mathbb{R}, \tag{3.6}$$

$$\gamma_S \leq 0, \forall S \subset V, S \neq \emptyset, \tag{3.7}$$

$$\rho_i \leq 0, \forall i \in V, \tag{3.8}$$

$$\lambda_{H,T} \leq 0, \forall H \subseteq V, \forall T \subseteq \delta(H). \tag{3.9}$$

Se ao invés de explicitamente considerarmos todo o conjunto de arestas E do grafo G que define o problema no PLM, considerarmos somente um conjunto E', $E' \subset E$, tal que $|E'| \ll |E|$, obtemos, então, o Problema Linear Mestre Restrito (PLMR), dado por

$$\min \left\{ \sum_{e \in E'} c_e x_e : x \in P_2' \right\}, \tag{3.10}$$

onde P_2' é definido por

$$P_2' = \{x \in \mathbb{R}^{|E'|} : x \text{ satisfaz (2.2)-(2.6)}\}. \tag{3.11}$$

Seja $\overline{x}$ uma solução ótima para o PLMR. A fim de fornecer um certificado de que $\overline{x}$ resolve o PLM, é preciso garantir que a solução dual associada, representada pelas variáveis duais $\overline{\sigma}$, $\overline{\gamma}$, $\overline{\rho}$ e $\overline{\lambda}$, satisfaz todas as restrições (3.5). Para isso, resolve-se o problema de precificação, definido como:

$$e^* \in \underset{e \in E}{arg\ min}\ \omega(e), \tag{3.12}$$

no qual $\omega(e)$, denominado *custo reduzido* de e, é definido por:

$$\omega(e) = (c_e - \overline{\sigma} - \sum_{S \subset V, S \neq \emptyset | e \in E(S)} \overline{\gamma}_S - \sum_{i \in V | e \in \delta(i)} \overline{\rho}_i - \sum_{H \subseteq V} \sum_{\substack{T \subseteq \delta(H) | \\ e \in E(H) \text{ ou } e \in T}} \overline{\lambda}_{H,T}). \tag{3.13}$$

Se $\omega(e^*) \geq 0$, a solução $\overline{x}$ também é solução ótima para o PLM. Caso contrário, foi encontrada uma aresta à qual associa-se uma restrição dual (3.5) violada pela solução dual $\overline{\sigma}$, $\overline{\gamma}$, $\overline{\rho}$ e $\overline{\lambda}$. Assim, como o programa dual (3.4) do PLM é inviável, não podemos caracterizar a otimalidade de $\overline{x}$ em relação à Relaxação Linear do PLM. Portanto, a aresta e^* deve ser incluída no conjunto E', e o problema PLMR é redefinido e reotimizado.

Um resultado importante a ser comentado é que, devido ao fato de o subproblema Lagrangeano (3.1) satisfazer a propriedade de integralidade [Geoffrion, 1974], teoricamente, o mesmo limite PL(P_2) é obtido seja qual for a estratégia escolhida dentre as descritas nesta e nas seções anteriores.

3.1.4 Métodos Híbridos

A maioria dos algoritmos propostos para o PAGMRG utilizou combinações de algumas das abordagens definidas nas seções anteriores em conjunto com um método de enumeração para tentar resolver o PAGMRG. No entanto, o único método que transfere informações entre os procedimentos foi proposto em Lucena et al. [2011]. Naquela referência, os autores propuseram, para resolver a Relaxação Linear do modelo (2.7), um algoritmo *Relax-and-cut* como preprocessamento para um BC. O método *Relax-and-cut*, além de aproximar o limite PL(P_2), foi capaz de fixar arestas subótimas do grafo de definição do problema. Além disso, o método oferece ao BC um conjunto de desigualdades válidas que, se incorporadas à uma Relaxação Linear para o problema, fornece um limite de programação linear pelo menos tão bom quanto o melhor limite Lagrangeano obtido pelo algoritmo *Relax-and-cut*.

Nesta dissertação, propomos três algoritmos híbridos que utilizam os métodos detalhados nas Seções 3.1.1, 3.1.2 e 3.1.3. Por exemplo, no primeiro algoritmo sugerido neste trabalho, GC e Planos de Corte são combinados para resolver PL(P_2). Durante o processo de obtenção do limite, tentamos obter soluções viáveis de boa qualidade utilizando uma solução do problema Lagrangeano (3.1), formulado e resolvido em termos das variáveis duais associadas às restrições dos PLMRs. Ao assim procedermos, estamos implicitamente utilizando uma heurística Lagrangeana, similar à proposta em Andrade et al. [2006] e Cunha & Lucena [2007], sem a utilização do Método do Subgradiente para determinar os multiplicadores. A solução de (3.1) também é utilizada na tentativa de fixar variáveis garantidamente subótimas. Esse algoritmo, bem como os outros propostos neste trabalho, serão detalhados a seguir.

3.2 Geração de Colunas para avaliar PL(P_2) na raiz de um BC

Nesta seção iremos detalhar o método híbrido que utiliza GC e BC para resolver o PAGMRG, denotado por GCBC. Esse algoritmo consiste de duas fases. Na primeira fase, utilizamos GC e um algoritmo de planos de corte que separa (2.3) e (2.6) para obter o limite de Relaxação Linear de P_2. Em meio ao processo de geração de colunas, utilizamos heurísticas para obtenção de um limite superior válido e tentamos fixar arestas subótimas. Na segunda etapa, todas as arestas que não foram fixadas são consideradas em um método BC para tentar resolver o problema.

O algoritmo baseado em GC utilizado neste e em todos os outros métodos propos-

tos nesta dissertação é detalhado nas próximas subseções. Por fim, na última subseção, os detalhes de implementação do BC são dados.

3.2.1 Fase 1: Algoritmo baseado em GC para avaliar PL(P_2)

Inicialmente, relaxamos as restrições SECs (2.3) e DBs (2.4) do poliedro (3.11). A fase inicial, para obtermos PL(P_2) utilizando GC, é composta pelos seguintes passos:

1. Escolha um conjunto inicial E' de arestas, de forma que seja possível encontrar uma solução básica viável para o PLMR, definido, inicialmente, a partir das restrições (2.2), (2.4) e (2.5).
2. Resolva o PLMR, e assuma que sua solução seja $\overline{x} \in [0,1]^m$.
3. Através de um algoritmo de separação de SECs e DBs, tente encontrar desigualdades violadas por $\overline{x}$. Caso sejam encontradas, adicione-as ao PLMR e retorne ao passo 2.
4. Através de heurísticas que utilizam informações (primais e duais) do PLMR, tente encontrar limites superiores válidos para o PAGMRG.
5. Tente fixar variáveis utilizando o limite oferecido pela Relaxação Lagrangeana L($\overline{\rho}, \overline{\lambda}$), no qual $\overline{\rho}$ e $\overline{\lambda}$ são as variáveis duais associadas, respectivamente, às restrições (2.4) e (2.6) do PLMR, em conjunto com o melhor limite superior até então obtido.
6. Verifique a otimalidade de $\overline{x}$ em relação ao PLM, isto é, resolva o problema de precificação (3.12) para gerar novas colunas a serem inseridas no modelo. Caso haja alguma coluna com custo reduzido negativo, adicione-a ao PLMR e retorne ao passo 2. Caso contrário, $\overline{x}$ é a solução ótima do PLM.
7. Para a solução ótima $\overline{x}$ do PLM, se $\overline{x}_e \notin \mathbb{B}$, para alguma aresta $e \in E$, tente fixar mais variáveis utilizando o custo reduzido de programação linear.

O problema de precificação é solucionado através do problema (3.12), que é resolvido verificando-se $\omega(e)$, para todo $e \in E \setminus E'$. Denote $\mathrm{C} = \{e \in E \setminus E' : \omega(e) < 0\}$ o conjunto de arestas com custo reduzido negativo em uma dada iteração do método GC. Ao invés de adicionarmos ao modelo somente a aresta de menor custo reduzido em C, adicionamos uma porcentagem ϵ de |C|. Essa abordagem tem como foco a redução do número de iterações da GC, uma vez que as arestas contidas em C provavelmente seriam adicionadas em iterações futuras, caso somente e^* fosse inserida no PLMR. Em

nossos testes computacionais, o melhor valor encontrado para ϵ de forma a não tornar a resolução da Relaxação Linear muito lenta foi $\epsilon = 10\%$.

A seguir, serão detalhados os métodos para a obtenção das colunas iniciais, o modo como separamos desigualdades SECs e DBs, as heurísticas para obtenção de limites superiores válidos e os esquemas de fixação de variáveis.

3.2.1.1 Gerando as colunas iniciais

No método de GC é necessário, inicialmente, determinar um conjunto inicial de arestas para E'. Essas arestas devem ser capazes de fornecer uma base para o primeiro PLMR. Nesta dissertação, consideramos $E' = T_1 \cup T_2 \cup \bigcup_{i \in V} J_i$ onde:

- T_1 é o conjunto de arestas em uma solução viável para o PAGMRG obtida por meio da heurística descrita na Seção 3.2.1.4. Para obter T_1, a heurística emprega os custos originais $\{c_e : e \in E\}$ das arestas de G e, caso não seja um grafo completo, adicione arestas artificiais com custo $+\infty$ para completá-lo. Dessa forma, garantimos que exista uma solução básica viável no início.

- T_2 é o conjunto das arestas na AGM de G obtida pelo Algoritmo de Kruskal [Kruskal, 1956].

- $J_i \subseteq \delta(i)$ é conjunto com até três arestas de menor custo em $\delta(i)$, $i \in V$, tal que $J_i \cap (T_1 \cup T_2) = \emptyset$.

O conjunto E', da forma como foi determinado, é capaz de gerar uma base para o poliedro que define o primeiro PLMR.

Proposição 1. *É possível obter uma base viável através das colunas de E'.*

Demonstração. Considere o vetor solução $\phi \in \mathbb{B}^m$ tal que $\phi_e = 1$ se $e \in E' \cap T_1$ e $\phi_e = 0$, caso contrário. Observe que ϕ satisfaz o sistema linear formado por (2.2), (2.4) e (2.5). Logo, ϕ é uma solução viável para P_2', $P_2' \neq \emptyset$, P_2' possui pontos extremos e P_2' não possui reta. □

3.2.1.2 Separação de SECs

O algoritmo para separar SECs de forma exata [Padberg & Wolsey, 1983] é computacionalmente caro ($O(n^4)$). Por isso, tentamos, inicialmente, encontrar SECs violadas através de um método heurístico, descrito a seguir.

A heurística aqui utilizada tem como base o Algoritmo de Kruskal [Kruskal, 1956] para encontrar uma AGM de um grafo G. Considere $\overline{x}$ a solução do PLMR a ser separada. O método funciona conforme descrito no Algoritmo 1.

Algoritmo 1: Heurística para obteção de SECs violadas

```
1 T' ← ∅; S ← ∅;
2 Ordene as arestas de E' de forma não crescente de {x̄_e : e ∈ E'} ;
3 enquanto (V, T') não induzir uma árvore geradora de (V, E') faça
4 |   Selecione a primeira aresta e' ∈ E' ainda não avaliada cuja inserção em T'
  |     não resulte em ciclo ;
5 |   T' ← T' ∪ {e'} ;
6 |   Defina S como sendo o conjunto de vértices da componente conexa
  |     resultante da inclusão de e' em T'. Se x̄ violar a SEC associada à S, faça
  |     S ← S ∪ {S}.
7 fim enquanto
8 retorna S
```

Se nenhuma desigualdade SEC violada for encontrada pela heurística, executamos o método exato descrito na Seção 3.1.1.1.

Ao final da execução do método exato, caso algum corte violado tenha sido encontrado, executamos um procedimento cujo objetivo é encontrar desigualdades violadas ortogonais à mais violada. Essa abordagem foi utilizada em Lucena & Resende [2004] e foi inspirada nos chamados *nested cuts* propostos por Koch & Martin [1998] para o Problema de Steiner. Para ilustrar a ideia do procedimento, assuma que k denote o índice correspondente ao corte mais violado obtido pelo método de Padberg & Wolsey [1983] e inicialize $\hat{x} = \overline{x}$. Para todo $e \in E$, se $\overline{x}_e > 0$ e $e \in E(S^k)$, faça $\hat{x}_e = 0$. O algoritmo de separação exata é executado novamente considerando $\hat{x}$ como o ponto a ser separado. É fácil perceber que os cortes violados encontrados utilizando $\hat{x}$ são válidos para o modelo. Além disso, tais cortes são ortogonais àqueles obtidos utilizando $\overline{x}$.

Com o objetivo de controlar o tamanho do PLMR e retardar o efeito de *tailing off* [Padberg & Rinaldi, 1991], adotamos uma estratégia, também utilizada por Valle et al. [2011] e Martinez & Cunha [2014], para a inclusão de desigualdades SECs violadas obtidas pelos algoritmos acima. Após a execução dos métodos de separação, os cortes são normalizados (utilizando a norma Euclidiana) e apenas a desigualdade mais violada é incluída no PLMR. Os cortes restantes são adicionados ao PLMR se forem suficientemente ortogonais ao corte mais violado. Consideramos que dois cortes normalizados são suficientemente ortogonais se o produto interno entre eles é menor que um parâmetro τ. Em nossos testes, adotamos $\tau = 0{,}2$.

As duas heurísticas utilizadas para encontrar desigualdades SECs violadas foram capazes de reduzir consideravelmente o tempo de execução do método para computar PL(P_2).

3.2.1.3 Separação de DBs

Ao invés de utilizar o algoritmo exato proposto por Letchford et al. [2004] para separar DBs, fizemos o uso de um método heurístico. A heurística aqui apresentada, a mesma utilizada em Lucena et al. [2011], consegue encontrar DBs violadas de forma bastante satisfatória, a um custo computacional relativamente baixo ($O(n^2)$).

Para um dado par H e T, uma desigualdade Blossom pode ser reescrita na forma

$$x(\delta(H) \setminus T) + \sum_{e \in T \subseteq \delta(H)} (1 - x_e) + \sum_{i \in H} (d_i - x(\delta(i))) \geq 1. \tag{3.14}$$

Considere $\overline{x}$ o ponto a ser separado, tal que $\overline{x}(\delta(i)) \leq d_i$, para todo $i \in V$. Assuma $\overline{G} = (V, \overline{E})$ como sendo o grafo de suporte de $\overline{x}$, onde $\overline{E} = \{e \in E' : \overline{x}_e \in (0, 1)\}$. Na heurística de separação, todas as arestas com $\overline{x}_e = 1$ são eliminadas de $\overline{G}$. A heurística consiste em avaliar cada componente conexa no grafo $\overline{G}$ resultante. Assuma que $\overline{G}$ tenha $p > 1$ componentes conexas, representados por $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$, para todo $i = 1, ..., p$. Se a folga $\sum_{j \in \overline{V}_i} (d_j - \overline{x}(\delta(j)))$, associada à $\overline{G}_i$, é menor que 1 e se $\overline{G}_i$ é ímpar, ou seja, $\sum_{j \in \overline{V}_i} (d_j - |\{e \in \delta(j) : \overline{x}_e = 1\}|)$ é ímpar, então, se definirmos $H = \overline{V}_i$ e $T = \{e \in \delta(H) : \overline{x}_e = 1\}$, obtemos uma DB garantidamente violada. O pseudocódigo deste método é apresentado no Algoritmo 2.

3.2.1.4 Limites superiores

Para se obter limites superiores válidos para o PAGMRG, utilizamos uma heurística que se beneficia de informações duais fornecidas pela solução corrente, logo após o Passo 3 de GC. A heurística aqui utilizada possui duas fases: primeiramente, uma solução viável é obtida através de um método construtivo baseado em uma modificação do algoritmo de Kruskal [Kruskal, 1956]. Na segunda etapa, a solução obtida pelo algoritmo construtivo é, se viável, submetida à uma busca local.

O método para se obter uma solução inicial inclui o mecanismo de *prevenção de inviabilidades* introduzido por Andrade et al. [2006]. Inicialmente, ordena-se as arestas de E em ordem não decrescente de custo. A cada iteração, uma aresta candidata $e = \{i, j\}$ só é aceita para integrar $F \subseteq E$ se as três condições a seguir forem satisfeitas:

- $F \cup \{i, j\}$ não forma ciclo;

Algoritmo 2: Heurística para separação de DBs violadas

Obtenha $\overline{G} = (V, \overline{E})$, onde $\overline{E} = \{e \in E' : \overline{x}_e \in (0, 1)\}$;
$\mathcal{B} \leftarrow \emptyset$;
para cada *componente conexa* $\overline{G_i} = (\overline{V}_i, \overline{E}_i)$ *de* $\overline{G}$ **faça**
  para todo $j \in \overline{V}_i$ **faça**
    $d'_j = |\{e \in \delta(j) : \overline{x}_e = 1\}|$;
  fim para todo
  $f = \sum_{j \in \overline{V}_i} (d_j - \overline{x}(\delta(j)))$;
  $b = \sum_{j \in \overline{V}_i} (d_j - d'_j)$;
  se *(*$f < 0$*) e (b é ímpar)* **então**
    $H \leftarrow \overline{V}_i$;
    $T \leftarrow \{e \in \delta(H) : \overline{x}_e = 1\}$;
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(H, T)\}$;
  fim se
fim para cada
retorna $\mathcal{B}$

- $|\delta(i) \cap F| < d_i$ e $|\delta(j) \cap F| < d_j$;
- se $|F| \neq (|V| - 2)$, $|\delta(i) \cap F| + |\delta(j) \cap F| > 2$.

A primeira condição garante que a inserção de uma aresta não forme um ciclo em F. A segunda condição garante que, na iteração do método de Kruskal modificado onde $e = \{i, j\}$ for inserida, a inclusão de e em F não violará a restrição de grau dos vértices i e j. O mecanismo de prevenção de inviabilidades (terceira condição), tenta assegurar a capacidade de conexão futura da componente conexa de F ao qual a aresta $\{i, j\}$, se adicionada, fará parte.

Para obter uma solução primal de boa qualidade, duas versões do algoritmo descrito acima foram implementadas. Seja $\overline{x}$ a solução corrente do PLMR após o Passo 3 e considere $\overline{\rho}$ e $\overline{\lambda}$ o vetor de variáveis duais (não necessariamente viáveis para o dual da Relaxação Linear do PLM) associadas às restrições (2.4) e (2.6) do PLMR nessa mesma etapa. A fase construtiva do primeiro algoritmo, opera sobre os *custos complementares* $\{c_e(1 - \overline{x}_e) : e \in E\}$ para obter uma árvore geradora válida para o PAGMRG. Já na segunda abordagem, utilizamos os custos complementares com base na solução do problema Lagrangeano L($\overline{\rho}$,$\overline{\lambda}$).

A etapa final da heurística consiste em refinar as soluções obtidas pelos métodos construtivos, através de uma Busca Local (BL). Na BL, os custos das arestas são restaurados ao seu valor original. Considere $T = (V, E_T)$ o subgrafo de G associado à solução obtida por uma das abordagens construtivas descritas acima. A ideia da

BL é avaliar as soluções pertencentes à *1-vizinhança* de T. Uma árvore geradora de G pertence à 1-vizinhança de T se difere de T por somente duas arestas. Dentre as árvores vizinhas, consideramos aquelas que também respeitam o grau máximo nos vértices. Assim, se uma árvore geradora viável com custo inferior ao da árvore representada por T for encontrada na vizinhança, T é substituído pelo subgrafo associado à nova árvore e o método de BL na 1-vizinhança se repete. Caso contrário, a BL é encerrada.

Uma decisão importante na implementação de uma BL é se devemos aceitar a primeira solução viável com custo inferior ao da árvore atual T (estratégia *first improvement*) ou se devemos escolher a melhor solução viável de toda 1-vizinhança de T (estratégia *best improvement*). Assim como em Cunha & Lucena [2007], a estratégia *first improvement* forneceu soluções tão boas quanto as obtidas pela estratégia *best improvement*, demandando menos tempo de CPU.

Com base em Cunha & Lucena [2007], alguns testes de dominância foram implementados com o intuito de reduzir o tempo computacional gasto pela BL. Em uma dada execução de BL, considerando a solução inicial T, inicialmente calculamos os seguintes parâmetros:

- $p(r, i)$: o custo da aresta de maior peso no caminho único entre i e um vértice raiz r previamente definido, para todo $i \in V$. Por definição, $p(r, r) = 0$.
- $a(i)$: o custo da aresta mais cara incidente à i em T, para todo $i \in V$.
- c^*: o custo da aresta mais cara em T, isto é, $c^* = \max\{c_e : e \in E_T\}$.

As arestas $\{e \in E : c_e \geq c^*\}$ podem ser descartadas da BL, uma vez que a inclusão dessas arestas, na vizinhança aqui considerada, é incapaz de reduzir o custo da árvore representada por T. Assuma $e = \{i, j\}$ a aresta candidata a reduzir o custo da solução corrente T. Se as extremidades de e não estão saturadas, ou seja, $|\delta(i) \cap E_T| < b_i$ e $|\delta(j) \cap E_T| < b_j$, qualquer aresta no caminho entre i e j pode ser eliminada a fim de produzir uma árvore viável. No entanto, podemos verificar de antemão se $c_{\{i,j\}} < p(i, r)$ ou $c_{\{i,j\}} < p(j, r)$. Caso nenhuma das condições forem satisfeitas, então a aresta de maior custo entre i e j é inferior à $c_{\{i,j\}}$.

Agora suponha que uma das extremidades de $e = \{i, j\}$, por exemplo, a extremidade j, esteja saturada e a outra não, ou seja, $|\delta(j) \cap E_T| = b_j$ e $|\delta(i) \cap E_T| < b_i$. Nesse caso, a aresta e só melhora o custo de T se $c_{\{i,j\}} < a(j)$.

É importante observar que, sempre que a solução T for modificada através da inclusão de uma aresta, os parâmetros $p(r, i)$, $a(i)$ e c^* utilizados nos testes de dominância devem ser recalculados. Isso pode ser feito de forma eficiente através da estrutura de

dados introduzida por Barr et al. [1979]. Nesta dissertação, o vértice raiz usado para inicializar essa estrutura de dados foi escolhido de forma aleatória.

3.2.1.5 Testes de Fixação de Variáveis

Ao executar um algoritmo para resolver uma instância do PAGMRG podemos, a princípio, se $|V| > 2$, eliminar do problema as arestas $\{i, j\} \in E$ caso $d_i = d_j = 1$, já que não há nenhuma solução válida para o PAGMRG que as contenham.

Além do preprocessamento descrito acima, nesta dissertação utilizamos dois métodos para eliminar arestas *subótimas* do grafo de definição do problema. Chamamos de *subótima* uma aresta que garantidamente não pertence a nenhuma solução ótima do PAGMRG. O primeiro algoritmo tem como base a árvore geradora $\overline{T} = (V, \overline{E})$ obtida ao se resolver o problema Lagrangeano $\mathrm{L}(\overline{\rho},\overline{\lambda})$, onde $\overline{\rho}$ e $\overline{\lambda}$ são as variáveis duais associadas, respectivamente, às restrições (2.4) e (2.6) na solução dual do PLMR após o Passo 3. Utilizando o limite inferior $\mathrm{L}(\overline{\rho},\overline{\lambda})$ em conjunto com um limite superior válido ub para o PAGMRG, podemos tentar caracterizar as arestas como pertences ou não à uma solução ótima do problema, através do *custo reduzido* (CR). O CR associado à uma aresta $e \in E$, em relação à $\overline{T}$, é dado pela diferença entre o custo de e e o custo da aresta mais cara no único ciclo de $(V, \overline{E} \cup \{e\})$. Se o valor de CR associado à uma aresta e for superior à $ub - \mathrm{L}(\overline{\rho}, \overline{\lambda})$, temos que e é uma aresta garantidamente subótima e pode ser excluída do problema. Se durante o processo de fixação de variáveis todas as arestas incidentes à um vértice $i \in V$, com exceção de uma, forem excluídas de G, o vértice i e sua única aresta incidente também podem ser removidos do grafo, já que esta aresta necessariamente deve pertencer à solução ótima.

Nesta dissertação, o procedimento descrito acima só é executado quando o *gap de dualidade*, dado por $\frac{ub-\mathrm{L}(\overline{\rho},\overline{\lambda})}{ub}$, for suficientemente pequeno. Em nossos testes, os melhores resultados foram obtidos quando executamos os testes de fixação de variáveis após obter gaps inferiores à 5%.

Ao final do cálculo de $\mathrm{LP}(P_2)$, realizamos um segundo procedimento para fixação de variáveis. Como $\mathrm{LP}(P_2)$ é um limitante inferior válido para o problema, também podemos utilizá-lo em conjunto com um limite superior na tentativa de eliminar arestas subótimas de G. Isto é feito tomando o custo reduzido de programação linear $\omega(e)$ ao final da última iteração da GC. Se $\omega(e)$, para $e \in E$, for maior do que $ub - \mathrm{LP}(P_2)$, a aresta e é subótima e pode ser removida de G.

3.2.1.6 Detalhes de implementação

Para resolver os Programas Lineares do Passo 2, foi utilizado o software ILOG CPLEX 12.5.

Os problemas de fluxo máximo necessários para a separação de SECs, foram resolvidos utilizando o Algoritmo de Dinic, cuja complexidade é $O(n^2m)$ [Ahuja et al., 1993].

Ao se trabalhar com algoritmos de Planos de Cortes são necessários cuidados na manutenção da matriz de desigualdades do modelo. Como o número de desigualdades SECs geradas pelo método de Planos de Corte é muito grande, convém adotar um parâmetro MAX_ITER de iterações subsequentes na qual uma SEC *ineficiente* permanece no modelo. Consideramos uma SEC ineficiente se a variável de folga associada é positiva e não básica. Caso uma SEC permaneça ineficiente por mais do que MAX_ITER iterações, ela é removida do modelo.

Após testes em algumas instâncias do PAGMRG, adotamos o valor MAX_ITER = 30.

3.2.2 Fase 2: Branch-and-cut

O BC discutido nesta seção utiliza como grafo de entrada a instância preprocessada pelo algoritmo da Seção 3.2.1. Além disso, o método é inicializado com a base que descreve a solução ótima da Relaxação Linear de P_2. As desigualdades válidas (SECs e DBs) associadas a essa base também são incluídas no BC. O melhor limite superior encontrado nas iterações do método da Fase 1 é utilizado como parâmetro de *cut off* do BC.

Essencialmente, o tratamento do nó raiz diferencia o BC aqui implementado daquele proposto em Lucena et al. [2011]. Ao contrário do que ocorre ao longo da árvore de enumeração, no nó raiz as variáveis de decisão são implicitamente consideradas. Na etapa de enumeração que ocorre após o cálculo de LP(P_2), todas as colunas (arestas) que não foram fixadas são incluídas explicitamente no modelo.

Diferentemente do nó raiz, nos demais nós as desigualdades SECs violadas são identificadas apenas pelo método exato de Padberg & Wolsey [1983], descrito na Seção 3.2.1.2. Não utilizamos a heurística baseada no método de Kruskal para encontrar SECs violadas pelo fato desta não apresentar os mesmos ganhos computacionais observados no nó raiz, devido ao baixo número de desigualdades válidas encontradas por tal heurística nesta etapa. As DBs são separadas conforme descrito anteriormente.

Assim como em Lucena et al. [2011], ao longo da árvore de enumeração, tenta-se obter melhores limites superiores através da heurística detalhada na Seção 3.2.1.4,

utilizando, na fase construtiva do algoritmo, os custos complementares $\{c_e(1 - \overline{x}_e) : e \in E\}$, onde $\overline{x}$ é a solução da Relaxação Linear ao final da resolução do nó. Esta é uma outra diferença com relação ao BC proposto por Lucena et al. [2011]. Naquele algoritmo a heurística é chamada após toda Relaxação Linear avaliada em cada nó.

O BC foi implementado utilizando a plataforma ILOG CPLEX 12.5 para gerenciar a árvore de enumeração. Todos os procedimentos de preprocessamento, heurísticas e desigualdades válidas fornecidas pelo CPLEX foram desativados. As rotinas de separação de SECs e DBs, bem como a rotina para obtenção de limites primais foram implementadas em *callbacks* fornecidas pela interface CONCERT.

A estratégia de seleção de nós da árvore de BB empregada foi a estratégia *melhor limite*, que explora o nó com menor valor de Relaxação Linear. Esta é a estratégia de seleção padrão do CPLEX. Outras estratégias de seleção foram testadas sem diferença significativa nos resultados. Assim como a forma de seleção de nós, utilizamos a regra de *branching* padrão do CPLEX, no caso, o próprio software decide como implementar o *branching*.

A opção de paralelismo do CPLEX não foi utilizada.

Foi imposto um limite de 4 horas para a solução do algoritmo híbrido GCBC. Assim, para a fase de enumeração do BC, foi empregado um limite de tempo $tl = 14400 - tgc$ segundos, onde tgc é o tempo de CPU (em segundos) gasto para resolver a Fase 1 do GCBC.

3.3 Um algoritmo *Branch-and-cut-and-price*

Um algoritmo BCP, como dito anteriormente, envolve a precificação de colunas e a identificação de planos de corte ao longo da árvore de busca do BB. Essa técnica foi empregada pela primeira vez em Nemhauser & Park [1991] para tratar o problema de Coloração de Arestas, e desde então vem sendo utilizada em diversos problemas de Otimização Combinatória [Fukasawa et al., 2006; Alves & de Carvalho, 2008; Uchoa et al., 2008].

No algoritmo BCP aqui proposto, em cada nó da árvore de enumeração, aplicamos o procedimento de GC e os métodos de separação de SECs (exato e heurísticos) e DBs descritos anteriormente. Dessa forma, a Relaxação Linear de cada nó da árvore de enumeração é obtida pelo algoritmo da Fase 1 do método descrito na seção anterior.

Nas próximas subseções, detalhamos características particulares do BCP proposto nesta dissertação. Sugestões de como realizar *branching*, como gerenciar o *cut-pool*, e como explorar a lista de candidatos são encontradas em Ladányi et al. [2001], referência

que serviu como base para a implementação do nosso BCP.

3.3.1 *Branching*

Ao final do processamento de um nó da árvore de enumeração, se não for possível podar o nó por inviabilidade ou por otimalidade, é necessário criar novos nós através de alguma regra de *branching*.

Em nossos algoritmos BCP, utilizamos uma abordagem de *branching* em variáveis. Assuma $\overline{x}$ como sendo a solução da Relaxação Linear de um determinado nó, obtida pelo método descrito na Seção 3.2.1, e considere $e \in E'$ a variável selecionada para o *branching*, onde E' é o conjunto de arestas explicitamente incluídas no modelo do nó em questão.

Ao fazer *branching* em variáveis, criamos dois novos nós: em um impomos $x_e = 0$ e no outro impomos $x_e = 1$. Devemos, então, definir qual aresta do conjunto $\mathrm{W} = \{e \in E' : \overline{x}_e \notin \mathbb{B}\}$ deve ser usada para definir o *branching*.

Uma técnica comumente utilizada na literatura consiste em escolher a aresta e cujo valor de Relaxação Linear mais se distancia da integralidade, no caso do PAGMRG, uma aresta com $\overline{x}_e$ próximo à 0,5.

Neste trabalho, entretanto, utilizamos a técnica conhecida como *strong branching* [Achterberg et al., 2005]. Essa técnica classifica cada aresta $e \in \mathrm{W}$ (ou um conjunto dessas arestas) candidata à realização do *branching* ao avaliar cada um dos novos nós obtidos caso selecionarmos a variável x_e para fazer *branching*. Assim, para cada aresta $e \in \mathrm{W}$, dois novos nós $S^0_{x_e}$ e $S^1_{x_e}$ são gerados, o primeiro impondo $x_e = 0$ e o segundo impondo $x_e = 1$. Avaliamos o valor dos programas lineares resultantes com as colunas e cortes identificados até então, em conjunto com os novos limites $x_e = 0$ ou $x_e = 1$ através do método Simplex. Optamos por não realizar *pricing* no *strong branching* para não elevar o tempo de CPU gasto pelo procedimento. O menor valor de função objetivo retornado pelos nós é, então, atribuído à l_e, onde l_e estima o limite inferior que será obtido ao selecionar a aresta e para fazer *branching*. Como estamos interessados em cada vez mais aproximar os limites inferiores e superiores, a aresta selecionada para o *branching* é a de maior valor de l_e.

A adoção de *strong branching* como regra de *branching* pode demandar muito tempo de processamento. Isso ocorre, principalmente, nos primeiros nós da árvore de BB, onde o número de variáveis fracionárias candidatas ao *branching* é, em geral, maior. Para reduzir o tempo gasto com a decisão de qual aresta empregar para o *branching*, adaptamos uma das estratégias sugeridas em Achterberg et al. [2005], dada a seguir. As arestas candidatas em W são analisadas na ordem em que foram precificadas. Se

não for obtida nenhuma nova *melhor aresta* candidata ao *branching* em κ iterações, o procedimento se encerra. Em nossos testes, adotamos $\kappa = 0{,}03|V|$. A redução no tempo de processamento ao se utilizar a abordagem descrita anteriormente foi muito significativa se comparada à adoção de uma estratégia de *strong branching completa*.

3.3.2 O gerenciamento do cut-pool

Nesta seção, detalhamos como os cortes gerados ao longo da árvore são gerenciados. A estratégia adotada neste trabalho é mais simples do que a sugerida por Ladányi et al. [2001].

Como dito, após os testes de fixação de variáveis da Seção 3.2.1.5, se somente uma aresta $e = \{i, j\}$ for a única incidente a um determinado vértice i, removemos o vértice i e a aresta e do grafo G e somamos à função objetivo o valor de c_e. Em consequência disso, excluímos do modelo a restrição (2.4) associada ao vértice i e reduzimos em uma unidade o valor de d_j.

As desigualdades (2.3) e (2.6) dos novos nós são herdadas do nó pai. Duas abordagens diferentes foram testadas:

- Os nós herdam todas as desigualdades remanescentes no *cut-pool* do nó pai. Essa abordagem preserva o contador que é comparado com o parâmetro MAX_ITER (definido na Seção 3.2.1.6) do nó pai para os novos subproblemas.

- Somente as restrições *justas* e as desigualdades cujas variáveis de folga são básicas no nó pai são passadas para os novos nós.

Em nossos testes, ao utilizarmos a segunda abordagem, obtivemos uma redução considerável nos tempos de processamento.

3.3.3 O gerenciamento da árvore de busca

Ao longo da execução do BB, é necessário decidir o próximo subproblema da lista de candidatos a ser processado. A escolha de uma estratégia implica diretamente no tamanho da árvore e na velocidade com que se encontra boas soluções viáveis.

Para tentar reduzir o tamanho da árvore de busca, devemos selecionar o nó candidato com o menor limite inferior associado (estratégia *melhor limite*). Dessa forma, a cada nó explorado, buscamos aumentar o melhor limite inferior global. Essa estratégia é a mesma utilizada pelo método BC descrito na Seção 3.2.2.

Em problemas nos quais uma solução viável é difícil de ser encontrada ou suspeita-se que a melhor solução é ruim, deve-se investir em outra forma de explorar a árvore.

Para obter boas soluções viáveis logo no início da enumeração, recomenda-se percorrer a árvore de enumeração através de uma *busca em profundidade*.

Ambas as abordagens foram testadas no nosso BCP. No entanto, através de testes computacionais, não ficou claro qual estratégia obteve o melhor desempenho. Para algumas instâncias a estratégia *melhor primeiro* obteve melhores tempos de processamento. Para outras, isso aconteceu com a *busca em profundidade*.

Os resultados computacionais apresentados no Capítulo 5 foram obtidos utilizando a estratégia *melhor limite*, também empregada no BC.

3.3.4 Detalhes de implementação

O processamento dos subproblemas é realizado pela Fase 1 (Seção 3.2.1) do algoritmo GCBC, a menos de um detalhe. Excetuando-se o nó raiz, o conjunto inicial de colunas de um subproblema, ao invés de ser obtido pelo procedimento da Seção 3.2.1.1, são as colunas do nó pai no momento em que o subproblema em questão foi gerado.

Nos algoritmos BCP propostos nesse trabalho, o software CPLEX só foi utilizado como resolvedor de Relaxação Linear. Os demais aspectos importantes de um BB como, por exemplo, *branching*, seleção de nós, gerenciamento dos cortes, foram definidos por procedimentos por nós implementados.

Por isso, é preciso decidir qual estrutura de dados utilizar para armazenar a árvore de enumeração. Essa estrutura deve ser capaz de, conforme o método de seleção de nós escolhido, adicionar e remover subproblemas da lista de candidatos eficientemente. Como o BCP foi testado com dois métodos diferentes de seleção de nós, duas estruturas de dados diferentes foram utilizadas em cada caso. Quando utilizamos o método *melhor limite*, uma *fila de prioridade* foi empregada para gerenciar os nós candidatos a serem processados. Já para a *busca em profundidade*, foi utilizado uma fila simples. A inserção de um subproblema na fila de prioridade tem complexidade logarítmica, ao passo que a remoção é $O(1)$. No caso da fila simples, os procedimentos de inserção e remoção de um nó tem complexidade $O(1)$.

3.4 Algoritmo BCP Híbrido

Nesta seção, apresentamos uma modificação do algoritmo BCP detalhado na Seção 3.3, denominada de BCP-BC. No algoritmo BCP-BC, a Fase 1 do método de GC não é realizada em toda a árvore de enumeração. Para determinados nós, ao invés de continuar gerando colunas, utilizamos o algoritmo BC (Fase 2 do GCBC) descrito na Seção 3.2.2.

Essa abordagem foi motivada pelo fato de, ao longo da execução do BCP, o número de arestas do grafo que define o problema nos nós se tornarem cada vez menor. Isso ocorre devido aos sucessivos testes de fixação de variáveis realizados até gerarmos o subproblema em questão.

Acreditamos que, em um dado momento, o custo adicional de manter as SECs e DBs, o custo computacional despendido para criar modelos no CPLEX e o tempo de CPU gasto para gerar colunas, não compensa a redução do tempo de execução ao não considerar explicitamente todas as variáveis no PLMR. Logo, incorporamos ao modelo que define o nó as arestas que ainda não foram fixadas e o resolvemos através de BC.

Para decidir se um nó será processado através do BCP ou do BC, utilizamos a seguinte ideia. Assuma $\hat{G} = (V, \hat{E})$ como sendo o grafo de definição desse nó, tal que $\hat{E} = E \setminus E_F$, onde E_F representa as arestas fixadas nos nós predecessores ao nó em questão. Se $\hat{E} >$ NUM_EDGE, devemos, então, continuar gerando colunas para esse nó. Caso contrário, utilizamos BC para resolvê-lo.

Em nossos testes computacionais, utilizamos dois valores diferentes para NUM_EDGE. Realizamos testes com NUM_EDGE = 3000 e NUM_EDGE = 5000 para todas as instâncias utilizadas neste trabalho. Os tempos de CPU obtidos ao utilizar NUM_EDGE = 5000 foram melhores se comparados aos obtidos utilizando NUM_EDGE = 3000. Números maiores e menores do que os valores assumidos para NUM_EDGE foram testados em um subconjunto de instâncias. No entanto, os resultados obtidos foram inferiores se comparados aos obtidos pelos valores previamente definidos.

Avaliamos também uma outra estratégia para decidir se um nó deve ser resolvido por BCP ou BC, que leva em conta a dimensão do grafo de definição do problema. Tal estratégia consiste em aplicar BC ao subproblema se o conjunto $\hat{E}$ possuir menos do que $\alpha \cdot |V|$ arestas. Caso contrário, continuamos gerando colunas para aquele subproblema através do BCP. Testamos essa abordagem para $\alpha \in \{3, 5, 7, 9, 11\}$ em um subconjunto de instâncias para o PAGMRG. Os resultados obtidos foram inferiores se comparados aos obtidos utilizando a primeira estratégia.

Capítulo 4

Experimentos Computacionais

Neste capítulo, apresentamos os experimentos computacionais conduzidos com os algoritmos propostos na dissertação, comparando-os com os algoritmos propostos na literatura do PAGMRG. Para tanto, utilizamos instâncias de teste da literatura, bem como novas e mais difíceis instâncias, aqui introduzidas.

4.1 Instâncias de Teste da Literatura

Um dos conjuntos de instâncias testados, denominado ANDINST, foi introduzido por Andrade et al. [2006]. Nesse conjunto, $|V|$ pode assumir valores entre 100 e 2000. Os vértices do grafo são pontos no plano Euclidiano. Suas coordenadas, sempre inteiras na faixa $[1, 480]$ e $[1, 640]$, foram sorteadas aleatoriamente por meio de uma distribuição uniforme. Os limites de grau $\{d_i : i \in V\}$ dos vértices foram escolhidos aleatoriamente em $\{1, 2, 3, 4\}$, com a condição de que no máximo 10% dos vértices tenham limite de grau unitário. Todas as instâncias são definidas em grafos completos e os custos são dados pela parte inteira da distância euclidiana entre as extremidades dos vértices.

Além do conjunto descrito acima, utilizamos dois grupos de instâncias propostos por Cunha & Lucena [2007], denotados por DE e DR. As instâncias do conjunto DE consistem de vértices correspondentes a pontos no plano Euclidiano, cujas coordenadas, sempre inteiras, são escolhidas aleatoriamente considerando uma distribuição uniforme no intervalo $[1, 1000]$. Para cada instância, os valores de d_i são selecionados aleatoriamente em $\{1, 2, 3\}$, considerando-se uma distribuição uniforme. Os grafos gerados são completos e os custos das arestas correspondem ao valor da distância Euclidiana entre os vértices, com os valores arredondados para baixo. Foram geradas três instâncias distintas para cada valor de $|V| \in \{100, 200, 300, 400, 500, 600\}$. O segundo conjunto de instâncias, DR, é composto de instâncias aleatórias geradas a partir das instâncias

DE. Para cada instância do grupo DE, mantemos os limites de grau de cada vértice e substituimos os custos Euclidianos das arestas por custos inteiros escolhidos de forma aleatória no intervalo $[1, 1000]$, considerando uma distribuição uniforme.

4.2 Novas Instâncias de Teste

Nesta dissertação, propomos dois novos conjuntos de instâncias, que podem ser consideradas mais difíceis do que as descritas anteriormente, uma vez que impomos, para cada instância criada, a restrição

$$\sum_{i \in V} d_i = 2(n-1). \tag{4.1}$$

Em instâncias com essa característica, as restrições de grau dos vértices estão sempre justas em qualquer solução viável para o PAGMRG. Em decorrência disso, uma Busca Local que utiliza a *1-vizinhança* se torna inócua, uma vez que, após a remoção de uma aresta, não é possível reconectar os componentes resultantes senão através da mesma aresta removida.

O primeiro conjunto de novas instâncias são Euclidianas. Tais instâncias, denominadas LH-E, consistem de grafos cujos vértices inteiros têm suas coordenadas geradas aleatoriamente em um quadrado de dimensões 1000×1000 no plano Euclidiano, considerando uma distribuição uniforme. O custo de cada aresta corresponde à distância Euclidiana entre os vértices, com os valores arredondados para cima. Para cada $|V| \in \{100, 200, ..., 1000\}$, foram geradas três instâncias distintas, uma para cada conjunto de valores que d_i pode assumir, a saber, $\{1, 2, 3\}$, $\{1, 2, 3, 4\}$ e $\{1, 2, 3, 4, 5\}$, sempre respeitando a restrição (4.1).

Já o segundo grupo de novas instâncias, designado por LH-R, utiliza-se de custos obtidos de forma aleatória no intervalo $[10, 500]$ para cada par de vértice. O número de vértices e os limites de grau atribuídos à estes foram definidos utilizando os mesmos critérios das instâncias Euclidianas.

4.3 Ambiente Computacional

Os algoritmos propostos neste trabalho foram implementados em C++ e os experimentos foram realizados em uma máquina com processador Intel®Xeon®E5645, 2,40GHz, com 32GB de memória RAM, executando o sistema operacional Ubuntu 12.04 (64 bits). O compilador g++ (versão 4.7.3) foi utilizado com a *flag* de otimização -O3 ativada.

Como dito no Capítulo anterior, os métodos utilizam o software ILOG CPLEX 12.5 [1], através de chamadas de função à biblioteca CONCERT [2].

4.4 Resultados Computacionais

Nesta seção apresentamos o desempenho computacional dos algoritmos propostos nesta dissertação. Inicialmente, comparamos somente os métodos aqui propostos. Após identificar o método que obteve melhores resultados, comparamos seu desempenho com vários métodos disponíveis na literatura, a saber: o algoritmo NDRC/BC4 em Lucena et al. [2011], o algoritmo de Programação por Restrições em Fages et al. [2013] e o método VNS-Lagrangeno associado à uma Árvore de Subgradiente em Freitas [2011] e Andrade & Freitas [2013].

Nas próximas seções, em todas as tabelas, denotamos por z o valor de função objetivo ótima, para uma determinada instância. As colunas lb e ub indicam, respectivamente, o melhor limite inferior e superior obtido, na raiz da árvore de enumeração, por um determinado método para a instância em questão. Os gaps de dualidade $(\text{ub} - \text{lb})/\text{ub}$ alcançados por um método e o número de arestas fixadas são sempre dados em valores percentuais. O tempo de CPU gasto por um determinado algoritmo é sempre representado em segundos.

4.4.1 Comparação entre os métodos propostos neste trabalho

Nas Tabelas 4.1, 4.2 e 4.3, apresentamos os resultados dos métodos GCBC, BCP e BCP-BC para as instâncias DE, DR e ANDINST, respectivamente. Para cada instância, a primeira coluna corresponde à identificação da mesma. O problema correspondente ao nó raiz é resolvido através do mesmo procedimento em todos os algoritmos propostos nesta dissertação. Assim, as cinco colunas subsequentes indicam, respectivamente, o limite inferior, o limite superior, o gap de dualidade, o percentual de arestas fixadas e o tempo de CPU referente ao nó raiz da árvore de enumeração. As duas próximas colunas indicam o número de nós da árvore de BC e o tempo total gasto pelo GCBC. As próximas colunas indicam, respectivamente, o número de nós e o tempo total demandado pelo algoritmo BCP. A próxima coluna, denominada Nós BCP, mostra o número de nós da árvore de enumeração do BCP-BC resolvidos pelo método da Seção 3.2.1. A coluna seguinte, denominada Nós BC, indica o número de nós do BCP-BC resolvidos

[1] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`. Visitado no dia 6 de março de 2014.

[2] `http://www-01.ibm.com/software/commerce/optimization/interfaces/`. Visitado no dia 6 de março de 2014.

somente por Planos de Corte, através do BC descrito na Seção 3.2.2. O número total de nós executados pelo BCP-BC é dado pela soma das colunas Nós BCP e Nós BC. A penúltima coluna informa o tempo de CPU gasto pelo BCP-BC. Finalmente, na última coluna, apresentamos o valor de função objetivo ótimo para a instância.

Como pode ser observado na Tabela 4.1, em 3 das 18 instâncias do conjunto DE conseguimos atestar a otimalidade na raiz da árvore de enumeração. O gap de dualidade médio observado na raiz foi de 0,632%. Pela Tabela 4.1, é possível observar claramente que, para as instâncias DE, o desempenho do método GCBC é superior ao dos algoritmos BCP e BCP-BC. Em 3 casos o método GCBC resolveu a instância em, no máximo, metade do tempo de processamento gasto pelos outros métodos. Para as 6 instâncias em que o tempo demandado pelo GCBC é maior do que o obtido pelo BCP-BC, observamos que o ganho não foi algorítmico, uma vez que a diferença entre os tempos de CPU é de, no máximo, 0,83 segundos, diferença muito inferior ao tempo total gasto para resolver a instância.

Pela Tabela 4.2, observamos que, para as instâncias DR, a otimalidade foi atestada na raiz da árvore de enumeração em 3 das 18 instâncias e o gap de dualidade médio nessa etapa é 4,694%. Para este conjunto, o tempo de CPU gasto pelo BCP-BC foi inferior aos outros métodos em 10 das 18 instâncias deste grupo. Em instâncias maiores, para $|V| \in \{500, 600\}$, o BCP-BC obteve um melhor desempenho em 4 de 6 instâncias. Na instância 500_2, em especial, as execuções dos métodos BCP e BCP-BC obtiveram uma vantagem significativa com relação ao tempo de CPU se comparados ao GCBC.

Embora instâncias de custos aleatórios não sejam necessariamente mais difíceis do que as de custos Euclidianos de mesmas dimensões, observamos que o gap de dualidade das instâncias DR são, de modo geral, maiores se comparado aos obtidos nas instâncias DE. Assim, menos arestas/variáveis são fixadas na raiz, o que faz com que o método GCBC tenha que explicitamente incluir essas variáveis durante o processo de enumeração, aumentando o tempo de CPU necessário para resolver os demais nós. Isso não afeta o método BCP-BC da mesma forma, já que a geração de colunas é levada adiante na árvore de BB, fazendo com que, após a obtenção de melhores limites superiores no decorrer da enumeração, arestas que foram mantidas no GCBC sejam desconsideradas por serem subótimas.

Para as instâncias ANDINST, cujos resultados são mostrados na Tabela 4.3, assim como Lucena et al. [2011], conseguimos atestados de otimalidade para a maioria das instâncias (15 de 25 instâncias) no nó raiz. A diferença entre os tempos de execução dos métodos aqui propostos não são significativas. Para as instâncias de grande porte, onde $|V| = 2000$, não conseguimos avaliar LP(P_2) com nenhum dos métodos, pois o algoritmo

atingiu o tempo limite de 4 horas. Isso ocorre em razão do tempo computacional gasto para separar as desigualdades SECs para grafos de grande porte e do custo de se resolver Relaxações Lineares com um número muito grande de linhas, sempre mantidas nos Programas Lineares. Para as instâncias com $|V| = 2000$ (e algumas instâncias LH-E e LH-R) os limites inferiores foram estimados, em cada iteração do método da Seção 3.2.1, por $z_{PLMR} + (n-1) \cdot \omega(e^*)$, onde z_{PLMR} corresponde ao valor de função objetivo do PLMR naquela iteração [Desaulniers et al., 2005].

Tabela 4.1. Comparativo entre os métodos propostos - instâncias DE

	Raiz					GCBC		BCP		BCP-BC			
Id	lb	ub	%gap	%ef	t(s)	Nós	t(s)	Nós	t(s)	Nós BCP	Nós BC	t(s)	z
100_1	8779,000	8779	0,000	-	0,20	0	0,20	1	0,20	1	0	0,20	8779
100_2	9629,000	9629	0,000	-	0,60	0	0,60	1	0,62	1	0	0,59	9629
100_3	10791,500	10798	0,060	96,08	0,99	1	1,01	3	1,08	1	1	1,02	10798
200_1	12029,000	12044	0,125	97,69	4,09	2	4,15	3	4,51	1	3	4,19	12031
200_2	12618,500	12755	1,070	86,96	6,89	40	10,07	35	17,44	1	46	10,51	12645
200_3	12221,000	12406	1,491	81,37	6,78	21	8,04	5	8,09	1	6	7,90	12229
300_1	14776,750	14852	0,507	94,34	32,70	5	34,15	6	36,86	1	5	33,32	14792
300_2	13922,000	13931	0,065	98,68	10,40	1	10,53	3	11,03	1	1	10,48	13931
300_3	14972,837	15194	1,456	79,76	29,51	33	36,15	83	158,73	27	116	96,05	14997
400_1	19239,000	19239	0,000	-	110,52	0	110,52	1	111,12	1	0	110,29	19239
400_2	17405,500	17555	0,852	89,21	65,24	57	76,13	13	87,01	11	9	86,24	17419
400_3	16074,500	16227	0,940	87,99	126,49	15	132,42	9	144,24	3	18	136,56	16091
500_1	19351,000	19388	0,191	97,68	171,53	2	173,22	3	182,85	1	2	172,78	19357
500_2	22397,500	22453	0,247	95,86	496,84	2	499,89	3	503,85	3	0	503,21	22400
500_3	19392,419	19705	1,586	71,33	251,12	146	435,60	217	1695,64	125	681	1256,50	19411
600_1	21915,722	22169	1,142	78,43	547,65	74	642,13	49	1293,36	43	36	1233,21	21930
600_2	21423,181	21725	1,389	72,52	666,02	35	702,65	21	884,48	19	15	879,87	21449
600_3	22236,962	22292	0,247	96,74	575,14	3	583,36	5	653,72	3	6	616,59	22243

Tabela 4.2. Comparativo entre os métodos propostos - instâncias DR

	Raiz					GCBC		BCP		BCP-BC			
Id	lb	ub	%gap	%ef	t(s)	Nós	t(s)	Nós	t(s)	Nós BCP	Nós BC	t(s)	z
100_1	2174,750	2245	3,129	91,01	0,40	8	0,48	5	0,55	1	11	0,60	2186
100_2	2674,000	2674	0,000	-	0,21	0	0,21	1	0,22	1	0	0,21	2674
100_3	2689,000	2689	0,000	-	0,33	0	0,33	1	0,32	1	0	0,32	2689
200_1	2139,000	2331	8,237	80,40	6,24	21	7,63	9	8,94	1	23	8,23	2141
200_2	2469,857	2731	9,562	75,34	4,90	14	5,74	3	5,62	1	7	5,54	2471
200_3	2402,750	2527	4,917	87,61	2,56	22	3,98	9	6,24	1	7	3,16	2405
300_1	2460,000	2570	4,280	89,01	13,78	11	17,45	12	30,41	1	37	18,53	2462
300_2	2312,000	2411	4,106	90,23	36,22	1	36,90	3	37,62	1	1	36,78	2312
300_3	2583,857	2934	11,934	67,84	23,51	16	29,38	8	33,09	7	2	32,84	2585
400_1	2815,000	3256	13,544	59,85	98,49	160	233,31	15	151,16	15	0	150,32	2817
400_2	2441,034	2569	4,981	87,94	37,20	30	47,95	33	113,68	21	235	148,39	2443
400_3	2387,000	2387	0,000	-	53,28	0	53,28	1	53,59	1	0	53,22	2387
500_1	2595,625	2602	0,245	98,83	165,98	25	200,07	25	542,01	1	20	198,03	2597
500_2	2650,000	2874	7,794	79,32	112,71	163	832,37	35	440,29	35	0	437,20	2652
500_3	2591,851	2669	2,891	92,52	106,40	23	125,94	15	260,56	13	117	353,18	2593
600_1	2819,520	2833	0,476	98,30	344,13	10	396,29	5	488,15	1	6	350,17	2820
600_2	2659,668	2885	7,810	78,96	419,82	204	1962,98	85	2285,15	77	153	2657,99	2662
600_3	2798,452	2815	0,588	98,05	385,37	103	561,86	79	1743,08	1	111	525,20	2800

Tabela 4.3. Comparativo entre os métodos propostos - instâncias ANDINST

	Raiz					GCBC		BCP		BCP-BC			
Id	lb	ub	%gap	%ef	t(s)	Nós	t(s)	Nós	t(s)	Nós BCP	Nós BC	t(s)	z
100_1	3790,000	3790	0,000	-	0,03	0	0,03	1	0,02	1	0	0,04	3790
100_2	3829,000	3829	0,000	-	0,05	0	0,05	1	0,04	1	0	0,04	3829
100_3	3916,000	3916	0,000	-	0,04	0	0,04	1	0,04	1	0	0,05	3916
200_1	5316,000	5316	0,000	-	0,75	0	0,75	1	0,76	1	0	0,75	5316
200_2	5647,000	5647	0,000	-	1,29	0	1,29	1	1,3	1	0	1,3	5647
200_3	5698,000	5698	0,000	-	3,03	0	3,03	1	3,07	1	0	3,02	5698
300_1	6476,500	6477	0,008	-	1,61	0	1,61	1	1,73	1	0	1,62	6477
300_2	6806,000	6815	0,132	98,46	5,57	3	5,73	3	5,88	1	2	5,65	6807
300_3	6430,000	6430	0,000	-	2,62	0	2,62	1	2,62	1	0	2,58	6430
400_1	7414,000	7414	0,000	-	4,7	0	4,7	1	4,75	1	0	4,7	7414
400_2	7781,000	7797	0,205	98,19	36,79	2	37,21	3	38,31	1	2	37,04	7782
400_3	7602,500	7605	0,033	99,25	45,09	1	45,28	4	45,93	1	1	44,94	7604
500_1	8272,000	8272	0,000	-	7,66	0	7,66	1	7,67	1	0	7,61	8272
500_2	8391,500	8394	0,030	99,38	150,9	2	151,24	3	155,45	1	2	150,19	8392
500_3	8502,000	8502	0,000	-	16,37	0	16,37	1	16,46	1	0	16,33	8502
600_1	9037,000	9037	0,000	-	3,82	0	3,82	1	3,85	1	0	3,82	9037
700_1	9786,000	9786	0,000	-	30,59	0	30,59	1	30,78	1	0	30,5	9786
800_1	10333,000	10333	0,000	-	109,56	0	109,56	1	110,15	1	0	109,02	10333
900_1	10918,000	10918	0,000	-	90,54	0	90,54	1	90,82	1	0	90,11	10918
1000_1	11407,000	11414	0,061	99,42	4721,49	5	4754,58	3	6173,44	1	37	5031,65	11408
2000_1	10671,088	15670	31,901	-	*	0	*	0	*	0	0	*	-
2000_2	8441,313	16253	48,063	-	*	0	*	0	*	0	0	*	-
2000_3	7214,865	16689	56,769	-	*	0	*	0	*	0	0	*	-
2000_4	2453,549	16373	85,015	-	*	0	*	0	*	0	0	*	-
2000_5	6944.221	16530	57.99	-	*	0	*	0	*	0	0	*	-

(∗) Execução interrompida ao atingir o tempo limite de 14400 segundos (4 horas)

Nas Tabelas 4.4 e 4.5, apresentamos os resultados computacionais alcançados pelos algoritmos GCBC, BCP e BCP-BC para as instâncias LH-E e LH-R, respectivamente, propostas nesta dissertação. A primeira coluna corresponde à identificação da instância. As próximas cinco colunas representam, respectivamente, o limite inferior, o limite superior, o gap de dualidade, o percentual de arestas fixadas e o tempo de CPU referente ao nó raiz da árvore de enumeração. As próximas colunas denotam o limite inferior, o limite superior, o percentual do gap de dualidade, o número de nós e o tempo de processamento do método GCBC. Nas três colunas seguintes, apresentamos o gap de dualidade, o número de nós e o tempo de CPU do algoritmo BCP. As três colunas subsequentes indicam, respectivamente, o número de nós BCP, a quantidade de nós BC e o tempo de execução do algoritmo BCP-BC. Finalmente, apresentamos o valor ótimo, se obtido, da instância em questão. Optamos por não mostrar os limites inferiores e superiores dos métodos BCP e BCP-BC para reduzir o tamanho da tabela. Além disso, pela Tabela 4.4, é possível constatar que, para instâncias LH-E, os limites obtidos por BCP e BCP-BC nunca fornecem gaps de dualidade melhores do que os obtidos pelo GCBC. Omitimos, também, os gaps de dualidade obtidos pelo algoritmo BCP-BC pelo fato de estes serem, coincidentemente, os mesmos gaps alcançados pelo BCP.

Analisando a Tabela 4.4, verificamos que em 8 das 30 instâncias do conjunto LH-E o valor ótimo foi alcançado na raiz, e o gap de dualidade médio (excetuando-se as instâncias que atingiram o tempo limite), nessa etapa, é de 0,781%. Para as instâncias LH-E com $|V| \in \{100, 200, ..., 600\}$, o gap de dualidade médio observado no nó raiz é de 0,571%, valor inferior ao obtido para as instâncias DE (0,632%). No entanto, os tempos de CPU gastos pelos métodos para resolver as instâncias LH-E é, em geral, maior do que os tempos gastos gastos para resolver as instâncias DE. As diferenças nos tempos de execução são, em parte, justificadas pelas diferenças no número médio de arestas fixadas na raiz, 88,31% para as instâncias DE e 77,45% para as LH-E. Analisando este resultado, em conjunto com os tempos de CPU demandados para resolver estas instâncias, nos leva a crer que as instâncias aqui propostas são mais difíceis de serem solucionadas se comparadas com as instâncias da literatura. Com relação aos algoritmos, o método GCBC obteve tempos de CPU tão bons quanto os outros dois em 22 das 30 instâncias deste conjunto. Em nenhuma instância obtivemos um ganho real ao usar o BCP, isto é, ao realizar GC em toda a árvore de busca. Para as 8 instâncias em que o tempo de CPU gasto pelo GCBC é superior ao gasto pelo BCP-BC, ou a instância foi resolvida na raiz ou os nós da árvore de enumeração, excetuando-se a raiz, foi resolvida pelo BC.

Os resultados da Tabela 4.5, para as instâncias LH-R, mostram que, ao contrário

Tabela 4.4. Comparativo entre os métodos propostos - instâncias LH-E

	Raiz					GCBC					BCP			BCP-BC			
Id	lb	ub	%gap	%ef	t(s)	lb	ub	%gap	Nós	t(s)	%gap	Nós	t(s)	Nós BCP	Nós BC	t(s)	z
100_3	11640,000	11640	0,000	-	1,75	11640,000	11640	0,000	0	1,75	0,000	1	1,73	1	0	1,72	11640
100_4	11748,000	11748	0,000	-	1,14	11748,000	11748	0,000	0	1,14	0,000	1	1,15	1	0	1,14	11748
100_5	12345,000	12345	0,000	-	1,48	12345,000	12345	0,000	0	1,48	0,000	1	1,48	1	0	1,48	12345
200_3	15168,750	15393	1,457	68,83	23,03	15177,000	15177	0,000	3	24,03	0,000	5	25,82	3	7	25,22	15177
200_4	14669,000	14680	0,075	97,72	10,62	14671,000	14671	0,000	0	10,68	0,000	3	10,92	1	0	10,60	14671
200_5	14963,000	14982	0,127	97,44	9,62	14967,000	14967	0,000	2	9,74	0,000	5	10,80	1	2	9,74	14967
300_3	15659,875	16155	3,065	44,29	24,23	15691,000	15691	0,000	23	32,56	0,000	109	147,02	99	86	154,92	15691
300_4	19730,333	20115	1,912	51,35	64,60	19731,000	19731	0,000	4	68,63	0,000	3	67,22	3	0	66,69	19731
300_5	18492,000	18492	0,000	-	37,70	18492,000	18492	0,000	0	37,70	0,000	1	37,92	1	0	37,68	18492
400_3	20451,000	20577	0,612	86,54	328,86	20454,000	20454	0,000	5	337,28	0,000	3	344,76	3	0	343,31	20454
400_4	21005,000	21005	0,000	-	158,29	21005,000	21005	0,000	0	158,29	0,000	1	159,15	1	0	157,80	21005
400_5	24907,000	24954	0,188	95,65	330,73	24911,000	24911	0,000	3	334,56	0,000	7	356,55	1	7	334,14	24911
500_3	20565,641	21070	2,394	43,11	534,17	20584,000	20584	0,000	156	746,00	0,000	85	1399,07	79	8	1382,08	20584
500_4	27118,500	27126	0,028	98,85	998,05	27124,000	27124	0,000	5	1003,41	0,000	5	1038,21	1	6	1014,55	27124
500_5	27389,000	27389	0,000	-	807,76	27389,000	27389	0,000	0	807,76	0,000	1	809,37	1	0	809,88	27389
600_3	24578,111	24683	0,425	90,68	1913,66	24583,000	24583	0,000	8	1939,91	0,000	9	2115,22	9	0	2149,73	24583
600_4	27141,000	27141	0,000	-	942,20	27141,000	27141	0,000	0	942,20	0,000	1	942,78	1	0	947,13	27141
600_5	24923,000	24923	0,000	-	674,22	24923,000	24923	0,000	0	674,22	0,000	1	673,31	1	0	670,33	24923
700_3	26009,725	26219	0,798	81,14	3662,08	26024,000	26024	0,000	42	3848,63	0,000	25	4743,17	23	24	4808,70	26024
700_4	28588,333	28812	0,776	81,40	3274,68	28592,000	28592	0,000	9	3307,98	0,000	7	3443,61	3	4	3438,27	28592
700_5	31397,047	31461	0,203	95,73	5768,12	31402,000	31402	0,000	17	5877,97	0,000	7	6169,37	7	0	6212,13	31402
800_3	27426,250	28217	2,802	18,78	3104,75	27433,000	27433	0,000	21	3501,19	0,000	14	3819,91	5	10	3576,03	27433
800_4	28377,063	28824	1,551	58,32	4689,80	28399,000	28399	0,000	127	5479,05	0,000	49	7816,23	49	0	7823,34	28399
800_5	32250,917	32257	0,019	99,39	6489,22	32253,000	32253	0,000	3	6496,47	0,000	3	6650,50	1	2	6491,94	32253
900_3	29637,508	31039	4,515	10,32	13391,39	29641,303	29970	1,097	4	*	4,515	3	*	2	0	*	-
900_4	34049,250	34075	0,076	98,43	12651,04	34054,000	34054	0,000	3	12668,80	0,000	3	12685,06	3	0	12831,39	34054
900_5	33350,000	33374	0,072	98,85	7757,92	33355,000	33355	0,000	2	7771,19	0,000	3	7836,40	1	2	7791,03	33355
1000_3	30310,879	32363	6,341	-	*	30310,879	32363	6,341	0	*	6,341	1	*	1	0	*	-
1000_4	0,000	33774	100,000	-	*	0,000	33774	100,000	0	*	100,000	1	*	1	0	*	-
1000_5	0,000	50247	100,000	-	*	0,000	50247	100,000	0	*	100,000	1	*	1	0	*	-

(*) Execução interrompida ao atingir o tempo limite de 14400 segundos (4 horas)

do GCBC, os algoritmos BCP e BCP-BC conseguiram atestados de otimalidade para as instâncias 700_3, 800_3 e 800_5. Além disso, para as instâncias não resolvidas na otimalidade, o BCP e o BCP-BC encontraram melhores gaps de dualidade para as instâncias 900_4 e 1000_3. Em alguns casos, por exemplo, as instâncias 500_3, 600_3 e 600_4, o GCBC conseguiu atestados de otimalidade em um tempo de CPU consideravelmente mais baixo. Três instâncias foram resolvidas na raiz da árvore de enumeração e o gap de dualidade médio (excetuando-se a instância 1000_4) obtido nessa fase é de 3,600%.

À exceção do conjunto de instâncias LH-R, o método GCBC, de modo geral, apresentou melhores resultados se comparados aos algoritmos BCP e BCP-BC. O BCP-BC, por sua vez, claramente dominou o BCP. Compilamos, na Tabela 4.6, o número de instâncias de cada conjunto que foram resolvidas mais rapidamente pelos métodos aqui propostos. A primeira coluna identifica o conjunto de instâncias. Nas outras colunas, apresentamos o número de instâncias em que um método dominou os outros. Somente contabilizamos as instâncias em que a diferença entre os tempos é maior do que um segundo. Assim, observando os dados da Tabela 4.6, em conjunto os resultados mostrados das Tabelas 4.1, 4.2, 4.3, 4.4 e 4.5, é possível verificar que o GCBC leva uma evidente vantagem em relação aos métodos BCP e BCP-BC.

4.4.2 Comparação de GCBC com algoritmos exatos da literatura

Nas próximas tabelas, comparamos os resultados obtidos pelo algoritmos GCBC com os melhores métodos disponíveis na literatura. Os métodos BCP e BCP-BC não serão aqui considerados, uma vez que estes apresentaram resultados inferiores em relação ao GCBC. Ao invés de comparar diretamente o método GCBC com o algoritmo híbrido proposto em Lucena et al. [2011], denotado por NDRC/BC4 naquele trabalho, utilizamos nossa própria implementação da fase BC4 daquele algoritmo, uma vez que o código fonte de BC4 não é mais disponível. O nosso algoritmo NDRC/BC utiliza o código fonte da fase NDRC em Cunha & Lucena [2007] e Lucena et al. [2011], mas implementa uma fase BC com algumas modificações em relação ao BC4.

Assim como em Lucena et al. [2011], as SECs e as DBs identificadas pelo NDRC foram repassadas ao nosso BC. No entanto, o BC4 proposto por Lucena et al. [2011] se difere do nosso, essencialmente, por utilizar um outro pacote para gerenciar a árvore de enumeração (utilizamos o CPLEX 12.5 enquanto aqueles utilizaram o XPRESS MP 16), por separar desigualdades válidas *Cutsets* e por utilizar formas diferentes de gerenciar o *cut-pool*. Logo, ao compararmos os algoritmos aqui propostos e os da literatura com

Tabela 4.5. Comparativo entre os métodos propostos - instâncias LH-R

	Raiz					GCBC					BCP			BCP-BC			
Id	lb	ub	%gap	%ef	t(s)	lb	ub	%gap	Nós	t(s)	%gap	Nós	t(s)	Nós BCP	Nós BC	t(s)	z
100_3	2143,600	2496	14,119	31,84	0,57	2149,000	2149	0,000	7	0,90	0,000	15	1,56	1	11	0,91	2149
100_4	2327,000	2327	0,000	-	0,48	2327,000	2327	0,000	0	0,48	0,000	1	0,47	1	0	0,47	2327
100_5	2416,500	2428	0,474	94,97	0,48	2419,000	2419	0,000	2	0,53	0,000	3	0,70	1	2	0,52	2419
200_3	3106,000	3398	8,593	44,64	5,45	3108,000	3108	0,000	65	12,46	0,000	13	9,62	13	0	9,53	3108
200_4	3144,000	3144	0,000	-	3,49	3144,000	3144	0,000	0	3,49	0,000	1	3,51	1	0	3,48	3144
200_5	3335,000	3376	1,214	91,80	6,39	3336,000	3336	0,000	4	6,77	0,000	7	7,96	1	10	7,13	3336
300_3	4110,000	4110	0,000	-	11,49	4110,000	4110	0,000	0	11,49	0,000	1	11,52	1	0	11,37	4110
300_4	4205,000	4443	5,357	60,05	19,80	4206,000	4206	0,000	20	25,12	0,000	7	25,72	3	8	23,39	4206
300_5	4342,000	4342	0,000	-	21,72	4342,000	4342	0,000	0	21,72	0,000	1	21,81	1	0	21,72	4342
400_3	5114,057	5630	9,164	11,34	156,96	5117,000	5117	0,000	196	534,11	0,000	133	743,01	111	546	1039,44	5117
400_4	5232,500	5540	5,551	46,84	184,76	5233,000	5233	0,000	7	219,32	0,000	7	235,70	7	0	233,84	5233
400_5	5354,600	5446	1,678	84,80	221,45	5356,000	5356	0,000	17	234,75	0,000	28	328,75	23	50	336,13	5356
500_3	5997,879	6211	3,431	61,28	384,81	5999,000	5999	0,000	140	794,59	0,000	101	1402,39	89	43	1350,59	5999
500_4	6193,000	6288	1,511	83,71	992,35	6194,000	6194	0,000	29	1191,26	0,000	15	1149,10	15	0	1140,58	6194
500_5	6167,500	6214	0,748	91,97	161,30	6168,000	6168	0,000	54	231,14	0,000	9	234,50	9	0	232,92	6168
600_3	6886,873	7189	4,203	44,81	483,96	6888,000	6888	0,000	120	1729,73	0,000	55	2329,26	39	321	3027,69	6888
600_4	7096,646	7239	1,966	75,98	801,07	7098,000	7098	0,000	849	4660,84	0,000	527	10583,59	108	5500	14399.86	7098
600_5	7228,937	7230	0,015	99,38	1297,80	7230,000	7230	0,000	48	1371,76	0,000	37	1769,10	1	51	1338,75	7230
700_3	7908,849	8214	3,715	45,04	2467,99	7909,518	7911	0,019	147	*	0,000	117	8903,09	117	0	8887,37	7910
700_4	8073,080	8531	5,368	22,42	2016,93	8073,765	8075	0,015	185	*	0,792	232	*	232	0	*	-
700_5	8108,444	8116	0,093	98,43	1741,99	8109,000	8109	0,000	34	1849,87	0,000	17	3971,83	1	35	2017,18	8109
800_3	8854,360	9205	3,809	36,03	5987,72	8854,600	8858	0,038	76	*	0,000	39	11924,09	39	0	11883,77	8855
800_4	8953,875	9264	3,348	46,65	1981,12	8955,000	8955	0,000	172	13855,33	0,954	93	*	93	0	*	8955
800_5	9110,522	9283	1,858	71,97	4798,86	9110,628	9112	0,015	128	*	0,000	35	11706,98	35	0	11668,62	9111
900_3	9335,329	10334	9,664	-	*	9335,329	10334	9,664	0	*	9,664	1	*	1	0	*	-
900_4	9969,968	10595	5,899	18,27	4157,61	9969,968	10226	2,504	40	*	1,475	64	*	64	0	*	-
900_5	10024,353	10030	0,056	98,82	6638,34	10025,000	10025	0,000	146	10914,15	0,000	47	*	1	97	8960,29	10025
1000_3	10783,034	11178	3,533	28,11	12017,10	10783,110	11178	3,533	4	*	2,977	6	*	6	0	*	-
1000_4	0,000	11610	100,000	-	*	0,000	11610	100,000	0	*	100,000	1	*	1	0	*	-
1000_5	10232,285	11235	8,925	-	*	10232,285	11235	8,925	0	*	8,925	1	*	1	0	*	-

(*) Execução interrompida ao atingir o tempo limite de 14400 segundos (4 horas)

Tabela 4.6. Número de instâncias dominadas por cada método

Tipo	GCBC	BCP	BCP-BC
DE	8	0	0
DR	5	0	4
ANDINST	1	0	1
LH-E	14	0	2
LH-R	8	0	7
Total	36	0	14

o NDRC/BC, estamos apresentando uma aproximação de como estes algoritmos se compariariam ao NDRC/BC4 em Lucena et al. [2011].

O algoritmo proposto em Freitas [2011] e Andrade & Freitas [2013], que utiliza uma heurística VNS-Lagrangeana em conjunto com uma Árvore de Subgradiente, é aqui designado por VNSL+ASG. Como o código fonte ou o arquivo binário do VNSL+ASG não foi disponibilizado, só comparamos o método VNSL+ASG com os outros algoritmos nas instâncias DE, DR e ANDINST. Isto foi feito utilizando os resultados divulgados naqueles trabalhos. O algoritmo VNSL+ASG foi implementado na linguagem C++ e executado em uma máquina Intel®Core 2 Duo™, 2,4 GHz e 3GB de memória RAM.

Finalmente, o método de Programação por Restrições utilizado em Fages et al. [2013] é aqui denotado por CHOCO, nome da biblioteca (para a linguagem de programação Java) de Programação por Restrições [3] utilizada naquele trabalho. O código fonte do método proposto por Fages et al. [2013] é fornecido como código de exemplo nessa biblioteca. Assim sendo, todos os resultados que divulgamos para CHOCO, foram obtidos no mesmo ambiente de testes empregado para avaliar GCBC, BCP, BCP-BC e NDRC/BC.

A Tabela 4.7 mostra os resultados alcançados pelos métodos da literatura e pelo GCBC para o conjunto de instâncias DE. As duas primeiras colunas correspondem, respectivamente, ao identificador da instância e ao valor de função objetivo ótimo para a instância. Com exceção do CHOCO, para cada algoritmo comparado informamos os gaps de dualidade e os tempos de CPU obtidos. Para os métodos GCBC e NDRC/BC informamos, também, o número de nós abertos na árvore do BC. Como os algoritmos VNSL+ASG e CHOCO não foram capazes de resolver algumas destas instâncias com garantia de otimalidade, informamos, para cada instância, o melhor limite superior obtido por estes métodos até atingir o tempo limite.

Pela Tabela 4.7, observamos que os algoritmos GCBC e NDRC/BC foram capazes

[3] `http://www.emn.fr/z-info/choco-solver/`. Visitado no dia 14 de março de 2014. Versão da biblioteca 3.1.0.

de encontrar o ótimo em todas as instâncias do conjunto DE. O VNSL+ASG conseguiu resolver apenas 8 instâncias e o CHOCO resolveu 11 das 18 instâncias do conjunto. Além disso, é possível verificar que o VNSL+ASG demanda tempos de processamento bastante elevados se comparados aos métodos GCBC e NDRC/BC, mesmo para as instâncias de pequenas dimensões, com $|V| \in \{100, 200, 300\}$. Devido a enorme diferença com relação aos tempos de CPU obtidos pelo método VNSL+ASG se comparados, por exemplo, aos do GCBC, acreditamos não se tratar apenas da diferença entre as máquinas utilizadas para executar os métodos. Comparando os métodos GCBC e NDRC/BC, é possível observar uma clara vantagem do GCBC em relação ao NDRC/BC. Com exceção da instância 400_3, em todas as outras instâncias, o algoritmo GCBC conseguiu atestados de otimalidade gastando menos tempo de CPU. Essa redução no tempo de resolução das instâncias DE ocorre, principalmente, devido a melhor qualidade dos limites (tanto duais quanto primais) obtidos pelo método baseado em GC (Tabela 4.1) em comparação com os limites alcançados pelo método NDRC (disponíveis em Cunha & Lucena [2007]) na etapa de preprocessamento.

Tabela 4.7. Comparativo entre os métodos da literatura e o GCBC para as instâncias DE

		GCBC			NDRC/BC			VNSL+ASG			CHOCO	
Id	z	%gap	Nós	t(s)	%gap	Nós	t(s)	ub	%gap	t(s)	ub	t(s)
100_1	8779	0	0	0,20	0	0	2,72	8779	0,000	50	8779	2,17
100_2	9629	0	0	0,60	0	0	1,76	9629	0,000	171	9629	2,19
100_3	10798	0	1	1,01	0	8	7,22	10798	0,000	1413	10798	12,67
200_1	12031	0	2	4,15	0	1	16,32	12031	0,000	3766	12031	12,86
200_2	12645	0	40	10,07	0	38	35,96	12665	0,292	†	12645	497,09
200_3	12229	0	21	8,04	0	5	33,19	12229	0,000	3147	12229	80,61
300_1	14792	0	5	34,15	0	11	69,23	14796	0,033	†	14792	364,51
300_2	13931	0	1	10,53	0	2	26,46	13931	0,000	26390	13931	74,30
300_3	14997	0	33	36,15	0	93	85,02	15009	0,273	†	14997	8727,62
400_1	19239	0	0	110,52	0	0	239,46	19239	0,000	103904	25536	∗
400_2	17419	0	57	76,13	0	117	249,80	17479	0,430	†	20664	∗
400_3	16091	0	15	132,42	0	10	81,68	16091	0,000	355773	16091	611,14
500_1	19357	0	2	173,22	0	6	181,54	19445	0,480	†	19357	390,11
500_2	22400	0	2	499,89	0	3	758,17	22754	1,593	†	29304	∗
500_3	19411	0	146	435,60	0	557	877,39	19493	0,583	†	21843	∗
600_1	21930	0	74	642,13	0	514	778,84	22039	0,584	†	25742	∗
600_2	21449	0	35	702,65	0	957	1550,39	21663	1,214	†	25741	∗
600_3	22243	0	3	583,36	0	15	843,03	22368	0,625	†	26768	∗

(∗) Execução interrompida ao atingir o tempo limite de 14400 segundos (4 horas)
(†) Execução interrompida ao atingir o tempo limite de 388800 segundos (108 horas)
Obs.: os dados das colunas referentes ao VNSL+ASG foram extraídos de Freitas [2011]

Os resultados para as instâncias DR são apresentados na Tabela 4.8. Para cada instância do conjunto, identificada pela primeira coluna, informamos o seu valor ótimo

na coluna seguinte. Para os métodos GCBC e NDRC/BC apresentamos o número de nós do BC e o tempo de CPU gasto para resolver a instância. Para os algoritmos VNSL+ASG e CHOCO, apresentamos somente os tempos de execução. Os gaps de dualidade foram omitidos nessa tabela, pois todos os métodos conseguiram atestados de otimalidade dentro do tempo limite estabelecido. Para este conjunto, o CHOCO obteve melhores tempos de processamento em 12 das 18 instâncias. Para a instância 600_2, em especial, o CHOCO conseguiu o valor ótimo em um tempo de CPU muito inferior ao dos outros métodos. Os tempos de execução do VNSL+ASG foram, novamente, muito piores se comparados aos outros métodos. Comparando os tempos de CPU dos algoritmos GCBC e NDRC/BC, observamos que este último foi superior em apenas 5 das 18 instâncias se comparado ao primeiro. A superioridade do GCBC em relação ao NDRC/BC, entretanto, não pode ser justificada pelos limites obtidos pela GC e pelo NDRC na etapa de preprocessamento. O NDRC introduzido por Cunha & Lucena [2007] obteve um gap de dualidade médio de 1,03%, muito inferior ao gap médio de 4,694% alcançado pelo método de GC aqui proposto, principalmente pela qualidade dos limites primais. Observamos, porém, que o tempo gasto pela GC para resolver o nó raiz, é, na maioria das instâncias, consideravelmente inferior ao tempo gasto pelo NDRC. Com isso, é possível observar um ganho estritamente algorítmico ao se utilizar GC ao invés de NDRC para as instâncias DR.

Na Tabela 4.9, apresentamos os resultados do GCBC e dos métodos NDRC/BC, VNSL+ASG e CHOCO para as instâncias ANDINST. Nessa tabela, exibimos as mesmas informações disponíveis na Tabela 4.7. Apresentamos também o melhor limite superior obtido pelo método GCBC, uma vez que não conseguimos provar a otimalidade em todas as instâncias ANDINST. Para esse conjunto, o método NDRC/BC conseguiu atestados de otimalidade para todas as instâncias, ao contrário dos outros métodos. Além disso, desconsiderando as instâncias em que nenhum outro método além do NDRC/BC encontrou o valor ótimo, este método resolveu, em menos tempo, 9 das 23 instâncias restantes. Embora o VNSL+ASG não consiga encontrar a solução ótima em todas as instâncias, podemos observar, através do gap de dualidade, que o método encontra soluções cujos custo são bastante próximos aos custos ótimos. O CHOCO, apesar de claramente superar o GCBC e o VNSL+ASG, obteve, na maioria dos casos, tempos de CPU maiores se comparado ao NDRC/BC.

Analisando os resultados computacionais para as instâncias ANDINST, cujos limites de grau são mais folgados, observamos como o GCBC se comporta à medida em que as dimensões das instâncias crescem. A partir de $|V| = 400$, as ordens de grandeza dos tempos de execução do GCBC crescem mais rapidamente que o NDRC/BC. Isso se deve ao fato de o número de restrições de grau (2.4) ter menos impacto na Rela-

Tabela 4.8. Comparativo entre os métodos da literatura e o GCBC para as instâncias DR

		GCBC		NDRC/BC		VNSL+ASG	CHOCO
Id	z	Nós	t(s)	Nós	t(s)	t(s)	t(s)
100_1	2186	8	0,48	7	3,98	24,00	5,09
100_2	2674	0	0,21	0	2,48	2,00	0,66
100_3	2689	0	0,33	1	4,56	18,00	0,93
200_1	2141	21	7,63	15	18,85	45,00	6,38
200_2	2471	14	5,74	4	18,99	117,00	7,08
200_3	2405	22	3,98	13	19,44	101,00	6,76
300_1	2462	11	17,45	12	54,29	241,00	11,79
300_2	2312	1	36,90	0	25,84	212,00	10,11
300_3	2585	16	29,38	5	75,37	311,00	13,92
400_1	2817	160	233,31	68	269,00	4259,00	216,69
400_2	2443	30	47,95	55	163,18	4557,00	56,75
400_3	2387	0	53,28	0	33,71	381,00	13,89
500_1	2597	25	200,07	25	192,44	3622,00	47,43
500_2	2652	163	832,37	50	942,56	3481,00	88,13
500_3	2593	23	125,94	8	200,67	2089,00	42,52
600_1	2820	10	396,29	6	369,44	4933,00	56,31
600_2	2662	204	1962,98	152	1365,85	41611,00	106,44
600_3	2800	103	561,86	104	567,91	13002,00	120,86

Obs.: os dados das colunas referentes ao VNSL+ASG foram extraídos de Freitas [2011]

xação Lagrangeana quando os limites de grau nos vértices são mais relaxados. Logo, para instâncias deste tipo, os tempo gasto para resolver as Relaxações Lineares e os algoritmos de separação no nó raiz do método GCBC supera o tempo de CPU gasto pelo NDRC para resolver o Dual Lagrangeano (3.2).

Os resultados obtidos pelos métodos GCBC, NDRC/BC e CHOCO para as instâncias LH-E são apresentados na Tabela 4.10. Para cada instância, apresentamos o valor ótimo, se encontrado, na segunda coluna. Apresentamos, em seguida, os melhor limite inferior e superior, o gap de dualidade, o número de nós do BC é o tempo de CPU para os algoritmos GCBC e NDRC/BC. Para o CHOCO, mostramos apenas o melhor limite superior e o tempo de execução do método. Assim como nas instâncias DE, os resultados obtidos pelo CHOCO foram bastante inferiores se comparados aos outros dois algoritmos. Em 21 das 30 instâncias do conjunto LH-E, o CHOCO atingiu o tempo limite preestabelecido sem conseguir encontrar a solução ótima. A diferença entre os melhores limites superiores fornecido por este método (no tempo limite de 4 horas) e o valor de função objetivo ótimo para instância (se conhecido) é, em média, cerca de 15,95%. Analisando a Tabela 4.10, é possível observar que o GCBC obteve, melhores tempos de CPU em 23 das 30 instâncias, se comparado com o NDRC/BC. Além disso, em 5 instâncias o GCBC foi o único método capaz de fornecer uma prova

Tabela 4.9. Comparativo entre os métodos da literatura e o GCBC para as instâncias ANDINST

		GCBC				NDRC/BC		VNSL+ASG			CHOCO	
Id	z	%gap	ub	Nós	t(s)	Nós	t(s)	%gap	ub	t(s)	ub	t(s)
100_1	3790	0,000	3790	0	0,03	0	0,03	0,000	3790	0,00	3790	0,41
100_2	3829	0,000	3829	0	0,05	0	0,04	0,000	3829	0,00	3829	0,15
100_3	3916	0,000	3916	0	0,04	0	0,11	0,000	3916	1,00	3916	0,36
200_1	5316	0,000	5316	0	0,75	0	0,28	0,000	5316	0,00	5316	0,75
200_2	5647	0,000	5647	0	1,29	0	1,00	0,000	5647	1,00	5647	1,11
200_3	5698	0,000	5698	0	3,03	0	0,67	0,000	5698	1,00	5698	0,54
300_1	6477	0,000	6477	0	1,61	0	0,97	0,000	6477	12,00	6477	1,56
300_2	6807	0,000	6807	3	5,73	3	5,96	0,000	6807	1249,00	6807	5,03
300_3	6430	0,000	6430	0	2,62	0	1,52	0,000	6430	1,00	6430	1,56
400_1	7414	0,000	7414	0	4,70	0	0,99	0,000	7414	1,00	7414	2,83
400_2	7782	0,000	7782	2	37,21	7	5,84	0,000	7782	1615,00	7782	26,55
400_3	7604	0,000	7604	1	45,28	1	7,76	0,000	7604	93,00	7604	5,66
500_1	8272	0,000	8272	0	7,66	0	2,36	0,000	8272	3,00	8272	4,13
500_2	8392	0,000	8392	2	151,24	1	5,46	0,000	8392	121,00	8392	4,45
500_3	8502	0,000	8502	0	16,37	0	6,40	0,000	8502	127,00	8502	9,03
600_1	9037	0,000	9037	0	3,82	0	2,29	0,000	9037	1,00	9037	6,03
700_1	9786	0,000	9786	0	30,59	0	4,33	0,000	9786	2,00	9786	10,04
800_1	10333	0,000	10333	0	109,56	0	6,39	0,000	10333	7,00	10333	13,39
900_1	10918	0,000	10918	0	90,54	0	7,87	0,000	10918	7,00	10918	17,10
1000_1	11408	0,000	11408	5	4754,58	1	24,98	0,008	11408	⋆	11408	25,19
2000_1	15670	31,901	15670	0	∗	0	93,91	0,000	15670	743,00	15670	189,06
2000_2	16242	48,063	16253	0	∗	2	551,76	0,018	16242	⋆	16458	∗
2000_3	16670	56,769	16689	0	∗	0	690,50	0,011	16670	⋆	16939	∗
2000_4	16370	85,015	16373	0	∗	2	661,96	0,012	16370	⋆	16589	∗
2000_5	16521	57,990	16530	0	∗	2	596,43	0,006	16521	13212,00	16521	612,55

(∗) Execução interrompida ao atingir o tempo limite de 14400 segundos (4 horas)
(⋆) Execução interrompida ao atingir o tempo limite de 86400 segundos (24 horas)
Obs.: os dados das colunas referentes ao VNSL+ASG foram extraídos de Freitas [2011] e Andrade & Freitas [2013]

de otimalidade. No entanto, devido as dificuldade inerente às instâncias deste conjunto e devido ao tamanho das mesmas, o GCBC não foi capaz de avaliar LP(P_2) para as instâncias 1000_4 e 1000_5, no limite de tempo preestabelecido. Para as outras duas instâncias em que o GCBC atingiu o tempo limite, observamos que os gaps de dualidade fornecidos por este são melhores do que os obtidos pelo NDRC/BC.

Finalmente, os resultados referentes às instâncias LH-R são apresentados na Tabela 4.11. As colunas dessa tabela fornecem as mesmas informações das colunas que compõe a Tabela 4.10. Para as instâncias LH-R, excetuando-se as instâncias com $|V| = 100$ e as instâncias em que o algoritmo não atingiu o tempo limite, o CHOCO obteve tempos de CPU consideravelmente melhores em todos os outros casos (19 de 30 instâncias). Além disso, somente o CHOCO foi capaz de retornar o valor ótimo para a instância 900_3. Comparando isoladamente os métodos GCBC e NDRC/BC, observamos uma vantagem para o GCBC, que conseguiu resolver em menos tempo 17

Tabela 4.10. Comparativo entre os métodos da literatura e o GCBC para as instâncias LH-E

		GCBC					NDRC/BC					CHOCO	
Id	z	lb	ub	%gap	Nós	t(s)	lb	ub	%gap	Nós	t(s)	ub	t(s)
100_3	11640	11640,000	11640	0,000	0	1,75	11640,000	11640	0,000	1	7,66	11640	10,593
100_4	11748	11748,000	11748	0,000	0	1,14	11748,000	11748	0,000	0	2,68	11748	1,298
100_5	12345	12345,000	12345	0,000	0	1,48	12345,000	12345	0,000	0	4,81	12345	1,633
200_3	15177	15177,000	15177	0,000	3	24,03	15177,000	15177	0,000	3	65,6	15177	19,189
200_4	14671	14671,000	14671	0,000	0	10,68	14671,000	14671	0,000	1	28,79	14671	46,607
200_5	14967	14967,000	14967	0,000	2	9,74	14967,000	14967	0,000	3	41,14	14967	47,052
300_3	15691	15691,000	15691	0,000	23	32,56	15691,000	15691	0,000	171	260,5	19360	*
300_4	19731	19731,000	19731	0,000	4	68,63	19731,000	19731	0,000	3	274,5	25572	*
300_5	18492	18492,000	18492	0,000	0	37,7	18492,000	18492	0,000	0	31,85	18492	550,265
400_3	20454	20454,000	20454	0,000	5	337,28	20454,000	20454	0,000	10	471,27	25855	*
400_4	21005	21005,000	21005	0,000	0	158,29	21005,000	21005	0,000	0	285,79	21005	266,745
400_5	24911	24911,000	24911	0,000	3	334,56	24911,000	24911	0,000	5	684,95	34174	*
500_3	20584	20584,000	20584	0,000	156	746	20584,000	20584	0,000	258	1169,29	25648	*
500_4	27124	27124,000	27124	0,000	5	1003,41	27124,000	27124	0,000	18	1885,4	36339	*
500_5	27389	27389,000	27389	0,000	0	807,76	27389,000	27389	0,000	0	755,56	27389	526,434
600_3	24583	24583,000	24583	0,000	8	1939,91	24583,000	24583	0,000	61	2411,82	33882	*
600_4	27141	27141,000	27141	0,000	0	942,2	27141,000	27141	0,000	5	4336,73	35602	*
600_5	24923	24923,000	24923	0,000	0	674,22	24923,000	24923	0,000	19	9711,18	30804	*
700_3	26024	26024,000	26024	0,000	42	3848,63	26024,000	26024	0,000	282	12292,07	37146	*
700_4	28592	28592,000	28592	0,000	9	3307,98	28592,000	28592	0,000	4	2982,29	37316	*
700_5	31402	31402,000	31402	0,000	17	5877,97	31380,561	32058	2,100	0	*	43808	*
800_3	27433	27433,000	27433	0,000	21	3501,19	27399,583	28912	5,200	0	*	34251	*
800_4	28399	28399,000	28399	0,000	127	5479,05	28399,000	28399	0,000	633	7536,54	38928	*
800_5	32253	32253,000	32253	0,000	3	6496,47	32239,245	32670	1,300	0	*	43360	*
900_3	-	29641,303	29970	1,097	4	*	29595,260	30845	4,100	0	*	40058	*
900_4	34054	34054,000	34054	0,000	3	12668,8	33922,172	35847	5,400	0	*	48348	*
900_5	33355	33355,000	33355	0,000	2	7771,19	33295,609	34223	2,700	0	*	44853	*
1000_3	-	30310,879	32363	6,341	0	*	31928,307	34115	6,400	0	*	45533	*
1000_4	-	0,000	33774	100,000	0	*	32932,910	33502	1,700	0	*	44693	*
1000_5	-	0,000	50247	100,000	0	*	43960,657	46971	6,400	0	*	63095	*

(*) Execução interrompida ao atingir o tempo limite de 14400 segundos (4 horas)

das 30 instâncias do conjunto, ao passo que o NDRC/BC resolveu gastando menos tempo apenas 8. No entanto, para a instância 1000_4, o GCBC não consegue resolver LP(P_2) por GC até o tempo limite preestabelecido, ao passo que o NDRC/BC retorna um gap de dualidade de 1,645%.

Uma das razões pelas quais o CHOCO apresenta dificuldade em resolver as instâncias euclidianas com restrições de grau mais apertadas – instâncias DE e LH-E – é a transitividade da distância entre os vértices. Sejam i, j e k vértices do grafo G, e assumindo que i é proximo de j e j é próximo de k, então i também é próximo de k. Isso impacta o comportamento da Relaxação Lagrangeana utilizada pelo método de Fages et al. [2013], devido à baixa qualidade dos custos marginais, fazendo com que o número de arestas filtradas seja baixo. O mesmo não ocorre para as instâncias de custo aleatório, cujas distâncias entre os vértices não são transitivas, pois a inclusão (ou remoção) de uma aresta é propagada ao longo do algoritmo, fazendo com que a filtragem seja mais eficiente.

À luz dos resultados expostos nesta seção, podemos verificar que o GCBC foi superior ao NDRC/BC na maioria das instâncias testadas. O CHOCO, por sua vez, foi mais eficiente ao resolver instâncias aleatórias. Entretanto, este algoritmo não é robusto, pois, para resolver instâncias com custos Euclidianos, tal método demanda muito tempo de CPU. Por fim, verificamos que os resultados obtidos pelo VNSL+ASG são bastante inferiores aos outros métodos em praticamente todas as instâncias de teste.

Tabela 4.11. Comparativo entre os métodos da literatura e o GCBC para as instâncias LH-R

		GCBC					NDRC/BC					CHOCO	
Id	z	lb	ub	%gap	Nós	t(s)	lb	ub	%gap	Nós	t(s)	ub	t(s)
100_3	2149	2149,000	2149	0,000	7	0,90	2149,000	2149	0,000	10	9,31	2149	5,01
100_4	2327	2327,000	2327	0,000	0	0,48	2327,000	2327	0,000	0	1,53	2327	0,57
100_5	2419	2419,000	2419	0,000	2	0,53	2419,000	2419	0,000	9	6,50	2419	4,46
200_3	3108	3108,000	3108	0,000	65	12,46	3108,000	3108	0,000	34	45,50	3108	8,71
200_4	3144	3144,000	3144	0,000	0	3,49	3144,000	3144	0,000	0	9,21	3144	2,76
200_5	3336	3336,000	3336	0,000	4	6,77	3336,000	3336	0,000	3	45,55	3336	5,65
300_3	4110	4110,000	4110	0,000	0	11,49	4110,000	4110	0,000	0	94,71	4110	9,95
300_4	4206	4206,000	4206	0,000	20	25,12	4206,000	4206	0,000	17	237,47	4206	15,54
300_5	4342	4342,000	4342	0,000	0	21,72	4342,000	4342	0,000	4	85,22	4342	9,04
400_3	5117	5117,000	5117	0,000	196	534,11	5117,000	5117	0,000	290	968,75	5117	182,72
400_4	5233	5233,000	5233	0,000	7	219,32	5233,000	5233	0,000	49	420,22	5233	13,78
400_5	5356	5356,000	5356	0,000	17	234,75	5356,000	5356	0,000	41	361,87	5356	25,15
500_3	5999	5999,000	5999	0,000	140	794,59	5999,000	5999	0,000	277	4390,80	8111	*
500_4	6194	6194,000	6194	0,000	29	1191,26	6194,000	6194	0,000	14	431,54	6194	310,84
500_5	6168	6168,000	6168	0,000	54	231,14	6168,000	6168	0,000	76	579,99	6168	33,18
600_3	6888	6888,000	6888	0,000	120	1729,73	6888,000	6888	0,000	75	2823,01	8696	*
600_4	7098	7098,000	7098	0,000	849	4660,84	7098,000	7098	0,000	838	8891,97	7098	740,12
600_5	7230	7230,000	7230	0,000	48	1371,76	7230,000	7230	0,000	61	908,84	7230	50,74
700_3	7910	7909,518	7911	0,019	147	*	7910,000	7910	0,000	163	8935,59	7910	1336,10
700_4	-	8073,765	8075	0,015	185	*	8073,633	8076	0,029	183	*	10606	*
700_5	8109	8109,000	8109	0,000	34	1849,87	8109,000	8109	0,000	32	1216,92	8109	62,48
800_3	8855	8854,600	8858	0,038	76	*	8855,000	8855	0,000	56	4604,00	8855	651,27
800_4	8955	8955,000	8955	0,000	172	13855,33	8954,078	8960	0,066	80	*	8955	1271,77
800_5	9111	9110,628	9112	0,015	128	*	9111,000	9111	0,000	87	3823,75	11740	*
900_3	9792	9335,329	10334	9,664	0	*	9792,000	9838	0,468	0	*	9792	11009,83
900_4	-	9969,968	10226	2,504	40	*	9970,237	10113	1,412	49	*	12504	*
900_5	10025	10025,000	10025	0,000	146	10914,15	10025,000	10025	0,000	83	10452,19	10025	1018,68
1000_3	-	10783,110	11178	3,533	4	*	10782,973	11275	4,364	1	*	12779	*
1000_4	-	0,000	11610	100,000	0	*	10914,424	11097	1,645	15	*	13549	*
1000_5	11061	10232,285	11235	8,925	0	*	11061,000	11061	0,000	182	9945,34	13692	*

(∗) Execução interrompida ao atingir o tempo limite de 14400 segundos (4 horas)

Capítulo 5

Conclusão e considerações finais

Nesta trabalho, avaliamos o uso de três algoritmos *Branch-and-cut-and-price* (BCP) para o Problema da Árvore Geradora de Custo Mínimo com Restrição de Grau (PAGMRG). Esse problema consiste em, dado um grafo não direcionado e valorado em suas arestas, encontrar uma Árvore Geradora Mínima que satisfaça as restrições de grau impostas aos vértices.

Até então, os algoritmos presentes na literatura para tratar o problema, em sua maioria, baseiam-se em Relaxação Lagrangeana. Nesta dissertação, entretanto, estudamos o uso de Geração de Colunas para resolver o PAGMRG. Para isso, implementamos três algoritmos que fazem uso desta técnica. O primeiro deles, denominado GCBC, utiliza Geração de Colunas para resolver o nó raiz em algoritmo um *Branch-and-cut*. Nos outros dois algoritmos, geramos colunas e planos de corte ao longo da árvore de *Branch-and-bound*. Esses dois algoritmos se diferenciam com relação ao processo de gerar colunas. Em um deles, as colunas são geradas ao longo de toda a árvore de enumeração, ao passo que no outro, geramos colunas enquanto o número de variáveis envolvidas (implícita e explicitamente) em um determinado nó ainda é grande.

Apresentamos, também, dois novos conjuntos de instâncias para o PAGMRG. Esses conjuntos são, em teoria, mais difíceis do que os encontrados na literatura, pois impomos que a soma dos graus máximos permitidos para os vértices seja igual à $2(n-1)$, valor correspondente à soma dos graus dos vértices em qualquer árvore geradora. Essa característica faz com que a busca local utilizada para encontrar soluções viáveis de boa qualidade em quase todos os trabalhos que abordam o PAGMRG se torne inócua. Através dos testes computacionais apresentados neste trabalho, constatamos que, de fato, as instâncias aqui propostas são mais difíceis do que as da literatura, fato este comprovado pelos tempos de CPU demandados resolver essas instâncias tanto pelos métodos aqui propostos quanto pelos melhores algoritmos da literatura. Além disso,

o valor ótimo para algumas das instâncias propostas nesta dissertação ainda não foi encontrado.

Dos três algoritmos aqui propostos, o GCBC apresentou resultados superiores. Comparando com os métodos da literatura, observamos que, de um modo geral, o GCBC também apresentou melhores resultados. O algoritmo GCBC apresentado nesta dissertação se destaca ao resolver instâncias com custos Euclidianos e limites mais restritos de grau para os vértices. Para instâncias desse tipo, os tempos de processamento obtidos pelo GCBC foram consideravelmente inferiores aos obtidos pelos métodos até então propostos para o PAGMRG. Para instâncias com custos aleatórios, entretanto, o método, em geral, não obteve melhores resultados do que alguns algoritmos da literatura. Um fator diretamente ligado à eficiência do GCBC é a dimensão do grafo de entrada da instância. Para instâncias de grande porte, com o número de vértices superior a 900, o GCBC na maioria das vezes não é capaz resolver a Relaxação Linear da formulação no nó raiz da árvore de enumeração.

Para trabalhos futuros, seria interessante investigar outras vizinhanças para a Busca Local, objetivando encontrar, de forma rápida, boas soluções primais para as instâncias propostas nesta dissertação, cuja busca local 1-vizinhança é ineficaz. Uma outra linha de pesquisa seria avaliar a resolução do PAGMRG através do método *Local Branching* proposto por Fischetti & Lodi [2003], utilizando-o em conjunto com o arcabouço proposto nesta dissertação. A paralelização dos algoritmos aqui propostos, com o objetivo de reduzir ainda mais os tempos de execução dos métodos, também pode ser estudada. Isso pode ser feito, por exemplo, resolvendo os nós da árvore de enumeração em diferentes núcleos de processamento ou executando, de forma paralela, os problemas de separação e o problema de precificação.

Referências Bibliográficas

Achterberg, T.; Koch, T. & Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, 33(1):42–54.

Ahuja, R. K.; Magnanti, T. L. & Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice hall.

Alves, C. & de Carvalho, J. V. (2008). A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, 35(4):1315–1328.

Andrade, R.; Lucena, A. & Maculan, N. (2006). Using lagrangian dual information to generate degree constrained spanning trees. *Discrete Applied Mathematics*, 154(5):703–717.

Andrade, R. C. & Freitas, A. T. (2013). Disjunctive combinatorial branch in a subgradient tree algorithm for the DCMST problem with VNS-Lagrangian bounds. *Electronic Notes in Discrete Mathematics*, 41(0):5–12.

Barr, R.; Glover, F. & Klingman, D. (1979). Enhancements of spanning tree labeling procedures for network optimization. *INFOR*, 17(1):16–34.

Caccetta, L. & Hill, S. (2001). A branch and cut method for the degree-constrained minimum spanning tree problem. *Networks*, 37(2):74–83.

Cunha, A. & Lucena, A. (2007). Lower and upper bounds for the degree-constrained minimum spanning tree problem. *Networks*, 50(1):55–66.

Cunha, A. S. (2006). *Árvores ótimas em grafos: modelos, algoritmos e aplicações*. Tese de doutorado, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Dantzig, G. B.; Fulkerson, D. R. & Johnson, S. M. (1954). Solution of a large scale traveling salesman problem. *Operations Research*, 2(4):393–410.

Dantzig, G. B. & Wolfe, P. (1960). Decomposition principle for linear programs. *Operations research*, 8(1):101–111.

de Carvalho, J. V. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273.

de Souza, M. C. & Martins, P. (2008). Skewed VNS enclosing second order algorithm for the degree constrained minimum spanning tree problem. *European Journal of Operational Research*, 191(3):677–690.

Desaulniers, G.; Desrosiers, J. & Solomon, M. M., editores (2005). *Column Generation*, volume 5 of *GERAD 25th anniversary series*. Springer.

Desrochers, M. & Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13.

Desrosiers, J.; Soumis, F. & Desrochers, M. (1984). Routing with time windows by column generation. *Networks*, 14(4):545–565.

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

Edmonds, J. (1971). Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136.

Edmonds, J. & Johnson, E. (1970). Matching: A well-solved class of integer linear programs. Em Guy, R.; Hanani, H.; Sauer, N. & Schonheim, J., editores, *Proceedings of the Calgary International Conference on Combinatorial Structures and their Applications*. Gorden and Breach.

Fages, J.-G.; Lorca, X. & Rousseau, L.-M. (2013). Une approche basée sur les contraintes pour résoudre le problème d'arbre recouvrant de coût minimum avec contraintes de degré. Em *Actes JFPC*, pp. 119–122.

Fischetti, M. & Lodi, A. (2003). Local branching. *Mathematical programming*, 98(1-3):23--47.

Freitas, A. T. (2011). Árvore de subgradiente com pré-fase VNS-Lagrangeana para o problema da árvore geradora mínima com restrição de grau máximo nos vértices. Dissertação de mestrado, Departamento de Computação, Universidade Federal do Ceará, Fortaleza, CE, Brasil.

Fukasawa, R.; Longo, H.; Lysgaard, J.; de Aragão, M. P.; Reis, M.; Uchoa, E. & Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511.

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

Gavish, B. (1982). Topological design of centralized computer networks - formulations and algorithms. *Networks*, 12(4):355–377.

Geoffrion, A. (1974). Lagrangean relaxation for integer programming. Em Balinski, M., editor, *Approaches to Integer Programming*, volume 2 of *Mathematical Programming Studies*, pp. 82–114. Springer Berlin Heidelberg.

Gilmore, P. & Gomory, R. (1963). A linear programming approach to the cutting stock problem-part ii. *Operations Research*, 11(6):863--888.

Gilmore, P. C. & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859.

Gomory, R. E. & Hu, T. C. (1961). Multi-terminal network flows. *Journal of the Society for Industrial & Applied Mathematics*, 9(4):551–570.

Held, M. & Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25.

Knowles, J. & Corne, D. (2000). A new evolutionary approach to the degree-constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4(2):125–134.

Koch, T. & Martin, A. (1998). Solving steiner tree problems in graphs to optimality. *Networks: An International Journal*, 32(3):207--232.

Kruskal, J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50.

Ladányi, L.; Ralphs, T. K. & Trotter Jr, L. E. (2001). Branch, cut, and price: Sequential and parallel. Em *Computational Combinatorial Optimization*, pp. 223–260. Springer.

Letchford, A. N.; Reinelt, G. & Theis, D. O. (2004). A faster exact separation algorithm for blossom inequalities. Em *Integer Programming and Combinatorial Optimization*, pp. 196–205. Springer.

Lucena, A. (1992). Steiner problem in graphs: Lagrangean relaxation and cutting-planes. *COAL Bulletin*, 21(2):2–8.

Lucena, A. (2005). Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140(1):375–410.

Lucena, A.; Maculan, N. & Cunha, A. S. (2011). Relax-and-cut as a preprocessor and warm starter to branch-and-cut. Em Mahjoub, A. R., editor, *Progress in Combinatorial Optimization*, capítulo 5, pp. 171–197. Wiley.

Lucena, A. & Resende, M. G. (2004). Strong lower bounds for the prize collecting steiner problem in graphs. *Discrete Applied Mathematics*, 141(1):277–294.

Magnanti, T. L. & Wolsey, L. A. (1995). Optimal trees. Em M.O. Ball, T.L. Magnanti, C. M. & Nemhauser, G., editores, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pp. 503–615. Elsevier.

Martinez, L. C. & Cunha, A. S. (2014). The min-degree constrained minimum spanning tree problem: Formulations and branch-and-cut algorithm. *Discrete Applied Mathematics*, 164(1):210–224.

Narula, S. & Ho, C. (1980). Degree-constrained minimum spanning tree. *Computers & Operations Research*, 7(4):239–249.

Nemhauser, G. L. & Park, S. (1991). A polyhedral approach to edge coloring. *Operations Research Letters*, 10(6):315–322.

Obruča, A. (1968). Spanning tree manipulation and the travelling salesman problem. *The Computer Journal*, 10(4):374–377.

Padberg, M. & Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100.

Padberg, M. W. & Wolsey, L. A. (1983). Trees and cuts. *North-Holland Mathematics Studies*, 75:511–517.

Papadimitriou, C. H. & Vazirani, U. V. (1984). On two geometric problems related to the travelling salesman problem. *Journal of Algorithms*, 5(2):231–246.

Prim, R. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401.

Ribeiro, C. & Souza, M. (2002). Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118(1):43–54.

Savelsbergh, M. & Volgenant, T. (1985). Edge exchanges in the degree-constrained minimum spanning tree problem. *Computers & Operations Research*, 12(4):341–348.

Uchoa, E.; Fukasawa, R.; Lysgaard, J.; Pessoa, A.; De Aragão, M. P. & Andrade, D. (2008). Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Mathematical Programming*, 112(2):443–472.

Valle, C. A.; Martinez, L. C.; Cunha, A. S. & Mateus, G. R. (2011). Heuristic and exact algorithms for a min-max selective vehicle routing problem. *Computers & Operations Research*, 38(7):1054–1065.

Volgenant, A. (1989). A lagrangean approach to the degree-constrained minimum spanning tree problem. *European Journal of Operational Research*, 39(3):325–331.

Zhou, G. & Gen, M. (1997). A note on genetic algorithms for degree-constrained spanning tree problems. *Networks*, 30(2):91–95.