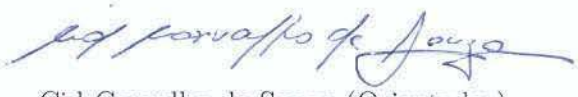
O método de geração de colunas aplicado a problemas de otimização em grafos

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Edna Ayako Hoshino e aprovada pela Banca Examinadora.

Campinas, 22 de dezembro de 2009.



Cid Carvalho de Souza (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Miriam Cristina Alves – CRB8 / 5094

Hoshino, Edna Ayako

H793m O método de geração de colunas aplicado a problemas de otimização em grafos/Edna Ayako Hoshino -- Campinas, [S.P. : s.n.], 2009.

Orientador : Cid Carvalho de Souza

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação .

1. Otimização combinatória. 2. Programação inteira. 3. Problema de roteamento de veículos. 4. Teoria dos grafos. I. Souza, Cid Carvalho de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Column generation technique applied to graph optimization problems

Palavras-chave em inglês (Keywords): 1. Combinatorial optimization. 2.Integer programming. 3. Vehicle routing problem. 4. Graph theory.

Área de concentração: Otimização combinatória

Titulação: Doutora em Ciência da Computação

Banca examinadora: Prof. Dr. Cid Carvalho de Souza (IC-UNICAMP)
Prof. Dr. Eduardo Uchoa Barboza (UFF)
Prof. Dr. Alexandre Salles da Cunha (DCC-UFMG)
Prof. Dr. Flávio Keidi Miyazawa (IC-UNICAMP)
Prof. Dr. Orlando Lee (IC-UNICAMP)

Data da defesa: 22/12/2009

Programa de Pós-Graduação: Doutorado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 22 de dezembro de 2009, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Eduardo Uchoa Barboza
Engenharia de Produção / UFF

Prof. Dr. Alexandre Salles da Cunha
Depto. Ciência da Computação / UFMG

Prof. Dr. Flávio Keidi Miyazawa
IC / UNICAMP

Prof. Dr. Orlando Lee
IC / UNICAMP

Prof. Dr. Cid Carvalho de Souza
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

O método de geração de colunas aplicado a problemas de otimização em grafos

Edna Ayako Hoshino

novembro de 2009

Banca Examinadora:

- Cid Carvalho de Souza (Orientador)
- Eduardo Uchoa Barboza, Dep. de Engenharia de Produção, Universidade Federal Fluminense (UFF)
- Alexandre Salles da Cunha, Dep. Ciência da Computação, Universidade Federal de Minas Gerais (UFMG)
- Flávio Keidi Miyazawa, Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)
- Orlando Lee, Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)
- Yoshiko Wakabayashi (suplente), Dep. Ciência da Computação, Universidade de São Paulo (USP)
- Zanoni Dias (suplente), Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)

Resumo

Nesta tese, dois problemas de otimização combinatória em grafos são modelados por programação linear inteira e resolvidos através de técnicas de geração de colunas. Os dois casos correspondem a generalizações de problemas clássicos em grafos e que ocorrem em muitas situações práticas. O primeiro, chamado problema dos anéis-estrelas capacitados, é uma generalização do problema de roteamento de veículos e modela situações reais encontradas nas áreas de logística de distribuição e de transporte. O segundo, conhecido por problema da coloração particionada, generaliza o problema da coloração de vértices em grafos e ocorre em aplicações no projeto de redes ópticas.

As formulações de programação linear inteira desenvolvidas neste trabalho para modelar ambos os problemas estão relacionadas à técnica da decomposição de Dantzig-Wolfe e usam uma quantidade exponencialmente grande de variáveis de decisão. Nestas formulações, cada uma das variáveis representa uma estrutura específica do problema sendo estudado. No problema dos anéis-estrelas capacitados, cada variável está associada a um anel-estrela e, no problema da coloração particionada, a um conjunto independente. As relaxações lineares destes tipos de modelos, em geral, apresentam limitantes duais mais apertados que outros modelos compactos, isto é, definidos para um número polinomial de variáveis.

Nesta tese, nós avaliamos estas novas formulações, comparando-as com outros modelos conhecidos para os problemas estudados. Além disso, nos dois casos, projetamos e implementamos algoritmos exatos do tipo *branch-and-price* e/ou *branch-and-cut-and-price* capazes de computar os modelos propostos. Experimentos computacionais foram realizados com estes algoritmos que confirmaram a adequação das técnicas aqui empregadas. Tanto para o problema dos anéis-estrelas capacitados quanto para o problema da coloração particionada, os resultados alcançados por nós foram comparados com aqueles reportados na literatura e mostraram que os algoritmos baseados em geração de colunas tiveram desempenho melhor que os algoritmos propostos anteriormente.

Abstract

In this thesis, two combinatorial optimization problems are modeled by integer linear programming and solved using the column generation technique. Both cases correspond to generalizations of classical problems in graphs that occur in many practical situations. The first, called capacitated ring-star problem, is a generalization of the vehicle routing problem and models real situations found in logistics and transportation. The second, known as the partition coloring problem, generalizes the vertex coloring problem in graphs and arises in design of fiber optics networks.

The integer linear programming formulations developed in this work to model both problems are related to the Dantzing-Wolfe decomposition and use exponential number of decision variables. In these formulations, each decision variable represents a specific structure of the problem under study. For the capacitated ring-star problem, each variable is assigned to a ring-star and, for the partition coloring problem, to an independent set. The linear relaxation of this kind of model in general leads to tighter dual bounds than the ones obtained from compact models, i.e., defined over a polynomial number of variables.

In this thesis, we evaluated both new formulations, comparing them to other known models for the respective problems. Moreover, in both cases, we designed and implemented exact *branch-and-price* and/or *branch-and-cut-and-price* algorithms that are able to solve the proposed models. Computational experiments were performed with these algorithms and showed that the used techniques were adequate. Both for the capacitated ring-star problem and for the partition coloring problem, we compared our results with those reported in the literature and showed that the algorithms based on column generation outperformed the previous ones.

Dedicatória

Dedico esta tese à minha filha Letícia.

Agradecimentos

Eu gostaria de agradecer aos amigos, professores, colegas de trabalho e familiares que, embora de formas diferentes, contribuíram para a concretização desta tese e deram suporte profissional, acadêmico e emocional durante o meu doutoramento. Embora difícil de nomear todos aqui, é impossível não mencionar alguns deles.

Primeiramente, gostaria de agradecer ao professor Cid pela atenção e dedicação com que orientou o meu trabalho. Acreditar no potencial do aluno, incentivar a pesquisa, contribuir para o crescimento profissional, acadêmico e pessoal do orientando são algumas de suas atividades marcantes, pelas quais sou muito grata.

Gostaria de agradecer aos órgãos de fomento, em especial à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, que incentivam as pesquisas no País. Também gostaria de mencionar o apoio que recebi da Universidade Federal de Mato Grosso do Sul e dos meus colegas de trabalho, durante o período em que estive afastada de minhas atividades.

Meus agradecimentos ao IC pela infra-estrutura física e apoio fornecido durante o doutorado. Aproveito para agradecer ao Yuri Frota, pós-doutorando do IC, pela sua colaboração no trabalho conjunto que desenvolvemos em parte desta tese.

Durante estes anos de doutoramento, tive a sorte de ganhar grandes amigos, que em graus e formas diferentes deixaram marcas (mesmo que de café) nesta tese. Obrigada, amigos!

Por fim e não menos importante, gostaria de agradecer a minha família, em especial à Letícia que compreendeu a minha ausência em alguns momentos de sua vida e ao Vagner pelo carinho e pela revisão da tese, e aos meus pais que são os responsáveis pela minha formação pessoal e acadêmica.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Problemas de otimização combinatória ocorrem em muitas situações práticas, por exemplo nas áreas de logística de transporte e distribuição, de produção e escalonamento, de alocação de recursos humanos, entre outros. Muitos destes problemas são clássicos e bem estudados como, por exemplo, roteamento de veículos e coloração de vértices. Novas variantes destes problemas surgem à medida que aplicações exigem restrições adicionais ou relaxam algumas restrições do problema. Nesta tese, estudamos dois novos problemas introduzidos na literatura recentemente. O primeiro trata-se de uma variante do problema de roteamento de veículos e foi introduzido em 2005 [4] para uma aplicação que ocorre na área de projeto de redes de fibras óticas. O segundo problema estudado refere-se a uma variante da coloração de vértices introduzido por Li e Simha em 2000 e originalmente motivado por uma aplicação de alocação de comprimentos de onda em redes de fibras óticas [33].

Em geral, aplicações do problema de roteamento de veículos ocorrem nas áreas de logísticas de transporte e distribuição. Na versão clássica capacitada, uma frota de m veículos idênticos localizada em um depósito central deve distribuir um tipo de mercadoria para atender as demandas de todos os clientes localizados em pontos geograficamente distintos. Cada cliente possui uma demanda que deve ser atendida por exatamente um dos veículos, que por sua vez, pode atender vários clientes, desde que a carga transportada não ultrapasse o limite de capacidade de carga do veículo. Nas aplicações deste problema, existe um custo, chamado custo de roteamento, associado a cada par de pontos (clientes e/ou depósito). Desta forma, o percurso ou rota de um veículo, que começa no depósito, visita alguns clientes e retorna ao depósito, implica num custo que é dado pela soma dos custos de roteamento dos trechos percorridos pelo veículo. O problema consiste em determinar uma rota para cada veículo que respeite sua capacidade de carga, atenda a demanda de todos os clientes e minimize a soma dos custos das rotas.

Em algumas situações reais como, por exemplo, em redes de fibras óticas, o custo

de roteamento é muito caro. Nestes casos, é desejável que as rotas visitem um número menor de clientes de modo que os demais clientes tenham sua demanda atendida por um meio alternativo e mais barato de distribuição. No caso específico do projeto de redes de fibras óticas, deseja-se interligar um conjunto de pontos terminais a um servidor central usando-se fibras óticas, que são aterradas e cujo custo de instalação é muito caro. Para não comprometer toda a rede de comunicação quando um trecho dela é interrompida, usa-se uma estrutura de anel, que equivale ao conceito de rota, para interligar os terminais ao servidor central. Como há um limite no número de terminais que podem ser conectados ao anel, o projeto da rede usualmente consiste na determinação de vários anéis. Para diminuir os custos de implantação da rede, é comum nessas aplicações deixar alguns terminais fora dos anéis e conectá-los à rede indiretamente através de meios físicos mais baratos, como cabos coaxiais. Uma vez que os terminais conectados indiretamente em um anel lembram uma topologia estrela, o termo anel-estrela é usado para designar a configuração destas redes. Também é comum, nestas aplicações, incluir pontos intermediários na rede apenas para diminuir custos de implantação dos anéis para que sirvam de ponto de conexão aos terminais mantidos fora dos anéis. O projeto de redes de fibras óticas visa determinar os anéis-estrelas para interligar os terminais a um custo baixo, considerando os custos de implantação dos anéis e os custos de interligação indireta dos terminais via cabo coaxial. Esta variante do roteamento de veículos, chamada problema dos anéis-estrelas capacitados, surgiu motivada por este tipo de aplicação e é objeto de estudo da tese. Pelo nosso conhecimento, o único algoritmo exato para resolver este problema é um algoritmo *branch-and-cut* proposto por Baldacci et al. em [4], no mesmo artigo em que o problema foi introduzido.

O segundo problema estudado também ocorre em redes de fibras óticas, mas trata uma situação bem diferente. Considere que a rede já está implantada e interliga vários terminais. Nesta rede, um par de terminais comunicam-se através de um caminho ótico usando um comprimento de onda único por todo o caminho desde a fonte até o destino. A Figura 1.1 ilustra cinco caminhos óticos, três deles interligando os terminais a e g e os demais interligando f e c. Comprimentos de onda distintos estão representados na figura por estilos de linha diferentes.

A estrutura deste tipo de rede utiliza tecnologias como *Wavelength Division Multiplexing*, que permite transmissões simultâneas na mesma fibra utilizando-se comprimentos de onda diferentes. Essa característica contribui para um uso mais eficiente da rede, mas por outro lado dificulta o estabelecimento das comunicações, uma vez que caminhos óticos que passam por um mesmo par de terminais (os quais são chamados de caminhos óticos conflitantes) devem ser associados a comprimentos de onda distintos. Os caminhos óticos $p1$ e $p4$ ilustrados na Figura 1.1 são conflitantes por causa do trecho em comum interligando os terminais b e d. O problema de determinar os caminhos óticos para os pares de

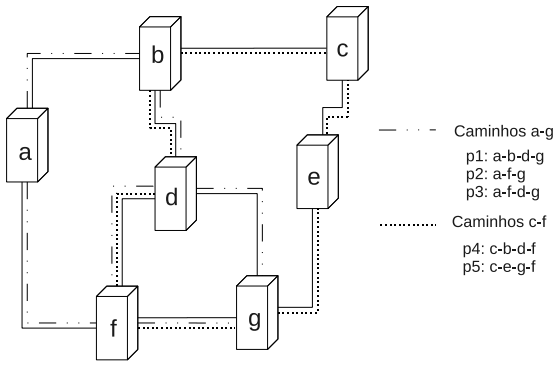


Figura 1.1: Exemplos de caminhos óticos

terminais que desejam comunicar-se entre si e comprimentos de onda aos caminhos óticos é chamado problema de roteamento e atribuição de comprimentos de onda, ou *routing and wavelength assigment problem* (RWA). Quando o conjunto de possíveis caminhos óticos entre os pares de terminais está pré-estabelecido e o critério de otimização é minimizar o número total de comprimentos de ondas, o problema é conhecido como RWA *off-line*. Esta variante foi resolvida por Li e Simha [33] usando-se uma estratégia de duas fases. Na primeira fase, um conjunto de k caminhos óticos possíveis para cada par de terminais que desejam comunicar-se entre si é estabelecido. Na segunda fase, um dos caminhos óticos (computados na fase anterior) e seu respectivo comprimento de onda são determinados para cada par de terminais. Esta última fase foi modelada por Li e Simha como uma variante da coloração de vértices, chamada problema de coloração particionada. Neste problema, um grafo e uma partição dos vértices do grafo são dados de entrada e o objetivo consiste em encontrar uma coloração própria e de cardinalidade mínima de apenas um subconjunto dos vértices do grafo de modo que exatamente um vértice de cada parte seja colorido. A redução de uma instância da segunda fase do RWA *off-line* em uma instância da coloração particionada consiste em construir um grafo de interferência $G = (V, E)$, em que cada caminho ótico torna-se um vértice em V e cada par de caminhos óticos conflitantes corresponde a uma aresta em E. A partição do grafo é definida colocando-se os vértices relativos a caminhos óticos interligando um mesmo par de terminais em uma mesma parte. A Figura 1.2 ilustra um exemplo de instância do problema de coloração particionada obtida aplicando-se a redução para os caminhos óticos da rede descritos na Figura 1.1.

Note que, quando a partição é formada por $|V|$ partes de cardinalidade unitária, um para cada vértice do grafo, o problema equivale ao problema clássico de coloração de vértices, do qual segue que a coloração particionada é NP-difícil. Esta variante de coloração de vértices é o segundo problema estudado nesta tese. Pelo nosso conhecimento, poucos trabalhos foram dedicados ao estudo deste problema. Li e Simha introduziram

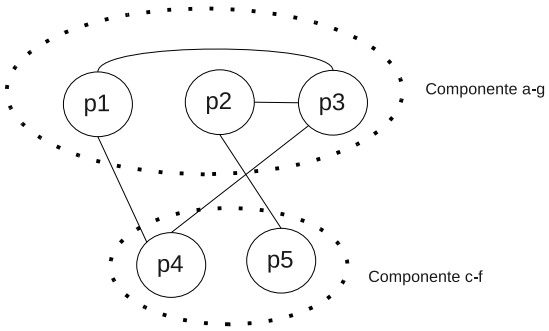


Figura 1.2: Segunda fase do RWA *off-line* modelado como uma instância da coloração particionada.

esta variante de coloração de vértices motivados pela sua aplicação ao problema do RWA *off-line* e propuseram heurísticas para a sua resolução, baseadas em heurísticas para a coloração de vértices. Mais tarde, Noronha e Ribeiro [41] propuseram uma heurística baseada em Busca Tabu para o PCP. O único algoritmo exato que pudemos encontrar na literatura para o problema é um algoritmo *branch-and-cut* proposto recentemente em [22].

Nesta tese, propomos novas formulações por programação linear inteira e novos algoritmos exatos para ambos os problemas. As formulações baseiam-se em modelos de cobertura utilizando-se uma quantidade exponencial de variáveis. Como já reportado na literatura para outros problemas, estes modelos têm a vantagem de apresentar limitantes duais apertados, motivo que nos levou a investigar a qualidade destas formulações para modelar os problemas estudados. Os algoritmos exatos propostos nesta tese baseiam-se no método da geração de colunas e consistem nos algoritmos *branch-and-price* e *branch-and-cut-and-price*. Ambos os algoritmos são avaliados por experimentos computacionais e seus desempenhos são comparados com aqueles reportados na literatura.

A tese está dividida em duas partes, cada uma delas discute um dos problemas estudados. Os fundamentos teóricos necessários para a compreensão da tese estão reunidos no Capítulo 2. A primeira parte da tese refere-se ao problema dos anéis-estrelas capacitados e está dividida nos Capítulos 3, 4, 5 e 6. No Capítulo 3 fazemos uma introdução para a primeira parte da tese e uma revisão da literatura e dos problemas correlatos. O Capítulo 4 apresenta algumas definições e a notação adotada, bem como uma nova formulação por cobertura de conjuntos de programação linear inteira (PLI). Os detalhes da implementação do algoritmo de *branch-and-cut-and-price* proposto e os resultados computacionais são analisados no Capítulo 5. O Capítulo 6, último desta parte da tese, apresenta as conclusões sobre os nossos estudos relativos ao problema dos anéis-estrelas capacitados. Os Capítulos 7, 8, 9 e 10 compreendem a segunda parte da tese referente ao problema da coloração particionada. O Capítulo 7 é dedicado para uma introdução à segunda parte da

tese, onde trabalhos correlatos e a imersão de nosso trabalho são discutidos. No Capítulo 8 revisamos os modelos matemáticos para o problema da coloração particionada e apresentamos uma nova formulação. No Capítulo 9 descrevemos os detalhes do algoritmo de *branch-and-price* que desenvolvemos para resolver o problema da coloração particionada e comparamos seu desempenho com o algoritmo *branch-and-cut* de Frota et al. Comentários finais sobre o trabalho realizado para o problema da coloração particionada e direções para pesquisas futuras são dados no Capítulo 10. Por fim, conclusões finais sobre a tese contendo nossas contribuições são apresentadas no Capítulo 11.

Capítulo 2

Fundamentação Teórica

Este capítulo trata da fundamentação teórica de vários aspectos da tese. As duas primeiras seções cobrem os principais conceitos e algoritmos relacionados a problemas de programação linear inteira e seguem as notações adotadas no livro [49]. A última seção é dedicada a algoritmos para o problema do caminho mínimo em grafos, os quais são usados na tese.

2.1 Programação Linear Inteira

Dado um conjunto finito $N = \{1, \ldots, n\}$, pesos c_j para cada $j \in N$ e uma coleção $\mathcal{F}$ de subconjuntos de N, um **problema de otimização combinatória** consiste em encontrar um subconjunto em $\mathcal{F}$ de peso mínimo, ou seja,

$$(COP) \min_{S \in \mathcal{F}} \sum_{j \in S} c_j.$$

Quando a coleção $\mathcal{F}$ pode ser descrita por um conjunto de inequações lineares temos um problema de **programação linear** (PL), cuja forma padrão é:

$$\begin{aligned} & \min \sum_{j \in N} c_j x_j \\ \text{sujeito a} \quad & \sum_{j \in N} a_{ij} x_j = b_i, \forall i \in M \\ & x_j \geq 0, \forall j \in N, \end{aligned} \tag{2.1}$$

onde x_j são as **variáveis de decisão** e M é o conjunto de índices das restrições (lineares) sobre estas variáveis que definem o problema. Para cada $i \in M$, as componentes do vetor a_i denotam os coeficientes de cada variável na restrição correspondente, enquanto b_i é refere-se ao seu termo independente.

Alternativamente, podemos escrever o problema de PL na seguinte forma matricial:

$$\begin{aligned} & \min cx \\ \text{sujeito a} \quad & Ax = b \\ & x \geq 0, \end{aligned} \tag{2.2}$$

onde $c : 1 \times n$, $x : n \times 1$, $A : m \times n$ e $b : m \times 1$. Em um problema de PL, os vetores c e b e a matriz A são constantes e definem o tamanho da entrada. A matriz A é denominada **matriz de coeficientes**, o vetor b define o **lado direito das restrições** (rhs), cx é chamada **função objetivo** e o vetor c é o **custo das variáveis** na função objetivo. Em algumas situações, usamos a notação $A_{.j}$ para denotar a j-ésima coluna da matriz A. A i-ésima linha de A é denotada por $A_{i.}$.

Um vetor x que satisfaz todas as restrições do problema de PL é chamado **solução viável**. O conjunto das soluções viáveis define um espaço do R^n chamada **região de viabilidade**. Uma solução viável que minimiza o valor da função objetivo é chamada **solução ótima**. O valor da função objetivo da solução ótima é chamado **valor ótimo**.

Um problema de PL pode ser resolvido por diferentes algoritmos, como o método *Simplex*, o método elipsóide e o algoritmo de pontos interiores, sendo estes dois últimos de complexidade polinomial no tamanho da entrada. O método *Simplex* é o mais utilizado na prática e é parte de muitos pacotes comerciais. Embora seu tempo de execução possa crescer exponencialmente com o tamanho da entrada, na prática, resolve muitos problemas em um número polinomial de iterações. A descrição do funcionamento e da corretude deste algoritmo depende de vários conceitos e foge do escopo desta tese. Para mais detalhes veja por exemplo [6, 40]. No entanto, descrevemos a seguir os principais resultados conhecidos nos quais o método *Simplex* se baseia e que são importantes para a compreensão dos métodos utilizados na tese.

1. A região de viabilidade de um PL é um poliedro convexo.
2. Se o poliedro de um PL é vazio dizemos que o PL é **inviável**.
3. Se o valor ótimo de um PL é finito dizemos que o PL é limitado e, caso contrário, é **ilimitado**.
4. Se o PL for limitado então existe uma solução ótima que é dada por um vértice (também chamada **ponto extremo**) deste poliedro.
5. Uma **solução básica** de um sistema linear $Ax = b$ é dado por uma divisão das variáveis de decisão em **básicas** e **não-básicas** (possivelmente após uma reordenação das colunas da matriz A e das entradas do vetor x) de modo que:

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$$

sendo $x_N = 0$, B inversível e $x_B = B^{-1}b \geq 0$.

Note que nestas condições temos que

$$Ax = \begin{bmatrix} B & N \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} = Bx_B + Nx_N = B(B^{-1}b) + N.0 = b.$$

A matriz B é chamada **base**.

6. Todo ponto extremo do poliedro $\{x \in R^n : Ax = b, x \geq 0\}$, ou seja, da região de viabilidade de um PL, corresponde a uma solução básica.

7. Um PL $\min\{cx : Ax = b, x \geq 0, x \in R^n\}$ pode ser resolvido através da resolução do PL abaixo escrito apenas em função das variáveis não-básicas (substituindo-se x_B por $B^{-1}b - B^{-1}Nx_N$):

$$\begin{array}{rl} & \min c_B B^{-1}b + (c_N - c_B B^{-1}N)x_N \\ \text{sujeito a} & B^{-1}Nx_N \leq B^{-1}b \\ & x_N \geq 0, \end{array} \tag{2.3}$$

$c_N - c_B B^{-1}N$ são chamados de **custos reduzidos** das variáveis não-básicas e denotados por $\overline{c}_N$.

8. Para um PL de minimização (maximização), a condição de otimalidade requer que todos os custos reduzidos sejam não negativos (não positivos).

9. Para saber se a condição de otimalidade é atendida num PL de minimização (maximização), o método *Simplex* executa o passo de **pricing** que consiste em encontrar a variável não-básica de menor (maior) custo reduzido.

10. A todo PL da forma

$$\begin{array}{rl} \text{(P)} & z^* = \min cx \\ \text{sujeito a} & Ax \geq b \\ & x \geq 0, \end{array} \tag{2.4}$$

chamado de **primal** está associado um PL **dual** dado por

$$\begin{array}{rl} \text{(D)} & w^* = \max ub \\ \text{sujeito a} & uA \leq c \\ & u \geq 0, \end{array} \tag{2.5}$$

onde cada variável de (P) está associada a uma restrição de (D) e cada variável de (D) (chamada **variável dual**) corresponde a uma restrição de (P). Uma solução viável é também chamada **primal viável**. Uma solução viável do problema dual é chamada **dual viável**.

11. O Teorema da **Dualidade Fraca** diz que, se x^* e u^* são soluções viáveis quaisquer de (P) e (D), respectivamente, então $cx^* \geq u^*b$.

12. Um par de problemas primal (P) e dual (D) tem uma das seguintes relações:

    - ambos são viáveis e limitados. Neste caso, tem-se que $z^* = w^*$ (**Teorema da Dualidade Forte**);
    - (P) é ilimitado e (D) é inviável;
    - (D) é ilimitado e (P) é inviável;
    - (P) e (D) são inviáveis.

13. Considere o par de problemas primal e dual, (P) e (D), reescrito na forma que se segue, usando-se variáveis s e t, chamadas **variáveis de folga**:

$$\begin{aligned} \text{(P)} \quad & z^* = \min cx \\ \text{sujeito a} \quad & Ax - s = b \\ & x \geq 0, s \geq 0, \end{aligned} \tag{2.6}$$

$$\begin{aligned} \text{(D)} \quad & w^* = \max ub \\ \text{sujeito a} \quad & uA + t = c \\ & u \geq 0, t \geq 0. \end{aligned} \tag{2.7}$$

    O **Teorema das folgas complementares** diz que (x^*, s^*) é uma solução ótima de (P) e (u^*, t^*) é uma solução ótima de (D) se e somente se $x^*t^* = 0$ e $u^*s^* = 0$.

14. Em uma iteração qualquer do método *Simplex*, as variáveis duais associadas às restrições são computadas por $u = c_B B^{-1}$. Com isso, o passo de *pricing* em um problema de minimização (maximização) se reduz a encontrar $j \in N$ tal que $\overline{c}_j = c_j - uA_{\cdot j}$ seja mínimo (máximo).

15. Se a condição de otimalidade não é satisfeita, o aumento da variável não-básica de custo reduzido mínimo (máximo), aqui denotada por x_s, pode provocar uma redução (aumento) no valor da função objetivo e, para PL com ótimo finito, causa a redução do valor de uma ou mais variáveis básicas. Suponha que x_r seja a primeira

variável básica, cujo valor é zerado. Então, uma nova base é obtida a partir da atual trocando-se a coluna correspondente a x_r por aquela associada a x_s, passo esse denominado de **pivoteamento**. Quando o PL é ilimitado, tem-se que $B^{-1}A_{.s} \leq 0$.

Com base nestes resultados, podemos sumarizar a idéia do algoritmo como segue. O método *Simplex* procura uma solução ótima apenas nos pontos extremos do poliedro, ou seja, calculando apenas as soluções básicas. Ele começa em uma base (que define um ponto extremo do poliedro) e sucessivamente muda de base em direção a um ponto extremo que pode melhorar o valor da função objetivo. A condição de otimalidade é detectada pelo passo de *pricing* que também é o responsável em encontrar a coluna correspondente a uma variável não básica que deve entrar na base para potencialmente melhorar o valor da função objetivo. O algoritmo 1 esboça o método *Simplex*.

Algoritmo 1: Algoritmo *Simplex*.

Input: Problema $P = \{\min cx : Ax = b, x \geq 0, x \in \mathbb{R}^n\}$
Output: Solução ótima x^* de P

Encontre uma base inicial B;
while *true* **do**
  /* passo de *pricing* */
  Escolha r tal que $r = \arg\min_{j \in N} \overline{c}_j = (c_j - c_B B^{-1} A_{.j})$
  if $\overline{c}_r \geq 0$ **then**
    Pare e retorne a solução básica ótima $x_B = B^{-1}b$ e $x_N = 0$
  if $y_r = B^{-1}A_{.r} \leq 0$ **then**
    Pare, P é ilimitado
  /* mudança de base */
  Escolha s tal que $s = \arg\min_{(y_r)_i > 0} (B^{-1})_{i.} b / (B^{-1})_{i.} A_{.r}$
  $B \leftarrow B \setminus \{A_{.s}\} \cup \{A_{.r}\}$;

Problemas de programação linear inteira são aqueles em que as variáveis de decisão são inteiras. Tais problemas são mais difíceis e precisam de algoritmos mais complexos que o *Simplex*. O restante desta seção revisa alguns destes algoritmos.

Deste ponto em diante, considere um problema (P) de PLI

$$z = \min\{c(x) : x \in S \cap \mathbb{Z}^n_+\},$$

onde S é um poliedro convexo como, por exemplo, $\{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$.

Note que os pontos extremos de S podem não ser pontos inteiros, ou seja, pontos em $\mathbb{Z}^n$. Seja $X = S \cap \mathbb{Z}^n_+$. Idealmente gostaríamos de conhecer o menor poliedro convexo

que contém X (ao qual dá-se o nome de **envoltória convexa** de X e denotada-se por $conv(X)$), uma vez que todos os pontos extremos deste poliedro são inteiros e, então, poderíamos utilizar o algoritmo *Simplex* para resolver o problema de PLI correspondente.

Alguns problemas de PLI, como o problema de cobertura de conjuntos, são bem estudados pois surgem como subproblemas em outros mais complexos. Dado um conjunto M de m elementos, digamos que $M = \{1, \ldots, m\}$, uma família $\mathcal{C}$ de subconjuntos de elementos de M e custos c_j associados a cada conjunto j em $\mathcal{C}$, o **problema da cobertura de conjuntos** consiste em encontrar um subconjunto S de conjuntos em $\mathcal{C}$ que cubram todos os elementos em M a um custo mínimo. Um elemento é dito **coberto** por S se pertencer a algum conjunto de S. O **modelo de cobertura** é uma formulação de PLI para modelar o problema de cobertura de conjuntos e consiste em:

$$\begin{aligned} & \min \sum_{j \in \mathcal{C}} c_j x_j \\ \text{sujeito a} \quad & \sum_{j \in \mathcal{C}} a_{ij} x_j \geq 1, \forall i \in M \\ & x_j \in \{0,1\}, \forall j \in \mathcal{C}, \end{aligned} \tag{2.8}$$

onde $a_{ij} = 1$ se e, somente, se o elemento i em M pertence ao conjunto j e $x_j = 1$ se e, somente, se j é escolhido para estar em S. As restrições (2.8), chamadas **restrições de cobertura**, garantem que cada elemento em M seja coberto pela solução.

2.1.1 Relaxações e Limitantes Primais e Duais

Sejam x^* uma solução viável e z^* o valor de cx^*. Para provar que x^* é uma solução ótima para (P) é necessário encontrar um limitante superior $\overline{z} \geq z$ e um inferior $\underline{z} \leq z$ tais que $\underline{z} = z^* = \overline{z}$. Em um PL de maximização (minimização), o limitante superior (inferior) é chamado **limitante dual** e o limitante inferior (superior) é chamado **limitante primal**. Portanto, um algoritmo para resolver um problema de PLI precisa encontrar limitantes duais e primais. Na verdade, estes limitantes são melhorados iterativamente até que $\overline{z} - \underline{z} < \epsilon$, onde ϵ é um valor não-negativo pequeno suficiente para provar a otimalidade, ou seja, $\overline{z} = \underline{z}$. A essa diferença entre os limitantes dá-se o nome de ***gap* de dualidade**.

O valor de qualquer solução viável dá um limitante primal para z. Um limitante dual é obtido por relaxações de (P). Um problema $z^R = \min\{f(x) : x \in T \cap R^n_+\}$ é uma **relaxação** de (P) se:

- $S \subseteq T$ e
- $f(x) \geq c(x), \forall x \in S$.

A **relaxação linear** de um problema de PLI consiste no problema de PL obtido ao relaxar-se (eliminar) as restrições de integralidade, ou seja, aquelas que obrigam que as variáveis de decisão sejam inteiras. No exemplo, a relaxação linear de (P) é $\min\{c(x) : x \in S\}$.

2.1.2 *Branch-and-Bound*

Considere novamente o conjunto S de soluções viáveis de um PLI. Seja $S_1 \cup S_2 \cup \ldots \cup S_K$ uma decomposição de S em conjuntos menores e de modo que $S = S_1 \cup S_2 \cup \ldots \cup S_k$ e seja $z^k = \min\{c(x) : x \in S_k\}$, para $k = 1, \ldots, K$. Logo, o valor de $z = \min_k z_k$. Note que cada um dos subproblemas pode ser decomposto em outros subproblemas menores sucessivamente, até que cada subconjunto contenha um número suficientemente pequeno de pontos para que o subproblema seja resolvido trivialmente. Uma forma de representar essa decomposição é através de uma **árvore de enumeração**, na qual cada nó representa um subconjunto do espaço de soluções e cada nó filho, uma componente de sua decomposição. A raiz da árvore representa o conjunto S e, portanto, todo o espaço de soluções do problema.

Enumerar todo o espaço de soluções em uma árvore de enumeração é custoso tanto em tempo quanto em espaço. Algoritmos de *branch-and-bound* tiram proveito da decomposição do problema e evitam a enumeração de toda a árvore de enumeração, através do uso de limitantes duais e primais, para resolver um problema de PLI.

Inicialmente, a árvore de enumeração de um algoritmo *branch-and-bound* contém apenas o nó raiz, que está associado ao problema (P). Um limitante dual é obtido resolvendo-se a relaxação linear de (P). Caso a solução ótima da relaxação linear de (P) seja inteira, a solução ótima de (P) foi encontrada. Caso contrário, uma operação de **branching** é realizada, decompondo S em dois ou mais subconjuntos $S_1, \ldots, S_K$, cada um deles definindo um novo nó na árvore de enumeração.

Em um algoritmo de *branch-and-bound*, limitantes duais e primais são gerados para cada nó da árvore de enumeração, à medida em que são criados. Um nó pode ser **podado** da árvore de enumeração se o problema associado a ele é inviável ou já se conhece sua solução ótima, ou se o limitante dual do nó é menor que um limitante primal para (P). Nós que não foram podados e ainda não foram decompostos por operação de *branching* são chamados **nós ativos**. Uma vez que os nós ativos são os únicos que podem gerar um limitante primal melhor para (P), a otimalidade pode ser detectada pelo algoritmo de *branch-and-bound* quando não há nós ativos ou quando o maior dos limitantes duais dos nós ativos for menor que o melhor limitante primal para (P). Portanto, quanto melhor forem os limitantes duais e primais, melhor será o desempenho de um algoritmo de *branch-and-bound*.

O Algoritmo 2 apresenta um esboço de um algoritmo *branch-and-bound* para o caso de árvores de enumeração binárias, isto é, aquelas onde cada *branching* dá origem a dois novos nós.

Algoritmo 2: Algoritmo de *Branch-and-Bound*.

Input: Problema $P = \{\min cx : x \in S \cap \mathbb{R}^n\}$
Output: Solução ótima ou melhor solução x^* de P e limitantes duais e primais

```
1  L = {P}; z̄ = +∞; z = −∞;
2  while L ≠ ∅ e z̄ − z ≥ ε do
3      /* passo de seleção de nó */
4      Escolha um problema P^i ∈ L
5      L = L − {P^i}
6      /* passo de otimização */
7      Resolva a relaxação linear de P^i
8      Seja x solução ótima e z o valor ótimo da relaxação linear de P^i
9      /* passo de poda */
10     if P^i inviável then
11         Poda por inviabilidade
12     else
13         if z ≥ z̄ then
14             Poda por limitante
15         else
16             if x inteiro then
17                 if z < z̄ then
18                     z̄ = z
19                     x* = x
20                 Poda por otimalidade
21             else
22                 /* passo de branching */
23                 Crie dois subproblemas P_1 e P_2 a partir de P^i usando uma regra de
                   branching e coloque em L
24     Atualize z = min{limitante dual de P' : P' ∈ L}
25 Devolva x*, z̄, z
```

O critério para escolha do próximo nó ativo a ser otimizado no passo (4) do algoritmo é conhecido como **critério de seleção de nó**. Os critérios mais usados são: busca por profundidade, busca em largura e melhor limitante. O primeiro consiste em otimizar na ordem definida pela busca em profundidade, enquanto o segundo pela ordem da busca

em largura. O último mantém a lista de nós ativos ordenada pelo limitante dual e otimiza primeiro o nó com melhor limitante dual (no caso de um problema de minimização (maximização), o nó com menor (maior) limitante dual é otimizado primeiro). Em geral, o critério de seleção de nó com melhor limitante apresenta um resultado melhor. No entanto, a busca por profundidade tem a vantagem de encontrar soluções primais mais cedo.

Strong Branching Em muitas situações pode haver mais de uma maneira de realizar a operação de *branching*, cada um deles causando um impacto diferente no desempenho do algoritmo de *branch-and-bound*. Na técnica de *strong branching* diferentes candidatos para a operação de *branching* são avaliados e, com base nesta avaliação, seleciona-se a variável de *branching* mais promissora. A avaliação consiste em realizar, para cada candidata, alguns passos de otimização dos nós filhos que seriam gerados ao ramificar-se a enumeração a partir daquela variável a fim de obter uma estimativa de seus limitantes duais, os quais são usados como critério para a escolha do candidato mais promissor. Deve-se notar que, no caso do método Simplex, estes passos de otimização correspondem a iterações (pivoteamentos) do método.

2.1.3 Algoritmos de Planos de Corte

Considere um PLI $P = \min\{cx : Ax = b, x \geq 0, x \in \mathbb{Z}^n\}$. Uma desigualdade $\pi x \leq \pi_0$ é uma **desigualdade válida** para P se $\pi x' \leq \pi_0$, para todo x' em $\{x \in Z^n : Ax = b, x \geq 0\}$. Dizemos que uma desigualdade é **violada** por um ponto x' se $\pi x' \leq \pi_0$.

Seja x^* uma solução ótima da relaxação linear de P. A idéia do algoritmo de planos de corte é encontrar uma desigualdade válida para P que é violada pelo ponto x^*. O problema de encontrar uma desigualdade válida violada por um ponto é chamado **problema de separação** e a rotina que encontra tal desigualdade é chamada **rotina de separação**. A inclusão da desigualdade violada como uma restrição adicional em P torna o poliedro mais apertado.

O algoritmo de planos de corte consiste em sucessivamente resolver a relaxação linear do problema e incluir uma desigualdade violada para separar o ponto ótimo fracionário até que uma solução inteira ótima seja encontrada.

2.1.4 *Branch-and-cut*

O algoritmo de *branch-and-cut* consiste em aplicar o algoritmo de planos de corte em cada nó da árvore de enumeração no algoritmo de *branch-and-bound*. Ele pode ser aplicado em duas situações. Em uma delas, a formulação por PLI do problema possui uma quantidade exponencial de restrições. Neste caso, apenas um subconjunto delas é usada inicialmente

e as demais restrições são adicionadas à medida que a rotina de separação encontra uma delas que é violada pela solução ótima. A outra situação ocorre quando deseja-se apertar a formulação incluindo mais desigualdades válidas ao modelo. Neste caso, algumas iterações do algoritmo de plano de corte são realizadas em cada nó da árvore de enumeração.

2.2 O Método da Geração de Colunas

O método da geração de colunas surgiu na década de 60, com a idéia introduzida por Dantzig e Wolfe para decompor programas lineares [10]. Posteriormente, o método foi empregado para resolver problemas de programação linear inteira, com o trabalho pioneiro de Gilmore e Gomory para o clássico problema de corte e empacotamento [26]. Desde a década de 80, muitos artigos utilizando com sucesso o método junto com algoritmos de *branch-and-bound* (*branch-and-price* e *branch-and-cut-and-price*) vêm sendo publicados e têm revelado a adequação do método na solução de alguns problemas difíceis [15, 43, 24, 47]. Nesta seção, fazemos uma revisão sobre o método e os algoritmos *branch-and-price* e *branch-and-cut-and-price*. Para mais detalhes do assunto veja, por exemplo, [5, 35, 14, 48].

Considere o seguinte programa linear, chamado **problema mestre** (PM) no contexto de geração de colunas:

$$\begin{aligned} & \min \sum_{j \in N} c_j x_j \\ \text{sujeito a} \quad & \sum_{j \in N} a_{ij} x_j \geq b_i, \forall i \in M \\ & x_j \geq 0, \forall j \in N. \end{aligned} \tag{2.9}$$

Como discutido na seção anterior, o passo de *pricing* em um algoritmo *Simplex* na resolução do programa linear consiste em encontrar $j = \arg\min\{\overline{c}_j = c_j - \sum_{i \in M} a_{ij}\pi_i\}$, onde π é a variável dual associada às restrições (2.9). Portanto, o passo de *pricing* toma tempo proporcional a $|N|$.

Métodos de geração de colunas são aplicados quando $|N|$ é muito grande, possivelmente exponencial no tamanho da entrada do problema. A idéia deste método consiste em resolver o problema mestre restrito a um subconjunto menor de colunas, $\tilde{N} \subseteq N$. Este problema menor é chamado **problema mestre restrito** (`PMR`). Obtido o valor das variáveis duais $\tilde{\pi}$ com a otimização do `PMR`, o método da geração de colunas realiza o passo de *pricing* resolvendo um outro problema de otimização, chamado **problema escravo** ou **problema de pricing**, ao invés de calcular, explicitamente, o custo reduzido $\overline{c}_j$, para cada $j \in N \setminus \tilde{N}$. Para tanto, as colunas a_j, $j \in N$, devem estar associadas a elementos de

um conjunto Λ e deve existir uma função $c : \Lambda \mapsto R$, que compute o valor de c_j. O passo de *pricing* pode, então, ser realizado resolvendo-se o seguinte problema de otimização $\overline{c}^* = \min\{c(a) - \tilde{\pi}a | a \in \Lambda\}$. Quando $\overline{c}^* \leq 0$, $\tilde{\pi}$ é dual viável, em relação ao `PM`, e portanto, a solução do `PMR` é também solução do `PM`. Caso contrário, a coluna a gerada pelo *pricing* (isto é, $a = \arg\min\{c(a) - \tilde{\pi}a | a \in \Lambda\}$) é incluída em $\tilde{N}$ e o `PMR` é novamente otimizado. O Algoritmo 3 é um esboço de um pseudocódigo para o método da geração de colunas.

Algoritmo 3: Algoritmo de geração de colunas.

Input: Problema $P = \{\min cx : Ax \geq b, x_j \geq 0, \forall j \in N\}$
Output: Solução ótima x^* de P

```
1  Seja Ñ ⊂ N
2  while true do
3      Resolva P^i = {min cx : Ax ≥ b, x_j ≥ 0, ∀j ∈ Ñ}
4      Seja x* solução primal ótima associada a P^i
5      Seja π̃ solução dual ótima associada a P^i
6      /* passo de pricing */
7      Resolva c̄* = min{c(a) − π̃a | a ∈ Λ}
8      if c̄* ≥ 0 then
9          Pare
10     Seja a = arg min{c(a) − π̃a | a ∈ Λ}
11     Ñ = Ñ + {a}
12 Devolva x*
```

A convergência do método da geração de colunas é garantida pelo fato de que cada coluna $a \in \Lambda$ é gerada, no máximo, uma vez pelo *pricing*, já que todas as colunas em $\tilde{N}$ têm custo reduzido não-positivo após a otimização do `PMR`.

A princípio, pode parecer que determinar $\overline{c}^*$ seja tão difícil quanto avaliar o custo reduzido de cada coluna em $N \backslash \overline{N}$. No entanto, para tomar vantagem do método, o modelo de programação linear do problema mestre é selecionado de tal forma que as colunas $a \in \Lambda$ possuam propriedades específicas da aplicação de modo que o problema de gerar colunas em Λ possa ser resolvido usando-se um algoritmo combinatório ou possa ser modelado por outro problema de PL com um número polinomial de restrições (usualmente, da ordem de $|M|$ e muito menor que $|N|$). Em geral, $a \in \Lambda$, representam objetos combinatórios como árvores, caminhos e outras estruturas bem definidas.

Exemplo

Vamos exemplificar o uso do método da geração de colunas no clássico problema de corte e empacotamento. Considere que barras de aço de L centímetros de comprimento devem ser cortadas para obter um conjunto M de pedaços de aço de comprimentos distintos. Vamos supor que cada item (pedaço de aço demandado) seja associado a um inteiro em $[1, n]$, com $|M| = n$. Supondo que cada item i, $1 \leq n \leq n$, tem l_i centímetros de comprimento, o problema de corte e empacotamento unidimensional consiste em determinar o menor número de barras necessárias para gerar todos os pedaços de aço.

Uma formulação natural para o problema consiste em associar uma variável de decisão x_j a cada padrão j de corte da barra. Um padrão de corte refere-se a uma forma de cortar uma barra e pode ser representado pelos itens gerados pelo corte. Se j é um padrão de corte, então $a_{ij} = 1$ se e, somente, se o item i é gerado pelo corte j. Seja N o conjunto de todos os padrões de cortes. Logo, o seguinte é um modelo para o problema de corte e empacotamento:

$$\begin{aligned} \text{(PM)} \quad & \min \sum_{j \in N} x_j \\ \text{sujeito a} \quad & \sum_{j \in N} a_{ij} x_j = 1, \forall i \in M \qquad (2.10) \\ & x_j \in \{0, 1\}, \forall j \in N. \qquad (2.11) \end{aligned}$$

Neste modelo, cada coluna representa um padrão de corte. Uma vez que N é, no máximo, 2^n, temos uma formulação com um grande número de colunas. Portanto, torna-se inviável resolver o passo de *pricing* de forma explícita, ou seja, calculando-se o custo reduzido de cada coluna $j \in N$. Este é um exemplo clássico em que o método da geração de colunas pode ser aplicado.

Apenas para ilustrar como a geração de colunas é aplicada, considere que na relaxação linear de (PM), π_i, $1 \leq i \leq M$, são as variáveis duais associadas a (2.10) do problema relaxado. Logo, o custo reduzido de uma coluna j é dado por:

$$\overline{c}_j = 1 - \sum_{i \in M} a_{ij} \pi_i.$$

Uma vez que cada coluna do problema mestre consiste em um padrão de corte, pode-se encontrar uma coluna de menor custo reduzido, resolvendo-se o seguinte problema de otimização:

$$\begin{aligned} \text{(Pricing)} \quad & \max \sum_{i \in M} a_i \pi_i \\ \text{sujeito a} \quad & \sum_{i \in M} l_i a_i \leq L & (2.12) \\ & a_i \in \{0,1\}, \forall i \in M, & (2.13) \end{aligned}$$

que equivale ao problema da mochila binária.

Seja $v(Pricing)$ o valor ótimo de (Pricing) e a^* sua solução ótima. Se $1 - v(Pricing) \leq 0$, a coluna correspondente a a^* é adicionada a $\overline{N}$, caso contrário o valor ótimo da relaxação linear de (PM) foi encontrada.

O exemplo anterior trata-se de um dos tipos de aplicação do método da geração de colunas, em que a formulação natural do problema contém uma quantidade exponencial de variáveis de decisão. Na próxima seção, fazemos uma revisão sobre o método da decomposição de Dantzig-Wolfe, que é aplicado sobre uma **formulação compacta**, isto é, sobre um modelo de programação linear com uma quantidade polinomial de variáveis e restrições, do problema e gera um modelo com uma quantidade exponencial de colunas.

2.2.1 Decomposição de Dantzig-Wolfe

Considere, inicialmente, o seguinte programa linear, que modela um problema Π:

$$\begin{aligned} & \min \sum_{j \in N} c_j x_j \\ \text{sujeito a} \quad & \sum_{j \in N} a_{ij} x_j \geq b_i, \forall i \in M & (2.14) \\ & x \in P = \{x \in \mathbb{R}^{|N|} | Dx \geq d, x \geq 0\} & (2.15) \end{aligned}$$

Suponha que $n = |N|$ e $m = |M|$ sejam limitados polinomialmente no tamanho da entrada do problema Π. Considere P não-vazio e limitado, para simplificar a explanação (o método da decomposição de Dantzig-Wolfe também se aplica nos demais casos [10]). Sejam Q o conjunto dos pontos extremos de P e p_q, para $q \in Q$, o vetor característico de q. Tipicamente, o tamanho de Q é exponencial no número de restrições e variáveis que descrevem P. Contudo, pelo Teorema de Caratheodory [49], sabe-se que cada ponto $x \in P$ pode ser representado pela combinação convexa dos pontos em Q, ou seja,

$$x = \sum_{q \in Q} p_q \lambda_q, \sum_{q \in Q} \lambda_q = 1, \lambda \in R_+^{|Q|}. \quad (2.16)$$

A substituição de x na formulação compacta por (2.16) e a aplicação das transformações lineares $c^q = cp_q$ e $a^q = Ap_q$, resulta no seguinte programa equivalente, chamado **formulação extensiva**:

$$\begin{aligned} \text{(P)} \quad & \min \sum_{q \in Q} c^q \lambda_q \\ \text{sujeito a} \quad & \sum_{q \in Q} a_i^q \lambda_q \geq b_i, \forall i \in M & (2.17) \\ & \sum_{q \in Q} \lambda_q = 1 & (2.18) \\ & \lambda_q \geq 0, \forall q \in Q. \end{aligned}$$

A restrição (2.18) é chamada **restrição de convexidade**. Uma vez que o número de colunas na formulação extensiva é muito grande, o método da geração de colunas é utilizado na resolução do problema, que é um caso bastante particular de problema mestre. Nele, as colunas são dadas por pontos extremos do poliedro P. Sejam $\overline{\pi}, \overline{\mu}$ as variáveis duais ótimas obtidas com a otimização do `PMR` correspondente ao problema linear dado pela formulação extensiva, onde μ é a variável dual associada à restrição de convexidade. Devido às transformações lineares acima no passo de *pricing*, a determinação de $\overline{c}^* = \min_{q \in Q}\{c^q - \overline{\pi} a^q - \overline{\mu}\}$ equivale a resolver o seguinte programa linear:

$$\overline{c}^* = \min\{(c - \overline{\pi}A)x - \overline{\mu} | Dx \geq d, x \geq 0\}.$$

Agora, considere o caso de um PLI em que $x \in X = P \cap Z_+$ na formulação compacta, ou seja, considere o caso em que o problema (P) anterior assume a seguinte forma:

$$\begin{aligned} & \min \sum_{j \in N} c_j x_j \\ \text{sujeito a} \quad & \sum_{j \in N} a_{ij} x_j \geq b_i, \forall i \in M & (2.19) \\ & x \in X = P \cap Z_+ \end{aligned}$$

Substituindo-se X por $conv(X)$, chega-se a um programa equivalente para o qual,

aplicando-se a decomposição de Dantzig-Wolfe, obtém-se a seguinte formulação extensiva:

$$\begin{aligned}
& \min \sum_{q \in Q} c^q \lambda_q \\
\text{sujeito a} \quad & \sum_{q \in Q} a_i^q \lambda_q \geq b_i, \forall i \in M && (2.20) \\
& \sum_{q \in Q} \lambda_q = 1 && (2.21) \\
& \lambda_q \geq 0, \forall q \in Q \\
& x = \sum_{q \in Q} p_q \lambda_q && (2.22) \\
& x \in Z_+^n.
\end{aligned}$$

Relaxando-se a restrição de integralidade de x, a restrição (2.22) torna-se desnecessária. Neste caso, o problema mestre resultante, passa a ser

$$\begin{aligned}
\text{(DW)} \quad & \min \sum_{q \in Q} c^q \lambda_q \\
\text{sujeito a} \quad & \sum_{q \in Q} a_i^q \lambda_q \geq b_i, \forall i \in M && (2.23) \\
& \sum_{q \in Q} \lambda_q = 1 && (2.24) \\
& \lambda_q \geq 0, \forall q \in Q,
\end{aligned}$$

Esta relaxação linear da formulação estendida pode ser resolvida por geração de colunas e o valor da solução ótima obtida dá um limitante inferior para o problema inteiro. Geoffrion [25] mostrou que este limitante, obtido pela decomposição de Dantzig-Wolfe, é sempre maior ou igual ao limitante dado pela relaxação linear da formulação compacta. De fato, Geoffrion mostrou que o limitante obtido pela decomposição de Dantzig-Wolfe coincide com o valor do limitante dado por outros dois métodos de decomposição, relaxação lagrangeana e algoritmos de planos de corte com separação exata de $conv(X)$. Portanto, a decomposição de Dantzig-Wolfe pode ser utilizada para obter limitantes mais apertados que aqueles da relaxação linear da formulação compacta, o que, como vimos, é útil em algoritmos de *branch-and-bound*. Geoffrion também mostrou que o limitante dual dado pela decomposição de Dantzig-Wolfe coincide com o valor da relaxação linear da formulação compacta quando $P = conv(X)$. Isso decorre da observação de que, enquanto a relaxação linear resolve a formulação compacta com $x \in P$, o método da geração de colunas, implicitamente, resolve a formulação compacta com $x \in conv(X)$. Portanto, escolhas

adequadas de se decompor uma formulação compacta na decomposição de Dantzig-Wolfe devem ser consideradas para obter limitantes duais mais apertados que aqueles gerados pela relaxação linear da formulação compacta.

2.2.2 Base Inicial

O método de geração de colunas precisa de um problema mestre restrito inicial, ou seja, de um conjunto de colunas iniciais que formem uma base. Encontrar uma base inicial pode não ser trivial, dependendo do problema. Nestes casos, pode ser utilizada uma idéia similar ao método de duas fases usada pelo método *Simplex* para gerar uma base inicial viável: criam-se variáveis artificiais com alto custo na função objetivo e com colunas que formam uma matriz identidade. Este conjunto de colunas iniciais garante uma solução inicial para o problema mestre restrito.

Além das colunas iniciais para formar uma base do problema mestre restrito, observa-se na prática que a adição de "boas" colunas na formulação inicial podem ajudar na convergência do método de geração de colunas [5].

2.2.3 Limitante Dual de Lasdon

Um limitante dual para o problema obtido através de sua relaxação linear só será calculado quando a geração de colunas determinar que não há colunas de custo reduzido negativo. Isso é particularmente ruim uma vez que a convergência do método de geração, em muitos casos, é lenta (efeito conhecido como **tailing-off**). Para contornar esta situação, pode-se interromper a geração de colunas quando a melhoria no valor ótimo do problema mestre restrito é pequena e usar o valor das variáveis duais e o valor ótimo do problema escravo atual para calcular um limitante dual [5, 14]. Um dos limitantes usualmente adotados é o de Lasdon [32].

Sejam π^* e μ^* os vetores duais ótimos do problema mestre restrito em alguma iteração do algoritmo de geração de colunas. Sejam $ZPric$ o valor ótimo do subproblema de *pricing*, $v(PM)$ o valor ótimo do problema mestre e $v(PMR)$ o valor ótimo do problema mestre restrito. Para fins de generalização, considere k, $k \geq 1$, o *rhs* da restrição de convexidade.

Lasdon [32] mostrou que

$$LB = v(PMR) + kZPric$$

é um limitante inferior para o valor ótimo da relaxação linear do problema mestre.

Uma vez que $ZPric$ define o custo reduzido mais negativo, tem-se que

$$\begin{aligned} & ZPric \leq c_j - \pi^* A_{.j} - \mu^*, \forall j \in N \\ \Rightarrow \quad & c_j - \pi^* A_{.j} - \mu^* - Zpric \geq 0, \forall j \in N \\ \Rightarrow \quad & \pi^* A_{.j} + \mu^* + Zpric \leq c_j, \forall j \in N. \end{aligned}$$

Portanto, $(\pi^*, \mu^* + Zpric)$ é dual viável. Por dualidade fraca, tem-se que

$$v(PM) \geq \sum_{i \in M} b_i \pi_i^* + k\mu^* + kZpric = v(PMR) + kZpric,$$

logo, LB é um limitante dual para o problema mestre.

2.2.4 Acelerando a Convergência do Método

Dentre os problemas que surgem ao usar o método da geração de colunas podemos citar: o efeito *tailing off*, grande oscilação no valor das variáveis duais e degenerescência. As próximas subseções discutem estratégias para estabilizar o valor dos duais para acelerar a convergência dos algoritmos de geração de colunas na tentativa de diminuir os efeitos de *tailing off* e técnicas para diminuir a degenerescência.

Degenerescência

Degenerescência ou **grau de degenerescência** é o nome que se dá ao número de soluções básicas degeneradas. Uma solução básica é **degenerada** quando alguma variável básica tem valor zero e, isso impede mudança no valor da solução básica. Degenerescência implica na existência de diferentes bases para representar o mesmo ponto extremo e é um efeito indesejado no método *Simplex* pois leva-o a realizar iterações que não progridem em direção à solução ótima, ou seja, retardam a sua convergência. A inclusão de colunas pelo método da geração de colunas também podem gerar soluções degeneradas, que, da mesma forma que no método *Simplex*, não são desejadas.

Uma forma de diminuir a degenerescência em um modelo de cobertura, por exemplo, consiste em perturbar o rhs. A perturbação é realizada somando-se um pequeno valor distinto para o rhs de cada restrição de cobertura. Essa técnica pode ser aplicada quando a solução ótima do problema perturbado também é ótimo para o problema original. Exemplo de aplicação desta técnica e resultados computacionais podem ser encontrados, por exemplo, em [28].

Métodos de Estabilização

Quando o problema mestre é degenerado, o dual dele tem um número infinito de soluções ótimas. Os resolvedores de PL retornam um ponto extremo do poliedro dual, levando as

variáveis duais a ter valores pequenos ou grandes demais. Este fato pode ser observado na grande oscilação nos valores das variáveis duais obtidos ao longo do processo de geração de colunas. Muitas aproximações melhores seriam obtidas se as variáveis duais tomassem valores no centro (interior) do poliedro dual. Para prevenir as variáveis duais de tomar valores extremos, uma estratégia é tentar limitar a distância percorrida pelas variáveis duais no espaço dual de uma iteração a outra. Existem duas técnicas para isso: *box* e limitantes com penalização. Du Merle et al [17] propuseram um método para estabilização dos algoritmos de geração de colunas que combina estas duas técnicas. Para expor o método de Du Merle et al, considere o problema de cobertura de conjuntos com a seguinte formulação e seu dual:

$$\begin{aligned} \text{(BoxP)} \quad & \min \sum_{j \in N} c_j x_j \\ \text{sujeito a} \quad & \sum_{j \in N} a_{ij} x_j \geq 1, \forall i \in M \\ & x_j \geq 0, \forall j \in N. \end{aligned}$$

$$\begin{aligned} \text{(BoxD)} \quad & \max \sum_{i \in M} \pi_i \\ \text{sujeito a} \quad & \sum_{i \in M} a_{ij} \pi_i \leq c_j, \forall j \in N \\ & \pi_i \geq 0, \forall i \in M. \end{aligned}$$

A idéia é impor limites (d^-, d^+) sobre as variáveis duais π_i associadas com as restrições de (BoxP), mas permitindo que elas tomem valores fora destes limites (usando-se variáveis w^- e w^+). Quando o valor das variáveis w^- ou w^+ são não-nulos (isto é, as variáveis duais π_i têm valores fora dos limites), uma penalização é aplicada na função objetivo (ε^- e ε^+). O problema mestre modificado e seu dual estabilizado é:

$$
\begin{aligned}
(\overline{BoxP})\ \min & \sum_{j\in N} c_j x_j + \sum_{i\in M}(-d_i^- y_i^- + d_i^+ y_i^+) \\
\text{sujeito a} \quad & \sum_{j\in N} a_{ij}x_j - y_i^- + y_i^+ \geq 1, \forall i \in M \\
& y_i^- \leq \varepsilon^-, \forall i \in M \\
& y_i^+ \leq \varepsilon^+, \forall i \in M \\
& x_j \geq 0, \forall j \in N \\
& y_i^- \geq 0, \forall i \in M \\
& y_i^+ \geq 0, \forall i \in M.
\end{aligned}
$$

$$
\begin{aligned}
(\overline{BoxD})\ \max & \sum_{i\in M} \pi_i - \varepsilon_i^- w_i^- - \varepsilon_i^+ w^+ i \\
\text{sujeito a} \quad & \sum_{i\in M} a_{ij}\pi_i \leq c_j, \forall j \in N \\
& \pi_i + w_i^- \geq d_i^-, \forall i \in M \\
& \pi_i - w_i^+ \leq d_i^+, \forall i \in M \\
& \pi_i, w_i^-, w_i^+ \geq 0, \forall i \in M.
\end{aligned}
$$

Este método de estabilização é conhecido como *Box Stabilization* pelo fato de que os valores de d_- e d_+ são definidos pelo centro Box_c do *box* (um hipercubo) e pelo lado ξ do *box* da seguinte forma:

$$d_- = Box_c - \xi$$

e

$$d_+ = Box_c + \xi.$$

Se uma boa estimativa do valor dos duais ótimos é conhecido, ele é usado para inicializar o valor de Box_c. Caso contrário, usa-se os valores dos duais iniciais em P.

O algoritmo de geração de colunas estabilizado consiste em, a cada iteração, resolver o problema mestre restrito $\overline{BoxP}$ no lugar de $(BoxP)$, incluir em $\overline{BoxP}$ a coluna de custo reduzido negativo gerada pelo *pricing* e atualizar os parâmetros d_-, d_+, ε_- e ε_+. Note que $(\overline{BoxP})$ é uma relaxação de $(BoxP)$ e que uma solução $(\overline{x}, w_-, w_+)$ de $(\overline{BoxP})$ é, também, uma solução de $(BoxP)$ sempre que $w_- = w_+ = 0$. Portanto, o algoritmo de geração de colunas estabilizado termina somente quando o *pricing* retorna uma coluna de custo reduzido positivo e $y_- = y_+ = 0$ (ou seja, ou (i) $\varepsilon_- = \varepsilon_+ = 0$ ou (ii) $d_- < \pi < d_+$).

Para garantir a convergência do algoritmo, deve-se escolher estratégias adequadas para a atualização dos parâmetros d_-, d_+, ε_- e ε_+.

Du Merle *et al* sugerem as seguintes estratégias de atualização:

- Diminua o valor de ε_- e ε_+ sempre que o *pricing* devolver uma coluna de custo reduzido positivo ou o valor dos duais for a melhor estimativa para os duais ótimos. Nos demais casos, aumenta-se o valor de ε_- e ε_+. Uma medida da qualidade do valor dos duais pode ser obtida através do cálculo dos limitantes duais de Lasdon.

- Atualize o valor de d_- e d_+ para os duais correntes sempre que o valor dos duais for a melhor estimativa para os duais ótimos (ou seja, faça $Box_c = \pi$, mantendo-se $\xi > 0$). Quando o *pricing* devolver uma coluna de custo reduzido positivo, aumente o valor de ξ, mantendo o centro do box.

Estas estratégias de atualização devem garantir a convergência do método. Desde que $BoxP$ seja limitado, em alguma iteração do processo de geração de colunas, o *pricing* retornará uma coluna de custo reduzido positivo e o valor de ε_- e ε_+ será decrementado. Portanto, em um número finito de iterações, a condição (i) ocorrerá e o algoritmo termina. Seja π^k o valor dos duais na iteração k, na qual o *pricing* retornou uma coluna de custo reduzido positivo. Note que $boxD$ é uma relaxação de $\overline{boxD}$. Portanto, π^k é viável para $boxD$ e $v(\overline{boxD}^k) \leq v(boxD)$. Se $v(\overline{boxD}^k) = v(boxD)$ então $\varepsilon_- = \varepsilon_+ = 0$ ou $w_- = w_+ = 0$. No primeiro caso, a condição (i) foi atingida na iteração k e o algoritmo termina. No último caso, após atualizar ξ (aumentar $\xi > 0$), a condição (ii) é atingida e o algoritmo termina. Caso $v(\overline{boxD}^k) < v(boxD)$, em alguma iteração posterior t, $v(\overline{boxD}^k) < v(\overline{boxD}^t)$, uma vez que o valor de ξ foi aumentado na iteração k. E, portanto, após um número finito de iterações l, $v(\overline{boxD}^{k+l}) = v(boxD)$.

A dificuldade na utilização deste método de estabilização está na determinação das estratégias de atualização e no valor dos parâmetros, que podem apresentar desempenhos diferentes quando aplicados a problemas distintos.

Interior Point Stabilization

Rousseau *et al* propuseram um método diferente, chamado *Interior Point Stabilization*, para estabilização do algoritmo de geração de colunas [42]. O método consiste em gerar vários pontos extremos no poliedro dual (ao invés de um só) e tomar a combinação convexa deles para obter um ponto interior do poliedro. A idéia para se chegar a base dual ótima consiste em identificar os conjuntos S e R^*, definidos a seguir, e usar complementaridade das folgas. Considere o modelo de cobertura e seja x^* a solução ótima de P e seja $S = \{i \in M | \sum_{j \in N} a_{ij} x_j^* > 1\}$. Por complementariedade das folgas tem se que $\pi_i = 0$, para

todo $i \in S$, na base dual ótima. Por outro lado, seja $R^* = \{j \in N | x_j^* > 0\}$, ou seja, o conjunto relativo às variáveis básicas não-nulas. Por complementariedade das folgas, segue que $\sum_{i \in M} a_{ij}\pi_i = c_j$, para todo $j \in R^*$, na base dual ótima.

Logo, do programa linear $D(u)$ a seguir pode-se calcular diversas soluções duais ótimas a partir de diferentes valores de u:

$$\begin{aligned}
\text{(D(u))} \quad & \max \sum_{i \in M} u_i \pi_i \\
\text{sujeito a} \quad & \sum_{i \in M} a_{ij}\pi_i = c_j, \forall j \in R^* \\
& \sum_{i \in M} a_{ij}\pi_i \leq c_j, \forall j \in N \setminus R^* \\
& \pi_i = 0, \forall i \in S \\
& \pi_i \geq 0, \forall i \in M \setminus S.
\end{aligned}$$

Após a geração de algumas soluções duais ótimas, toma-se uma combinação convexa delas (por exemplo, a média) para se obter uma solução dual no interior da face dual ótima.

Um programa linear alternativo para computar soluções duais ótimas consiste no dual de $D(u)$, que é bastante similar à formulação original de P:

$$\begin{aligned}
\text{(P(u))} \quad & \min \sum_{j \in N} c_j x_j \\
\text{sujeito a} \quad & \sum_{j \in N} a_{ij} x_j \geq u_i, \forall i \in M \setminus S \\
& \sum_{j \in N} a_{ij} x_j \geq -\infty, \forall i \in S \qquad (2.25) \\
& x_j \geq 0, \forall j \in N \setminus R^* \\
& x_j \text{ qualquer}, \forall j \in R^*. \qquad (2.26)
\end{aligned}$$

A restrição (2.25) faz com que $\pi_i = 0$, para todo $i \in S$ e a restrição (2.26) faz com que a restrição correspondente no dual seja satisfeita na igualdade.

Inicialmente, tome $u_i = 1$, para todo $i \in M$, $S = \emptyset$ e $R^* = \emptyset$ para obter os conjuntos S e R^*. Depois, atribui-se diferentes valores para u_i, $i \in M \setminus S$, resolvendo-se cada um dos respectivos $P(u)$. Por fim, calcula-se uma combinação convexa dos duais ótimos obtidos com a solução de cada $P(u)$.

Outros Métodos de Estabilização

Um método que reduz o problema da oscilação dos duais foi proposto por Ben Amor, Desrosiers e Valério de Carvalho [3], que consiste na utilização de desigualdades válidas, chamadas desigualdades duais-ótimas, para as soluções ótimas do dual do problema mestre restrito, que em P representa a inclusão de colunas adicionais. A idéia deste método é que a inclusão destas colunas restringe os valores dos multiplicadores duais e, assim, reduz sua oscilação no espaço dual, acelerando a convergência em direção aos multiplicadores duais ótimos. Os resultados computacionais da aplicação deste método no problema de corte e empacotamento mostrou a efetividade do método, que reduziu significativamente o tempo e o número de iterações para a convergência do algoritmo de geração de colunas. A utilização deste método é dependente da aplicação uma vez que desigualdades duais-ótimas devem ser encontradas.

Gondzio e Sarkissian [27] mostram como usar método de ponto interior na geração de colunas. A idéia é utilizar o método primal-dual inviável para resolver a relaxação linear do problema mestre restrito, abrindo-se mão da otimalidade deste. A vantagem deste método é a rapidez na obtenção dos multiplicadores duais e na qualidade deles para a geração de colunas.

2.2.5 *Pricing* Parcial

Em algumas situações, o passo de *pricing* é aquele que consome mais tempo de processamento (muitas vezes o problema de *pricing* é NP-difícil). Um meio de contornar esse problema consiste em utilizar uma heurística para resolver o problema de *pricing* de um problema de minimização (maximização) ao invés de resolvê-lo de forma exata e, assim, espera-se gerar colunas de custo reduzido negativo (positivo) a um custo menor de processamento. Deve-se notar, no entanto, que a heurística pode falhar em encontrar uma coluna de custo reduzido negativo (positivo). Portanto, um algoritmo exato ainda é necessário para resolver o *pricing*, mas potencialmente será usado poucas vezes, somente quando a heurística de *pricing* falhar.

2.2.6 *Branch-and-Price*

Considere um problema de PLI com uma formulação contendo um número exponencial de variáveis de decisão. A relaxação linear do problema pode ser resolvida usando-se o algoritmo de geração de colunas. No entanto, ainda faz-se necessário usar um método exato de resolução de problemas de PLI, como por exemplo o algoritmo *branch-and-bound*, para obter uma solução inteira ótima. Neste caso, o algoritmo de geração de colunas é usado em cada nó da árvore de enumeração apenas para resolver a relaxação linear e,

assim, obter um limitante dual do nó. A esse algoritmo que une o algoritmo *branch-and-bound* e o método da geração de colunas dá-se o nome de **branch-and-price**.

Em um algoritmo *branch-and-price*, regras de *branching* usuais em um algoritmo *branch-and-bound* não são adequadas. Seja x_j uma variável de decisão com valor fracionário f na solução ótima da relaxação linear de um nó. Uma regra de *branching* usual em um algoritmo de *branch-and-bound* é realizar *branching* sobre x_j, incluindo uma restrição $x_j \leq \lfloor f \rfloor$ em um nó filho e a restrição $x_j \geq \lceil f \rceil$ em outro nó filho. Esse tipo de *branching* tem a vantagem de eliminar o ponto fracionário no espaço de soluções dos nós filhos. No entanto, isso não é verdade em um algoritmo *branch-and-price* pelo fato de que a coluna associada à variável de decisão fracionária pode ser gerada posteriormente. Regras de *branching* mais efetivas são aquelas que envolvem as variáveis de decisão do problema de *pricing*. Neste caso, deve-se apenas reescrever a desigualdade de *branching* em termos das variáveis do problema mestre antes de adicioná-la na formulação do nó filho.

2.2.7 *Branch-and-Cut-and-Price*

Um algoritmo ***branch-and-cut-and-price*** consiste em um algoritmo *branch-and-bound*, no qual cada nó da árvore de enumeração é resolvido usando-se o método da geração de colunas e algoritmos de planos de corte. Um algoritmo de plano de corte introduz desigualdades válidas ao problema mestre na tentativa de fortalecer a formulação do problema enquanto a geração de colunas é usada para resolver a relaxação linear em cada nó. O algoritmo é usado quando a formulação por PLI do problema possui uma quantidade exponencial de variáveis. Não há regras quanto à ordem em que os algoritmos de planos de cortes e algoritmos de geração de colunas devem ser executados em cada nó.

Quando as desigualdades válidas adicionadas pelo algoritmo de plano de cortes dentro de um algoritmo *branch-and-cut-and-price* não alteram o problema de *pricing* dizemos, no sentido definido em [46], que trata-se de um algoritmo ***branch-and-cut-and-price* robusto**. Ou seja, a inclusão das desigualdades pode afetar apenas os pesos das variáveis do problema de *pricing*. A vantagem de utilizar um algoritmo *branch-and-cut-and-price* robusto é que um único algoritmo pode ser usado para resolver o problema de *pricing* em toda iteração e em todo nó da árvore de enumeração do algoritmo de *branch-and-cut-and-price*.

2.3 Algoritmos de Rotulação

Nesta seção revisamos os algoritmos para resolução de variantes do problema de caminho mínimo, que surgem como problemas de *pricing* em vários problemas de PLI, como em

problemas de roteamento de veículos. Nós utilizamos estes algoritmos na resolução do problema dos anéis-estrelas capacitados, descrito no Capítulo 4.

Dado um grafo $G = (V, E)$, custos c_e para cada aresta e em E e um vértice fonte s, o **problema do caminho mínimo** consiste em encontrar um caminho não-orientado em G de s a cada vértice de G, cujo custo dado pela soma dos custos das arestas no caminho seja mínimo. Os principais algoritmos para resolver este problema são conhecidos como algoritmos de rotulação (ou do inglês, *labeling algorithms*). Estes algoritmos são iterativos e, em cada iteração, rotulam um vértice do grafo com um valor que é um limitante superior para a distância de um caminho mínimo de s ao vértice. O Algoritmo 4 é um pseudocódigo para algoritmos desse tipo. Eles diferem no critério de escolha do próximo vértice a ser removido de $\mathcal{N}$. Quando o vértice de menor rótulo é escolhido primeiro, tem-se um algoritmo *labeling setting*. Esse algoritmo foi proposto originalmente e independentemente por Dijkstra(1959), Dantzig(1960), Hillier e Whiting (1960).

Algoritmo 4: Algoritmo *Labeling*.

Input: grafo $G = (V, E)$, vértice s, custos c_{ij}, para todo $ij \in E$
Output: $d(i)$: distância do caminho mínimo de s ao vértice i, para todo $i \in V$.

```
1 N = {(s)}; S = ∅
2 d(i) = ∞, ∀i ∈ V \ {s}; d(s) = 0
3 while N ≠ ∅ do
4     Escolha um vértice i ∈ N
5     N = N − {i}
6     for w em V adjacente a i do
7         if d(w) > d(i) + c_iw then
8             d(w) = d(i) + c_iw
9             S = S ∪ {w}
```

O algoritmo de Dijskstra resolve o problema da árvore de caminhos mínimos definido sobre grafos livres de arestas de custos negativos. Ele mantém duas listas $\mathcal{N}$ e $\mathcal{S}$. Um rótulo associado a um vértice i em $\mathcal{S}$ representa a distância mínima de s a i, enquanto que em $\mathcal{N}$ um limitante superior para essa quantidade. A cada iteração, o vértice de menor rótulo é removido de $\mathcal{N}$ e seu rótulo torna-se definitivo, ou seja, o vértice é incluído na lista $\mathcal{S}$. O rótulo do vértice removido de $\mathcal{N}$ é usado para atualizar os rótulos dos vértices adjacentes a ele e que estão em $\mathcal{N}$ [2]. O Algoritmo 5 é um típico algoritmo **label setting**.

Uma das características principais do algoritmo *label setting* é que o rótulo dos vértices em $\mathcal{S}$, uma vez definidos, não são mais alterados. Isso só ocorre por conta da função custo ser não negativa. Todo vértice em $\mathcal{S}$ tem rótulo definitivo, ou seja representa a distância

Algoritmo 5: Algoritmo *Labeling Setting*.

Input: grafo $G = (V, E)$, vértice s, custos c_{ij}, para todo $ij \in E$
Output: $d(i)$: distância do caminho mínimo de s ao vértice i, para todo $i \in V$.

```
1 N = {(s)}; S = ∅
2 d(i) = ∞, ∀i ∈ V \ {s}; d(s) = 0
3 while N ≠ ∅ do
4     Escolha um vértice i ∈ N com d(i) mínimo
5     N = N − {i}
6     for w em V adjacente a i em N do
7         if d(w) > d(i) + c_iw then
8             d(w) = d(i) + c_iw
9             S = S ∪ {w}
```

do caminho mínimo de s ao vértice. Essa característica torna o algoritmo atrativo, uma vez que vértices processados em uma iteração não serão mais processados em iterações futuras, o que tende a manter baixa a sua complexidade.

Parte I

O problema dos anéis-estrelas capacitados

Capítulo 3

Introdução

O **problema de roteamento de veículos**, do inglês *vehicle routing problem* (VRP), introduzido por Dantzig e Ramser [11] está relacionado a aplicações em que determinadas mercadorias, localizadas em um **depósito**, devem ser distribuídas em pontos dispersos geograficamente, chamados **clientes**. Em tais aplicações, cada cliente precisa de uma demanda fixa de um tipo de mercadoria e uma frota de m veículos idênticos localizados no depósito está disponível para efetuar a distribuição das mercadorias e, assim, atender as demandas dos clientes. Em geral, os veículos possuem uma capacidade limitada de carga, o que restringe o número de clientes que um veículo pode visitar. Neste contexto, o termo **rota** é usado para designar a seqüência de clientes visitados por um veículo. Associado a cada rota, existe um custo, chamado **custo de roteamento**, que refere-se ao custo de visitar os clientes na seqüência definida pela rota. O VRP consiste em determinar m rotas, uma para cada veículo, para atender as demandas dos clientes com o objetivo de minimizar os custos de roteamento. Este problema tem muitas aplicações práticas e diferentes variações. A variante em que os veículos têm capacidade limitada de carga é conhecida por *capacitated vehicle routing problem* (CVRP) e aquela em que a demanda de um cliente pode ser atendida por dois ou mais veículos é conhecida por *split delivery VRP*. Ambas as variantes têm sido bastante estudadas nos últimos anos. Em algumas aplicações, o custo de roteamento pode ser muito caro. Nestas situações, pode ser mais viável traçar rotas que não visitam todos os clientes e dispor de um serviço de entrega adicional e menos custoso para entregar posteriormente a mercadoria de um ponto da rota até o cliente que ficou fora da rota. Exemplos de tais aplicações incluem logística de distribuição de jornais, fretamento de veículos, projeto de redes de fibras óticas, entre outros.

O problema dos anéis-estrelas capacitados(CmRSP) pode ser visto como essa variante do VRP, que adicionalmente permite pontos com demanda nula, chamados **pontos de *Steiner***, ocorrerem nas rotas para diminuir custos de roteamento. No CmRSP, a

demanda de cada cliente não visitado pelas rotas é atendida indiretamente através da associação do cliente a um ponto da rota. Esta associação, chamada **conexão**, implica num custo, que, do ponto de vista da aplicação, pode ser visto como o custo de deslocar as mercadorias de um ponto da rota até o cliente não visitado. No contexto de CmRSP, uma rota é chamada **anel**, o conjunto das conexões até uma rota é denominado **estrela** e as duas juntas formam uma configuração que lembra a topologia estrela e é designada por **anel-estrela**. Como no CVRP, os veículos possuem uma capacidade limitada de carga, que restringe o número de clientes que um veículo pode atender direta e indiretamente. O CmRSP consiste em encontrar m anéis-estrelas disjuntos nos vértices, um para cada veículo, respeitando-se sua capacidade, para atender as demandas dos clientes com o objetivo de minimizar os custos de roteamento e de conexão.

O CmRSP é NP-difícil uma vez que ele é uma generalização do CVRP. Este problema foi introduzido por Baldacci, Dell'Amico e Salazar [4] como um modelo para aplicações no contexto de redes e telecomunicações, como o projeto de redes de fibras ópticas (veja Capítulo 1). Duas formulações de programação linear inteira baseadas na formulação *two-index* e *two-commodity flow* do VRP [44] foram propostas para modelar o CmRSP. Pelo nosso conhecimento, o único algoritmo exato para resolver este problema é um algoritmo *branch-and-cut* proposto no mesmo artigo em que o problema foi introduzido.

Neste trabalho, uma formulação cobertura de conjuntos para modelar o CmRSP é proposta e algoritmos exatos para resolver o problema baseados neste modelo são comparados com o algoritmo *branch-and-cut* proposto por Baldacci, Dell'Amico e Salazar. A formulação proposta consiste num programa linear inteiro com uma quantidade exponencial de variáveis, cujas colunas na matriz de coeficientes correspondem a anéis-estrelas. Modelos de cobertura de conjuntos, em geral, apresentam limitantes duais mais apertados que a de outros modelos, como já reportado na literatura para o VRP [44]. Resolver a relaxação linear desses modelos para obter limitantes duais implica no uso do método da geração de colunas. Neste método, a relaxação linear do problema é obtida resolvendo-se repetidamente dois subproblemas: o problema mestre e o subproblema de *pricing*, que na formulação do CmRSP também é um problema NP-difícil. Para contornar essa complexidade, o subproblema de *pricing* é substituído por um problema relaxado, usando-se uma idéia similar ao que foi usado com sucesso para o VRP com janela de tempo [31] e para o CVRP [24]. A relaxação consiste, basicamente, em permitir a repetição de vértices ao longo dos anéis-estrelas. A vantagem no uso dessa relaxação é que uma rotina de programação dinâmica (PD) pode ser projetada para resolvê-la em tempo pseudopolinomial. Mas, por outro lado, o valor do limitante dual torna-se fraco. Para obter um limitante dual mais apertado, outras relaxações são consideradas que, embora, ainda permitam repetições de vértices, proíbem determinados tipos de repetições. A rotina de PD pode ser adaptada para resolver essas relaxações, usando-se uma idéia similar àquela usada por

Irnich e Villeneuve [31] para resolver o problema chamado *non-elementary shortest path problem with resource constraints* ao proibir k-ciclos. Três relaxações são avaliadas nesse trabalho em um algoritmo *branch-and-price* para resolver de forma exata o CmRSP. Os resultados computacionais mostram que o *branch-and-price* é competitivo com o único algoritmo exato conhecido para o problema e nenhum deles domina o outro, no sentido em que em algumas instâncias o *branch-and-price* apresenta desempenho melhor e vice-versa. Para tirar proveito de ambos os métodos, um algoritmo *branch-and-cut-and-price* é proposto e os resultados mostram que ele é altamente competitivo.

O texto desta parte da tese referente ao CmRSPestá organizado da seguinte forma. O Capítulo 4 contém as notações e definições necessárias para a descrição do trabalho que foi realizado. Este capítulo apresenta ainda a nossa formulação por cobertura de conjuntos de programação linear inteira (PLI) para o CmRSP, além de cobrir em detalhes os subproblemas relaxados e os algoritmos de programação dinâmica propostos para resolvê-los em tempo pseudo-polinomial. Os detalhes da implementação do algoritmo de *branch-and-cut-and-price* proposto aqui e os resultados computacionais são analisados no Capítulo 5. O Capítulo 6, último desta parte da tese, apresenta as conclusões sobre os nossos estudos relativos ao CmRSP.

Capítulo 4

O Problema dos Anéis-estrelas Capacitados

Neste capítulo, o problema dos anéis-estrelas capacitados (CmRSP) é formalmente definido. Em seguida, duas formulações de programação linear inteira são apresentadas como modelos para o problema. A primeira trata-se de uma formulação compacta proposta por Baldacci et al. enquanto a segunda refere-se a um novo modelo baseado naquele para o problema da cobertura de conjuntos. A formulação proposta consiste num programa linear inteiro com uma quantidade exponencial de variáveis, cujas colunas na matriz de coeficientes correspondem a anéis-estrelas. Resolver a relaxação linear desses modelos pode ser viável pelo uso do método da geração de colunas (veja Capítulo 2), que consiste em resolver repetidamente o problema mestre e o problema escravo. No modelo proposto para o CmRSP o problema escravo é um problema NP-difícil. Uma seção do capítulo é dedicada para expor as soluções propostas para lidar com a NP-dificuldade do problema escravo e, assim, resolver o modelo. A última seção apresenta as desigualdades válidas que podem sem incluídas nos modelos para apertar suas relaxações lineares dentro de um algoritmo de *branch-and-cut* ou *branch-and-cut-and-price*.

4.1 Notações e Definições

Inicialmente, considere as seguintes notações e definições usadas neste capítulo.

Um grafo misto é aquele que admite arcos orientados e não-orientados. Denote por $G = (V, E \cup A)$ um grafo misto, onde E é o conjunto dos arcos não-orientados e A, o dos orientados. O termo aresta é usado na tese para denotar os arcos não-orientados. Dado $R \subseteq E$, considere $V[R]$ denotando o subconjunto dos vértices que são extremos de arestas em R. O conjunto dos arcos não-orientados incidentes em um vértice i é denotado por $\delta(i)$. Um *caminho* $P = (p_1, p_2, \ldots, p_k)$ é uma seqüência de vértices $p_1, \ldots, p_k$ tal que cada

(p_i, p_{i+1}), $1 \leq i \leq k-1$, é uma aresta do grafo. Se os vértices do caminho forem dois-a-dois disjuntos tem-se um *caminho elementar*. Algumas vezes, o termo *caminho não-elementar* será usado para se referir a um caminho para distingui-lo de um *caminho elementar*. Um *ciclo* (elementar) é um caminho (elementar) em que $p_1 = p_k$. O *comprimento* de um caminho é definido pelo número de arestas no caminho. Um *laço* é um ciclo de comprimento 1.

Dados vetores x e c de dimensões $1 \times n$ e $n \times 1$, respectivamente, a notação $c(x)$ será usada para denotar $\sum_{i=1}^{n} c_i x_i$.

Considere um grafo misto $G = (\{0\} \cup U \cup W, E \cup A)$, onde 0 denota um depósito, U um conjunto de clientes e W um conjunto de pontos de *Steiner*. O conjunto dos vértices de G é denotado por V e o conjunto $V \setminus \{0\}$ por V'. O conjunto E de arcos não-orientados é completo enquanto o conjunto A de arcos orientados não é necessariamente completo e representa um conjunto de conexões. Cada **conexão** é orientada de um nó cliente a um nó em V'.

Denote por C_i o conjunto dos nós aos quais um cliente i pode se conectar. Portanto, $A = \{ij | i \in U, j \in C_i\}$. Cada arco não-orientado e é associado a um **custo de roteamento** não-negativo c_e e cada arco orientado ij é associado a um **custo de conexão** não-negativo w_{ij}. Cada cliente possui uma demanda de um produto localizado no depósito. Pontos de *Steiner* possuem demanda nula. Neste trabalho, consideramos apenas demandas unitárias nos clientes, uma vez que este foi o caso originalmente proposto na literatura e para os quais existem experimentos reportados. No entanto, não é difícil estender os modelos apresentados e as soluções aqui propostas para o caso não unitário.

Um **anel-estrela** é um par (R, S), onde $R \subseteq E$ é um ciclo disjunto nos vértices de G que passa pelo depósito e $S \subseteq \{uv \in A |\ u \notin V[R] \text{ e } v \in V[R]\}$.

Um anel-estrela (R, S) é **q-capacitado** se $|V[R \cup S] \cap U| \leq q$.

O custo de um anel-estrela $p = (R, S)$ é $c_p = \sum_{e \in R} c_e + \sum_{ij \in S} w_{ij}$.

A Figura 4.1 mostra dois anéis-estrelas 9-capacitados, um de custo 43 e outro de custo 55. Os pontos de *Steiner* estão representados por círculos preenchidos na figura. Os arcos e arestas de cada anel-estrela estão com padrões diferenciados. Por exemplo, na Figura 4.1, o anel-estrela $(\{(0,a), (a,b), (b,c), (c,0)\}, \{ec, fc\})$ está destacado com linhas pontilhadas.

Um cliente i é **coberto** por um anel-estrela (R, S) se $i \in V[R \cup S]$. O termo **pendurado** é utilizado no texto para designar um cliente i coberto pelo anel-estrela devido a um arco $ij \in S$.

Dizemos que um anel-estrela (R, S) é **canônico** se para cada ponto de *Steiner* j em R existir, pelo menos, um arco ij em S, qualquer que seja $i \in U$. Por exemplo, o anel-estrela $(\{(0,g),(g,h),(h,i),\ (i,j),\ (j,k),\ (k,0)\},\ \{mi,ni,qk,pk\})$ não é canônico, uma vez que o

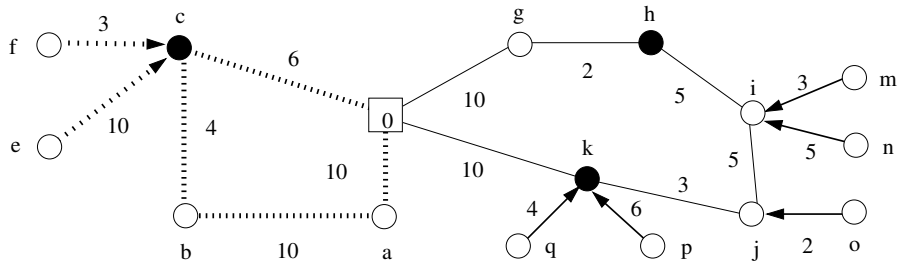


Figura 4.1: Dois exemplos de anéis-estrelas.

ponto de *Steiner* h aparece no anel sem nenhum cliente pendurado nele.

4.2 Definição do Problema

Dados um grafo misto $G = (V, E \cup A)$, custos de roteamento $c : E \mapsto \mathbb{R}_+$, custos de conexão $w : E \mapsto \mathbb{R}_+$, inteiros não-negativos Q e m, o **problema dos anéis-estrelas capacitados** consiste em encontrar m anéis-estrelas Q-capacitados, $p_1 = (R_1, S_1)$, $p_2 = (R_2, S_2)$, ..., $p_m = (R_m, S_m)$, tais que:

- $V[R_i \cup S_i] \cap V[R_j \cup S_j] = \{0\}$, $\forall i \neq j \in [1..m]$;
- $U \subseteq \bigcup_{i=1}^{m} V[R_i \cup S_i]$;
- $\sum_{i=1}^{m} c_{p_i}$ é mínima.

Quando a função de custos de roteamento é métrica, ou seja, satisfaz as desigualdades triangulares, a seguinte proposição é verdadeira:

Proposicao 4.1. *Toda instância do* CmRSP *admite uma solução ótima em que todo anel-estrela da solução é canônico.*

Demonstração. Suponha, por contradição, que existe um anel-estrela não canônico em uma dada solução ótima de uma instância do CmRSP. Seja (R, S) um desses anéis-estrelas com $R = \{(v_0, v_1), \ldots, (v_{i-1}, v_i), (v_i, v_{i+1}), \ldots, (v_{k-1}, v_k)\}$ e $v_i \in W$. Uma vez que os custos de roteamento satisfazem as desigualdades triangulares, tem-se que o custo do anel-estrela (R', S) com $R' = \{(v_0, v_1), \ldots, (v_{i-1}, v_{i+1}), \ldots, (v_{k-1}, v_k)\}$, é menor ou igual ao custo de (R, S), que contradiz a hipótese. □

Pela Proposição 4.1, é suficiente procurar pela melhor combinação de m anéis-estrelas Q-capacitados e canônicos de uma instância do CmRSP, para obter uma solução ótima do problema.

4.3 Modelagem Matemática

Esta seção apresenta duas formulações de programação linear inteira para o CmRSP, a formulação compacta proposta por Baldacci et al. [4] e uma nova baseada no modelo de cobertura de conjuntos. Neste último modelo, uma variável de decisão é utilizada para cada anel-estrela Q-capacitado e canônico. Obter uma solução deste modelo consiste em escolher m das variáveis de decisão, cujos anéis-estrelas correspondentes cobrem todos os clientes do grafo.

4.3.1 Formulação Compacta

Esta seção apresenta o modelo para o CmRSP proposto por Baldacci et al. [4]. Considere uma variável de decisão inteira x_e, associada a cada aresta $e \in E$, que assume valor um se e, somente, se a aresta e pertence a algum anel da solução. Seja z_{ij} uma variável de decisão binária, associada a cada arco $ij \in A$, que é igual a um se e, somente, se o cliente i está conectado ao vértice j na solução. Uma variável binária y_i é definida para cada vértice i e assume valor um se e, somente, se o vértice i pertencer a algum anel da solução. A formulação compacta para o CmRSP é descrita a seguir:

$$\text{(CF)} \quad \min \sum_{e \in E} c_e x_e + \sum_{ij \in A} w_{ij} z_{ij}$$

$$\text{sujeito a} \quad \sum_{e \in \delta(0)} x_e = 2m \tag{4.1}$$

$$\sum_{e \in \delta(i)} x_e = 2y_i, \forall i \in V \setminus \{0\} \tag{4.2}$$

$$\sum_{ij \in A} z_{ij} + y_i = 1, \forall i \in U \tag{4.3}$$

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q}\Big(\sum_{i \in U \cap S} y_i + \sum_{i \in U} \sum_{j \in S \cap C_i} z_{ij}\Big), \forall S \subseteq V \setminus \{0\} : S \neq \{\} \tag{4.4}$$

$$y \in \{0,1\}^{|V'|}, z_{ij} \in \{0,1\}, x_{ij} \in \{0,1,2\}. \tag{4.5}$$

O número de arestas incidentes em um vértice é dado pelas restrições (4.1) e (4.2). A restrição (4.1) força que o número de anéis-estrelas em uma solução seja igual a m enquanto (4.2) garante que duas arestas incidam em vértice se ele pertencer a algum anel. A restrição (4.3) força um cliente i estar em um anel ou ser coberto através de uma conexão ij. Note que S, em uma restrição (4.4), define um corte $\delta(S)$ no grafo e o total de arestas desse corte que pertence à solução é exatamente o valor do somatório do lado esquerdo dessas restrições. Uma vez que a intersecção de um corte com um ciclo é sempre

uma quantidade par de arestas, as restrições de capacidade fracionária (4.4) forçam o número de anéis-estrelas necessários para cobrir todos os clientes em S de modo que a capacidade dos anéis-estrelas não seja violada.

Este modelo tem um número polinomial de variáveis, mas exponencial de restrições por causa das restrições (4.4). No lugar destas restrições, Baldacci et al. usaram um conjunto de desigualdades válidas, que são incluídas no modelo através de rotinas de separação dentro de um algoritmo de *branch-and-cut*. Na Seção 4.5, estas desigualdades são apresentadas e discutidas com mais detalhes.

4.3.2 Modelo de Cobertura de Conjuntos

No modelo de cobertura de conjuntos, o problema é dividido em dois subproblemas: o problema mestre e o problema escravo. O primeiro trata-se efetivamente do modelo por cobertura de conjuntos. Como discutido no Capítulo 2, um problema mestre restrito, que contém apenas um subconjunto das variáveis de decisão do problema mestre, é utilizado no lugar do problema mestre, uma vez que este possui uma quantidade exponencial de variáveis de decisão. O problema escravo tem por finalidade gerar um anel-estrela, que não está na formulação do problema mestre restrito e cuja inclusão poderá gerar uma solução de custo menor.

O Problema Mestre

Denote por $\mathcal{P}$ o conjunto dos anéis-estrelas Q-capacitados e canônicos. Um anel-estrela (R, S) pode ser representado por dois vetores característicos r e s, de dimensões $(|U| + |E|) \times 1$ e $|A| \times 1$, respectivamente. r_i indica o número de ocorrências do vértice i no anel R, r_{ij} o número de ocorrências da aresta (i, j) em R, enquanto s_i e s_{ij} indicam, respectivamente, o número de ocorrências do vértice i e do arco ij em S. Defina a variável de decisão binária λ_p para cada anel-estrela p em $\mathcal{P}$ tal que $\lambda_p = 1$ se e, somente, se o anel-estrela associado a p é selecionado para compor uma solução ótima do problema. O modelo por cobertura de conjuntos para o CmRSP é dado pela seguinte formulação PLI:

$$
\begin{aligned}
\text{(SC)} \quad & \min \sum_{p \in \mathcal{P}} c_p \lambda_p & \\
\text{sujeito a} \quad & \sum_{p \in \mathcal{P}} \lambda_p = m & (4.6)\\
& \sum_{p \in \mathcal{P}} (r_i^p + \sum_{j \in C_i} s_{ij}^p) \lambda_p \geq 1, \forall i \in U & (4.7)\\
& \sum_{p \in \mathcal{P}} \sum_{ij \in E} r_{ij}^p \lambda_p \leq 2, \forall i \in V' & (4.8)\\
& \sum_{p \in \mathcal{P}} r_e^p \lambda_p \leq u_e, \forall e \in E & (4.9)\\
& \sum_{p \in \mathcal{P}} s_{ij}^p \lambda_p \leq 1, \forall ij \in A & (4.10)\\
& \lambda_p \in \{0,1\}, \forall p \in \mathcal{P}, & (4.11)
\end{aligned}
$$

onde $u_e = 2$ se e é incidente no depósito e $u_e = 1$, caso contrário. O tratamento diferenciado para as arestas incidentes no depósito nas restrições (4.9) é necessário para permitir anéis de comprimento dois, da forma $0 - i - 0$, onde i é um vértice qualquer do grafo.

A restrição de cardinalidade (4.6) determina o número de anéis-estrelas que devem ser selecionados para compor a solução ótima. O modelo de cobertura é dado pelas restrições (4.7) que impõem que todos os clientes sejam cobertos. Note que um modelo de partição de conjuntos, no qual as inequações (4.7) são substituídas por restrições de igualdade, poderia ser utilizado. O modelo de partição de conjuntos obriga cada cliente ser coberto exatamente por um único anel-estrela. Em geral, esse modelo apresenta o inconveniente de ser bastante degenerado, o que torna lenta a convergência da geração de colunas [18, 35].

Embora as restrições de cobertura permitam que um cliente seja coberto múltiplas vezes, é importante notar que sempre existe uma solução inteira ótima na qual isso não ocorre. Suponha, por contradição, que uma solução possui um cliente coberto múltiplas vezes. Devido às restrições (4.8), que garantem que, no máximo, duas arestas incidem em cada vértice, a multiplicidade tem que ocorrer quando o vértice está pendurado. Neste caso, uma solução de custo menor ou igual pode ser obtida, bastando remover as demais ocorrências do cliente nos anéis-estrelas em que ele aparece pendurado (já que o custo de conexão é não-negativo).

As restrições de aresta (4.9) e de arco (4.10) equivalem a impor um limitante superior para a ocorrência de arestas e de arcos nos anéis-estrelas, respectivamente. A rigor, essas restrições e as restrições (4.8) não são necessárias para descrever o modelo do CmRSP. No entanto, sem elas, não haveria uma correspondência um-para-um entre uma coluna do modelo e um anel-estrela.

Por fim, as restrições de integralidade das variáveis de decisão λ são dadas por (4.11).

O Problema Escravo

Sejam π, μ, ν, β e γ as variáveis duais associadas às restrições (4.6), (4.7), (4.8), (4.9) e (4.10), respectivamente. Logo, o custo reduzido associado a um anel-estrela p é:

$$\overline{c}_p = \sum_{e \in E} \tilde{c}_e r_e^p + \sum_{ij \in A} \tilde{w}_{ij} s_{ij}^p - \sum_{i \in V'} \tilde{p}_i r_i^p + \pi$$

onde $\tilde{c}_{ij} = c_{ij} - \beta_{ij} - k_i - k_j$, $k_0 = 0$, $k_i = \nu_i$, se $i \neq 0$, $\tilde{w}_{ij} = w_{ij} - \gamma_{ij} - \mu_i$, $\tilde{p}_i = \mu_i$, para $i \in U$ e $\tilde{p}_i = 0$, para $i \in W$.

Portanto, o problema escravo consiste em determinar um anel-estrela Q-capacitado e canônico de custo reduzido mínimo. É fácil provar que este problema escravo é NP-difícil, uma vez que ele generaliza uma variante do problema do caixeiro viajante com lucros [19], chamada *Profitable Tour Problem* (PTP) [12], que é NP-difícil. Dado um grafo $G = (V, E)$ com custos c_e, associado a cada aresta $e \in E$, lucros v_i, para cada vértice $i \in V$, e com um vértice especial, representando um depósito, o PTP consiste em encontrar um ciclo elementar C, que passa através do depósito e minimiza o custo do ciclo menos a soma dos lucros coletados, ou seja, $\sum_{e \in C} c_e - \sum_{i \in V(C)} v_i$. Uma instância do PTP pode ser polinomialmente reduzida a uma instância do problema escravo do CmRSP tomando-se o depósito como 0, $U := V \setminus \{0\}$, $W := \emptyset$, $A := \emptyset$, $\tilde{c_e} := c_e$, $\tilde{p_i} := v_i$ e $Q = n$, o que prova a NP-dificuldade do problema escravo do modelo por cobertura de conjuntos proposto para o CmRSP.

4.4 Relaxação do Problema Escravo

Dada a NP-dificuldade do problema escravo, um problema relaxado é considerado em seu lugar. Esta idéia tem sido usada em problemas de roteamento de veículos, árvore geradora mínima capacitada e variantes do problema do caixeiro viajante [24, 47, 30].

Três relaxações para o problema escravo são propostas. Basicamente, elas consistem em permitir a repetição de vértices nos anéis-estrelas e são baseadas nas idéias utilizadas por Irnich e Villeneuve [31] para resolver o problema do caminho mínimo elementar com restrição de recursos, cujo problema relaxado tem a vantagem de possuir um algoritmo de programação dinâmica de tempo pseudopolinomial [13].

Antes de apresentar os problemas escravos relaxados e os respectivos algoritmos, iremos discutir uma breve revisão da literatura para o problema do caminho mínimo elementar com restrição de recursos, sua relaxação e seu algoritmo pseudopolinomial.

4.4.1 Preliminares

Seja $G = (V, E)$ um grafo com custos c_e, em cada aresta e e um vértice especial s. Considere o problema de encontrar caminhos de custos mínimos partindo do vértice s que obedeçam determinadas restrições de recursos. Neste contexto, um recurso representa uma quantidade como tempo, capacidade de carga de veículos, entre outros, que é consumido em cada aresta de um caminho. A soma dos recursos consumidos ao longo das arestas de um caminho de s até um vértice i define os **recursos acumulados** em i e é denotado por T_i.

As restrições de recursos impõem quantidades mínimas e máximas para os recursos acumulados em cada vértice do caminho.

Os recursos são representados por vetores em $\mathbb{R}^R$, onde R é o número de recursos diferentes. Dados vetores T e S em $\mathbb{R}^R$, tem-se que $T \leq S$ se $T^i \leq S^i$, para todo $1 \leq i \leq R$.

Denote por $t^r : E \mapsto \mathbb{R}$, $\forall r \in R$, o consumo de recursos definido sobre cada aresta do grafo e seja $[a_i, b_i]$, $\forall i \in V$, com $a_i, b_i \in \mathbb{R}^R$, sendo $a_i^r \leq b_i^r$, um intervalo definido para cada vértice do grafo, o qual determina as quantidades mínimas e máximas para os recursos acumulados no vértice.

Um caminho $P = (v_0, v_1, \ldots, v_p)$ é **recurso-viável** se existirem vetores de recursos $T_0, T_1, \ldots, T_p$, tais que $T_i \in [a_{v_i}, b_{v_i}]$, $\forall i = 0, \ldots, p$, e $T_{i+1} \geq T_i + t_{i,i+1}$, $\forall i = 0, \ldots, p-1$.

Considere o exemplo da Figura 4.2, em que há apenas dois recursos. Em cada aresta está representado o vetor de consumo de recursos (t^1, t^2) associado à aresta e em cada vértice, o intervalo $[(a^1, a^2), (b^1, b^2)]$, que determina a quantidade mínima e máxima permitida de recursos acumulados no vértice. O caminho $P = (0, 1, 3)$ não é recurso-viável, pois quaisquer vetores de recursos acumulados T_0, T_1 e T_3 serão tais que $T_3^2 \geq 6$, o qual viola o limite máximo permitido do segundo recurso no vértice 3, que é 5. Por outro lado, o caminho $(0, 2, 3)$ é recurso-viável, uma vez que os vetores de recursos acumulados $T_1 = (0, 0)$, $T_2 = (2, 1)$ e $T_3 = (6, 3)$ satisfazem as restrições de recursos.

Note que verificar se um caminho $P = (v_0, v_1, \ldots, v_p)$ é recurso-viável pode ser realizado computando os vetores de recursos acumulados em cada vértice e verificando se os limitantes não são violados. Para cada vértice v_i em P, o vetor T_i pode ser computado pelas equações:

$$T(0) = a_{v_0} \tag{4.12}$$

e

$$T(i) = \max\{a_{v_i}, T(i-1) + t_{i-1,i}\}, \forall 1 \leq i \leq p. \tag{4.13}$$

A cada caminho recurso-viável $P = (v_0, v_1, \ldots, v_p)$, associamos o único vetor de recursos $res(P) = T(p)$, computados pelas equações (4.12) e (4.13).

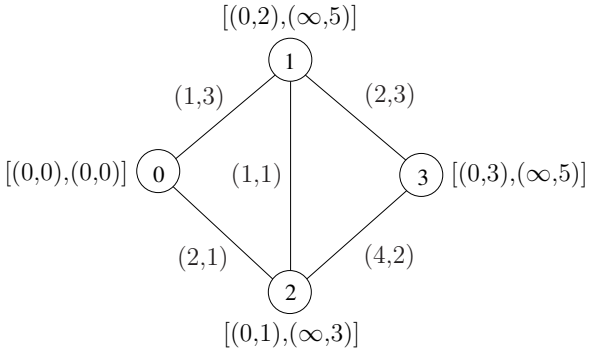


Figura 4.2: Exemplo de caminho mínimo elementar com restrição de recursos.

Dados vetores T e S em $\mathbb{R}^R$, diz-se que T **domina** S (denota-se $T \prec_{\text{dom}} S$) se $T \leq S$. Um caminho P_1 em $\mathcal{F}(s, v)$ domina outro caminho P_2 em $\mathcal{F}(s, v)$ se $res(P_1) \prec_{\text{dom}} res(P_2)$.

Um vetor de recursos $T \in \mathbb{R}^R$ é **pareto-optimal** se nenhum outro vetor $S \in \mathbb{R}^R$ satisfaz $S \prec_{\text{dom}} T$. Um caminho P é pareto-optimal se $res(P)$ é pareto-optimal.

O **problema do caminho mínimo elementar com restrição de recursos** (ESPRC), do inglês *non-elementary shortest path problem with resource constraints*, consiste em encontrar caminhos pareto-optimais em $\mathcal{F}(s, v) \cap \mathcal{G}$, para cada vértice v em V. Aqui, $\mathcal{F}(s, v)$ é o conjunto dos caminhos de s a v que são recursos-viáveis e $\mathcal{G}$, o conjunto dos caminhos elementares em G. O ESPRC é NP-difícil no sentido forte [16]. No **problema do caminho mínimo com restrições de recursos** (SPRC), que é uma relaxação do ESPRC, o conjunto $\mathcal{G}$ é constituído por todos os caminhos, não necessariamente elementares, em G.

O algoritmo de programação dinâmica, proposto por Desrochers e Soumis [13], para resolver o SPRC é apresentado em 6. Basicamente, ele consiste em manter duas listas de caminhos, $\mathcal{N}$ e $\mathcal{S}$. A lista $\mathcal{N}$ contém caminhos ainda não processados enquanto a lista $\mathcal{S}$ contém caminhos já processados. Um caminho é considerado processado sempre que ele foi utilizado em alguma iteração para estender e construir novos caminhos. O algoritmo é iterativo e termina quando não há caminhos a serem processados. A cada iteração um caminho em $\mathcal{N}$ é escolhido para ser processado. No passo de extensão, o caminho escolhido é usado como prefixo para construir novos caminhos. No passo de dominância, regras são aplicadas para eliminar caminhos não promissores, para os quais pode-se provar que não são prefixos de caminhos pareto-optimais. Esse passo tem por finalidade diminuir a quantidade de caminhos em $\mathcal{N}$ e assim evitar uma explosão combinatória do algoritmo.

Algoritmo 6: Algoritmo para o SPRC.

Input: grafo $G = (V, E)$, vértice s
Output: caminhos pareto-optimais em $\mathcal{F}(s, v) \cap \mathcal{G}$, para cada $v \in V$

1 $\mathcal{N} = \{(s)\}$; $\mathcal{S} = \emptyset$
2 **while** $\mathcal{N} \neq \emptyset$ **do**
3 Escolha um caminho $P = (v_0, v_1, \ldots, v_p) \in \mathcal{N}$ com $res(P) = \min_{P' \in \mathcal{N}} res(P')$
4 $\mathcal{N} = \mathcal{N} - \{P\}$
  `/* passo de extensão */`
5 **for** w *em* V *adjacente a* v_p **do**
6 **if** $(P, w) \in \mathcal{F}(s, w) \cap \mathcal{G}$ **then**
7 Adicione (P, w) a $\mathcal{N}$
8 $\mathcal{S} = \mathcal{S} + \{P\}$
  `/* passo de dominância */`
9 **if** $\nexists P' \in \mathcal{N} : res(P') = res(P)$ **then**
10 Aplique algoritmo de dominância para caminhos em $\mathcal{N} \cup \mathcal{S}$.
11 Identifique $S \subseteq \mathcal{S}$

Se existir um recurso r tal que $t^r_{ij} > 0$, para todo $(i, j) \in E$, é suficiente escolher um caminho P de menor $res^r(P)$ no passo 3 para garantir que os caminhos em $\mathcal{S}$ após o passo de dominância sejam pareto-optimais. Uma vez que $t^r > 0$, um caminho P somente produzirá caminhos estendidos (P, w) com $res^r((P, w)) > res^r(P)$ e, portanto, novos caminhos nunca dominam caminhos já processados em iterações anteriores. Essa característica é similar ao que ocorre com os vértices incluídos na lista $\mathcal{S}$ nos algoritmos *Label setting* (veja Capítulo 2).

Se nenhuma regra de dominância é aplicada, o algoritmo constrói todos os caminhos em $\mathcal{F}(s, v) \cap \mathcal{G}$, para todo v em V. Se $\mathcal{G}$ é o conjunto de todos os caminhos não-elementares em G, a dominância entre vetores de recursos pode ser aplicada. No entanto, veremos adiante que dependendo da estrutura proibida em $\mathcal{G}$, manter apenas caminhos pareto-optimais não é suficiente para garantir que a solução ótima seja encontrada.

Veremos nas próximas seções, diferentes relaxações para o problema escravo do CmRSP, baseadas na relaxação do ESPRC, e seus respectivos algoritmos, construídos com base no algoritmo pseudopolinomial do SPRC.

4.4.2 Anel-estrela Relaxado Q-capacitado

Uma relaxação do problema escravo do CmRSP consiste em permitir repetição de vértices nos anéis-estrelas. Para formalizar este problema relaxado introduziremos algumas de-

finições.

Um **anel-estrela relaxado** é um par (R, S), onde $R \subseteq E$ define um ciclo sem auto-laços, não necessariamente disjunto nos vértices, que passa pelo depósito e $S \subseteq \{(u, v) \in A | v \in V[R]\}$. Ou seja, em um anel-estrela relaxado, vértices podem ocorrer múltiplas vezes tanto no anel quanto na estrela.

Um **anel-estrela relaxado q-capacitado** é um anel-estrela relaxado em que o total de ocorrências dos clientes no anel e na estrela é menor ou igual a q.

A Figura 4.3 mostra um exemplo de anel-estrela relaxado 8-capacitado.

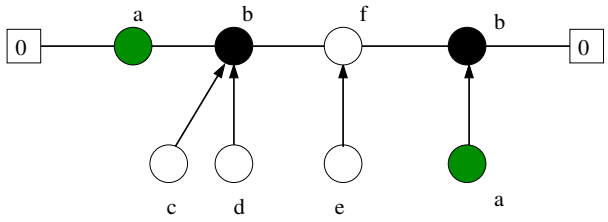


Figura 4.3: Exemplo de anel-estrela relaxado 8-capacitado.

O **peso** de um anel-estrela relaxado q-capacitado $p = (R, S)$ consiste em $\tilde{c}_p = \sum_{e \in E} \tilde{c}_e x_e + \sum_{ij \in A} \tilde{w}_{ij} z_{ij} - \sum_{i \in V'} \tilde{p}_i y_i$, onde x_e, z_{ij} e y_i são, respectivamente, o número de ocorrências de e em R, de ij em S e de i em $V[R]$.

O **problema escravo relaxado**, chamado **problema do anel-estrela relaxado Q-capacitado**, consiste em encontrar um anel-estrela relaxado Q-capacitado de peso mínimo.

Seja $\mathcal{P}'$ o conjunto dos anéis-estrelas relaxados Q-capacitados e canônicos. A formulação (SC) continua válida mesmo após substituir o conjunto $\mathcal{P}$ de anéis-estrelas Q-capacitados e canônicos pelo conjunto $\mathcal{P}'$. A validade do modelo vem de vários fatores. Primeiro, deve-se constatar que $\mathcal{P} \subseteq \mathcal{P}'$. Segundo, devido às restrições (4.8), nenhuma solução inteira é formada por anéis-estrelas com repetição de vértices nos anéis. Por último, devido ao custo de conexão ser não-negativo, existe uma solução ótima inteira em que todo vértice pendurado não ocorre em nenhum anel da solução e ocorre, no máximo, uma vez em cada estrela.

Antes de apresentar um algoritmo para resolver o problema escravo relaxado, algumas notações e definições são necessárias.

Um **passeio-estrela** é um subgrafo similar a um anel-estrela relaxado, exceto que um ciclo aberto ocorre no lugar do ciclo fechado do anel. Um **(j, q)-passeio-estrela** é um passeio-estrela com uma das extremidades no depósito e o outro em um vértice j e tal que o total de ocorrências de clientes no anel e na estrela é exatamente q. A função peso também é definida para um passeio-estrela da mesma forma que em um anel-estrela relaxado. O anel-estrela da Figura 4.3 sem a aresta $(b, 0)$ é um exemplo de passeio-estrela.

Por definição, considere que $(0,0)$-passeio-estrela representa o caminho contendo apenas o depósito. Além disso, (j,q)-passeio-estrela não está definido para os seguintes valores: $j=0$ e $q>0$, $j\neq 0$ e $q=0$.

O algoritmo de programação dinâmica para resolver o problema escravo relaxado consiste em construir uma matriz F de ordem $V'\times Q$, em que cada entrada $F(j,q)$ corresponde ao peso de um (j,q)-passeio-estrela de peso mínimo.

Note que um anel-estrela relaxado pode ser obtido com a inclusão da aresta $(j,0)$ em um (j,q)-passeio-estrela. Logo, o valor ótimo do problema escravo relaxado, ou seja, o peso do anel-estrela relaxado Q-capacitado de peso mínimo, pode ser encontrado da seguinte forma:

$$\min_{j\in V',q\in[1..Q]} F(j,q)+\tilde{c}_{j0}.$$

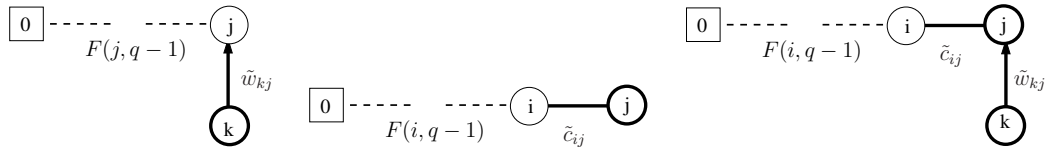
O valor de $F(j,q)$ pode ser definido recursivamente com base no fato de que todo (j,q)-passeio-estrela pode ser construído estendendo-se um $(i,q-1)$-passeio-estrela através das seguintes operações (veja Figura 4.4):

(a) inclusão de um arco $(k,j)\in A$, se $i=j$;

(b) inclusão de uma aresta $(i,j)\in E$, se $i\neq j$ e $j\in U$;

(c) inclusão de uma aresta $(i,j)\in E$ e de um arco $(k,j)\in A$, se $i\neq j$ e $j\in W$.

As condições para cada uma das operações vêm do fato de que um anel-estrela relaxado não pode conter auto-laços e por estarmos nos restringindo apenas a anéis-estrelas canônicos.



(a) incluindo arco (b) incluindo aresta (c) incluindo arco e aresta

Figura 4.4: Construção de (j,q)-passeio-estrela.

Logo, a definição recursiva de $F(j,q)$, para $j\neq 0$ e $1\leq q\leq Q$, é dada por:

$$\min\big\{\min_{k\in U} F(j,q-1)+\tilde{w}_{kj},\ \min_{i\in V,i\neq j} F(i,q-1)+\tilde{c}_{ij}-\tilde{p}_j\big\},$$

se $j\in U$, e

$$\min\big\{\min_{kj\in A} F(j,q-1)+\tilde{w}_{kj},\ \min_{ij\in E,kj\in A} F(i,q-1)+\tilde{w}_{kj}+\tilde{c}_{ij}\big\},$$

caso contrário.

Por definição de (j, q)-passeio-estrela, temos que $F(0, 0) = 0$ e $F(0, q > 0) = F(j \neq 0, 0) = \infty$.

O Algoritmo 7 (página 52) mostra um pseudocódigo para computar a matriz F. Note que para obter um anel-estrela relaxado Q-capacitado de peso mínimo é necessário manter em cada entrada de $F(j, q)$ um (j, q)-passeio-estrela de peso mínimo, além de seu peso. No algoritmo, representamos por $F(j, q)$.ring as arestas que formam o anel, $F(j, q)$.star, os arcos da estrela e $F(j, q)$.peso o peso do (j, q)-passeio-estrela de peso mínimo.

Comparando o Algoritmo 7 com o Algoritmo 6, vemos que o primeiro é uma versão do segundo, no qual as operações (a), (b) e (c) compreendem o passo de extensão e a linha (25) compreende o passo de dominância.

O passo de extensão para construir (j, q)-passeios-estrelas, para algum vértice j e inteiro q, toma tempo $O(|U||V|)$, por causa da operação (c), que é a mais custosa das operações. Uma vez que $|\mathcal{C}|$ é $O(|U||V|)$, o passo de dominância também é $O(|U||V|)$. Logo, o Algoritmo 7 para resolver o problema escravo relaxado é $O(|U||V|^2Q)$.

4.4.3 Anel-estrela Q-capacitado Livre de k-ciclos

Embora a relaxação do problema escravo considerada na seção anterior nos tenha permitido obter um algoritmo pseudopolinomial para a geração de colunas, ela gera limitantes duais fracos. Uma forma de obter limitantes duais melhores consiste em utilizar outras relaxações, ainda baseadas na idéia de permitir repetições de vértices, mas proibindo a ocorrência de determinadas estruturas. Essa idéia novamente é baseada nas relaxações utilizadas para o problema ESPRC, para o qual limitantes melhores são obtidos, usando-se o SPRC com proibição de k-ciclos. Um **k-ciclo** é um ciclo de comprimento menor ou igual a k. Nessa relaxação k-ciclos são proibidos dentro dos caminhos não-elementares. O algoritmo de programação dinâmica para resolvê-la é baseado no algoritmo para resolver o problema do anel-estrela relaxado Q-capacitado e tem sua complexidade de tempo alterada por um fator, que é uma função de k.

O **problema do anel-estrela relaxado Q-capacitado com proibição de k-ciclos** (kcyc-RSP) consiste em encontrar um anel-estrela relaxado Q-capacitado de peso mínimo que não contenha k-ciclos dentro dos anéis.

Da mesma forma que fizemos para o problema do anel-estrela relaxado Q-capacitado, iremos construir a matriz F através da construção de (j, q)-passeio-estrela pelas operações (a), (b) e (c). Desta vez, as operações (b) e (c) estão condicionadas para evitar a formação de k-ciclos. Considere um $(i, q-1)$-passeio-estrela associado a uma entrada da matriz F. Uma vez que é possível recuperar os $k-1$ vértices predecessores de i no caminho a partir das informações mantidas em F, a condição para evitar a formação de k-ciclos consiste

Algoritmo 7: Algoritmo para o problema do anel-estrela relaxado Q-capacitado.

Input: grafo G, inteiro Q, custos $\tilde{w}$, $\tilde{c}$ e $\tilde{p}$
Output: matriz F de ordem $V \times Q$

```
1  F(0,0).custo= 0; F(0,0).ring={}; F(0,0).star={}
2  F(0,q>0).custo= ∞; F(0,q>0).ring={}; F(0,q>0).star={}
3  for i ∈ V' do
4      F(i,0).custo= ∞; F(i,0).ring= {}; F(i,0).star= {}
5  for q = 1 to Q do
6      for j ∈ V' do
7          C = ∅
           /* operação (a) */
8          for k ∈ U : j ∈ C_k do
9              P.ring=F(j,q-1).ring
10             P.star=F(j,q-1).star +{kj}
11             P.custo=F(j,q-1).custo+w̃_kj
12             C = C ∪ {P}
13         for i ∈ V' : i ≠ j do
14             if i ∈ U then
                   /* operação (b) */
15                 P.ring=F(i,q-1).ring +{(i,j)}
16                 P.star=F(i,q-1).star
17                 P.custo=F(i,q-1).custo+c̃_ij − p̃_j
18                 C = C ∪ {P}
19             else
20                 for k ∈ U : j ∈ C_k do
                       /* operação (c) */
21                     P.ring=F(i,q-1).ring +{(i,j)}
22                     P.star=F(i,q-1).star +{kj}
23                     P.custo=F(i,q-1).custo+c̃_ij + w̃_kj − p̃_j
24                     C = C ∪ {P}
25         Seja P em C tal que P = arg min_{P'∈C} P'.custo
26         F(j,q).custo=P.custo; F(j,q).ring=P.ring; F(j,q).star=P.star
```

em permitir estender o $(i, q-1)$-passeio-estrela para construir um (j, q)-passeio-estrela somente se j for diferente de i e dos $k-1$ vértices predecessores de i.

Uma vez que estamos interessados em proibir a ocorrência de k-ciclos apenas no anel de um passeio-estrela, iremos nos referir aos passeios-estrelas por caminhos. Um caminho é **livre de k-ciclos** se não possuir k-ciclos.

Quando uma estrutura, como os k-ciclos, é proibida de ocorrer dentro de caminhos, a idéia de guardar apenas um caminho de peso mínimo em cada entrada $F(j, q)$ não é mais suficiente. Para exemplificar, considere dois caminhos com extremidades 0 e b, $P = (0, a, b)$ e $\overline{P} = (0, c, b)$, onde o peso de P é menor que de $\overline{P}$. Note que o caminho P não pode ser utilizado para obter um caminho com extremidade em a e de comprimento 3, uma vez que ele conteria um 2-ciclo. Portanto, se apenas o caminho de peso mínimo é mantido em $F(b, 2)$, caminhos como $(0, c, b, a)$ não serão construídos. O exemplo dá a entender que todos os caminhos devem ser mantidos em $F(j, q)$ para garantir que um caminho ótimo seja gerado. Na realidade, veremos que existem regras de dominância para determinar quais caminhos devem ser mantidos em cada entrada $F(j, q)$.

Considere a mesma notação usada na Seção 4.4.1. Aqui, o conjunto $\mathcal{G}$ é formado por todos os caminhos em G livres de k-ciclos.

Dados caminhos $P = (p_1, p_2, \ldots, p_k)$ e $\overline{P} = (\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_l)$, o caminho $(p_1, p_2, \ldots, p_k = \overline{p}_1, \overline{p}_2, \ldots, \overline{p}_l)$ é denotado por $(P, \overline{P})$. Dados dois caminhos P e $\overline{P}$ em $\mathcal{G}$, tem-se que $\overline{P}$ **estende** P se $(P, \overline{P}) \in \mathcal{G}$. O conjunto de todos os caminhos que podem estender um caminho P é denotado por $\mathcal{E}(P)$.

Lema 4.2 (Irnich e Villeneuve [31])**.** *Sejam $P_1, \ldots, P_t$ e P caminhos em $\mathcal{F}(s, v) \cap \mathcal{G}$ tais que:*

- *$P_i \prec_{dom} P$, para todo $i \in \{1, \ldots, t\}$;*
- *$\mathcal{E}(P) \subseteq \mathcal{E}(P_1) \cup \ldots \cup \mathcal{E}(P_t)$.*

Seja $\overline{P} = (v, \ldots, w)$ um caminho arbitrário em $\mathcal{G}$. Se $(P, \overline{P}) \in \mathcal{G} \cap \mathcal{F}(s, w)$ então o mesmo acontece para algum P_i, $i \in \{1, \ldots, t\}$, ou seja, $(P_i, \overline{P}) \in \mathcal{G} \cap \mathcal{F}(s, w)$ com $res(P_i, \overline{P}) \leq res(P, \overline{P})$.

Nas condições do lema anterior, dizemos que P é um **caminho inútil**.

O resultado do lema anterior é que caminhos inúteis não são necessários para construir caminhos recursos-viáveis livres de k-ciclos de peso mínimo. Em outras palavras, caminhos inúteis não precisam ser mantidos na matriz F.

Dado um subconjunto $\mathcal{C}$ de caminhos em $\mathcal{G} \cap \mathcal{F}(s, v)$, dizemos que um caminho P em $\mathcal{G} \cap \mathcal{F}(s, v)$ é um **caminho útil** em $\mathcal{C}$ se

$$\mathcal{E}(P) \not\subseteq \bigcup_{P' \in \mathcal{C}: P' \prec_{\mathrm{dom}} P} \mathcal{E}(P'). \tag{4.14}$$

Por definição, P é um caminho útil em $\mathcal{C}$ se $\mathcal{C} = \emptyset$. O conjunto dos caminhos úteis em $\mathcal{G} \cap \mathcal{F}(s, v)$ é dado por:

$$U(v) = \{P \in \mathcal{G} \cap \mathcal{F}(s, v) : \mathcal{E}(P) \not\subseteq \bigcup_{P' \in \mathcal{G} \cap \mathcal{F}(s,v): P' \prec_{\text{dom}} P} \mathcal{E}(P')\}. \tag{4.15}$$

Dada uma função CaminhosÚteis($\mathcal{C}$), que computa os caminhos úteis em uma coleção de caminhos $\mathcal{C}$, o Algoritmo 8 (página 55) constrói a matriz F com os passeios-estrelas livres de k-ciclos de pesos mínimos.

Uma particularidade desse algoritmo é que caminhos detectados como úteis não são removidos até o final do algoritmo, ou seja, continuam úteis mesmo com a inclusão de novos caminhos em iterações posteriores do algoritmo, ou seja, trata-se de um algoritmo de Label-setting. Isso ocorre porque caminhos gerados em iterações posteriores não dominam os caminhos já gerados, devido ao fato do consumo de recursos $t_{ij} > 0$, para toda aresta (i, j) ou arco ij, o que implica em um consumo estritamente positivo a cada passo de extensão do algoritmo. Uma vantagem que advém disso, é que apenas caminhos úteis são estendidos para gerar novos caminhos, diminuindo bastante a quantidade de caminhos gerados.

Complexidade de tempo do algoritmo. Seja $\alpha(k)$ um limite para o total de caminhos em qualquer $F(j, q)$ e $T(k)$ o tempo que a função CaminhosÚteis gasta para identificar caminhos úteis.

Dados um vértice j em V' e um valor de q, $0 \leq q \leq Q$, vamos analisar a complexidade de tempo para construir todos os (j, q)-passeios-estrelas por cada uma das operações. O tempo gasto pela operação (a), nas linhas (7) a (13), é, no máximo, $\alpha(k)|U|$. Uma vez que verificar a formação de um k-ciclo na linha (16) toma tempo $k-1$, as operações (b) e (c), nas linhas (14) a (29), gastam tempo de, no máximo, $(k+|U|)\alpha(k)|V|$. Portanto, o tempo gasto pelo Algoritmo 8 é limitado a $\alpha(k)|U||V|^2Q + T(k)|V|Q$. Ambas as funções $\alpha(k)$ e $T(k)$ são definidas em termos da constante k e, portanto, não aumentam a complexidade assintótica do algoritmo que resolve o problema escravo relaxado. No entanto, veremos que este fator em k não é pequeno.

Nas próximas subseções discutiremos diferentes implementações para a função CaminhosÚteis, o valor de $\alpha(k)$ e de $T(k)$.

Identificação de Caminhos Úteis

Irnich e Villeneuve [31] propuseram um modo eficiente de codificar $\mathcal{E}(P)$ para computar o conjunto dos caminhos úteis dado pela equação (4.15). Os parágrafos que seguem resumem o algoritmo de Irnich e Villeneuve.

Algoritmo 8: Algoritmo para o problema do anel-estrela relaxado Q-capacitado livre de k-ciclos.

Input: grafo G, inteiro Q, custos $\tilde{w}$, $\tilde{c}$ e $\tilde{p}$
Output: matriz F de ordem $V \times Q$

```
1  F(0,0).custo = 0; F(0,0).ring = {}; F(0,0).star = {}
2  F(0,q>0).custo = ∞; F(0,q>0).ring = {}; F(0,q>0).star = {}
3  for i ∈ V' do
4      F(i,0).custo = ∞; F(i,0).ring = {}; F(i,0).star = {}
5  for q = 1 to Q do
6      for j ∈ V' do
7          C = ∅
8          for P' ∈ F(j,q-1) do
9              /* operação (a) */
10             for u ∈ U : j ∈ C_u do
11                 P.ring=P'.ring
12                 P.star=P'.star +{uj}
13                 P.custo=P'.custo +w̃_uj
14                 C = C ∪ {P}
15         for i ∈ V' : i ≠ j do
16             for P' ∈ F(i,q-1) do
17                 if j ≠ k-1 predecessores de i em P'.ring then
18                     if i ∈ U then
19                         /* operação (b) */
20                         P.ring=P'.ring +{(i,j)}
21                         P.star=P'.star
22                         P.custo=P'.custo +c̃_ij
23                         C = C ∪ {P}
24                     else
25                         for u ∈ U : j ∈ C_u do
26                             /* operação (c) */
27                             P.ring=P'.ring +{(i,j)}
28                             P.star=P'.star +{uj}
29                             P.custo=P'.custo +c̃_ij + w̃_uj
30                             C = C ∪ {P}
31         F(j,q) = CaminhosÚteis(C)
```

Defina um ***set form*** como um conjunto de caminhos para os quais algum conjunto finito de posições estão fixadas. Um *set form* pode ser codificado por uma cadeia s de elementos em $V \cup \{.\}$, com $s_i =$ "." representando que a i-ésima posição dos caminhos não está fixada e, caso contrário, que a i-ésima posição dos caminhos está fixada com o vértice s_i. Por exemplo, o *set form* $(..a)$ denota o conjunto de caminhos $\{(aaa), (aba), (baa), (bba)\}$, se $V = \{a, b\}$.

Dado um caminho $P \in \mathcal{F}(s, v) \cap \mathcal{G}$, um caminho $\overline{P} = (v_1, \ldots, v_k, \ldots, v_t)$ pertence a $\mathcal{E}(P)$ somente se $\overline{P} \in \mathcal{G}$ e $(v_1, \ldots, v_k) \in \mathcal{E}(P)$. Portanto, apenas os k primeiros vértices de $\overline{P}$ são relevantes para determinar se $\overline{P} \in \mathcal{E}(P)$. Assim, um conjunto de *set forms* de comprimento k pode ser utilizado para representar o complemento de $\mathcal{E}(P)$.

Um ***self-hole set*** de um caminho P, denotado por $H(P)$, é o conjunto dos *set forms* que representam o conjunto de caminhos $\overline{P}$ que não podem estender P, ou seja, tais que $(P, \overline{P}) \notin \mathcal{G}$. Por exemplo, no caso em que $\mathcal{G}$ é definido sobre os caminhos livres de 4-ciclo, o *self-hole set* de um caminho $P = (a, b, c, d, e)$ é dado por $H(P) = \{(b...), (c...), (.c..), (d...), (.d..), (..d.), (e...), (.e..), (..e.), (...e)\}$. Note que alguns *set forms* podem ser descartados de um *self-hole set* $H(P)$ se o conjunto dos caminhos que ele representa já estiver contido em outro *set form* de $H(P)$, que é o caso, por exemplo, de $(bc..)$. Todo *self-hole set* de um caminho $P \in \mathcal{F}(s, v) \cap \mathcal{G}$ contém um único *set form* especial, em que a k-ésima posição é diferente de "." (de fato, a k-ésima posição é o vértice v). Uma maneira de diminuir a codificação dos *set forms* consiste em considerar implicitamente a existência de *set forms* contendo o vértice v em todo *self-hole set* de caminhos em $\mathcal{F}(s, v) \cap \mathcal{G}$ e codificar os demais *set forms* usando cadeias de comprimento $k - 1$.

Uma vez que $H(P)$ é o complemento de $\mathcal{E}(P)$, pode-se calcular $U(v)$ através de:

$$U(v) = \{P \in \mathcal{G} \cap \mathcal{F}(s, v) : \bigcap_{P_i \in \mathcal{G} \cap \mathcal{F}(s,v) : P_i \prec_{\mathrm{dom}} P} H(P_i) \not\subseteq H(P)\}. \tag{4.16}$$

O algoritmo de Irnich e Villeneuve para identificar caminhos úteis consiste em rotinas para calcular *self-hole set* e para realizar as operações de intersecção e continência de subconjuntos entre *self-hole sets*. A operação de intersecção de *self-hole sets* é $O(kpq)$, onde p e q são os tamanhos dos *self-hole set*. Irnich e Villeneuve provaram que o número de *set forms* obtidos como resultado de uma dessas operações de intersecção é limitado a $(k-1)!^2$. Devido a esse limitante, ao fato de $p = |H(P)|$, para algum caminho P, e $|H(P)| \leq k^2$, o custo da operação de intersecção é limitado a $kk^2(k-1)!^2$, ou seja, a $k(k!)^2$.

Por simplicidade, considere que existem apenas dois recursos e que o segundo recurso é tal que $t_{ij}^2 > 0$ para toda aresta ij.

Seja $\mathcal{C}$ uma seqüência $\{P_1, \ldots, P_t\}$ de caminhos em $\mathcal{G} \cap \mathcal{F}(s, v)$, tais que $res^2(P_i) = res^2(P_{i+1})$, $1 \leq i \leq t-1$. Denote por I_j, para $1 \leq j \leq t$, o conjunto resultante

da intersecção dos *self-hole sets* dos caminhos $P_1, \ldots, P_j$ de uma seqüência $\mathcal{C}$, ou seja, $I_j = \bigcap_{i=1}^{j} H(P_i)$. Por definição, considere $I_0 = \{(\ldots..)\}$, com $(\ldots..)$ denotando o *set form* codificado apenas por ".". Note que, pela definição da operação de intersecção, $I_0 \supseteq I_1 \supseteq \ldots \supseteq I_t$.

Suponha que uma seqüência $\mathcal{C}$ satisfaça $P_1 \prec_{\text{dom}} \ldots \prec_{\text{dom}} P_t$. Neste caso, temos que $res^1(P_1) \leq res^1(P_2) \leq \ldots \leq res^1(P_t)$. Logo, se $I_{i-1} = I_i$, para algum $1 \leq i \leq t$, temos que P_i é um caminho inútil em $\mathcal{C}$. A prova segue do fato de $P_1 \prec_{\text{dom}} \ldots \prec_{\text{dom}} P_i$ e $I_{i-1} = I_i$ implicar que $I_{i-1} \subseteq H(P_i)$.

Irnich e Villeneuve conjecturaram que $\alpha(k)$, ou seja, o número máximo de caminhos úteis em qualquer seqüência $\mathcal{C}$ de caminhos, é $k!$. A conjectura é válida para $k = 2$ devido a um trabalho anterior de Houck [29]. De modo diferente, Irnich e Villeneuve mostraram que a conjectura também vale para $k = 3$ usando um digrafo chamado **digrafo de intersecção**. Calcular o valor de $\alpha(k)$ equivale a determinar a maior seqüência $\mathcal{C}$ de caminhos tais que $I_0 \supset I_1 \ldots \supset I_{\alpha(k)}$ e $P_1 \prec_{\text{dom}} \ldots \prec_{\text{dom}} P_{\alpha(k)}$. O digrafo de intersecção é usado para representar todas as seqüências de caminhos $\mathcal{C}$ com a propriedade $I_{i-1} \supset I_i$, para todo i de 1 ao tamanho da seqüência.

No digrafo de intersecção, cada nó é um possível conjunto resultante da intersecção dos *self-hole sets* de uma seqüência qualquer de caminhos em $\mathcal{G} \cap \mathcal{F}(s, v)$ e um arco (I_i, I_j) pertence ao digrafo se e, somente, se existir um caminho P em $\mathcal{G} \cap \mathcal{F}(s, v)$ tal que $I_j = I_i \cap H(P)$ e $I_j \neq I_i$. A raiz do digrafo representa a intersecção do *self-hole set* do caminho (s, v), ou seja, a raiz é o vértice I_0. Desta forma, um caminho orientado $(I_0, I_1, \ldots, I_l)$ no digrafo, a partir da raiz até um nó I_l, define uma seqüência de caminhos $\mathcal{C} = \{P_1, \ldots, P_l\}$, na qual cada P_j, $1 \leq j \leq l$, é um caminho associado ao arco (I_{j-1}, I_j). Note que a seqüência $\mathcal{C}$ satisfaz $I_0 \supset I_1 \supset \ldots \supset I_l$. Portanto, desde que a seqüência de caminhos sejam tais que $res^2(P_i) = res^2(P_{i+1})$, $1 \leq i \leq l-1$, e $P_1 \prec_{\text{dom}} \ldots \prec_{\text{dom}} P_l$, o comprimento do maior caminho orientado no digrafo dá o valor de $\alpha(k)$.

Note ainda que o vértice v não ocorre nos *set form* de um *self-hole set* de um caminho em $\mathcal{G} \cap \mathcal{F}(s, v)$. Logo, um único digrafo de intersecção é suficiente para representar todas as coleções de caminhos em $\mathcal{G} \cap \mathcal{F}(s, v)$, qualquer que seja o vértice v.

A Figura 4.5 mostra uma parte do digrafo de intersecção para $k = 3$. Uma vez que apenas os $k-1$ vértices predecessores de v em um caminho P são relevantes para o cálculo de $H(P)$, cada arco do digrafo é rotulado pelos $k-1$ vértices predecessores de v em P. Na figura, o símbolo mais à direita do rótulo de um arco representa o vértice que precede imediatamente v no caminho em $\mathcal{G} \cap \mathcal{F}(s, v)$ associado ao arco. Os símbolos a, b, c e d são usados para representar quaisquer vértices distintos do grafo. Por exemplo, o rótulo *ba* no arco ligando o nó $\{(..)\}$ ao nó $\{(a.), (.a), (b.)\}$ indica que o arco está associado aos caminhos em $\mathcal{G} \cap \mathcal{F}(s, v)$ da forma $(s, \ldots, b, a, v)$, onde b e a representam quaisquer dois vértices distintos. Um exemplo de caminho associado a este arco é $P = (s, x, y, z, v)$.

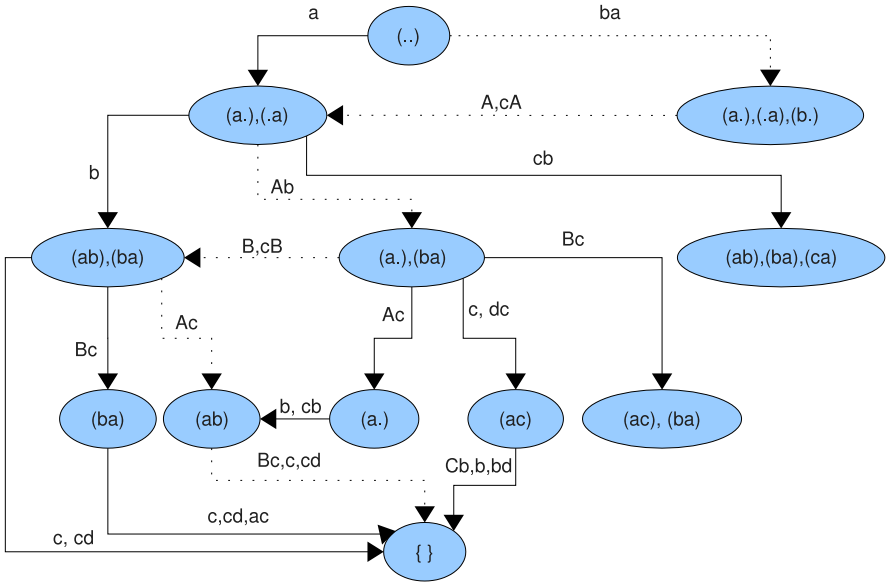


Figura 4.5: Digrafo de intersecção para 3-ciclos.

Note que $H(P) = \{(y.), (z.), (.z)\}$. Um símbolo em letra maiúscula no rótulo de um arco é usado para representar o mesmo vértice representado pelo símbolo correspondente em letra minúscula que ocorre no caminho orientado da raiz até aquele arco. Por exemplo, o rótulo A no arco que liga o nó $\{(a.), (.a), (b.)\}$ ao nó $\{(a.), (.a)\}$ representa o mesmo vértice representado por a no arco de rótulo ba, que liga $\{(..)\}$ a $\{(a.), (.a), (b.)\}$. Se P é o caminho associado ao último arco, temos que $A = a = z$.

O caminho orientado representado pelos arcos pontilhados no digrafo da Figura 4.5 define uma seqüência $\mathcal{C}$ de caminhos de comprimento $k! = 6$, que é o comprimento do maior caminho orientado no digrafo e, portanto, serve como prova da validade da conjectura de Irnich e Villeneuve para $k = 3$. Um exemplo de seqüência de comprimento 6 é $\mathcal{C} = \{(s, b, a, v), (s, c, a, v), (s, a, b, v), (s, c, b, v), (s, a, c, v), (s, b, c, v)\}$.

O Algoritmo 9 apresenta a função CaminhosUteis proposta por Irnich e Villeneuve [31].

Algoritmo 9: Algoritmo CaminhosUteis.

Input: seqüência de caminhos $\mathcal{C}$

Output: seqüência de caminhos úteis em $\mathcal{C}$

1 Ordene os caminhos em $\mathcal{C}$ e obtenha $\{P_1, \ldots, P_t\}$ tal que $res^2(P_i) < res^2(P_{i+1})$, $\forall i = 1, \ldots, t-1$.
2 $I_0 = \{(..)\}$
3 **for** $i = 1$ *to* t **do**
4 Calcule $H(P_i)$
5 Compute $I_i = H(P) \cap I_{i-1}$
6 **if** $I_{i-1} == I_i$ **then**
7 Remova P_i de $\mathcal{C}$, pois P_i é inútil

O passo (1) do algoritmo toma tempo $t \log t$. Os passos (4) e (6) gastam no total tk^2, enquanto o passo (5) é limitado a $tk(k!)^2$. Uma vez que t é, no máximo, $\alpha(k)$, tem-se que o custo do algoritmo para identificar caminhos úteis de Irnich e Villeneuve é $T(k) = \alpha(k)(\log \alpha(k) + k^2 + k(k)!^2)$.

Identificação de Caminhos Úteis Usando Autômatos Finitos

Nesta subseção apresentamos um novo algoritmo para identificar caminhos úteis. Ele utiliza autômatos finitos determinísticos para reconhecer caminhos úteis. Antes de apresentar o algoritmo proposto, uma breve revisão sobre autômatos é discutida.

Basicamente, um **autômato finito determinístico** (AFD) é uma máquina de estados projetada para reconhecer palavras de uma determinada linguagem. Dado um alfabeto, que é um conjunto de símbolos, uma palavra consiste em uma seqüência de símbolos do alfabeto e uma linguagem é um conjunto de palavras. Dado um AFD M, $L(M)$ representa a linguagem reconhecida pelo autômato M. A entrada de um AFD M é uma palavra w e a saída consiste em uma resposta afirmativa caso w pertença a $L(M)$, ou negativa, em caso contrário. Um AFD é formalmente definido por um quíntupla $M = (Q, \Sigma, \delta, q_0, F)$, onde Q define um conjunto finito de estados, Σ é o alfabeto da linguagem, δ é uma função em $Q \times \Sigma \mapsto Q$, q_0 é o estado inicial do AFD e F é o conjunto de estados finais. O processamento de um AFD é apresentado no Algoritmo 10. Ele consiste em, a partir do estado inicial, mudar de estado a cada símbolo da palavra de entrada, usando a função de transição δ. Quando uma transição não está definida para o estado atual e o símbolo atual da palavra de entrada, o AFD pára com resposta negativa. Caso todos os símbolos da palavra de entrada tenham sido processados com sucesso e um estado final tenha sido atingido, o AFD pára com resposta afirmativa.

Algoritmo 10: Algoritmo AFD.

Input: $M = (Q, \Sigma, \delta, q_0, F)$, palavra $w = a_1 \dots a_n$
Output: sim, caso w pertença a $L(M)$ e não, caso contrário.

```
1 q = q_0
2 i = 1
3 while i ≤ n e ∃δ(q, a_i) do
4 |   q = δ(q, a_i)
5 |   i = i + 1
6 if i < n ou q ∉ F then
7 |   devolva NÃO
8 else
9 |   devolva SIM
```

A idéia do novo algoritmo para identificar caminhos úteis consiste em construir um AFD $\overline{M}$ para reconhecer a linguagem constituída por todas as seqüências de caminhos $\mathcal{C} = \{P_1, \dots, P_t)$ em $\mathcal{G} \cap \mathcal{F}(s, v)$, tais que $res^2(P_i) = res^2(P_{i+1})$, $1 \leq i \leq t-1$, $P_1 \prec_{\text{dom}} \dots \prec_{\text{dom}} P_t$ e $I_0 \supset I_1 \supset \dots \supset I_t$. Neste caso, o alfabeto do AFD consiste no conjunto dos caminhos elementares de comprimento $k-1$, os quais representam caminhos em $\mathcal{G} \cap \mathcal{F}(s, v)$. O conjunto dos estados do autômato é formado pelos nós do digrafo de intersecção e os arcos no digrafo definem as transições do autômato. O estado inicial é a raiz do digrafo e o conjunto dos estados finais contém todos os estados do autômato.

O algoritmo proposto é apresentado no Algoritmo 11.

Algoritmo 11: Algoritmo CaminhosUteis-AFD.

Input: seqüência de caminhos $\mathcal{C}$
Output: seqüência de caminhos úteis em $\mathcal{C}$

```
1 Ordene os caminhos em C e obtenha {P_1, ..., P_t} tal que res^2(P_i) < res^2(P_{i+1}),
  ∀i = 1, ..., t − 1.
2 q = q_0
3 i = 1
4 while i ≤ t do
5 |   if ∃δ(q, a_i) then
6 |   |   q = δ(q, a_i)
7 |   else
8 |   |   Remova caminho P_i de C, pois é inútil
9 |   i = i + 1
```

No Algoritmo 11, os estados q_0 e as transições δ são aquelas definidas para o AFD $\overline{M}$. Note que o processamento nas linhas (4) a (9) correspondem ao processamento do AFD, exceto que ele não pára quando uma transição não está definida.

O tempo para processar uma palavra w em um AFD é $|w|$, isto é, o comprimento da palavra [1]. No caso da palavra formada pela seqüência de caminhos $\mathcal{C}$, esse comprimento é no máximo $\alpha(k)$, uma vez que esse valor é o comprimento do maior caminho no digrafo. No entanto, uma vez que cada símbolo do AFD corresponde aos $k-1$ vértices de um caminho, o tempo para identificar o próximo símbolo da cadeia de entrada do AFD, ou seja, o passo (5) do Algoritmo 11 é proporcional a $k-1$. Da mesma forma que no Algoritmo 9, o custo do passo (1) é $\alpha(k)\log\alpha(k)$ e, portanto, o custo total de CaminhosUteis usando-se AFD é $T(k) = \alpha(k)(\log\alpha(k)+k)$. Comparando-se com o custo do Algoritmo 9, houve uma redução de $\alpha(k)(k+k(k)!^2)$ no valor de $T(k)$ ao usar AFD.

4.4.4 Anel-estrela Q-capacitado Livre de k-*stream*

A proibição de k-ciclos evita a repetição de vértices ao longo do anel, mas não evita que ela ocorra entre os vértices do anel e da estrela e, também, ao longo da estrela. Por exemplo, a repetição do vértice a no anel-estrela da Figura 4.6 não é proibida através de k-ciclos, qualquer que seja o valor de k. Uma forma alternativa de apertar a relaxação do problema escravo consiste em evitar esse tipo de repetição.

Considere os vértices do anel-estrela formando uma única seqüência de vértices, aqui chamada de ***stream***. Fixada uma orientação dos vértices pendurados em cada vértice num anel-estrela (R, S), existe uma única *stream* associada a (R, S), que é definida pela ordem em que os vértices ocorrem no anel e pela orientação dos vértices pendurados, da seguinte forma:

- se $(i, j) \in R$ e i ocorre antes de j no anel, então, os vértices pendurados em j aparecem na *stream*, imediatamente, após i e antes de j.

Por exemplo, a *stream* associada ao anel-estrela da Figura 4.6 é `0acdbefab0`.

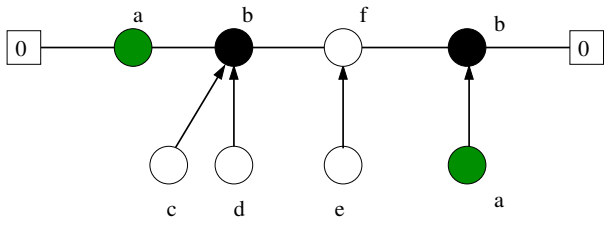


Figura 4.6: Exemplo de uma *stream* de um anel-estrela relaxado.

O **problema do anel-estrela relaxado Q-capacitado com proibição de k-*stream*** (kstream-RSP) consiste em encontrar um anel-estrela relaxado Q-capacitado de peso mínimo que não contenha k-***stream***, ou seja, ciclos de comprimento menor ou igual a k na *stream* associada ao anel-estrela. Note que, para um k fixo, proibir k-*stream* não implica necessariamente na eliminação de k-ciclos. No entanto, alguns k-ciclos que ocorrem no anel podem corresponder a k-ciclos na *stream* e, portanto, os limitantes duais obtidos pelo modelo do kstream-RSP podem ser mais apertados que aqueles gerados pelo kcyc-RSP.

Uma vez que cada uma das 3 operações que estendem um anel-estrela aumenta em pelo menos uma unidade o comprimento da *stream* associada, o algoritmo para computar a matriz F para passeios-estrelas livres de k-ciclos continua sendo um algoritmo *Label Setting* e pode ser utilizado para encontrar caminhos mínimos livres de k-*streams* sem nenhuma alteração adicional.

4.5 Desigualdades Válidas

Nesta seção discutimos as desigualdades válidas propostas em [4] para a formulação compacta apresentada na subseção 4.3.1 o CmRSP. Neste trabalho, estas desigualdades são estendidas e utilizadas para fortalecer o modelo por cobertura de conjuntos apresentado nas seções anteriores dentro de um algoritmo de *branch-and-cut-and-price*. Nas descrições das desigualdades que se seguem, considere as variáveis x, y e z definidas como no modelo dado na subseção 4.3.1.

4.5.1 Desigualdades de Conectividade

Desigualdades de conectividade impõem um número mínimo de anéis com arestas no corte $\delta(S)$, onde S é um subconjunto de vértices em V'. O limitante é definido pelo dobro do número de clientes em S que pertencem a anéis ou que estão pendurados em vértices em S.

Uma desigualdade de conectividade é da forma:

$$\sum_{e \in \delta(S)} x_e \geq 2(y_u + \sum_{j \in C_u \cap S} z_{uj}), \forall S \subseteq V', \forall u \in S \cap U. \tag{4.17}$$

Estas desigualdades, quando incluídas na formulação compacta (CF), ajudam a impor conectividade e, consequentemente, a evitar a formação de subciclos, uma vez que isso não é garantido pela restrição (4.4) quando o lado direito da restrição é menor que dois.

4.5.2 Desigualdades Anel Multi-Estrela

Seja S um subconjunto de vértices em V'. As desigualdades anel multi-estrela são obtidas das restrições (4.4) e do fato de que, pelo menos, duas arestas dos anéis-estrelas devem entrar em S não apenas quando um cliente em S pertence a algum anel-estrela ou quando algum cliente é pendurado a vértices em S mas, também, quando alguma aresta é usada para ligar um vértice de S a um cliente externo a S.

As desigualdades anel multi-estrela são definidas por:

$$\sum_{e\in\delta(S)} x_e \geq \frac{2}{Q}\Big(\sum_{i\in U\cap S} y_i + \sum_{i\in U}\sum_{j\in S\cap C_i} z_{ij} + \sum_{i\in U\setminus S}\sum_{j\in S} x_{ij}\Big), \forall S \subseteq V'. \tag{4.18}$$

4.5.3 Desigualdades de Capacidade

Três desigualdades de capacidade também foram estudadas em [4] e obtidas a partir de algum tipo de arredondamento das restrições (4.4). São elas: desigualdades C_i-anel-capacitado, capacidade-arredondada I e capacidade-arredondada II.

C_i-anel-capacitado Seja S um subconjunto de vértices de V'. As desigualdades C_i-anel-capacitado são obtidas pelo fato de, em algumas situações, o conjunto C_i, isto é, os vértices aos quais um cliente i pode se conectar, estar completamente contido em S. Neste caso, um limitante inferior para o número de anéis-estrelas passando por S é dado por:

$$\left\lceil \frac{|\{i \in U : C_i \subseteq S\}|}{Q} \right\rceil.$$

Usando este fato, as desigualdades C_i-anel-capacitado são caracterizadas por:

$$\sum_{e\in\delta(S)} x_e \geq 2\left\lceil \frac{|\{i \in U : C_i \subseteq S\}|}{Q} \right\rceil, \forall S \subseteq V' : S \cap U \neq \emptyset. \tag{4.19}$$

Considere que $mod(a, b)$ denote o resto da divisão inteira de a por b.

O seguinte lema é usado para obter as desigualdades capacidade-arredondada I e II:

Lema 4.3. *Se α, β e γ são inteiros não-negativos com $\alpha > \gamma$ e $mod(\alpha, \gamma) \neq 0$ então vale:*

$$\left\lceil \frac{\alpha - \beta}{\gamma} \right\rceil \geq \left\lceil \frac{\alpha}{\gamma} \right\rceil - \frac{\beta}{mod(\alpha, \gamma)}.$$

Capacidade-arredondada I As desigualdades capacidade-arredondada I são derivadas de (4.4), do Lema 4.3 e do fato do somatório do lado direito de (4.4) satisfazer:

$$\sum_{i\in U}\sum_{j\in S} z_{ij} + \sum_{i\in U\cap S} y_i = |U| - (\sum_{i\in U}\sum_{j\in C_i\setminus S} z_{ij} + \sum_{i\in U\setminus S} y_i).$$

Substituindo o resultado acima em (4.4) temos que:

$$\sum_{e\in\delta(S)} x_e \geq 2\left\lceil \frac{|U| - (\sum_{i\in U}\sum_{j\in C_i\setminus S} z_{ij} + \sum_{i\in U\setminus S} y_i)}{Q} \right\rceil.$$

Usando o Lema 4.3 com $\alpha = |U|$, $\beta = \sum_{i\in U}\sum_{j\in C_i\setminus S} z_{ij} + \sum_{i\in U\setminus S} y_i$ e $\gamma = Q$, obtemos as desigualdades capacidade-arredondada I:

$$\sum_{e\in\delta S} x_e \geq 2(\left\lceil \frac{|U|}{Q} \right\rceil - \frac{(\sum_{i\in U}\sum_{j\in C_i\setminus S} z_{ij} + \sum_{i\in U\setminus S} y_i)}{mod(|U|, Q)}), \forall S \subseteq V' : S\cap U \neq \emptyset. \quad (4.20)$$

Capacidade-arredondada II As desigualdades capacidade-arredondada II são obtidas de forma análoga às desigualdades capacidade-arredondada I. Note que

$$\sum_{i\in U\cap S}\sum_{j\in C_i\cap S} z_{ij} + \sum_{i\in U\cap S} y_i = |U\cap S| - \sum_{i\in U\cap S}\sum_{j\in C_i\setminus S} z_{ij},$$

que substituindo em (4.4) gera a seguinte desigualdade:

$$\sum_{e\in\delta(S)} x_e \geq 2\left\lceil \frac{|U\cap S| - \sum_{i\in U\cap S}\sum_{j\in C_i\setminus S} z_{ij}}{Q} \right\rceil.$$

Usando o Lema 4.3 para $\alpha = |U\cap S|$, $\beta = \sum_{i\in U\cap S}\sum_{j\in C_i\setminus S} z_{ij}$ e $\gamma = Q$ obtemos as desigualdades capacidade-arredondada II:

$$\sum_{e\in\delta S} x_e \geq 2(\left\lceil \frac{|U\cap S|}{Q} \right\rceil - \frac{\sum_{i\in U\cap S}\sum_{j\in C_i\setminus S} z_{ij}}{mod(|U\cap S|, Q)}), \forall S \subseteq V' : S\cap U \neq \emptyset. \quad (4.21)$$

Todas as desigualdades válidas descritas acima podem ser incluídas na formulação (SC) para apertar sua relaxação linear, bastando para isso, reescrevê-las em termos das variáveis λ. Isso pode ser realizado utilizando-se as equações abaixo:

$$x_e = \sum_{p\in P} r_e^p \lambda_p, \quad z_{ij} = \sum_{p\in P} s_{ij}^p \lambda_p, \quad \text{e} \quad y_i = \sum_{p\in P} r_i^p \lambda_p.$$

Capítulo 5

Implementação

Neste capítulo apresentamos os detalhes de implementação e os resultados dos testes empíricos do algoritmo *branch-and-cut-and-price* para resolver o CmRSP, baseado no modelo proposto no Capítulo 4.

5.1 Branch-Cut-and-Price

No código do *branch-and-price* e *branch-and-cut-and-price*, os seguintes critérios de implementação foram adotados.

Base inicial. Uma vez que a formulação do problema mestre do CmRSP não possui uma base inicial trivial, um total de $|U|$ variáveis artificiais foram adicionadas ao modelo. Para cada vértice $i \in U$, uma variável artificial t_i é introduzida ao modelo com coeficiente 1 na restrição de cobertura de i e com coeficientes nulos nas demais restrições. Um limitante superior para o custo de uma solução do CmRSP foi utilizado como coeficiente destas variáveis na função objetivo. A base inicial é então formada pelas variáveis artificiais e as variáveis de folga das restrições do modelo. Para acelerar a convergência da geração de colunas, um conjunto adicional de colunas são colocadas na base inicial. Essa idéia é conhecida como "povoar" a base e tem sido usada em algoritmos de geração de colunas. As colunas adicionais são geradas por uma heurística primal para o CmRSP. As colunas que fazem parte das soluções geradas são usadas para povoar a base. A heurística utiliza, em parte, a metaheurística *Greedy Randomized Adaptative Search Procedure* (GRASP) [20].

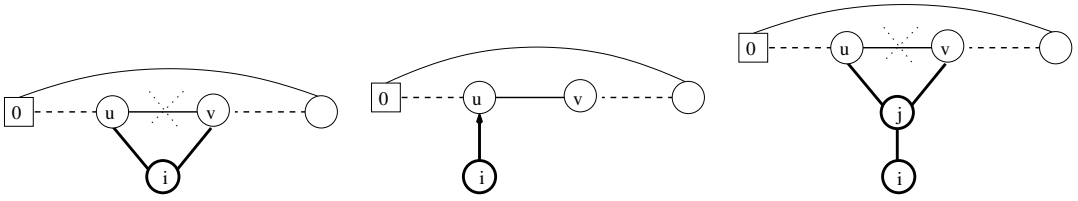
Antes da heurística, uma breve revisão sobre GRASP é apresentada. Basicamente, um GRASP consiste em duas fases: construtiva e busca local. Na fase construtiva, candidatos para a construção de uma solução são inicialmente ponderados segundo uma função gulosa e adaptativa. Uma lista restrita de candidatos (RCL) é formada com os candidatos melhor ponderados e um deles é selecionado aleatoriamente para ser incluído

na solução. A função que pondera os candidatos é adaptativa no sentido em que muda os pesos dos candidatos à medida que a solução é construída. Na segunda fase são aplicadas sucessivas operações de busca local para melhorar o peso da solução construída na primeira fase.

A heurística utilizada para "povoar" a base consiste num algoritmo GRASP sem a fase de busca local. Optou-se por remover essa fase por ser a mais custosa em termos de tempo de processamento e, no caso do CmRSP, por apresentar poucos ganhos na qualidade das soluções geradas.

O Algoritmo 12 é um pseudocódigo em alto nível para a heurística proposta. A idéia do algoritmo é construir os m anéis-estrelas iterativamente, incluindo um cliente não coberto em um dos anéis-estrelas a cada iteração. Inicialmente, m anéis-estrelas são construídos, cada um deles cobrindo um cliente diferente. Essa etapa é realizada nas linhas (1) a (11). RS é um vetor com m posições, em que cada uma delas contém as seguintes informações de um anel-estrela: arestas do anel, arcos da estrela, custo do anel-estrela e o total de clientes cobertos pelo anel-estrela. Após a construção dos m anéis-estrelas iniciais, cada cliente não coberto é iterativamente selecionado e incluído em um dos anéis-estrelas. A escolha do candidato é realizada de forma aleatória sob uma lista RCL de candidatos promissores, como descrito no Algoritmo 13. Aqui, um candidato refere-se a uma das seguintes três operações (veja Figura 5.1) para estender um anel-estrela e assim cobrir um determinado cliente:

- (a): inclusão de arestas. Implica em incluir um cliente i entre dois vértices u e v no anel;
- (b): inclusão de arcos. Consiste em pendurar um cliente i em um vértice u que está no anel;
- (c): inclusão de arestas e arcos. Um *Steiner* j é incluído entre dois vértices u e v no anel e um cliente i é pendurado em j.



(a) incluindo arestas (b) incluindo arco (c) incluindo arco e arestas

Figura 5.1: Operações para estender um anel-estrela e cobrir um cliente.

Algoritmo 12: Heurística Inicial.

Input: grafo $G = (\{0\} \cup U \cup W, E \cup A)$, inteiros Q, m, custos de roteamento c e de conexão w

Output: m anéis-estrelas Q-capacitados

```
1  Cobertos=∅
2  for r = 1 to m do
3      for i ∈ U\Cobertos do
4          v.entra={(0,i),(i,0)}; v.arco={}; v.custo=c_0i + c_i0
5          Candidatos = Candidatos ∪{v}
6          for ij ∈ A e j ∈ W\Cobertos do
7              v.entra={(0,j),(j,0)}; v.arco={ij}; v.custo=c_0j + c_j0 + w_ij
8              Candidatos = Candidatos ∪{v}
9      v=SelecionaCandidato(Candidatos)
       /* inclui o candidato escolhido no anel-estrela r */
10     RS[r].anel=v.entra; RS[r].estrela=v.arco; RS[r].custo=v.custo; RS[r].total=1
11     Cobertos=Cobertos ∪V[v.entra] ∪ V[v.arco]
12 while U\Cobertos ≠ ∅ do
       /* Recalcula menor custo para incluir cada cliente não coberto */
13     Candidatos= ∅
14     for i ∈ U\Cobertos do
15         v.custo = ∞
16         for r = 1 to m do
17             if RS[r].total< Q then
18                 for (u,v) ∈ RS[r].anel do
                       /* operacao a */
19                     if v.custo > RS[r].custo + c_ui + c_iv − c_uv then
20                         v.sai = {(u,v)}; v.entra={(u,i),(i,v)}; v.arco={}; v.local=r;
                           v.custo=RS[r].custo+c_ui + c_iv − c_uv
                       /* operacao b */
21                     if iu ∈ A e v.custo >RS[r].custo + w_iu then
22                         v.sai = {}; v.entra={}; v.arco={iu}; v.local=r;
                           v.custo=RS[r].custo+w_iu
23                     for ij ∈ A e j ∈ W\Cobertos do
                           /* operacao c */
24                         if v.custo >RS[r].custo + c_uj + c_jv − c_uv + w_ij then
25                             v.sai = {(u,v)}; v.entra={(u,j),(j,v)}; v.arco={ij};
                               v.local=r; v.custo=RS[r].custo+c_uj + c_jv − c_uv + w_ij
26         Candidatos = Candidatos ∪{v}
       /* ver continuação na próxima página */
```

Algoritmo 12: Heurística Inicial (continuação)

```
12 while U\Cobertos ≠ ∅ do
13     ...
27     v=SelecionaCandidato(Candidatos)
       /* inclui o candidato escolhido no melhor local */
28     RS[v.local].anel=RS[v.local].anel +v.entra −v.sai
29     RS[v.local].estrela=RS[v.local].estrela +v.arco
30     RS[v.local].custo=v.custo
31     RS[v.local].total=RS[v.local].total +1
32     Cobertos=Cobertos ∪V[v.entra] ∪ V[v.arco]
33 Devolva RS[1],..., RS[m]
```

Nas linhas (12) a (26) do Algoritmo 12, um candidato de menor peso é construído para cada cliente não coberto. Em seguida, na linha (27), um dos candidatos é selecionado. A operação selecionada é então executada, estendendo-se um dos anéis-estrelas para cobrir um cliente. Por fim, a lista Cobertos é atualizada com os vértices que foram adicionados ao anel-estrela na última operação. Esse processo se repete até que todos os clientes tenham sido cobertos.

Algoritmo 13: SelecionaCandidato

```
   Input: Uma lista L
   Output: um elemento de L
   /* escolhe um dos candidatos promissores aleatoriamente */
1  menor= min{v.custo : v ∈ L }
2  RCL= {v ∈ L : v.custo ≤ (1 + ε)∗menor}
3  Escolha aleatoriamente um elemento v em RCL
4  Devolva v
```

A construção dos m anéis-estrelas iniciais nas linhas (1) a (11) é $O(m(|U| + |A|))$. Considere um cliente i não coberto. Gerar um candidato para i pela operação (a) ou (b) leva tempo $O(mQ)$, enquanto que pela operação (c) é $O(mQ|C_i|)$. Logo, o tempo gasto pelo algoritmo nas linhas (14) a (26) é $O(mQ|A||U|)$ e, portanto, o Algoritmo 12 é $O(mQ|A||U|^2)$.

Regra de *Branching*. A regra de *branching* adotada é derivada da idéia de Lysgaard et al. [34] e baseia-se na seguinte desigualdade válida para o CmRSP:

$$\sum_{e\in\delta(S)}\sum_{p\in P} r_e^p \lambda_p \geq 2\left\lceil \frac{|\{i\in U|\hat{C}_i\subseteq S\}|}{Q}\right\rceil, S\subseteq V', S\neq\emptyset, \tag{5.1}$$

onde $\hat{C}_i = C_i \cup \{i\}$.

O lado direito da Inequação (5.1) totaliza o número de arestas no corte $(S,\overline{S})$ que fazem parte da solução. O termo $|\{i\in U|\hat{C}_i\subseteq S\}|$ contabiliza o número de clientes que precisam ser cobertos pelos vértices em S. Assim, o valor de $\left\lceil\frac{|\{i\in U|\hat{C}_i\subseteq S\}|}{Q}\right\rceil$ é um limite inferior para o número de anéis-estrelas necessários para cobrir os clientes em S. Portanto, o lado esquerdo da Inequação (5.1) define o número mínimo de arestas de cortes em $(S,\overline{S})$.

Denote por $f(S)$ a diferença entre o lado esquerdo e direito da Inequação (5.1). Uma vez que o valor do lado esquerdo da inequação é sempre par, uma solução fracionária λ^* em um nó da árvore de enumeração pode violar a inequação causando $0 < f(S) < 2$. Neste caso, os seguintes cortes podem ser usados para *branching*: $f(S) = 0$ and $f(S) \geq 2$. A heurística de separação para encontrar este corte de *branching* segue de [4] e é descrita pelo Algoritmo 14.

Algoritmo 14: Heurística de separação de cortes de *branching*.

```
Input: solucao λ*, grafo G = (V, E ∪ A)
Output: corte de branching
for ij ∈ E do
    x*_ij = Σ_{p∈P} a^p_ij λ*_p
fS = ∞
for u ∈ U do
    (S', fS') =CalculaCorte(x*, u, G)
    if |fS − 1.0| > |fS' − 1.0| then
        fS = fS'
        S = S'
if fS > 0 e fS < 2 then
    Devolva restrições de branching:
    Σ_{ij∈δ(S)} x_ij = 2
    Σ_{ij∈δ(S)} x_ij ≥ 4
```

A idéia da heurística consiste em procurar um subconjunto S de vértices que mais viole a Inequação (5.1), ou seja, tal que $f(S)$ mais se aproxime de 1.0. Para diminuir o

custo computacional do cálculo de $f(S)$, computa-se o valor das variáveis, da formulação compacta, x^*_{ij}, para cada aresta (i, j), como descrito nas linhas (1) e (2) do Algoritmo 14. Nas linhas (3) a (8) a rotina CalculaCorte(x^*, u, G) é chamada para construir um subconjunto S tal que $u \in S$ e o valor $f(S)$ mais se aproxima de 1.0. Dentre os subconjuntos construídos, o melhor deles é selecionado e usado para gerar as restrições de *branching* nas linhas (9) a (12).

Note que a heurística pode falhar na procura por uma Inequação (5.1) violada. Neste caso, a regra de *branching* consiste em fixar uma aresta do grafo a participar ou não do anel de algum anel-estrela. As seguintes restrições são adicionadas, um em cada nó filho: $\sum_{p\in\mathcal{P}} a^p_{ij}\lambda_p \geq 1$ e $\sum_{p\in\mathcal{P}} a^p_{ij}\lambda_p \leq 0$.

A rotina CalcularCorte(x^*, u, G), descrita no Algoritmo 15, consiste em um algoritmo guloso que utiliza uma função peso $f : V \mapsto \mathbb{R}$ para cada vértice i do grafo e representa o incremento no valor de $f(S)$ ao incluir i em S. O algoritmo é iterativo e em cada iteração um vértice que dá a melhor contribuição para que fS se aproxime de 1.0 é escolhido e incluído em S. A cada inclusão de um vértice em S, o peso de cada vértice é atualizado. Esse processo se repete enquanto houver melhoria no valor de $f(S)$, ou seja, enquanto $f(S)$ se aproximar de 1.0.

O subconjunto S é inicializado na linha (1) com todos os vértices que cobrem o cliente u e, desta forma, garantir que o lado esquerdo da Inequação (5.1) seja maior que zero. O valor de $f[i]$ computado nas linhas (5) e (6) para cada vértice i representa o incremento no valor de fS devido às arestas de cortes que surgirão quando i entrar em S e o decremento com as arestas que deixarão de ser de corte.

Duas variáveis são mantidas e atualizadas a cada iteração do algoritmo para permitir que o valor do lado direito da Inequação (5.1) seja calculado mais rapidamente. Para cada cliente i é computado o valor de `falta`$[i]$, que representa o número de vértices que faltam ser incluídos em S para que i esteja inteiramente coberto por vértices de S, ou seja, para que $\hat{C}_i \subseteq S$. Para cada vértice j é calculado o valor de `contrib`$[j]$, que representa o aumento no lado direito da Inequação (5.1), caso o vértice j seja incluído em S.

Nas linhas (10) e (11), o valor de fS é atualizado e o vértice que apresentou a melhor contribuição para fS é incluído em S. Os únicos valores de `falta` e `contrib` afetados com a inclusão do vértice `menor` em S são aqueles relativos aos vértices em $\hat{C}_{\texttt{menor}} \cup \{j \in U :$ `menor` $\in C_j\}$ (ver linhas (12) a (20). Se `menor` é um cliente, tem-se que `menor` $\in \hat{C}_{\texttt{menor}}$ e, portanto, o valor de `falta`[`menor`] deve ser decrementado em uma unidade. Todo cliente j coberto por `menor` também tem o valor de `falta`$[j]$ decrementado em uma unidade, uma vez que um dos vértices que o cobre foi incluído em S. Em ambos os casos, se o valor de `falta` tornar-se igual a 1 (um), a contribuição (valor de `contrib`) de algum vértice é incrementada. Por fim, nas linhas (21) a (22) o valor de $f[i]$ é atualizado para cada vértice i tendo como base o fato de que a aresta $(i,$ `menor`$)$ era contabilizada em $f[i]$

Algoritmo 15: Algoritmo CalculaCorte.

Input: solução x^*, vértice u, grafo G
Output: subconjunto $S \subset V$ e valor de $f(S)$

$S = \hat{C}_u$
$\texttt{falta}[i] = |\hat{C}_i \setminus S|, \forall i \in U$
$\texttt{contrib}[j] = |\{i \in U : \texttt{falta}[i] = 1 \text{ e } j \in \hat{C}_i\}|, \forall j \in V'$
$fS = \sum_{ij \in \delta(S)} x^*_{ij} - 2 \left\lceil \frac{|\{i \in U : \texttt{falta}[i]=0\}|}{Q} \right\rceil$
`/* Calcula parte da contribuição de cada vértice para` fS `se for incluído em` S `*/`
for $i \in V' \setminus S$ **do**
  $f[i] = \sum_{j \notin S} x^*_{ij} - \sum_{j \in S} x^*_{ij}$
`/* Calcula novo fS que melhor se aproxima de 1.0 */`
$\texttt{dif} = \min_{i \in V \setminus S} |fS + f[i] - 2\big(\left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}| + \texttt{contrib}[i]}{Q} \right\rceil - \left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}|}{Q} \right\rceil \big) - 1.0|$
while $\texttt{dif} < |fS - 1.0|$ **do**
  $\texttt{menor} = \arg\min_{i \in V \setminus S} |fS + f[i] - 2\big(\left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}| + \texttt{contrib}[i]}{Q} \right\rceil - \left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}|}{Q} \right\rceil \big) - 1.0|$
  `/* Atualiza` fS `*/`
  $fS = fS + f[\texttt{menor}] - 2\big(\left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}| + \texttt{contrib}[\texttt{menor}]}{Q} \right\rceil - \left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}|}{Q} \right\rceil \big)$
  $S = S + \{\texttt{menor}\}$
  `/* Atualiza contribuição de cada vértice */`
  if $\texttt{menor} \in U$ **then**
    $\texttt{falta}[\texttt{menor}] = \texttt{falta}[\texttt{menor}] - 1$
    if $\texttt{falta}[\texttt{menor}] = 1$ **then**
      for $j \in \hat{C}_{\texttt{menor}} \setminus S$ **do**
        $\texttt{contrib}[j] = \texttt{contrib}[j] + 1$
  for $i \in U : \texttt{menor} \in \hat{C}_i$ **do**
    $\texttt{falta}[i] = \texttt{falta}[i] - 1$
    if $\texttt{falta}[i] = 1$ **then**
      $\texttt{contrib}[i] = \texttt{contrib}[i] + 1$
  for $i \in V' \setminus S$ **do**
    $f[i] = f[i] - 2x^*_{i,\texttt{menor}}$
  `/* Calcula novo fS que melhor se aproxima de 1.0 */`
  $\texttt{dif} = \min_{i \in V \setminus S} |fS + f[i] - 2\big(\left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}| + \texttt{contrib}[i]}{Q} \right\rceil - \left\lceil \frac{|\{j \in U : \texttt{falta}[j]=0\}|}{Q} \right\rceil \big) - 1.0|$
Devolva $(S,\ fS)$

como uma futura aresta de corte (quando i fosse adicionado a S) e também já está sendo contabilizada em fS como uma aresta de corte.

Os cálculos nas linhas (1) a (3) são $O(|A|)$, enquanto as linhas (4) a (6) tomam tempo $O(|V|^2)$. O laço da linha (8) se repete no máximo $|V|$ vezes e, portanto, uma vez que as operações internas ao laço tomam tempo $O(|V|)$, tem-se que as linhas (8) a (23) levam tempo $O(|V|^2)$. Logo, a complexidade de tempo da heurística de separação de cortes de *branching* é $O(|U||V|^2)$.

Cálculo do Limitante Dual de um Nó. Durante a operação de *branching*, a relaxação linear de cada novo nó é otimizada e, então, uma única iteração da geração de colunas é processada para que o cálculo do limitante dual de Lasdon [32] (ver Capítulo 2) possa ser efetuado. O limitante dual do nó é, então, escolhido como o maior entre o limitante de Lasdon e o limitante dual do nó pai.

Seleção de Nó. O limitante dual em cada nó da árvore de enumeração é utilizado como critério para selecionar o próximo nó ativo da árvore de *branch-and-bound* a ser otimizado. O nó ativo com menor limitante dual é selecionado.

Pricing. No passo de *pricing*, nem toda coluna de custo reduzido negativo encontrada é adicionada à formulação do problema mestre reduzido. Uma coluna de custo reduzido negativo é adicionada somente se o anel-estrela correspondente à coluna: (1) não é um anel-estrela relaxado; ou (2) tiver o menor custo reduzido dentre todos os anéis-estrelas com mesma extremidade (correspondentes às colunas geradas).

Método de Estabilização Nenhum dos métodos de estabilização discutidos no Capítulo 2 foram habilitados, uma vez que testes preliminares indicaram que a inclusão destes métodos não foi útil para acelerar a convergência do método da geração de colunas nesta implementação.

5.2 Rotinas de Separação

No algoritmo *branch-and-cut-and-price* separamos as seguintes desigualdades válidas: desigualdades de conectividade (4.17), de anel multi-estrela (4.18), C_i-anel-capacitado (4.19), capacidade-arredondada I (4.20) e as capacidade-arredondada II (4.21) descritas na Seção 4.5. Para separar essas desigualdades, utilizamos as mesmas rotinas de separação descritas em [4], exceto que as desigualdades capacidade-arredondada II violadas foram separadas usando a mesma estratégia descrita em [4] para separar as desigualdades C_i-anel-capacitado. Esta seção descreve tais rotinas em mais detalhes.

Dada uma solução ótima do problema mestre restrito (x^*, z^*, y^*), considere $G^* = (V^*, E^*)$ o grafo suporte correspondente, onde $V^* = \{0 \cup \hat{V}^*$, $\hat{V}^* = \{i \in V' : y_i^* > 0\}$ e $E^* = \{(i,j) \in E : x_{i,j}^* > 0\}$.

Separação das desigualdades de conectividade. Seja u um vértice em U e $S \subseteq V'$. Uma vez que $y_u + \sum_{j \in C_u} z_{uj} = 1$ e $\sum_{j \in C_u} z_{uj} = \sum_{j \in C_u \cap S} z_{uj} + \sum_{j \in C_u \setminus S} z_{uj}$, temos que

$$y_u + \sum_{j \in C_u \cap S} z_{uj} = 1 - \sum_{j \in C_u \setminus S} z_{uj}.$$

Logo, a desigualdade (4.17) pode ser reescrita na seguinte forma:

$$\sum_{e \in \delta(S)} x_e + 2(y_u + \sum_{j \in C_u \setminus S} z_{uj}) \geq 2, \forall S \subseteq V', \forall u \in S \cap U.$$

Para encontrar a desigualdade (4.17) mais violada por (x^*, z^*, y^*) resolve-se o problema de corte $0 - u$ mínimo sob o grafo não-orientado e capacitado $\bar{G} = (\bar{V}, \bar{E})$, onde $\bar{V} = (V^* \cap V')$ e $\bar{E} = \{(i,j) \in E^* : i, j \in V\}$. Toda aresta (i,j) é associada a uma capacidade de $x_{i,j}^*$, exceto as arestas (u,j) que têm capacidade de $x_{u,j}^* + 2z_{uj}^*$.

Seja $(S, \bar{V} \setminus S)$ um corte $0 - u$ mínimo em $\bar{G}$ com $u \in S$. Logo, se a capacidade do corte for estritamente menor que 2, o conjunto S define a desigualdade (4.17) mais violada. Por outro lado, se a capacidade do corte mínimo for maior ou igual a 2 não existe desigualdade (4.17) violada por (x^*, z^*, y^*).

Separação das desigualdades anel multi-estrela. Uma vez que

$$\begin{aligned} \textstyle\sum_{e \in \delta(S)} x_e &= \textstyle\sum_{i \in V \setminus S} \sum_{j \in S} x_{ij} \\ &= \textstyle\sum_{i \in U \setminus S} \sum_{j \in S} x_{ij} + \sum_{i \in V \setminus (S \cup U)} \sum_{j \in S} x_{ij} \end{aligned}$$

e

$$\sum_{i \in U} (y_i + \sum_{j \in C_i \cap S} z_{ij} + \sum_{j \in C_i \setminus S}) = |U|,$$

a desigualdade (4.18) pode ser reescrita na seguinte forma:

$$\sum_{i \in V \setminus (S \cup U)} \sum_{j \in S} x_{ij} + (1 - \frac{2}{Q}) \sum_{i \in U \setminus S} \sum_{j \in S} x_{ij}) + \frac{2}{Q} (\sum_{i \in U} \sum_{j \in C_i \setminus S} z_{ij}) \geq \frac{2|U|}{Q}, \forall S \subseteq V'.$$

Considere o grafo orientado capacitado $\bar{G} = (\bar{V}, \bar{E})$, com $\bar{V} = (V^* \cap V') \cup \{t\}$, onde t é um vértice destino fictício e $\bar{E} = \{(i,j), (j,i) : i, j \in V, (i,j) \in E^*\} \cup \{(j,t) : j \in V^* \cap V'\}$.

Cada arco (i, j), $i, j \neq t$, tem capacidade $(1 - 2/Q)x^*_{ij}$, se $i \in U$, caso contrário, x^*_{ij}, e cada arco (j, t) tem capacidade $(2/Q)\sum_{i \in U} z^*_{ij}$.

Se $(S, V \setminus S)$ é um corte $0 - t$ mínimo em $\bar{G}$ com capacidade estritamente menor que $2|U|/Q$ então o conjunto $S \setminus \{t\}$ define a desigualdade (4.18) mais violada. Caso a capacidade do corte seja menor ou igual a $2|U|/Q$, não existe tal desigualdade violada por (x^*, z^*, y^*).

Separação das desigualdades de capacidade. A desigualdade (4.20) pode ser reescrita na seguinte forma:

$$\sum_{e \in \delta S} x_e + \alpha(\sum_{i \in U} \sum_{j \in V' \setminus S} z_{ij} + \sum_{i \in U \setminus S} y_i) \geq \beta, \forall S \subseteq V' : S \cap U \neq \emptyset.$$

onde $\alpha = 2/mod(|U|, Q)$ e $\beta = 2\lceil(\rceil |U|/Q)$.

Dados α e β, a desigualdade (4.20) mais violada pode ser encontrada resolvendo-se o corte $0 - t$ mínimo sobre o grafo não-orientado e capacitado $\bar{G} = (\bar{V}, \bar{E})$ com $\bar{V} = (V^* \cap V') \cup \{t\}$, onde t é um vértice destino fictício e $\bar{E} = \{(i, j) \in E^* : i, j \in V\} \cup \{(j, t) : j \in V^* \cap V'\}$. Toda aresta (j, t) é associada a uma capacidade $\alpha \sum_{i \in U} z_{ij}$, se $j \in W$ ou a $\alpha(\sum_{i \in U} z_{ij} + y_j)$, se $j \in U$, e as demais arestas (i, j) à capacidade de $x^*_{i,j}$. Seja $(\bar{S}, \bar{V} \setminus \bar{S})$ um corte $0-t$ mínimo em $\bar{G}$ e suponha que $t \in \bar{S}$. Se a capacidade do corte for estritamente menor que β, temos que o conjunto $S = \bar{S} \setminus \{t\}$ define a desigualdade (4.20) mais violada. Por outro lado, nenhuma desigualdade violada existe caso a capacidade de corte seja maior ou igual a β.

5.3 Instâncias de Teste

Os testes foram realizados sobre três classes de instâncias, denotadas por A,B e C. As duas primeiras classes são definidas em [4]. A terceira classe foi definida para conter instâncias com um número maior de conexões. Cada uma das classes contém 96 instâncias geradas a partir das instâncias `eil51.tsp`, `eil76.tsp` e `eil101.tsp` da TSPLIB [45], conforme especificado em [4]. As instâncias denominadas por `eil26.tsp` contêm 26 vértices e foram geradas a partir dos 26 primeiros pontos de `eil51.tsp`.

As instâncias da TSPLIB contêm as coordenadas de um conjunto de pontos no plano. Cada ponto é associado a um vértice do grafo. Seguindo o padrão adotado por Baldacci et al. em [4], o primeiro ponto define as coordenadas do depósito. Os próximos $|U|$ pontos definem os clientes e os restantes, os pontos de *Steiner*. Um grupo de instâncias é gerado a partir das instâncias da TSPLIB e de dois parâmetros, α e m, da seguinte forma. O total de clientes, ou seja, $|U|$, é dado por $\lfloor \alpha(n-1) \rfloor$, com $\alpha \in \{0.25, 0.5, 0.75, 1.0\}$. A

capacidade Q do anel-estrela é definida por $\left\lceil \frac{|U|}{0.9m} \right\rceil$ e o total de anéis-estrelas é fixado em $m \in \{3, 4, 5, 7, 10, 14\}$. Instâncias geradas com $Q = 1$ são desconsideradas.

Os custos de conexão w_{ij} e de roteamento c_{ij} de um par de vértices i e j são definidos com base na distância euclidiana das coordenadas dos pontos associados aos vértices, e_{ij}. Os custos são inteiros e são obtidos pelo arredondamento da distância euclidiana. Dois tipos de instâncias foram geradas para cada uma das classes, dependendo da função de distância usada no arredondamento. Os padrões `EUC_2D` e `CEIL_2D` da TSPLIB [45] foram utilizados como função de distância.

No padrão `EUC_2D`, a distância euclidiana é arredondada para o inteiro mais próximo, enquanto que na `CEIL_2D` é feito o arredondamento para cima. No caso das instâncias usando o padrão `CEIL_2D`, os custos são obtidos pelas fórmulas $c_{ij} = \beta * \lceil e_{ij} \rceil$ e $w_{ij} = (10 - \beta) * \lceil e_{ij} \rceil$. Este arredondamento garante que as desigualdades triangulares sejam satisfeitas pelos custos. Já no caso das instâncias com padrão `EUC_2D`, os custos são obtidos por $c_{ij} = \beta * \texttt{round}(e_{ij})$ e $w_{ij} = (10 - \beta) * \texttt{round}(e_{ij})$.

O padrão `EUC_2D` foi usado por Baldacci et al. nos experimentos reportados em [4] e um pós-processamento sobre as distâncias euclidianas e_{ij} é executado após o arredondamento, como descrito no Algoritmo 16, que nos foi repassado por comunicação pessoal. Mais adiante, veremos que contrariamente ao que foi dito, este procedimento não garante que as desigualdades triangulares sejam satisfeitas.

Algoritmo 16: Triangularização.

Input: custos c_{ij}, para toda aresta ij de um grafo $G = (V, E)$
Output: custos c_{ij} modificados tais que $c_{ij} \leq c_{iv} + c_{vj}, \forall i, j, v \in V$

```
1 mudou = true
2 while mudou do
3     mudou = false
4     for i, j, v ∈ V do
5         if c_ij > c_iv + c_vj then
6             c_ij = c_iv + c_vj
7             mudou=true
```

Em ambos os casos, o valor de β depende da classe de instância. Nas classes A e C, o valor de β é fixado em 5, ou seja, os custos de conexão e os custos de roteamento para um mesmo par de vértices são iguais. Na classe B, o custo de roteamento é mais caro que o custo de conexão. Neste caso, o valor de β foi fixado em 7, seguindo o mesmo padrão de [4].

Nem todo par de vértices define uma conexão. Uma conexão ij é definida na instância

apenas se o custo da conexão for menor ou igual a um limitante, definido por um percentual da média dos custos de conexão. Este limitante é dado por `fator` $*(\sum_{i \in U} \sum_{j \in V'} w_{ij})/(|U| * (|V| - 2))$. O valor de `fator` usado em [4] é 0.2 e foi utilizado para definir as conexões nas classes A e B. Para a classe C foi utilizado um `fator` de 0.5.

A Tabela 5.1 reúne as configurações de cada uma das instâncias geradas no padrão `CEIL_2D` que foram utilizadas nos experimentos. A coluna `inst` indica a instância da TSPLIB utilizada para gerar a instância teste. Para cada valor de n, α e m, as colunas $|U|$, Q mostram o total de clientes e a capacidade dos anéis-estrelas, conforme especificado anteriormente. A coluna $|A^1|$ representa o total de conexões válidas em cada instância da classe A e B, enquanto a coluna $|A^2|$ dá o total de conexões para instâncias da classe C. Note que a classe C reúne instâncias com um número alto de conexões, conforme desejado.

5.4 Resultados Computacionais

Para fins de comparação, implementamos um algoritmo *branch-and-cut*, aqui denotado por BC, baseado no algoritmo proposto por Baldacci, Del'Amico e Salazar [4]. O código BC é mais simples que o descrito em [4], não fazendo uso de heurística primal e nem de *strong branching*. Utilizamos as seguintes famílias de desigualdades válidas descritas em [4] para separar pontos fracionários: desigualdades de conectividade, anel multi-estrela e de capacidade (veja inequações (4.17, 4.18, 4.19, 4.20, 4.21) na Seção 4.5). Essas desigualdades foram separadas usando-se as mesmas rotinas de separação utilizadas pelo nosso algoritmo *branch-and-cut-and-price* e que estão descritas na Seção 5.2. O intuito com essa implementação foi permitir uma avaliação mais justa entre os algoritmos *branch-and-cut* e *branch-and-cut-and-price* para o CmRSP.

Os algoritmos *branch-and-cut* e *branch-and-price* foram implementados na linguagem C e compilados com gcc 4.1.2. O *branch-and-cut* utiliza a biblioteca do `Xpress-MP` versão 17.01.01 para implementar todo o suporte de *branch-and-cut* enquanto o algoritmo *branch-and-price* utiliza apenas o resolvedor de programação linear do `Xpress-MP` para obter as relaxações lineares.

Os experimentos foram realizados em um Pentium IV 3.4GHz com 4GB de RAM rodando Linux 2.6.24.

O algoritmo *branch-and-price* foi avaliado por três implementações diferentes, cada uma delas usando uma relaxação diferente para o subproblema de *pricing*. O primeiro código, chamado `BPr`, resolve o subproblema de *pricing* relaxado, conforme discutido na Seção 4.4.2. Os outros dois, chamados `BP3c` e `BP3s`, resolvem, respectivamente, o problema do anel-estrela relaxado Q-capacitado com proibição de k-ciclos e o problema do anel-estrela relaxado Q-capacitado com proibição de k-streams, ambos para $k = 3$.

inst	n	α	m	$\|U\|$	Q	$\|A^1\|$	$\|A^2\|$	Inst	n	α	m	$\|U\|$	Q	$\|A^1\|$	$\|A^2\|$
eil26	26	0.25	3	6	3	0	25	eil76	76	0.25	3	18	7	11	198
			4	6	2	0	25				4	18	5	11	198
			5	6	2	0	25				5	18	4	11	198
		0.5	3	12	5	0	48				7	18	3	11	198
			4	12	4	0	48				10	18	2	11	198
			5	12	3	0	48				14	18	2	11	198
			7	12	2	0	48			0.5	3	37	14	24	442
			10	12	2	0	48				4	37	11	24	442
		0.75	3	18	7	0	72				5	37	9	24	442
			4	18	5	0	72				7	37	6	24	442
			5	18	4	0	72				10	37	5	24	442
			7	18	3	0	72				14	37	3	24	442
			10	18	2	0	72			0.75	3	56	21	46	651
			14	18	2	0	72				4	56	16	46	651
		1	3	25	10	0	94				5	56	13	46	651
			4	25	7	0	94				7	56	9	46	651
			5	25	6	0	94				10	56	7	46	651
			7	25	4	0	94				14	56	5	46	651
			10	25	3	0	94			1	3	75	28	70	846
			14	25	2	0	94				4	75	21	70	846
											5	75	17	70	846
											7	75	12	70	846
											10	75	9	70	846
											14	75	6	70	846
eil51	51	0.25	3	12	5	2	96	eil101	101	0.25	3	25	10	22	415
			4	12	4	2	96				4	25	7	22	415
			5	12	3	2	96				5	25	6	22	415
			7	12	2	2	96				7	25	4	22	415
			10	12	2	2	96				10	25	3	22	415
		0.5	3	25	10	5	200				14	25	2	22	415
			4	25	7	5	200			0.5	3	50	19	77	754
			5	25	6	5	200				4	50	14	77	754
			7	25	4	5	200				5	50	12	77	754
			10	25	3	5	200				7	50	8	77	754
			14	25	2	5	200				10	50	6	77	754
		0.75	3	37	14	6	283				14	50	4	77	754
			4	37	11	6	283			0.75	3	75	28	134	1135
			5	37	9	6	283				4	75	21	134	1135
			7	37	6	6	283				5	75	17	134	1135
			10	37	5	6	283				7	75	12	134	1135
			14	37	3	6	283				10	75	9	134	1135
		1	3	50	19	12	366				14	75	6	134	1135
			4	50	14	12	366			1	3	100	38	194	1700
			5	50	12	12	366				4	100	28	194	1700
			7	50	8	12	366				5	100	23	194	1700
			10	50	6	12	366				7	100	16	194	1700
			14	50	4	12	366				10	100	12	194	1700
											14	100	8	194	1700

Tabela 5.1: Configuração das instâncias testes do padrão `CEIL_2D`.

Um autômato finito determinístico para reconhecer caminhos úteis foi incorporado na implementação `BP3s` no lugar do algoritmo de intersecção de set forms, conforme discutido na Seção 4.4, e é chamado aqui por `BP3sA`.

A Tabela 5.2 resume os resultados computacionais das quatro implementações do BP para resolver as instâncias da classe A usando-se a distância `CEIL_2D`. As instâncias estão descritas pelas colunas `inst`, `m`, `U` e `Q`, que correspondem ao nome da instância, o total de anéis, o número de clientes e a capacidade dos veículos, respectivamente. O valor da melhor solução inteira encontrada por algum dos algoritmos ou pela heurística inicial é mostrado na coluna `z*`. A coluna `raiz` representa o percentual de `z*` em relação ao limitante dual obtido na raiz da árvore de *branch-and-bound* por cada um dos algoritmos.

Pode-se notar pela colunas $\texttt{raiz}^1$, $\texttt{raiz}^2$ e $\texttt{raiz}^3$, que os limitantes duais obtidos com o modelo de set covering são bastante apertados, mesmo no modelo relaxado. A melhora no limitante ao usar as relaxações com proibição de k-ciclos e k-streams fica mais evidente nas instâncias `eil51.tsp` e `eil76.tsp`.

Na Tabela 5.3 é apresentado um comparativo entre pares de códigos BP. Para cada par de código são apresentadas três informações, que referem-se à média de desempenho em cada grupo de 9 instâncias com um mesmo número de vértices. As três colunas em cada cabeçalho `Código A` $\times$ `Código B` têm o seguinte significado. A coluna GAP representa a redução média do gap de dualidade obtida pelo `Código B` em relação ao `Código A`, tomando-se todas as instâncias não resolvidas na otimalidade por ambos os códigos. Para obter a redução média do gap, calcula-se a redução do gap de dualidade, dado pela diferença do gap de dualidade obtido pelo `Código A` e pelo `Código B`, em cada instância e computa-se a média delas. Desta vez, considerando-se apenas as instâncias em que ambos os códigos encontraram a solução ótima, a coluna TIME mostra a taxa de *speed-up* médio, que é definida pela média dos tempos de execução do `Código A` dividida pela média dos tempos de execução do `Código B`. O conteúdo da coluna OPT é da forma x/y, onde y é o número de instâncias resolvidas na otimalidade pelo `Código B` e $y - x$ é o total de instâncias resolvidas pelo `Código A`. Com estas definições, algumas entradas da tabela podem ser indefinidas e são indicadas pelo símbolo "*".

Nota-se que quanto melhor é a relaxação do subproblema de *pricing* maior é o número de instâncias resolvidas na otimalidade. O uso de um problema relaxado mais apertado também ajuda a reduzir o tempo total de processamento.

Com relação aos códigos BP, nota-se que o desempenho geral melhora à medida em que se utiliza uma relaxação mais apertada no subproblema de *pricing*. Inspecionando a coluna OPT nos dois primeiros grupos de colunas, nota-se que quanto mais complexas as estruturas proibidas na relaxação, maior é o número de instâncias resolvidas na otimalidade. O ganho é mais evidente quando se compara a coluna TIME. Relaxações mais apertadas levam a *speed-ups* mais altos de tempo de execução. Isto pode ser ex-

inst	m	U	Q	z*	raiz[1]	raiz[2]	raiz[3]
eil26.tsp	3	12	5	254	96.6%	99.4%	99.4%
eil26.tsp	4	12	4	271	99.4%	100.0%	100.0%
eil26.tsp	5	12	3	304	100.0%	100.0%	100.0%
eil26.tsp	3	18	7	312	96.2%	97.7%	97.7%
eil26.tsp	4	18	5	349	98.0%	99.9%	99.9%
eil26.tsp	5	18	4	387	99.2%	99.6%	99.6%
eil26.tsp	3	25	10	346	96.3%	98.0%	98.0%
eil26.tsp	4	25	7	379	97.1%	99.0%	99.0%
eil26.tsp	5	25	6	398	100.0%	100.0%	100.0%
eil51.tsp	3	25	10	343	92.2%	95.3%	98.0%
eil51.tsp	4	25	7	378	94.0%	97.1%	98.9%
eil51.tsp	5	25	6	395	98.4%	99.7%	100.0%
eil51.tsp	3	37	14	406	93.3%	95.2%	98.0%
eil51.tsp	4	37	11	437	93.5%	95.4%	98.0%
eil51.tsp	5	37	9	467	94.1%	96.5%	98.2%
eil51.tsp	3	50	19	496	95.7%	96.5%	96.7%
eil51.tsp	4	50	14	530	96.1%	97.3%	97.5%
eil51.tsp	5	50	12	560	95.8%	97.0%	97.3%
eil76.tsp	3	37	14	469	83.5%	84.8%	88.4%
eil76.tsp	4	37	11	490	89.4%	90.9%	93.6%
eil76.tsp	5	37	9	501	97.0%	99.3%	100.0%
eil76.tsp	3	56	21	575	81.3%	82.0%	85.4%
eil76.tsp	4	56	16	626	81.6%	82.6%	85.1%
eil76.tsp	5	56	13	584	95.2%	96.9%	98.8%
eil76.tsp	3	75	28	618	94.5%	95.2%	97.8%
eil76.tsp	4	75	21	758	81.6%	82.2%	84.0%
eil76.tsp	5	75	17	811	81.3%	82.0%	83.7%

Tabela 5.2: Limitantes duais na raiz obtidos pelos algoritmos (1) `BPr`, (2) `BP3c` e (3) `BP3s` nas instâncias com distância `CEIL_2D` da classe A

	BPr × BP3c			BP3c × BP3s			BP3s × BP3sA		
inst	GAP	TIME	OPT	GAP	TIME	OPT	GAP	TIME	OPT
eil26	*	3.4	0/9	*	0.9	0/9	*	3.1	0/9
eil51	0.9	8.1	0/2	0.5	18.8	3/5	0.3	2.6	1/6
eil76	0.9	*	1/1	3.1	45.0	0/1	0.3	2.7	0/1

Tabela 5.3: Resumo comparativo entre os códigos `BPr`, `BP3c`, `BP3s` e `BP3sA`.

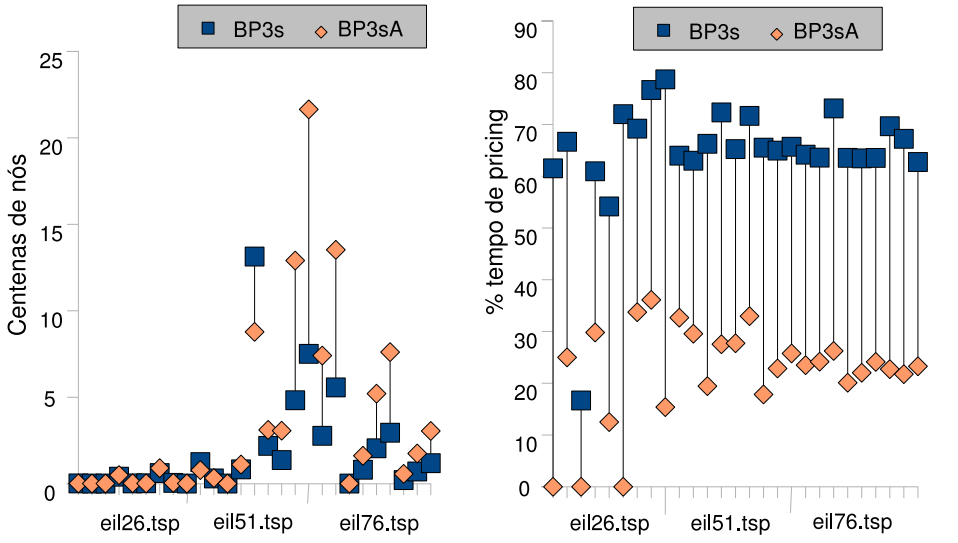
plicado pelo fato de que os limitantes duais das relaxações tornam-se mais apertados, o que permite podar mais cedo a árvore de enumeração. Isso pode ser confirmado pela redução do gap de dualidade, mostrado nas colunas GAP, que embora pequenas, sempre ocorrem. Destes resultados, pode-se concluir que `BP3s` domina os outros dois códigos de BP avaliados. Para ter uma implementação mais eficiente deste código, avaliamos o uso do AFD descrito na Seção 4.4.3. O ganho de desempenho com o uso do AFD pode ser notado na coluna `TIME`. O `BP3s` é aproximadamente 3 vezes mais lento que `BP3sA`. Essa diferença de desempenho fica mais evidente na Figura 5.2, onde comparamos o número de nós explorados na árvore de enumeração (em (a)) e o percentual do tempo total gasto com o subproblema de *pricing* (em (b)). Houve uma redução para aproximadamente 30% no percentual de tempo gasto com geração de colunas, ao usar o AFD. Essa redução de tempo permitiu ao `BP3sA` explorar uma porção maior do espaço de soluções e, assim, diminuir o gap de dualidade e resolver uma instância a mais na otimalidade.

Uma vez que temos identificado o `BP3sA` como a melhor implementação do nosso BP, comparamos este código com o código `BC`. Na Tabela 5.4 é apresentado um comparativo entre o código BP e BC. As colunas do cabeçalho desta tabela são como definidas na Tabela 5.3.

	BC × BP3sA		
inst	GAP	TIME	OPT
eil26	*	1.7	0/9
eil51	0.7	0.4	-1/6
eil76	0.0	*	1/1

Tabela 5.4: Resumo comparativo entre os códigos `BP3sA` e `BC`.

Os dados mostrados na Tabela 5.4 não são tão claros. Uma inspeção mais minuciosa revela que os dois códigos se comportam de forma muito diferente um do outro. Embora um mesmo número de instâncias tenham sido resolvidas na otimalidade pelos dois códigos, há variação nas instâncias em que a otimalidade foi comprovada por cada algoritmo e o tempo total gasto por cada um deles. Na Figura 5.3(a), considerando-se apenas as



(a) # nós processados por instância

(b) Tempo de *pricing* por instância

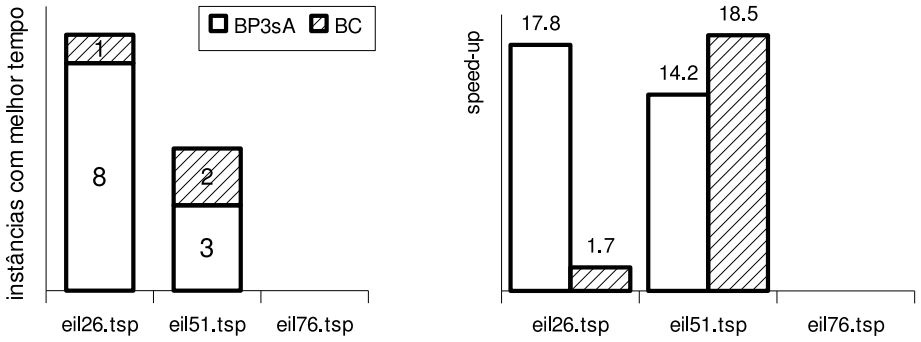
Figura 5.2: Comparação entre os códigos `BP3s` e `BP3sA`.

instâncias resolvidas à otimalidade por ambos os códigos, são ilustrados quantas vezes cada código foi mais rápido que o seu oponente. Embora um número maior de instâncias tenham sido resolvidas mais rápido pelo `BP3sA`, a taxa de *speed-up* não é expressiva em alguns casos, como pode ser notado na Figura 5.3(b). Nessa figura, mostramos a taxa de *speed-up* médio para cada código naquelas instâncias em que ele foi o mais rápido. Os resultados mostram que nenhum dos dois códigos domina o outro. Há instâncias que parecem ser mais adequadas ao uso do `BC` do que do `B3sA` e vice-versa.

Embora o desempenho médio de `BP3sA` não tenha sido melhor que `BC`, o limitante dual gerado pela sua relaxação linear na raiz da árvore de enumeração foi quase em sua totalidade (exceto por uma instância) mais apertado que o do `BC`. A Figura 5.4(a) mostra o percentual de melhoria no limitante dual e a Figura 5.4(b) o total de instâncias com limitante dual na raiz da árvore de enumeração mais apertado para cada código.

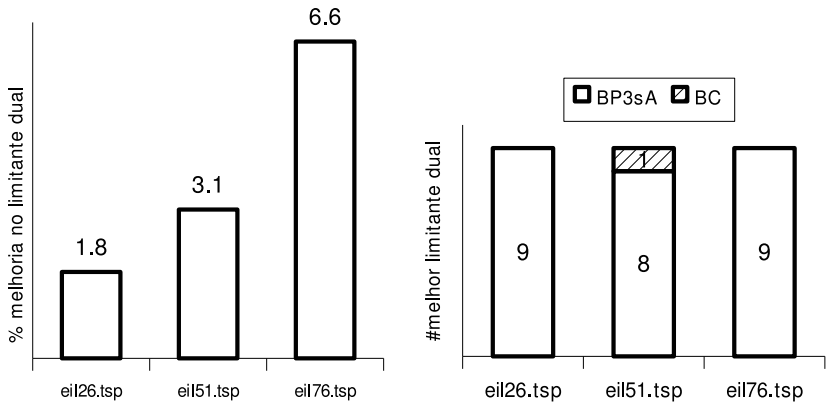
Com base nessas análises, passamos a realizar experimentos para comparar o desempenho de um algoritmo BCP, que tenta explorar a vantagem do limitante dual apertado dado pelo modelo de *set covering* do algoritmo BP e os cortes do algoritmo BC.

Inicialmente, comparamos o desempenho do BP e do BCP nas instâncias da classe A, cujos resultados estão sumarizados na Tabela 5.5. As colunas na Tabela 5.5 têm os mesmos significados que aquelas na Tabela 5.3. O algoritmo BCP apresentou um desempenho superior ao BP em todos os aspectos avaliados. Em especial, BCP resolveu 3 instâncias da Classe A na otimalidade a mais e foi em média 3 vezes mais rápido que



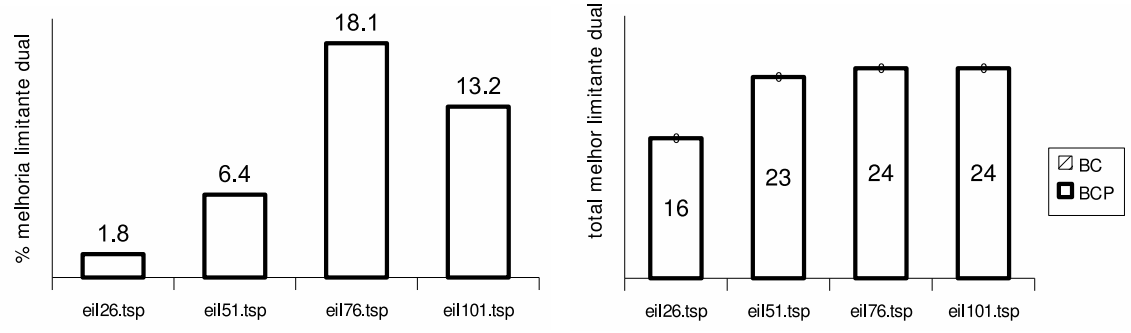
(a) Total de instâncias melhores (b) Taxa de speedup

Figura 5.3: Comparação entre `BP3sA` e `BC`.



(a) percentual de melhoria (b) total de instâncias melhores

Figura 5.4: Comparativo do limitante dual gerado por `BP3sA` e `BC`.



(a) percentual de melhoria (b) total de instâncias melhores

Figura 5.5: Comparativo do limitante dual gerado por BCP e BC na classe A.

BP.

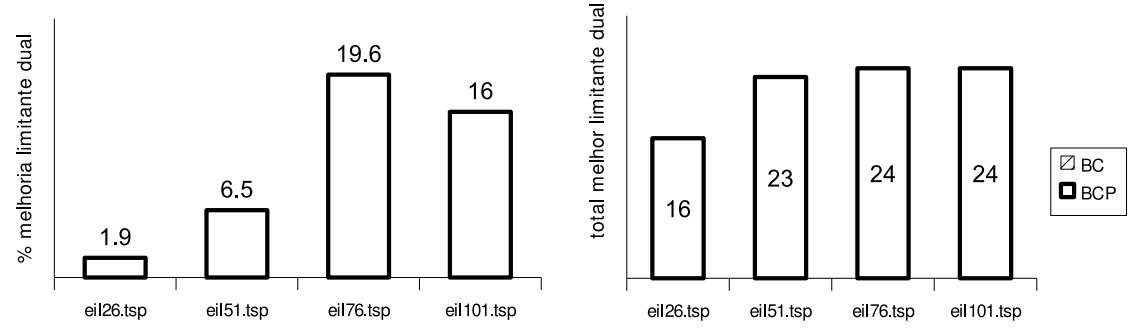
	BP × BCP (Classe A)		
inst	GAP	TIME	OPT
eil26.tsp	*	2.5	0/9
eil51.tsp	1.0	5.9	2/8
eil76.tsp	0.7	1.5	1/2
eil101.tsp	*	*	0/0

Tabela 5.5: Resumo comparativo entre os códigos `BCP` e `BP` em 36 instâncias da Classe A.

Os resultados evidenciaram a superioridade de desempenho de um algoritmo BCP em relação a um algoritmo BP. Em seguida, passamos a realizar testes para comparar o desempenho do BCP e de um BC.

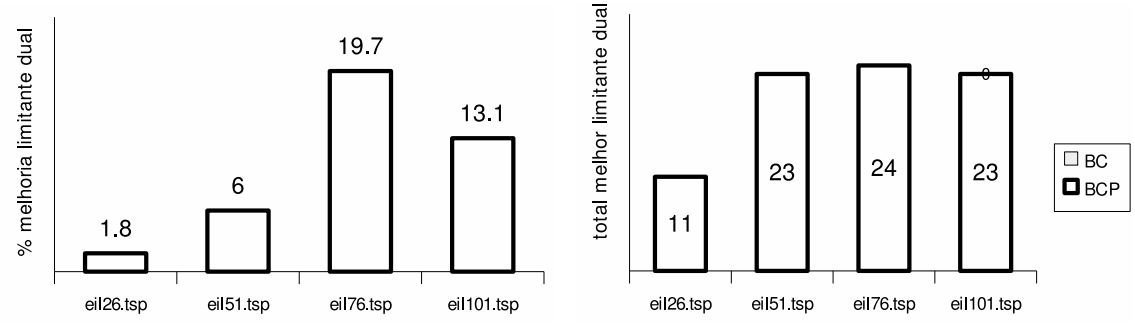
O algoritmo BCP apresentou um desempenho em geral melhor que o algoritmo BC. Conforme esperado, o limitante dual obtido na raiz da árvore de *branch-and-bound* foi sempre mais apertado nos experimentos que fizemos. A Figura 5.5(b) mostra o total de instâncias da classe A com melhor limitante dual gerado na raiz por cada algoritmo. Em alguns casos, este limitante foi apertado o suficiente para fechar o gap de otimalidade na raiz da árvore de *branch-and-bound*. O percentual de melhoria no valor desse limitante é mostrado na Figura 5.5(a).

As Tabelas 5.6, 5.7, 5.8, 5.9, 5.10 e 5.11 apresentam os desempenhos dos algoritmos BCP e BC para as instâncias das classes A, B e C, respectivamente. As colunas são como descritas na Tabela 5.2, exceto as colunas `gap`, `time` e `nodos`. Aqui, a coluna `gap` reporta o percentual do gap entre `z*` e o valor do melhor limitante dual obtido pelo algoritmo. O tempo total gasto em segundos é mostrado na coluna `tempo`. A coluna `nodos` representa



(a) percentual de melhoria (b) total de instâncias melhores

Figura 5.6: Comparativo do limitante dual gerado por BCP e BC na classe B.



(a) percentual de melhoria (b) total de instâncias melhores

Figura 5.7: Comparativo do limitante dual gerado por BCP e BC na classe C.

o total de nós explorados na árvore de *branch-and-bound*.

Sumarizando os resultados das Tabelas 5.6, 5.7, 5.8, 5.9, 5.10 e 5.11, temos que o ganho com limitantes duais mais apertados também é evidente quando se compara o gap final nas instâncias não resolvidas na otimalidade por ambos os algoritmos. Em apenas 5 das 26 instâncias em aberto, o BC teve um gap final menor. A média dos tempos gastos pelo BCP para resolver na otimalidade as instâncias da classe A com 51 vértices foi 3.6 vezes menor que a do BC e o total de instâncias resolvidas na otimalidade foi bem mais expressivo (26 instâncias a mais foram resolvidas na otimalidade pelo BCP).

As Figuras 5.6 e 5.7 apresentam os mesmos dados comparativos que as Figuras 5.5, mas para as classes B e C, respectivamente. Nota-se que instâncias da classe C são mais difíceis que instâncias da classe A e B, e, ainda assim, o BCP gerou limitantes duais na raiz mais apertados do que o BC.

inst	m	U	Q	z*	gap[1]	gap[2]	tempo[1]	tempo[2]	nodos[1]	nodos[2]	raiz[1]	raiz[2]
eil26.tsp	3	6	3	178	0.0	0.0	0.0	0.1	1	3	178.0	173.9
eil26.tsp	4	6	2	198	0.0	0.0	0.0	0.1	1	3	198.0	195.0
eil26.tsp	5	6	2	215	0.0	0.0	0.0	0.0	1	1	215.0	215.0
eil26.tsp	3	12	5	254	0.0	0.0	0.0	0.8	2	23	252.5	246.7
eil26.tsp	4	12	4	271	0.0	0.0	0.0	0.3	1	6	271.0	268.6
eil26.tsp	5	12	3	304	0.0	0.0	0.0	0.1	1	3	304.0	300.8
eil26.tsp	7	12	2	373	0.0	0.0	0.0	0.1	1	3	373.0	368.9
eil26.tsp	10	12	2	429	0.0	0.0	0.0	0.0	1	1	429.0	429.0
eil26.tsp	3	18	7	312	0.0	0.0	0.7	4.2	25	108	306.7	297.5
eil26.tsp	4	18	5	349	0.0	0.0	0.1	2.5	2	30	348.5	339.7
eil26.tsp	5	18	4	387	0.0	0.0	0.1	3.8	5	79	385.5	370.2
eil26.tsp	7	18	3	462	0.0	0.0	0.0	1.6	1	44	462.0	449.0
eil26.tsp	10	18	2	590	0.0	0.0	0.0	0.3	1	5	589.5	586.2
eil26.tsp	14	18	2	680	0.0	0.0	0.0	0.1	1	1	680.0	680.0
eil26.tsp	3	25	10	346	0.0	0.0	4.1	6.4	58	266	339.9	336.5
eil26.tsp	4	25	7	379	0.0	0.0	0.2	2.3	3	55	378.2	372.0
eil26.tsp	5	25	6	398	0.0	0.0	0.1	2.0	1	35	398.0	390.4
eil26.tsp	7	25	4	490	0.0	0.0	0.1	8.1	2	79	489.0	474.4
eil26.tsp	10	25	3	601	0.0	0.0	0.0	4.1	1	78	601.0	575.7
eil26.tsp	14	25	2	771	0.0	0.0	0.0	0.1	1	1	771.0	771.0
eil51.tsp	3	12	5	254	0.0	0.0	0.2	47.2	3	195	252.3	237.0
eil51.tsp	4	12	4	271	0.0	0.0	0.1	2.3	1	11	271.0	254.7
eil51.tsp	5	12	3	303	0.0	0.0	0.1	3.2	1	11	303.0	280.8
eil51.tsp	7	12	2	373	0.0	0.0	0.1	3.8	1	11	373.0	337.8
eil51.tsp	10	12	2	420	0.0	0.0	0.1	1.6	1	7	420.0	398.0
eil51.tsp	3	25	10	343	0.0	0.0	5.9	86.5	43	690	338.4	328.5
eil51.tsp	4	25	7	378	0.0	0.0	1.3	80.4	16	352	376.6	347.7
eil51.tsp	5	25	6	395	0.0	0.0	0.2	4.6	1	13	395.0	371.4
eil51.tsp	7	25	4	489	0.0	0.0	0.2	175.9	3	0	488.0	446.7
eil51.tsp	10	25	3	595	0.0	0.0	0.1	47.6	1	175	595.0	535.6
eil51.tsp	14	25	2	769	0.0	0.0	0.1	5.3	1	19	769.0	717.5
eil51.tsp	3	37	14	406	0.0	0.0	34.2	6.9	27	42	400.8	396.8
eil51.tsp	4	37	11	437	0.0	0.0	7.2	71.4	16	478	434.5	417.2
eil51.tsp	5	37	9	467	0.0	0.2	67.8	1801.1	377	11237	459.1	436.4
eil51.tsp	7	37	6	547	0.0	1.1	5.8	1801.1	91	8143	541.4	490.5
eil51.tsp	10	37	5	626	0.0	1.1	10.0	1801.1	199	7725	619.4	583.3
eil51.tsp	14	37	3	829	0.0	0.0	1.0	898.5	39	3150	824.4	768.0
eil51.tsp	3	50	19	496	0.6	0.0	1800.2	136.2	351	1878	487.1	485.1
eil51.tsp	4	50	14	530	0.0	0.0	347.1	909.7	281	7876	523.1	510.2
eil51.tsp	5	50	12	560	0.0	0.5	400.6	1801.1	575	9625	553.5	531.9
eil51.tsp	7	50	8	648	0.0	2.2	225.8	1801.1	797	5847	640.4	596.9
eil51.tsp	10	50	6	754	0.0	2.6	80.5	1801.1	671	6385	747.1	686.7
eil51.tsp	14	50	4	954	0.0	2.0	35.3	1801.1	600	4993	948.3	864.7

Tabela 5.6: Resultados obtidos pelos algoritmos (1) BCP e (2) BC nas instâncias métricas com 26 e 51 vértices da classe A

inst	m	U	Q	z*	gap[1]	gap[2]	tempo[1]	tempo[2]	nodos[1]	nodos[2]	raiz[1]	raiz[2]
eil76.tsp	3	18	7	338	0.0	3.0	201.9	1801.4	263	1311	325.9	262.5
eil76.tsp	4	18	5	399	0.0	4.5	16.7	1801.2	71	1201	392.7	296.7
eil76.tsp	5	18	4	460	0.0	3.1	12.6	1801.4	95	1149	454.1	322.9
eil76.tsp	7	18	3	558	0.0	3.7	0.6	1801.2	1	1317	558.0	390.5
eil76.tsp	10	18	2	746	0.0	1.4	0.4	1801.0	1	1373	746.0	545.6
eil76.tsp	14	18	2	814	0.0	1.2	0.4	1801.1	1	1459	814.0	635.5
eil76.tsp	3	37	14	469	10.1	10.9	1800.1	1801.7	315	1339	415.5	397.0
eil76.tsp	4	37	11	490	4.0	6.1	1800.0	1801.2	645	1885	460.2	419.7
eil76.tsp	5	37	9	501	0.0	0.4	2.8	1801.1	1	2171	501.0	433.8
eil76.tsp	7	37	6	641	0.0	4.7	1.0	1801.3	1	1795	641.0	489.9
eil76.tsp	10	37	5	748	0.0	5.6	3.4	1801.0	41	1697	746.0	584.7
eil76.tsp	14	37	3	1045	0.0	3.2	0.5	1801.1	1	1647	1044.0	825.7
eil76.tsp	3	56	21	575	15.0	14.1	1800.2	1801.2	107	3327	493.9	482.5
eil76.tsp	4	56	16	626	15.5	15.5	1800.2	1800.0	425	1343	536.3	510.7
eil76.tsp	5	56	13	584	0.0	1.6	1355.8	1801.3	439	2377	578.7	521.5
eil76.tsp	7	56	9	772	10.1	15.6	1800.4	1800.0	3323	951	693.7	597.9
eil76.tsp	10	56	7	824	0.0	5.4	436.5	1801.1	1450	1997	817.8	697.4
eil76.tsp	14	56	5	1030	0.0	4.6	124.0	1801.1	887	1941	1024.5	865.8
eil76.tsp	3	75	28	618	1.5	1.0	1800.2	1801.2	49	6733	607.0	599.7
eil76.tsp	4	75	21	758	17.9	17.7	1801.3	1801.1	147	4603	640.6	617.2
eil76.tsp	5	75	17	811	18.2	20.0	1800.8	1801.1	313	2953	682.8	634.6
eil76.tsp	7	75	12	882	11.5	16.5	1800.2	1801.1	1209	2113	785.4	698.7
eil76.tsp	10	75	9	1029	11.0	18.0	1800.3	1801.1	3225	1951	921.0	808.1
eil76.tsp	14	75	6	1180	0.0	6.6	769.9	1800.0	2385	1093	1173.8	971.9
eil101.tsp	3	25	10	381	0.0	0.3	48.4	1801.2	5	1025	377.1	341.2
eil101.tsp	4	25	7	433	0.0	3.3	134.6	1801.3	69	939	425.6	362.8
eil101.tsp	5	25	6	469	0.0	4.5	91.0	1801.3	145	911	461.5	384.8
eil101.tsp	7	25	4	576	0.0	4.7	40.5	1801.3	269	751	570.8	439.8
eil101.tsp	10	25	3	693	0.0	4.1	2.5	1801.8	4	769	691.3	541.9
eil101.tsp	14	25	2	905	0.0	2.0	1.0	1801.3	4	773	904.0	716.1
eil101.tsp	3	50	19	614	16.1	16.1	1801.0	1800.0	5	532	528.2	504.4
eil101.tsp	4	50	14	661	17.0	18.2	1800.2	1801.5	15	1157	563.5	519.5
eil101.tsp	5	50	12	706	18.9	21.3	1801.7	1801.2	19	1043	591.9	536.4
eil101.tsp	7	50	8	786	13.1	18.4	1800.1	1801.1	81	861	692.4	582.5
eil101.tsp	10	50	6	819	0.0	5.8	440.9	1801.6	107	857	811.9	664.4
eil101.tsp	14	50	4	1044	0.2	5.8	1800.1	1800.0	1415	354	1034.6	816.9
eil101.tsp	3	75	28	648	1.7	0.0	1801.3	1543.6	5	1365	633.2	624.6
eil101.tsp	4	75	21	801	19.6	19.7	1801.9	1800.0	13	848	669.5	641.6
eil101.tsp	5	75	17	855	21.4	22.8	1800.3	1801.3	35	1421	702.3	658.8
eil101.tsp	7	75	12	956	21.3	25.5	1800.6	1801.3	77	1051	784.7	700.3
eil101.tsp	10	75	9	1046	16.4	21.8	1800.3	1800.0	183	536	894.1	778.6
eil101.tsp	14	75	6	1293	15.2	21.9	1800.1	1801.2	1225	823	1117.1	923.6
eil101.tsp	3	100	38	838	17.4	16.9	1803.0	1801.3	1	2883	713.1	692.7
eil101.tsp	4	100	28	868	17.5	17.0	1815.2	1801.6	17	2697	738.2	716.6
eil101.tsp	5	100	23	909	17.9	18.1	1800.1	1801.2	47	1899	769.0	733.5
eil101.tsp	7	100	16	1018	19.3	22.1	1800.2	1801.2	195	1065	850.0	781.3
eil101.tsp	10	100	12	1119	17.1	20.7	1800.1	1801.3	595	1027	953.0	856.9
eil101.tsp	14	100	8	1350	15.0	20.6	1800.6	1801.2	1827	935	1169.0	1014.6

Tabela 5.7: Resultados obtidos pelos algoritmos (1) BCP e (2) BC nas instâncias métricas com 76 e 101 vértices da classe A

inst	m	U	Q	z*	gap[1]	gap[2]	tempo[1]	tempo[2]	nodos[1]	nodos[2]	raiz[1]	raiz[2]
eil26.tsp	3	6	3	1246	0.0	0.0	0.0	0.1	1	3	1246.0	1217.2
eil26.tsp	4	6	2	1386	0.0	0.0	0.0	0.1	1	3	1386.0	1365.0
eil26.tsp	5	6	2	1505	0.0	0.0	0.0	0.0	1	1	1505.0	1505.0
eil26.tsp	3	12	5	1778	0.0	0.0	0.0	0.6	4	13	1767.5	1727.0
eil26.tsp	4	12	4	1897	0.0	0.0	0.0	0.3	1	6	1897.0	1878.3
eil26.tsp	5	12	3	2128	0.0	0.0	0.0	0.1	1	3	2128.0	2105.7
eil26.tsp	7	12	2	2611	0.0	0.0	0.0	0.1	1	3	2611.0	2582.3
eil26.tsp	10	12	2	3003	0.0	0.0	0.0	0.0	1	1	3003.0	3003.0
eil26.tsp	3	18	7	2184	0.0	0.0	0.7	3.6	33	101	2146.7	2074.6
eil26.tsp	4	18	5	2443	0.0	0.0	0.1	2.8	2	37	2439.5	2378.1
eil26.tsp	5	18	4	2709	0.0	0.0	0.1	2.6	2	78	2698.5	2591.1
eil26.tsp	7	18	3	3234	0.0	0.0	0.0	2.1	1	60	3234.0	3143.2
eil26.tsp	10	18	2	4130	0.0	0.0	0.0	0.3	1	5	4126.5	4103.5
eil26.tsp	14	18	2	4760	0.0	0.0	0.0	0.1	1	1	4760.0	4760.0
eil26.tsp	3	25	10	2422	0.0	0.0	4.2	3.9	49	252	2379.0	2355.5
eil26.tsp	4	25	7	2653	0.0	0.0	0.2	2.0	2	62	2647.4	2597.0
eil26.tsp	5	25	6	2786	0.0	0.0	0.1	1.8	1	35	2786.0	2732.6
eil26.tsp	7	25	4	3430	0.0	0.0	0.1	13.1	3	252	3423.0	3311.5
eil26.tsp	10	25	3	4207	0.0	0.0	0.0	3.8	1	64	4207.0	4029.8
eil26.tsp	14	25	2	5397	0.0	0.0	0.0	0.1	1	1	5397.0	5397.0
eil51.tsp	3	12	5	1778	0.0	0.0	0.1	59.4	3	359	1759.8	1634.5
eil51.tsp	4	12	4	1897	0.0	0.0	0.1	5.7	1	23	1897.0	1783.3
eil51.tsp	5	12	3	2121	0.0	0.0	0.1	4.8	1	15	2121.0	1978.7
eil51.tsp	7	12	2	2611	0.0	0.0	0.1	2.3	1	7	2611.0	2367.5
eil51.tsp	10	12	2	2940	0.0	0.0	0.1	2.6	1	9	2940.0	2789.5
eil51.tsp	3	25	10	2354	0.0	0.0	7.8	35.3	35	181	2329.6	2256.0
eil51.tsp	4	25	7	2606	0.0	0.0	0.8	47.6	9	110	2594.1	2448.7
eil51.tsp	5	25	6	2718	0.0	0.0	0.3	3.2	1	15	2718.0	2560.6
eil51.tsp	7	25	4	3400	0.0	0.0	0.3	380.7	6	1366	3390.3	3061.9
eil51.tsp	10	25	3	4111	0.0	0.0	0.1	31.3	1	155	4111.0	3786.2
eil51.tsp	14	25	2	5353	0.0	0.0	0.1	6.4	1	27	5353.0	5103.4
eil51.tsp	3	37	14	2795	0.0	0.0	157.9	15.8	77	62	2757.8	2706.4
eil51.tsp	4	37	11	3022	0.0	0.0	53.2	222.3	89	1440	2958.2	2836.7
eil51.tsp	5	37	9	3236	0.0	0.3	296.7	1801.2	997	7345	3162.6	2980.7
eil51.tsp	7	37	6	3775	0.0	1.3	3.2	1801.1	41	7237	3751.5	3428.4
eil51.tsp	10	37	5	4335	0.0	0.9	8.1	1801.1	157	7979	4295.0	4054.4
eil51.tsp	14	37	3	5759	0.0	0.0	1.6	990.9	71	3930	5726.6	5386.7
eil51.tsp	3	50	19	3393	0.5	0.0	1801.6	138.3	471	1230	3326.0	3286.0
eil51.tsp	4	50	14	3624	0.0	0.0	529.5	1729.1	273	10422	3583.5	3442.0
eil51.tsp	5	50	12	3853	0.0	1.4	365.0	1801.1	479	7465	3801.6	3567.1
eil51.tsp	7	50	8	4474	0.0	2.8	279.7	1801.0	977	5417	4431.8	4069.3
eil51.tsp	10	50	6	5216	0.0	3.0	84.3	1801.1	760	5979	5166.2	4707.1
eil51.tsp	14	50	4	6612	0.0	1.6	45.7	1800.0	922	2343	6574.7	6049.1

Tabela 5.8: Resultados obtidos pelos algoritmos (1) BCP e (2) BC nas instâncias métricas com 26 e 51 vértices da classe B

inst	m	U	Q	z*	gap[1]	gap[2]	tempo[1]	tempo[2]	nodos[1]	nodos[2]	raiz[1]	raiz[2]
eil76.tsp	3	18	7	2319	0.0	3.2	357.5	1800.0	909	661	2242.3	1773.0
eil76.tsp	4	18	5	2729	0.0	3.1	22.6	1800.0	215	555	2687.7	2002.2
eil76.tsp	5	18	4	3172	0.0	4.5	20.5	1801.1	321	1225	3115.0	2209.8
eil76.tsp	7	18	3	3824	0.0	3.5	0.4	1801.1	3	1333	3822.2	2732.1
eil76.tsp	10	18	2	5137	0.0	2.1	0.3	1801.1	1	1503	5137.0	3806.5
eil76.tsp	14	18	2	5613	0.0	1.1	0.2	1801.3	1	1285	5613.0	4467.9
eil76.tsp	3	37	14	3242	10.3	11.8	1800.2	1801.2	549	2341	2893.8	2705.5
eil76.tsp	4	37	11	3807	16.6	21.3	1800.2	1801.2	1161	2133	3226.9	2850.0
eil76.tsp	5	37	9	3495	0.0	2.5	4.9	1801.2	5	2075	3494.9	2976.7
eil76.tsp	7	37	6	4451	0.0	6.0	1.0	1800.0	1	1691	4451.0	3440.9
eil76.tsp	10	37	5	5177	0.0	6.4	16.6	1801.2	147	1717	5157.0	3945.6
eil76.tsp	14	37	3	7235	0.0	4.5	1.9	1801.0	32	1661	7198.0	5520.1
eil76.tsp	3	56	21	3698	12.2	11.1	1800.5	1801.1	91	4733	3258.7	3182.2
eil76.tsp	4	56	16	4310	19.1	20.1	1800.0	1801.2	205	2471	3579.4	3328.8
eil76.tsp	5	56	13	4848	24.6	29.0	1801.0	1801.1	561	2187	3847.4	3439.1
eil76.tsp	7	56	9	5159	9.5	16.6	1800.7	1801.2	2369	2023	4665.5	3937.8
eil76.tsp	10	56	7	5541	0.0	5.7	183.1	1801.0	545	1973	5512.4	4464.7
eil76.tsp	14	56	5	7032	0.5	5.4	1800.2	1800.0	12030	908	6942.5	5769.6
eil76.tsp	3	75	28	4010	1.5	0.6	1800.9	1801.2	13	4831	3947.5	3849.7
eil76.tsp	4	75	21	4921	16.2	17.3	1800.9	1801.1	77	2513	4213.6	3992.4
eil76.tsp	5	75	17	5423	19.3	22.9	1800.3	1801.1	261	2107	4528.8	4137.0
eil76.tsp	7	75	12	5953	12.9	19.7	1800.2	1800.0	1071	2722	5241.1	4530.8
eil76.tsp	10	75	9	6930	11.8	18.5	1800.4	1801.2	2629	1773	6158.0	5176.7
eil76.tsp	14	75	6	9100	14.3	20.6	1800.3	1800.0	7189	1649	7904.9	6803.2
eil101.tsp	3	25	10	2629	0.0	3.2	71.6	1802.4	12	723	2608.4	2274.5
eil101.tsp	4	25	7	2972	0.0	5.1	72.4	1801.7	29	631	2951.4	2424.1
eil101.tsp	5	25	6	3237	0.0	5.8	67.7	1801.4	178	637	3184.2	2576.6
eil101.tsp	7	25	4	3986	0.0	6.3	416.8	1801.4	1578	509	3930.1	2954.2
eil101.tsp	10	25	3	4803	0.0	5.5	27.0	1801.5	193	513	4764.6	3630.9
eil101.tsp	14	25	2	6244	0.0	3.3	1.5	1801.4	4	555	6240.0	4886.1
eil101.tsp	3	50	19	4136	21.4	21.1	1800.1	1800.0	3	322	3405.7	3193.3
eil101.tsp	4	50	14	4432	18.4	22.5	1803.4	1801.3	9	667	3692.9	3290.0
eil101.tsp	5	50	12	4613	16.2	21.7	1800.5	1801.1	9	639	3968.9	3424.7
eil101.tsp	7	50	8	5351	13.6	21.4	1813.7	1801.8	123	523	4683.9	3815.3
eil101.tsp	10	50	6	6283	13.2	21.6	1800.3	1801.4	351	477	5521.1	4387.6
eil101.tsp	14	50	4	8180	15.3	24.1	1800.2	1802.6	1847	493	7056.2	5461.5
eil101.tsp	3	75	28	4707	21.0	16.0	1802.5	1801.2	7	957	3865.9	3843.9
eil101.tsp	4	75	21	5185	23.6	22.9	1803.2	1801.3	5	765	4156.7	3963.8
eil101.tsp	5	75	17	5424	20.5	23.6	1800.0	1801.2	7	701	4440.4	4096.9
eil101.tsp	7			6211	20.4	27.1	1800.5	1801.2	9	583	5072.1	4437.0
eil101.tsp	10	75	9	6926	16.5	22.9	1803.7	1801.2	231	521	5922.0	5011.7
eil101.tsp	14	75	6	8623	15.9	23.5	1800.6	1801.5	1403	477	7407.6	6018.1
eil101.tsp	3	100	38	5130	15.8	11.9	1802.2	1801.2	5	1159	4423.5	4375.5
eil101.tsp	4	100	28	5523	17.6	16.1	1803.4	1801.2	9	1057	4660.8	4487.0
eil101.tsp	5	100	23	5931	19.1	19.9	1800.6	1801.1	5	925	4922.1	4618.1
eil101.tsp	7	100	16	6677	18.9	23.3	1802.4	1801.5	71	689	5605.3	4973.9
eil101.tsp	10	100	12	7456	17.0	22.6	1800.4	1801.5	241	657	6357.9	5521.5
eil101.tsp	14	100	8	8871	13.0	19.2	1800.1	1801.9	565	519	7836.2	6443.4

Tabela 5.9: Resultados obtidos pelos algoritmos (1) BCP e (2) BC nas instâncias métricas com 76 e 101 vértices da classe B

inst	m	U	Q	z*	gap[1]	gap[2]	tempo[1]	tempo[2]	nodos[1]	nodos[2]	raiz[1]	raiz[2]
eil26.tsp	3	6	3	159	0.0	0.0	0.0	0.0	1	1	159.0	159.0
eil26.tsp	4	6	2	177	0.0	0.0	0.0	0.0	1	1	177.0	177.0
eil26.tsp	5	6	2	193	0.0	0.0	0.0	0.0	1	1	193.0	193.0
eil26.tsp	3	12	5	226	0.0	0.0	0.1	0.6	1	9	226.0	223.2
eil26.tsp	4	12	4	243	0.0	0.0	0.1	0.5	1	11	243.0	239.7
eil26.tsp	5	12	3	270	0.0	0.0	0.0	0.1	1	3	269.3	269.3
eil26.tsp	7	12	2	324	0.0	0.0	0.0	0.0	1	1	324.0	324.0
eil26.tsp	10	12	2	406	0.0	0.0	0.0	0.0	1	1	406.0	406.0
eil26.tsp	3	18	7	289	0.0	0.0	3.4	14.8	19	300	282.3	269.6
eil26.tsp	4	18	5	324	0.0	0.0	1.1	6.3	38	106	320.8	308.5
eil26.tsp	5	18	4	353	0.0	0.0	0.1	1.7	1	31	352.8	340.1
eil26.tsp	7	18	3	414	0.0	0.0	0.1	1.8	12	45	412.0	400.9
eil26.tsp	10	18	2	500	0.0	0.0	0.0	0.0	1	1	500.0	500.0
eil26.tsp	14	18	2	625	0.0	0.0	0.0	0.0	1	1	625.0	625.0
eil26.tsp	3	25	10	327	0.0	0.0	3.1	3.8	9	75	323.9	312.5
eil26.tsp	4	25	7	362	0.0	0.0	5.4	75.7	49	1070	358.6	341.4
eil26.tsp	5	25	6	385	0.0	0.0	0.3	26.5	5	416	383.7	370.2
eil26.tsp	7	25	4	460	0.0	0.0	0.1	13.6	1	232	460.0	445.6
eil26.tsp	10	25	3	547	0.0	0.0	0.0	0.9	1	13	546.5	540.1
eil26.tsp	14	25	2	683	0.0	0.0	0.0	0.0	1	1	683.0	683.0
eil51.tsp	3	12	5	226	0.0	0.0	2.9	6.6	3	17	225.6	216.8
eil51.tsp	4	12	4	241	0.0	0.0	0.5	3.8	1	11	241.0	232.7
eil51.tsp	5	12	3	270	0.0	0.0	0.2	4.2	2	15	269.0	256.7
eil51.tsp	7	12	2	319	0.0	0.0	0.0	3.0	1	29	319.0	307.0
eil51.tsp	10	12	2	373	0.0	0.0	0.1	0.9	1	6	373.0	372.1
eil51.tsp	3	25	10	325	0.0	0.0	372.5	45.8	53	138	322.1	304.0
eil51.tsp	4	25	7	359	0.0	0.0	34.9	220.3	11	593	355.3	329.5
eil51.tsp	5	25	6	383	0.0	0.0	3.3	343.8	5	1316	381.2	354.8
eil51.tsp	7	25	4	457	0.0	0.0	0.4	161.5	1	332	457.0	422.1
eil51.tsp	10	25	3	539	0.0	0.0	0.3	41.8	2	231	538.5	513.3
eil51.tsp	14	25	2	669	0.0	0.0	0.2	2.6	1	7	669.0	655.0
eil51.tsp	3	37	14	397	0.0	0.0	1083.0	275.1	93	1148	392.8	379.0
eil51.tsp	4	37	11	427	0.5	0.0	1800.1	1799.0	295	6053	421.1	400.9
eil51.tsp	5	37	9	446	0.0	0.0	10.6	572.6	7	1702	444.0	425.8
eil51.tsp	7	37	6	530	0.0	1.1	39.3	1801.1	172	5549	525.1	481.3
eil51.tsp	10	37	5	598	0.0	0.2	8.6	1801.1	61	8023	594.5	551.8
eil51.tsp	14	37	3	765	0.0	0.0	1.1	137.6	28	446	762.7	727.1
eil51.tsp	3	50	19	648	37.6	36.7	1800.1	1801.1	157	13359	466.0	455.0
eil51.tsp	4	50	14	505	0.0	0.6	751.5	1801.1	239	6275	501.0	470.0
eil51.tsp	5	50	12	688	29.6	31.3	1800.7	1801.1	1017	5321	525.7	489.1
eil51.tsp	7	50	8	623	0.6	3.3	1800.0	1801.1	3725	5167	611.5	561.3
eil51.tsp	10	50	6	721	0.0	2.1	34.1	1801.1	199	5079	716.5	635.9
eil51.tsp	14	50	4	905	0.0	1.7	3.5	1801.1	75	4601	903.3	821.9

Tabela 5.10: Resultados obtidos pelos algoritmos (1) BCP e (2) BC nas instâncias métricas com 26 e 51 vértices da classe C

inst	m	U	Q	z*	gap[1]	gap[2]	tempo[1]	tempo[2]	nodos[1]	nodos[2]	raiz[1]	raiz[2]
eil76.tsp	3	18	7	404	25.1	26.3	1801.6	1801.2	29	1533	319.1	255.9
eil76.tsp	4	18	5	385	0.0	2.9	429.9	1801.9	45	1211	380.8	280.1
eil76.tsp	5	18	4	439	0.0	2.1	258.8	1802.0	83	1069	432.5	320.5
eil76.tsp	7	18	3	527	0.0	1.3	4.2	1801.1	27	1447	523.3	371.9
eil76.tsp	10	18	2	676	0.0	0.0	0.6	1113.6	4	1176	675.5	499.0
eil76.tsp	14	18	2	745	0.0	0.0	0.7	687.8	14	1123	743.5	610.8
eil76.tsp	3	37	14	623	51.6	50.8	1801.3	1801.1	7	2157	404.1	380.0
eil76.tsp	4	37	11	676	48.9	51.2	1801.3	1801.2	25	1539	451.8	396.9
eil76.tsp	5	37		491	0.0	1.7	25.2	1800.0	3	1213	490.9	418.2
eil76.tsp	7	37	6	626	0.0	4.9	586.8	1801.0	419	1521	620.0	468.4
eil76.tsp	10	37	5	723	0.0	3.7	824.6	1801.0	1059	1553	716.0	565.6
eil76.tsp	14	37	3	969	0.0	0.3	1.5	1801.2	3	1375	968.0	771.7
eil76.tsp	3	56	21	488	0.8	0.0	1800.6	650.9	35	1078	482.0	459.0
eil76.tsp	4	56	16	775	47.3	47.3	1801.0	1801.4	69	2015	522.1	474.3
eil76.tsp	5	56		566	0.0	1.6	599.1	1800.0	100	1235	562.3	498.4
eil76.tsp	7	56	9	926	35.6	42.2	1800.2	1800.0	925	849	676.5	553.2
eil76.tsp	10	56	7	1058	32.3	38.3	1800.4	1801.3	2213	1935	794.3	628.6
eil76.tsp	14	56	5	1273	27.7	32.9	1800.4	1801.1	4805	1767	988.8	845.1
eil76.tsp	3	75	28	878	51.4	51.1	1800.9	1801.1	33	4915	578.0	564.2
eil76.tsp	4	75	21	922	49.2	50.9	1800.1	1801.1	99	2221	614.4	579.5
eil76.tsp	5	75	17	966	45.9	49.3	1801.1	1800.0	201	1311	657.3	594.0
eil76.tsp	7	75	12	1001	31.2	37.7	1800.1	1801.2	497	1849	756.9	657.3
eil76.tsp	10	75	9	1114	23.6	31.5	1800.5	1801.2	2007	1827	893.4	770.8
eil76.tsp	14	75	6	1380	20.7	26.8	1800.2	1801.1	2281	1649	1139.5	947.9
eil101.tsp	3	25	10	504	41.6	38.1	1802.0	1801.2	3	673	355.2	316.9
eil101.tsp	4	25	7	485	20.9	21.3	1801.0	1801.4	13	613	398.4	338.6
eil101.tsp	5	25	6	554	28.5	29.4	1800.5	1801.2	11	571	429.0	361.9
eil101.tsp	7	25	4	654	21.1	26.7	1842.3	1801.4	41	553	538.1	423.4
eil101.tsp	10	25	3	631	0.0	1.9	9.2	1801.7	1	601	630.8	510.8
eil101.tsp	14	25	2	789	0.0	0.0	1.7	1144.2	1	426	789.0	669.5
eil101.tsp	3	50	19	783	57.5	53.2	1802.0	1801.5	4	891	490.8	483.4
eil101.tsp	4	50	14	830	54.6	53.7	1801.1	1801.1	6	865	534.9	497.3
eil101.tsp	5	50	12	803	41.1	42.9	1801.1	1801.5	10	795	563.9	515.8
eil101.tsp	7	50	8	944	41.5	46.8	1802.1	1801.2	9	691	665.7	563.8
eil101.tsp	10	50	6	1054	33.4	40.9	1835.4	1801.3	123	739	786.9	644.6
eil101.tsp	14	50	4	993	0.0	4.4	17.3	1801.2	1	659	993.0	783.5
eil101.tsp	3	75	28	995	66.4	60.0	1801.9	1802.5	3	1127	596.3	602.4
eil101.tsp	4	75	21	1047	66.7	62.3	1800.1	1801.5	3	1185	627.6	616.2
eil101.tsp	5	75	17	1082	61.0	61.3	1801.8	1801.1	3	993	671.3	634.3
eil101.tsp	7	75	12	1157	54.1	57.4	1803.8	1801.6	5	883	750.8	678.2
eil101.tsp	10	75	9	1243	42.5	48.7	1800.6	1801.3	107	889	870.0	747.1
eil101.tsp	14	75	6	1468	34.7	42.5	1803.7	1801.5	369	763	1086.1	883.5
eil101.tsp	3	100	38	1094	60.6	58.8	1804.4	1801.2	2	1271	680.9	667.7
eil101.tsp	4	100	28	1140	59.9	59.7	1800.9	1801.2	3	1871	712.5	685.8
eil101.tsp	5	100	23	1154	55.3	55.9	1826.9	1801.2	11	1143	742.1	702.2
eil101.tsp	7	100	16	1236	50.4	53.0	1800.8	1801.6	93	991	820.2	742.7
eil101.tsp	10	100	12	1330	42.7	47.5	1800.0	1801.4	351	981	928.8	818.6
eil101.tsp	14	100	8	1527	33.8	40.2	1800.0	1801.5	1173	791	1138.3	945.9

Tabela 5.11: Resultados obtidos pelos algoritmos (1) BCP e (2) BC nas instâncias métricas com 76 e 101 vértices da classe C

Um comparativo para cada classe de instâncias, agrupando os resultados para instâncias com mesmo número de vértices, é sumarizado na Tabela 5.12. As colunas GAP, TIME e OPT são como definidos para a Tabela 5.3.

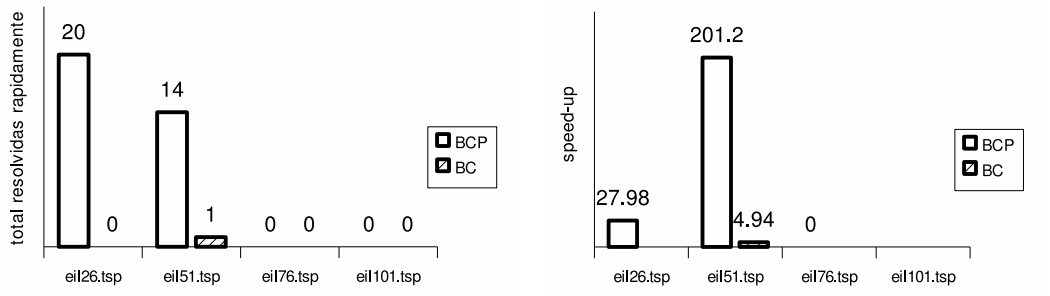
	BC × BCP (Classe A)			BC × BCP (Classe B)			BC × BCP (Classe C)		
inst	GAP	TIME	OPT	GAP	TIME	OPT	GAP	TIME	OPT
eil26.tsp	*	6.8	0/20	*	6.6	0/20	*	10.5	0/20
eil51.tsp	*	5.9	6/22	*	4.7	6/22	1.2	1.2	4/19
eil76.tsp	2.0	*	14/14	3.5	*	11/11	3.5	1418.5	7/10
eil101.tsp	2.7	*	6/7	3.6	*	6/6	1.5	669.1	2/3

Tabela 5.12: Resumo comparativo entre os códigos BCP e BC nas diferentes classes.

O total de instâncias resolvidas na otimalidade reportado na Tabela 5.12, reforça a impressão de que as instâncias da classe B e C são mais difíceis que aquelas da classe A. Houve uma redução no número de instâncias resolvidas na otimalidade e observa-se que o algoritmo BC apresentou um desempenho um pouco melhor nas instâncias da classe C. Entretanto, pelas colunas TIME e OPT, nota-se que o BCP teve um desempenho geral bem mais expressivo que o BC nas instâncias resolvidas na otimalidade. Nas instâncias não resolvidas na otimalidade por ambos os algoritmos, a redução do gap de dualidade com o código BCP em relação ao BC não foi tão expressiva, mas sempre houveram ganhos, como pode ser notado pela coluna GAP.

As diferenças de desempenho nas classes A, B e C são mostradas nas Figuras 5.8, 5.9 e 5.10. No lado esquerdo dessas figuras é apresentado o total de instâncias resolvidas mais rapidamente por cada algoritmo. No lado direito, a média de *speed-up*, definido pela razão entre o tempo gasto pelo código mais lento e o código mais rápido, é apresentada em cada caso. Além de resolver mais instâncias na otimalidade (como mostrado na Tabela 5.12) do que o algoritmo BC, pode ser observado pelos gráficos destas figuras, que o BCP resolveu as mesmas instâncias que BC usando muito menos tempo de processamento.

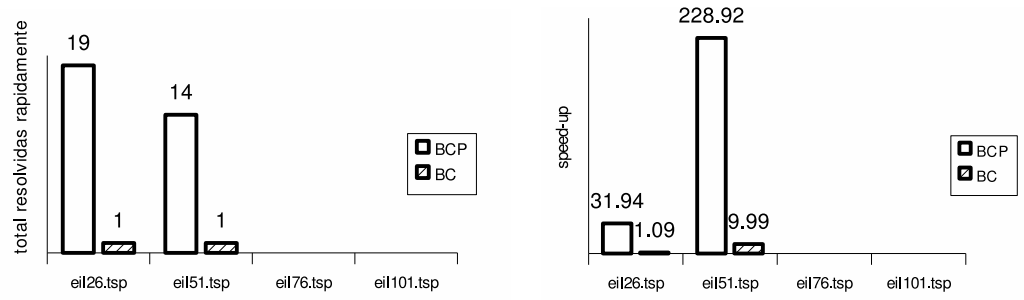
Observando a Tabela 5.12, pode-se notar que a dificuldade do problema está relacionada com o tamanho da instância, uma vez que o total de instâncias resolvidas na otimalidade diminui à medida que o número de vértices do grafo aumenta. Com relação à dificuldade do problema, também notamos que os algoritmos apresentam desempenhos diferentes conforme aumenta-se o número de anéis. Note que, pela construção das instâncias descritas na Seção 5.3, o aumento no número de anéis implica na redução da capacidade de cada anel. Essa diferença de comportamento dos algoritmos BCP e BC fica mais evidente quando analisamos os resultados da Tabela 5.13. Nela é apresentado o percentual de instâncias resolvidas na otimalidade em cada grupo de instâncias com o mesmo número de anéis. Nota-se um nítido aumento da diferença no número de instâncias resolvidas por



(a) Instâncias resolvidas mais rapidamente

(b) Taxa de *speed-up*

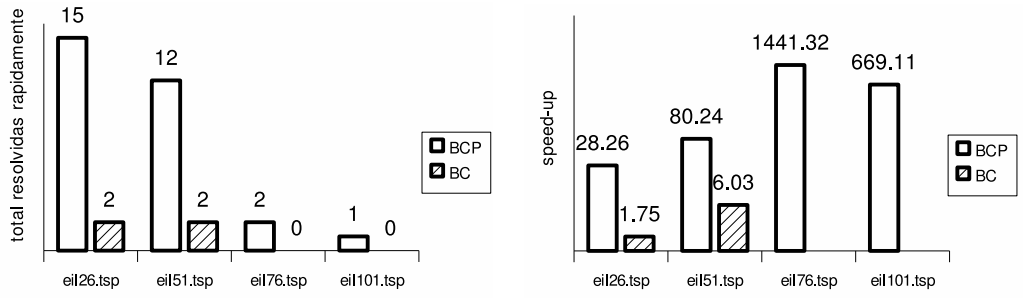
Figura 5.8: Comparação entre BCP e BC na classe A.



(a) Instâncias resolvidas mais rapidamente

(b) Taxa de *speed-up*

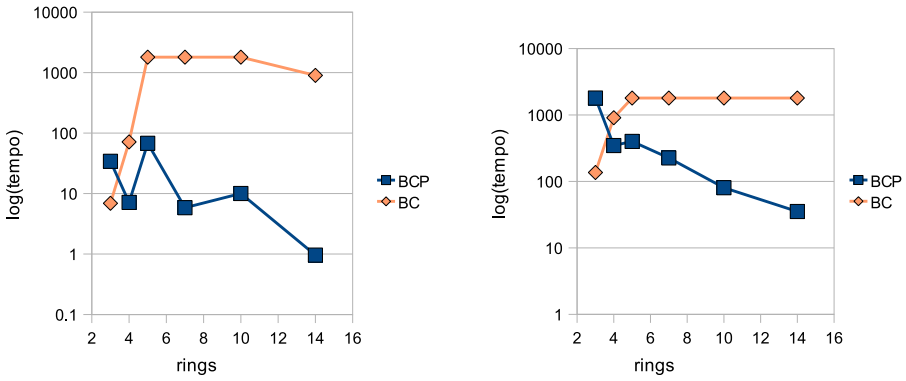
Figura 5.9: Comparação entre BCP e BC na classe B.



(a) Instâncias resolvidas mais rapidamente

(b) Taxa de speedup

Figura 5.10: Comparação entre BCP e BC na classe C.



(a) instância eil51.tsp $\alpha = 0.75$ (b) instância eil51.tsp $\alpha = 1.0$

Figura 5.11: Dificuldade em relação ao aumento no número de anéis.

ambos os códigos à medida que aumenta-se o número de anéis.

código	$m = 3$	$m = 4$	$m = 5$	$m = 7$	$m = 10$	$m = 14$
BCP	56%	63%	69%	67%	73%	62%
BC	50%	50%	38%	33%	33%	31%

Tabela 5.13: Total de instâncias resolvidas na otimalidade.

Essa diferença fica mais evidente na Figura 5.11, que mostra o comportamento dos algoritmos BCP e BC quando aumenta-se o número de anéis (m) em uma instância da classe A. Observando as curvas nos gráficos desta figura, nota-se que o desempenho do BCP melhora sensivelmente com o aumento do número de anéis, ao contrário que o BC.

Todos os testes apresentados até este ponto referem-se às instâncias com distância `CEIL_2D`. Alguns dos testes foram repetidos, desta vez nas instâncias com distâncias `EUC_2D`, para permitir a comparação do desempenho do BCP com os resultados reportados em [4]. No entanto, vale lembrar que trata-se de uma comparação que exige muitos cuidados, uma vez que os testes não apenas foram executados em máquinas distintas, como também usando-se linguagens de programação, compiladores, resolvedores de programação linear e sistemas operacionais diferentes. Além disso, a implementação em [4] fez uso de *strong branching* e heurísticas primais, os quais não estão implementados no BCP. Mas, por questão de completude, os testes comparativos foram realizados assim mesmo. Para ajustar a diferença entre as freqüências de relógio dos processadores das máquinas em que os experimentos foram executados, multiplicamos os tempos reportados em [4] por 0.65.

Note que o pós-processamento descrito no Algoritmo 16 não garante que as desigual-

dades triangulares sejam satisfeitas, uma vez que a violação da desigualdade triangular na linha (5) é verificada apenas para uma das três arestas que formam um triângulo (ciclo de comprimento três). No entanto, tanto o BCP quanto o *branch-and-cut* de Baldacci et. al. exigem que os anéis-estrelas sejam canônicos. Por outro lado, o BC não impõe essa restrição, o que o tornou incomparável com os demais códigos para instâncias com distâncias `EUC_2D`. Portanto, os resultados que seguem comparam o BCP com o *branch-and-cut* de Baldacci et. al.

Na Tabela 5.14 é apresentado o desempenho global dos dois códigos. A coluna FAST dá o total de instâncias em que cada código executou mais rapidamente e a coluna SPEED-UP a média dos *speed-up* nestas situações. O total de instâncias resolvidas na otimalidade é dada na coluna OPT. O número de vezes em que cada código obteve melhor limitante dual e o *gap* final são dados nas colunas LB e GAP, respectivamente.

	Classe A					Classe B				
código	FAST	SPEED-UP	OPT	LB	GAP	FAST	SPEED-UP	OPT	LB	GAP
BC	10	16.8	35	25	0.5	8	20.4	28	28	1.3
BCP	18	16.0	28	20	0.7	16	43.6	29	17	2.4

Tabela 5.14: Resumo comparativo de [4] e BCP.

O BCP resolveu mais instâncias em menos tempo que BC e obteve melhor *speed-up*. Entretanto, o total de instâncias resolvidas na otimalidade foi maior para BC do que BCP. Os resultados reportados em [4] mostram a importância do *strong branching* e da heurística primal para o desempenho do *branch-and-cut*. Isso sugere que adicionar essas duas características ao código BCP deve contribuir para melhorar o seu desempenho ainda mais.

Como observamos anteriormente, a comparação entre os resultados do BCP e aqueles reportados em [4] é delicada e pode incorrer a análises equivocadas, por conta das diferenças no ambiente computacional, bibliotecas, resolvedores e estilo de programação. Por outro lado, a comparação realizada entre os códigos BP, BC e BCP é justa e dá evidências no ganho expressivo de desempenho alcançado ao unificar o modelo de cobertura de um algoritmo de *branch-and-price* e as desigualdades válidas conhecidas para o modelo compacto (subseção 4.3.1) dentro de um algoritmo *branch-and-cut* dentro de um algoritmo *branch-and-cut-and-price*.

Capítulo 6

Conclusões

Um modelo de cobertura de conjuntos para o CmRSP e três relaxações para o subproblema de *pricing* que surge neste modelo foram propostos nesta tese. O desempenho de um algoritmo *branch-and-price* desenvolvido a partir deste modelo foi analisado e comparado com aquele de um algoritmo *branch-and-cut* que implementamos com base no único algoritmo exato descrito na literatura para o problema.

Como esperado, os limitantes duais gerados pelo modelo na raiz da árvore de *branch-and-price* foram mais apertados que aqueles gerados pelo algoritmo de *branch-and-cut*. As relaxações que proíbem k-ciclos e k-streams foram importantes para apertar o limitante dual. Os experimentos mostraram que os limitantes duais apertados ajudaram o *branch-and-price* a reduzir o tempo de execução e o *gap* de otimalidade. Uma das contribuições para melhorar o desempenho do BP, foi o uso de autômatos finitos determinísticos para identificar caminhos úteis. Ele diminuiu, em média, 2.7 vezes o tempo total gasto com a geração de colunas. Vale notar que não foi possível encontrar na literatura um outro exemplo de uso de um AFD no âmbito de geração de colunas.

Na média, o algoritmo de BP apresentou um desempenho similar ao BC. Em algumas instâncias, o BP superou o desempenho do BC e vice-versa, o que indica que um não domina o desempenho da outra. No entanto, um algoritmo *branch-and-cut-and-price* unindo ambos os algoritmos é bastante competitivo. A diferença de desempenho em relação ao BC é bastante expressiva quando o número de anéis-estrelas e, consequentemente, a capacidade dos veículos é alterada. Em instâncias onde a frota é maior, o algoritmo de *branch-and-cut-and-price* apresentou um desempenho bem superior àquele do algoritmo *branch-and-cut*.

Parte II

O problema da coloração particionada

Capítulo 7

Introdução

Dado um grafo $G = (V, E)$ e uma partição dos vértices, o **problema da coloração particionada** (PCP) consiste em encontrar um subconjunto de vértices V' que contenha exatamente um vértice de cada parte da partição e tal que o número cromático do grafo induzido por V' seja mínimo.

A Figura 7.1 mostra um exemplo de instância do PCP. As linhas pontilhadas são usadas para delinear as partes da partição, denotadas por Q_1, Q_2 e Q_3.

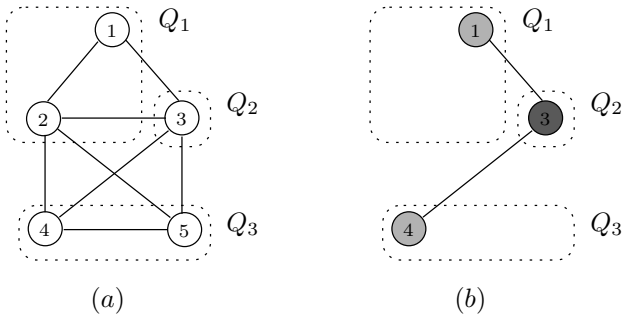


Figura 7.1: (a) Exemplo de instância do PCP e (b) uma solução ótima com duas cores.

É fácil ver que PCP é NP-difícil uma vez que ele generaliza o problema da coloração de vértices.

Li e Simha [33] foram os primeiros a estudar o problema, motivados por um problema que ocorre em redes óticas. Nestas redes, fibras óticas capazes de transmissão simultânea via múltiplos comprimentos de onda são usadas para conectar terminais da rede. Uma conexão entre um par de terminais é realizada através da determinação do caminho (ou rota) da rede ligando os terminais e a associação de um comprimento de onda. O problema que surge é a necessidade de atribuir comprimentos de ondas distintos para rotas que compartilham trechos comuns na rede (rotas conflitantes). Este problema é conhecido

como o **problema de roteamento e de associação de comprimentos de ondas**, do inglês *routing and wavelenght assigment problem* (RWA). Li e Simha resolveram uma versão *off-line* do problema, em que todas as rotas são conhecidas a priori para todo par de terminais que necessitam de conexão. O algoritmo de Li e Simha consiste em duas fases. Na primeira fase, todas as rotas alternativas para cada par de terminais da rede que precisam de conexão são computadas. Na segunda fase, uma rota e um comprimento de onda para cada conexão são determinados através da redução do problema ao PCP. A redução consiste em construir um grafo de interferência $G = (V, E)$, onde cada rota torna-se um vértice em V e arestas são colocadas em E para todo par de vértices relativos a rotas conflitantes. A partição do grafo é definida colocando-se rotas referentes ao mesmo par de terminais em uma mesma parte. Mais detalhes da aplicação e um exemplo desta redução são apresentados no Capítulo 1.

Até onde sabemos, poucos trabalhos foram dedicados ao PCP. Li e Simha [33] estenderam heurísticas clássicas do problema da coloração de vértices para obter um conjunto de heurísticas para o PCP. Mais tarde, Noronha e Ribeiro [41] propuseram uma heurística baseada em Busca Tabu para o PCP. O único algoritmo exato que pudemos encontrar na literatura para o problema é um algoritmo *branch-and-cut* proposto recentemente em [22].

Diferentemente de PCP, muitos trabalhos são conhecidos para o problema da coloração. Mehrotra e Trick [36] propuseram um algoritmo de *branch-and-price* para o problema da coloração, usando o modelo clássico por conjuntos independentes (IS). Um trabalho similar foi desenvolvido para um problema correlato, chamado problema da multi-coloração [37]. Algoritmos *branch-and-cut* para o problema da coloração foram propostos mais recentemente [38, 21, 9, 7]. Campêlo et al. [9] propuseram uma nova formulação chamada formulação por representantes, que é mais simples e compacta que as demais. Mais tarde, Campêlo et al [8] unificaram as formulações (IS) e por representantes para modelar o problema da coloração.

Em [22], Frota et al. estenderam a formulação por representantes para modelar o PCP e propuseram um algoritmo *branch-and-cut* para resolvê-lo. Nosso objetivo é usar a mesma idéia que Campêlo et al. [8] para unificar as formulações (IS) e por representantes para modelar o PCP e avaliar um algoritmo *branch-and-price* para resolver esse modelo unificado, comparando-o com o algoritmo *branch-and-cut* proposto em [22] para o problema.

Os capítulos a seguir descrevem o trabalho que fizemos em relação ao PCP para atender os objetivos acima. Assim, no Capítulo 8 modelos matemáticos para o PCP são apresentados e analisados. No capítulo seguinte, apresentamos os detalhes do algoritmo de *branch-and-price* que desenvolvemos para resolver o PCP e comparamos seu desempenho com o algoritmo *branch-and-cut* de Frota et al. Por fim, no Capítulo 10, fazemos

comentários finais sobre o trabalho realizado e damos direções para pesquisas futuras.

É preciso notar que esta pesquisa referente ao PCP foi realizada em conjunto com o Dr. Yuri Frota, pós-doutorando do IC da Unicamp.

Capítulo 8

Modelagem Matemática

Neste capítulo apresentamos uma definição formal e três modelos matemáticos para o problema da coloração particionada. O primeiro modelo é baseado no modelo por conjuntos independentes de Mehrotra e Trick [36]. O segundo trata-se do modelo por representantes proposto por Frota et al [22]. O último é um novo modelo para o problema, que unifica os dois primeiros.

Inicialmente, considere as seguintes notações e conceitos usados no texto.

Seja $G = (V, E)$ um grafo com $V = \{1, \ldots, n\}$, onde $n = |V|$.

A **antivizinhança** de um vértice i em G é o conjunto de vértices $\{j \in V : ij \notin E\}$ e é denotado por $A(i)$. Denote por $A'(i)$ o conjunto $A(i) \cup \{i\}$.

Seja Q uma partição de V. A parte da partição que contém um vértice u é denotado por $Q[u]$. Considere também $A_Q(u) = \{v \in A(u) : Q[u] \neq Q[v]\}$ e $A'_Q(u) = A_Q(u) + \{u\}$.

Dado um subconjunto S de vértices em V, o **subgrafo induzido** por S é o subgrafo de G restrito aos vértices S e arestas de G com extremidades em S. O subgrafo induzido por S é denotado por $G[S]$.

Um subconjunto S de vértices em V é um **conjunto independente** em G se todo par de vértices em S são não adjacentes.

Uma **coloração** dos vértices de G é uma função $f : V \mapsto N$. Se $f(i) \neq f(j)$, para todo par de vértices adjacentes i e j em V, tem-se que f é uma **coloração própria** de G. Uma coloração própria usando no máximo k inteiros (ou cores) distintos é chamada uma **k-coloração**. O **número cromático** de G, denotado por $\chi(G)$, é o menor inteiro para o qual G admite uma $\chi(G)$-coloração.

Uma definição formal do problema pode ser enunciada como segue. Considere um grafo $G = (V, E)$ e uma partição $Q = (Q_1, \ldots, Q_q)$ de V. O problema da coloração particionada consiste em encontrar um subconjunto de vértices S tal que:

- $|Q_k \cap S| = 1$, para todo Q_k em Q;

- o número cromático de $G[S]$ seja mínimo.

Nas próximas seções apresentamos os três modelos para o PCP.

8.1 Formulação por Conjunto Independente

Considere $\mathcal{P}$ o conjunto de todos os conjuntos independentes em G e seja λ_p uma variável de decisão associada a cada conjunto independente p em $\mathcal{P}$. O seguinte programa linear inteiro é a formulação por conjunto independente para o PCP:

$$
\begin{aligned}
\text{(PC)} \quad & \min \sum_{p \in \mathcal{P}} \lambda_p \\
\text{sujeito a} \quad & \sum_{p \in \mathcal{P}} a_i^p \lambda_p \leq 1, \forall i \in V && (8.1) \\
& \sum_{p \in \mathcal{P}} (\sum_{i \in Q_k} a_i^p) \lambda_p = 1, \forall Q_k \in Q && (8.2) \\
& \lambda_p \in \{0,1\}, \forall p \in \mathcal{P} && (8.3)
\end{aligned}
$$

onde $a_i^p = 1$ se e, somente, se i pertence ao conjunto independente associado a p.

As restrições (8.1) não são necessárias, mas podem ser mantidas para definir qual vértice de cada parte é selecionado para compor a solução. As restrições (8.2) forçam que cada parte da partição tenha exatamente um vértice colorido. As restrições de integralidade (8.3) forçam as variáveis de decisão serem inteiras.

Sejam π e α variáveis duais associadas às restrições 8.1 e 8.2, respectivamente. Logo, o custo reduzido de um conjunto independente p é dado por:

$$\tilde{c}_p = 1 - \sum_{i \in V} (\pi_i + \alpha_{Q[i]}) a_i^p.$$

Problema do Conjunto Independente de Peso Máximo

Note que o problema de *pricing* consiste em encontrar um conjunto independente p que minimize $\tilde{c}_p$. Uma vez que $\min_{p \in \mathcal{P}} \tilde{c}_p = 1 - \max_{p \in \mathcal{P}} \sum_{i \in V} (\pi_i + \alpha_{Q[i]}) a_i^p$, podemos redefinir o problema de *pricing* como o problema de encontrar um conjunto independente de peso máximo.

Seja $w_i = \pi_i + \sum_{C \in \mathcal{C}: i \in C} \alpha_C$ o peso definido para cada vértice i do grafo. Note que pode haver pesos negativos, uma vez que α é irrestrito em sinal.

Logo, o problema de *pricing* pode ser modelado pelo seguinte programa linear inteiro:

$$\begin{aligned} \text{(PC-pric)} \quad & \max \sum_{i \in v} w_i x_i \\ \text{sujeito a} \quad & x_i + x_j \leq 1, \forall ij \in E' && (8.4) \\ & x_i \in \{0,1\}, \forall i \in V && (8.5) \end{aligned}$$

Modelo por Cliques Maximais Ao invés das restrições de arestas (8.4) uma formulação mais forte baseada em cliques maximais pode ser utilizada [39].

Se C é uma clique em G, vale a seguinte desigualdade para o problema do conjunto independente de peso máximo:

$$\sum_{i \in C} x_i \leq 1.$$

No modelo por cliques maximais não é necessário utilizar todas as restrições de cliques. Para garantir que vértices adjacentes não sejam escolhidos para compor uma solução é suficiente considerar um conjunto de cliques maximais que cobrem todas as arestas do grafo.

Logo, se $\mathcal{C}$ é um conjunto de cliques maximais cobrindo as arestas do grafo, o seguinte PLI é um modelo por cliques maximais para o conjunto independente de peso máximo:

$$\begin{aligned} \text{(PC-pric)} \quad & \max \sum_{i \in v} w_i x_i \\ \text{sujeito a} \quad & \sum_{i \in C} x_i \leq 1, \forall C \in \mathcal{C} && (8.6) \\ & x_i \in \{0,1\}, \forall i \in V. && (8.7) \end{aligned}$$

8.2 Formulação por Representantes

De modo diferente, a formulação usada em [22] é baseada num modelo para o problema da coloração, que é chamada **formulação por representantes**. Nesse modelo, um vértice, chamado **representante**, dentre todos os vértices coloridos com a mesma cor é escolhido para representar aquela cor. Para tanto, utiliza-se variáveis de decisão binárias x_{uv}, para cada vértice $u \in V$ e $v \in A'(u)$ de modo que $x_{uv} = 1$ se e, somente, se u é o representante da cor do vértice v. Note que se u é um representante, tem-se que $x_{uu} = 1$. A seguinte

formulação por representante é um modelo para o problema da coloração:

$$\text{(CP)} \quad \min \sum_{u \in V} x_{uu}$$

$$\text{sujeito a} \quad \sum_{v \in A'(u)} x_{vu} \geq 1, \forall u \in V \tag{8.8}$$

$$x_{uv} + x_{uw} \leq x_{uu}, \forall u \in V, \forall vw \in E, \text{ com } v, w \in A(u) \tag{8.9}$$

$$x_{uv} \leq x_{uu}, \forall u \in V, \forall v \in A(u) \text{ tal que } v \text{ é isolado em } A(u) \tag{8.10}$$

$$x_{uv} \in \{0, 1\}, \forall u \in V, \forall v \in A'(u). \tag{8.11}$$

A formulação por representante para modelar o PCP é:

$$\text{(PCP)} \quad \min \sum_{u \in V} x_{uu}$$

$$\text{sujeito a} \quad \sum_{u \in Q_k} \sum_{v \in A'_p(u)} x_{vu} \geq 1, \forall Q_k \in Q \tag{8.12}$$

$$x_{uv} + x_{uw} \leq x_{uu}, \forall u \in V, \forall vw \in E, \text{ com } v, w \in A_p(u) \text{ e } Q[v] \neq Q[w] \tag{8.13}$$

$$x_{uv} \leq x_{uu}, \forall u \in V, \forall v \in A_p(u) \text{ tal que } v \text{ é isolado em } A_p(u) \tag{8.14}$$

$$x_{uv} \in \{0, 1\}, \forall u \in V, \forall v \in A_p^{'}(u). \tag{8.15}$$

As restrições (8.12) forçam cada parte da partição ter, pelo menos, um vértice colorido. Aqui, restrições de cobertura foram adotadas no modelo ao invés das restrições de partição, pelo fato de ser conhecido na literatura que esses últimos são, em geral, mais difíceis de se resolver. Note que, neste caso, uma fase de pós-processamento é necessária para remover vértices coloridos de modo que exatamente um vértice esteja colorido em cada parte da partição. As restrições (8.13) evitam que dois vértices adjacentes recebam a mesma cor, enquanto as restrições (8.14) junto com as restrições (8.13) garantem que um vértice seja colorido apenas se existir um representante para aquela cor.

Note que em toda solução viável qualquer vértice pode assumir o papel de representante, gerando um problema de simetria que é inerente às formulações por representantes. Uma maneira de contornar esse problema consiste em definir uma ordem total $\ll$ nos vértices do grafo de tal modo que um vértice u pode representar um outro vértice v somente se $u \ll v$.

Considere a **antivizinhança positiva** de i, denotado por $A_{\ll}(i)$, como o conjunto de vértices que podem ser representados por i, ou seja, $A_{\ll}(i) = \{j \in A_Q(i) : i \ll j\}$. Denote por $A'_{\ll}(i) = A_{\ll}(i) \cup \{i\}$. A formulação assimétrica por representantes para o PCP pode ser obtida da formulação (PCP) considerando $A_{\ll}$ no lugar de A_p e $A'_{\ll}$ ao invés de A'_Q.

8.3 Nova Formulação

Recentemente, Campêlo et al. propuseram um modelo para o problema da coloração que une as duas formulações, formulação por conjunto independente e por representante [8]. Usando essa mesma idéia, propomos uma formulação para o PCP que unifica as formulações (PC) e (PCP):

$$
\begin{aligned}
\text{(PCPr)} \quad & \min \sum_{p \in \mathcal{P}} \lambda_p \\
\text{sujeito a} \quad & \sum_{p \in \mathcal{P}} r_i^p \lambda_p \leq 1, \forall i \in V && (8.16) \\
& \sum_{p \in \mathcal{P}} a_i^p \lambda_p \leq 1, \forall i \in V && (8.17) \\
& \sum_{p \in \mathcal{P}} (\sum_{i \in Q_k} a_i^p) \lambda_p \geq 1, \forall Q_k \in Q && (8.18) \\
& \lambda_p \in \{0,1\}, \forall p \in \mathcal{P} && (8.19)
\end{aligned}
$$

onde $r_i^p = 1$ se e, somente, se i é o representante do conjunto independente associado a p.

As restrições (8.16) evitam que um vértice seja representante de mais de uma cor. As restrições (8.17) forçam cada vértice ser associado a, no máximo, uma cor. As restrições (8.18) garantem que cada parte da partição tenha pelo menos um vértice colorido, enquanto as restrições de integralidade (8.19) forçam as variáveis de decisão serem inteiras. Novamente, como no modelo (PCP), modelo de cobertura foi adotado ao invés do modelo de partição e, portanto, uma fase de pós-processamento deve ser aplicada para gerar uma solução ótima do PCP.

Sejam π, μ e α as variáveis duais associadas às restrições (8.16), (8.17) e (8.18), respectivamente, na relaxação linear de (PCPr). Logo, o dual da relaxação linear de (PCPr) é:

$$
\begin{aligned}
\text{(dual-PCPr)} \quad & \max \sum_{i \in V} (\pi_i + \mu_i) + \sum_{Q_k \in Q} (\alpha_{Q_k}) \\
\text{sujeito a} \quad & \sum_{i \in V} (\pi_i r_i^p + \mu_i a_i^p) + \sum_{Q_k \in Q} \alpha_{Q_k} \sum_{i \in Q_k} a_i^p \leq 1, \forall p \in P && (8.20) \\
& \pi_i, \mu_i \leq 0, \forall i \in V, \alpha_{Q_k} \geq 0, \forall Q_k \in Q && (8.21)
\end{aligned}
$$

Portanto, o custo reduzido de p, é dado por:

$$\overline{c}_P = 1 - \sum_{i \in V} (\mu_i + \alpha_{Q[i]}) a_i^p - \sum_{i \in V} \pi_i r_i^p.$$

O subproblema de *pricing* consiste em $\min_{p\in\mathcal{P}} \overline{c}_P$, que equivale a resolver o seguinte problema:

$$\text{(IS)} \quad \max \sum_{i\in V}(w_i x_i + \pi_i y_i)$$

sujeito a (8.22)

x é o vetor característico de um conjunto independente (8.23)

$y_i \in 0, 1, \forall i \in V$ (8.24)

$y_i = 1$ se e, somente, se i é o representante do conjunto independente, (8.25)

onde, $w_i = \mu_i + \alpha_{Q[i]}$.

Uma vez que $\mathcal{P}$ pode ser particionado em $|V|$ conjuntos disjuntos $P(i)$, onde $P(i) = \{P \in \mathcal{P} : i \text{ é representante de } P\}$, o problema (IS) pode ser resolvido pelos seguintes $|V|$ subproblemas independentes:

$$\text{(IS(k))} \quad \pi_k + \max \sum_{i\in A_{\ll}(k)} w_i x_i$$

sujeito a (8.26)

x é o vetor característico de um conjunto independente em $P(k)$. (8.27)

Note que $x_k = 1$ em qualquer solução do IS(k).

Uma observação importante é o porquê da necessidade de colocar as restrições (8.16) no modelo, já que não são necessárias para a caracterização de uma coluna do modelo (um conjunto independente). A explicação está no fato de que estamos interessados em colocar os cortes do modelo por representantes, que são dados em termos das variáveis compactas x_{uv}. A introdução de um corte da forma $\sum_{u\in V}\sum_{v\in V} B_{uv}x_{uv} \geq b$ em (PCPr) tem duas implicações:

- há a necessidade de reescrever o corte em termos das variáveis de decisão λ do (PCPr), para que o mesmo possa ser adicionado ao modelo;
- a introdução do corte causa mudanças nos coeficientes da função objetivo do problema de *pricing* (inerente a qualquer tipo de corte introduzido no problema mestre).

Sabe-se que $x_{uv} = \sum_{p\in\mathcal{P}} r_u^p a_v^p \lambda_p$. No entanto, o uso destas restrições de acoplagem não é viável devido à não linearidade de $r_u^p a_v^p$. A linearidade não é necessária quando introduz-se um corte, uma vez que a_v^p e r_u^p são constantes naquele instante (são calculadas apenas para as colunas que estão atualmente no problema mestre). A necessidade da linearidade fica evidente na geração de colunas, quando o dual associado ao corte é usado para modificar

a função objetivo do problema de *pricing*. Neste caso, a_u^p e r_u^p correspondem às variáveis x_u e y_u do problema de *pricing* (IS).

Ao introduzir as restrições (8.16) no modelo e dividir o problema de *pricing* em $|V|$ subproblemas (IS(k)), a distribuição do dual associado ao corte entre as variáveis x_u e y_u torna-se trivial, uma vez que $r_u^p = 1$, ou seja, é constante, no subproblema de *pricing* IS(u). Seja β o dual associado ao corte. Logo, com a introdução do corte, a função objetivo de p em $P(k)$ torna-se:

$$\pi_k + \max \sum_{i \in N(k)} (w_i + B_{ki}\beta)x_i.$$

8.4 Desigualdades Válidas

Famílias de desigualdades válidas para o modelo (PCP), chamadas cortes externos e cortes internos, foram estudadas [9, 7] e avaliadas [23] para verificar o ganho que a introdução destes cortes causam nos limitantes duais do modelo. O objetivo desta seção é mostrar que a maioria dessas desigualdades já é válida no modelo unificado e discutir os tipos de desigualdades que podem fortalecer o modelo.

Cortes externos Dado um vértice u em V e um subconjunto K de $A_{\ll}(u)$ com uma estrutura especial, como o fato de K definir uma clique ou um buraco ímpar, ou ainda um anti-buraco ímpar, **cortes externos** consistem em limitantes superiores para a quantidade de vértices que podem ser representados por u. A seguinte inequação dá a estrutura geral de um corte externo:

$$\sum_{v \in K} \frac{x_{uv}}{\alpha_v} \leq x_{uu}, \tag{8.28}$$

onde α_v é o tamanho do maior conjunto independente em $G[K]$ que contém v.

Cortes internos São caracterizados por inequações que dão um limitante inferior para o número cromático de subgrafos de G. Diferente do problema da coloração de vértices em que todo vértice deve ser colorido, no PCP não há nenhuma garantia, exceto pelo vértices elementares, de um vértice estar colorido em qualquer solução viável do PCP. Um vértice v é elementar se $|Q[v]| = 1$, ou seja, v é o único vértice da parte a qual pertence e, portanto, precisa estar colorido em qualquer solução viável do PCP.

Seja H um subconjunto de vértices elementares de G com uma estrutura especial, como buracos ímpares, e seja $\chi(G[H])$ o número cromático de $G[H]$. A seguinte inequação define um **corte interno**:

$$\sum_{v \in H} x_{vv} + \sum_{v \in H, w \in A_{\ll}(v) \setminus H} x_{wv} \geq \chi(G[H]). \tag{8.29}$$

Frota et al. [22] mostraram que a adição dos cortes externos no modelo por representantes (PCP) têm grande impacto para fortalecer os limitantes duais e acelerar o desempenho do algoritmo *branch-and-cut*, enquanto que os cortes internos apresentaram um impacto muito pequeno. Isso se explica pelo fato dos cortes internos dependerem da existência de vértices elementares no grafo.

No entanto, os cortes externos já são válidos na formulação (PCPr). Isso ocorre por causa da decomposição considerada na construção do modelo, em que cada coluna da matriz de coeficientes na formulação (PCPr) representa um conjunto independente. Neste caso, (PCPr) equivale ao modelo contendo a envoltória convexa dos pontos que representam os conjuntos independentes do grafo [49]. Desta forma, os cortes internos são os únicos cortes que poderiam separar pontos fracionários e fortalecer o modelo (PCPr). No entanto, Frota et al. mostraram que os cortes internos têm fraco impacto no desempenho de um algoritmo *branch-and-cut* [22]. Além disso, em testes preliminares nenhum dos cortes internos foi violado. Com base nesses fatos, desenvolvemos um algoritmo *branch-and-price* ao invés de um algoritmo *branch-and-cut-and-price* para resolver o PCP, o qual está descrito no próximo capítulo. Vale, portanto, notar que neste caso, as restrições (8.16) não são necessárias, mas optou-se por mantê-las no modelo para possibilitar a inclusão de outros cortes internos em trabalhos futuros.

Capítulo 9

Implementação

Este capítulo reúne os resultados dos experimentos conduzidos para avaliar um algoritmo *branch-and-price* usando o modelo unificado para o problema da coloração particionada apresentado no Capítulo 8.

9.1 Instâncias de Teste

Os testes foram realizados nas mesmas instâncias utilizadas em [22]. Elas estão agrupadas em três classes: aleatórias, NSF e RINGS. As Tabelas 9.1 e 9.2 reúnem algumas informações sobre as instâncias aleatórias e da classe NSF, respectivamente. As características das instâncias da classe RINGS estão reunidas nas Tabelas 9.3, 9.4 e 9.5. As colunas **n**, **m**, **C**, **d** e **c** representam o total de vértices, o número de arestas, o número de partes na partição, a densidade de arestas e o número de componentes conexos em cada instância, respectivamente.

9.2 Geração de Colunas

Inicialmente, realizamos testes para identificar o melhor algoritmo para resolver o problema de *pricing*. Três algoritmos exatos foram avaliados:

- SOLVER- consiste na utilização de um resolvedor de PLI para resolver o modelo por cliques maximais para o problema do conjunto independente máximo descrito na Seção 8.1;

- TRICK - algoritmo combinatório enumerativo proposto por Mehrotra e Trick [36] para resolver o problema do conjunto independente máximo;

instancia	**n**	**m**	**C**	**d**	**c**		instancia	**n**	**m**	**C**	**d**	**c**
n20p5t2s1	20	98	10	0.49	1		n90p1t2s1	90	404	45	0.1	1
n20p5t2s2	20	100	10	0.5	1		n90p1t2s2	90	402	45	0.1	1
n20p5t2s3	20	96	10	0.48	1		n90p1t2s3	90	427	45	0.11	1
n20p5t2s4	20	94	10	0.47	1		n90p1t2s4	90	391	45	0.1	1
n20p5t2s5	20	102	10	0.51	1		n90p1t2s5	90	444	45	0.11	1
n40p5t2s1	40	402	20	0.5	1		n90p2t2s1	90	786	45	0.19	1
n40p5t2s2	40	398	20	0.5	1		n90p2t2s2	90	801	45	0.2	1
n40p5t2s3	40	387	20	0.48	1		n90p2t2s3	90	838	45	0.21	1
n40p5t2s4	40	397	20	0.5	1		n90p2t2s4	90	777	45	0.19	1
n40p5t2s5	40	420	20	0.53	1		n90p2t2s5	90	827	45	0.2	1
n60p5t2s1	60	909	30	0.51	1		n90p3t2s1	90	1183	45	0.29	1
n60p5t2s2	60	872	30	0.48	1		n90p3t2s2	90	1202	45	0.3	1
n60p5t2s3	60	881	30	0.49	1		n90p3t2s3	90	1246	45	0.31	1
n60p5t2s4	60	884	30	0.49	1		n90p3t2s4	90	1171	45	0.29	1
n60p5t2s5	60	907	30	0.5	1		n90p3t2s5	90	1236	45	0.31	1
n70p5t2s1	70	1241	35	0.51	1		n90p4t2s1	90	1597	45	0.39	1
n70p5t2s2	70	1189	35	0.49	1		n90p4t2s2	90	1571	45	0.39	1
n70p5t2s3	70	1200	35	0.49	1		n90p4t2s3	90	1627	45	0.4	1
n70p5t2s4	70	1194	35	0.49	1		n90p4t2s4	90	1604	45	0.4	1
n70p5t2s5	70	1228	35	0.5	1		n90p4t2s5	90	1643	45	0.41	1
n80p5t2s1	80	1592	40	0.5	1		n90p6t2s1	90	2447	45	0.6	1
n80p5t2s2	80	1547	40	0.48	1		n90p6t2s2	90	2382	45	0.59	1
n80p5t2s3	80	1589	40	0.5	1		n90p6t2s3	90	2449	45	0.6	1
n80p5t2s4	80	1580	40	0.49	1		n90p6t2s4	90	2407	45	0.59	1
n80p5t2s5	80	1616	40	0.51	1		n90p6t2s5	90	2458	45	0.61	1
n90p5t2s1	90	2019	45	0.5	1		n90p7t2s1	90	2845	45	0.7	1
n90p5t2s2	90	1963	45	0.48	1		n90p7t2s2	90	2797	45	0.69	1
n90p5t2s3	90	2045	45	0.51	1		n90p7t2s3	90	2831	45	0.7	1
n90p5t2s4	90	2014	45	0.5	1		n90p7t2s4	90	2821	45	0.7	1
n90p5t2s5	90	2057	45	0.51	1		n90p7t2s5	90	2834	45	0.7	1
n100p5t2s1	100	2494	50	0.5	1		n90p8t2s1	90	3257	45	0.8	1
n100p5t2s2	100	2428	50	0.49	1		n90p8t2s2	90	3188	45	0.79	1
n100p5t2s3	100	2513	50	0.5	1		n90p8t2s3	90	3274	45	0.81	1
n100p5t2s4	100	2442	50	0.49	1		n90p8t2s4	90	3213	45	0.79	1
n100p5t2s5	100	2500	50	0.5	1		n90p8t2s5	90	3226	45	0.8	1
n120p5t2s1	120	3593	60	0.5	1		n90p9t2s1	90	3631	45	0.9	1
n120p5t2s2	120	3544	60	0.49	1		n90p9t2s2	90	3614	45	0.89	1
n120p5t2s3	120	3613	60	0.5	1		n90p9t2s3	90	3615	45	0.89	1
n120p5t2s4	120	3536	60	0.49	1		n90p9t2s4	90	3619	45	0.89	1
n120p5t2s5	120	3623	60	0.5	1		n90p9t2s5	90	3634	45	0.9	1

Tabela 9.1: Instâncias aleatórias.

instancia	**n**	**m**	**C**	**d**	**c**		instancia	**n**	**m**	**C**	**d**	**c**
nsf_p0.1_s2	22	27	16	0.11	5		nsf_p0.6_s2	154	1409	113	0.12	1
nsf_p0.1_s3	29	40	23	0.1	5		nsf_p0.6_s3	153	1371	112	0.12	1
nsf_p0.1_s4	38	67	30	0.09	3		nsf_p0.6_s4	161	1613	113	0.12	1
nsf_p0.1_s5	27	30	20	0.08	6		nsf_p0.6_s5	139	1184	103	0.12	1
nsf_p0.2_s1	37	69	31	0.1	2		nsf_p0.7_s1	180	2117	124	0.13	1
nsf_p0.2_s2	52	128	44	0.09	3		nsf_p0.7_s2	202	2670	144	0.13	1
nsf_p0.2_s3	51	130	40	0.1	1		nsf_p0.7_s3	177	1925	131	0.12	1
nsf_p0.2_s4	57	176	40	0.11	1		nsf_p0.7_s4	187	2151	135	0.12	1
nsf_p0.2_s5	66	242	44	0.11	1		nsf_p0.7_s5	159	1606	118	0.13	1
nsf_p0.3_s1	63	220	49	0.11	1		nsf_p0.8_s1	201	2549	147	0.13	1
nsf_p0.3_s2	87	452	64	0.12	1		nsf_p0.8_s2	221	3115	153	0.13	1
nsf_p0.3_s3	80	366	58	0.11	1		nsf_p0.8_s3	208	2692	147	0.12	1
nsf_p0.3_s4	80	397	59	0.12	1		nsf_p0.8_s4	209	2821	147	0.13	1
nsf_p0.3_s5	85	363	63	0.1	1		nsf_p0.8_s5	184	2014	139	0.12	1
nsf_p0.4_s1	91	527	66	0.13	1		nsf_p0.9_s1	216	2921	161	0.13	1
nsf_p0.4_s2	112	739	82	0.12	1		nsf_p0.9_s2	231	3324	167	0.12	1
nsf_p0.4_s3	101	606	73	0.12	1		nsf_p0.9_s3	217	2849	166	0.12	1
nsf_p0.4_s4	99	559	76	0.11	1		nsf_p0.9_s4	226	3118	165	0.12	1
nsf_p0.4_s5	112	741	80	0.12	1		nsf_p0.9_s5	234	3379	169	0.12	1
nsf_p0.5_s1	124	999	87	0.13	1		nsf_p1.0_s1	250	3948	182	0.13	1
nsf_p0.5_s2	130	1017	99	0.12	1		nsf_p1.0_s2	251	3973	182	0.13	1
nsf_p0.5_s3	122	848	92	0.11	1		nsf_p1.0_s3	238	3419	182	0.12	1
nsf_p0.5_s4	118	803	89	0.12	1		nsf_p1.0_s4	257	4261	182	0.13	1
nsf_p0.5_s5	132	1071	93	0.12	1		nsf_p1.0_s5	248	3827	182	0.12	1

Tabela 9.2: Instâncias NSF.

instancia	**n**	**m**	**C**	**d**	**c**		instancia	**n**	**m**	**C**	**d**	**c**
ring_n4p0.1s5	6	4	3	0.24	2		ring_n4p0.9s5	20	64	10	0.32	2
ring_n4p0.2s3	8	11	4	0.35	2		ring_n4p1s3	24	92	12	0.32	2
ring_n4p0.2s4	6	3	3	0.18	4		ring_n4p1s4	24	92	12	0.32	2
ring_n4p0.2s5	8	7	4	0.23	2		ring_n4p1s5	24	92	12	0.32	2
ring_n4p0.3s1	6	3	3	0.18	3		ring_n7p0.1s1	8	12	4	0.39	2
ring_n4p0.3s3	10	14	5	0.29	2		ring_n7p0.1s3	8	10	4	0.32	2
ring_n4p0.3s4	8	9	4	0.29	2		ring_n7p0.1s4	12	22	6	0.31	2
ring_n4p0.3s5	8	7	4	0.23	2		ring_n7p0.1s5	8	9	4	0.29	2
ring_n4p0.4s1	6	3	3	0.18	3		ring_n7p0.2s1	18	57	9	0.35	2
ring_n4p0.4s2	8	8	4	0.26	2		ring_n7p0.2s2	16	41	8	0.32	2
ring_n4p0.4s3	12	21	6	0.3	2		ring_n7p0.2s3	18	58	9	0.36	2
ring_n4p0.4s4	10	12	5	0.24	3		ring_n7p0.2s4	24	98	12	0.34	2
ring_n4p0.4s5	8	7	4	0.23	2		ring_n7p0.2s5	20	67	10	0.34	2
ring_n4p0.5s1	8	8	4	0.26	2		ring_n7p0.3s1	24	102	12	0.36	2
ring_n4p0.5s2	8	8	4	0.26	2		ring_n7p0.3s2	22	82	11	0.34	2
ring_n4p0.5s3	12	21	6	0.3	2		ring_n7p0.3s3	24	102	12	0.36	2
ring_n4p0.5s4	12	19	6	0.27	2		ring_n7p0.3s4	26	115	13	0.34	2
ring_n4p0.5s5	8	7	4	0.23	2		ring_n7p0.3s5	30	158	15	0.35	2
ring_n4p0.6s1	10	12	5	0.24	2		ring_n7p0.4s1	30	160	15	0.36	2
ring_n4p0.6s2	10	14	5	0.29	2		ring_n7p0.4s2	36	230	18	0.36	2
ring_n4p0.6s3	14	29	7	0.3	2		ring_n7p0.4s3	32	185	16	0.36	2
ring_n4p0.6s4	18	50	9	0.31	2		ring_n7p0.4s4	38	259	19	0.36	2
ring_n4p0.6s5	10	13	5	0.27	2		ring_n7p0.4s5	36	231	18	0.36	2
ring_n4p0.7s1	16	36	8	0.28	2		ring_n7p0.5s1	38	264	19	0.37	2
ring_n4p0.7s2	14	29	7	0.3	2		ring_n7p0.5s2	46	387	23	0.37	2
ring_n4p0.7s3	18	51	9	0.32	2		ring_n7p0.5s3	40	289	20	0.36	2
ring_n4p0.7s4	22	76	11	0.32	2		ring_n7p0.5s4	44	352	22	0.36	2
ring_n4p0.7s5	12	20	6	0.28	2		ring_n7p0.5s5	42	318	21	0.36	2
ring_n4p0.8s1	20	60	10	0.3	2		ring_n7p0.6s1	48	431	24	0.37	2
ring_n4p0.8s2	18	51	9	0.32	2		ring_n7p0.6s2	52	500	26	0.37	2
ring_n4p0.8s3	18	51	9	0.32	2		ring_n7p0.6s3	48	418	24	0.36	2
ring_n4p0.8s4	24	92	12	0.32	2		ring_n7p0.6s4	62	710	31	0.37	2
ring_n4p0.8s5	14	29	7	0.3	2		ring_n7p0.6s5	50	450	25	0.36	2

Tabela 9.3: Instâncias RINGS (parte 1).

instancia	**n**	**m**	**C**	**d**	**c**		instancia	**n**	**m**	**C**	**d**	**c**
ring_n7p0.7s1	58	631	29	0.38	2		ring_n10p0.5s1	76	1099	38	0.38	2
ring_n7p0.7s2	62	718	31	0.37	2		ring_n10p0.5s2	94	1700	47	0.38	2
ring_n7p0.7s3	64	758	32	0.37	2		ring_n10p0.5s3	78	1179	39	0.39	2
ring_n7p0.7s4	72	968	36	0.37	2		ring_n10p0.5s4	76	1114	38	0.39	2
ring_n7p0.7s5	56	572	28	0.37	2		ring_n10p0.5s5	92	1621	46	0.38	2
ring_n7p0.8s1	70	924	35	0.38	2		ring_n10p0.6s1	98	1837	49	0.38	2
ring_n7p0.8s2	72	979	36	0.38	2		ring_n10p0.6s2	106	2170	53	0.39	2
ring_n7p0.8s3	66	807	33	0.37	2		ring_n10p0.6s3	102	2027	51	0.39	2
ring_n7p0.8s4	80	1200	40	0.38	2		ring_n10p0.6s4	104	2084	52	0.39	2
ring_n7p0.8s5	62	707	31	0.37	2		ring_n10p0.6s5	102	2002	51	0.38	2
ring_n7p0.9s1	76	1083	38	0.38	2		ring_n10p0.7s1	122	2865	61	0.39	2
ring_n7p0.9s2	78	1148	39	0.38	2		ring_n10p0.7s2	138	3719	69	0.39	2
ring_n7p0.9s3	76	1084	38	0.38	2		ring_n10p0.7s3	130	3288	65	0.39	2
ring_n7p0.9s4	80	1200	40	0.38	2		ring_n10p0.7s4	132	3388	66	0.39	2
ring_n7p0.9s5	76	1078	38	0.37	2		ring_n10p0.7s5	114	2510	57	0.39	2
ring_n7p1s1	84	1330	42	0.38	2		ring_n10p0.8s1	146	4148	73	0.39	2
ring_n7p1s2	84	1330	42	0.38	2		ring_n10p0.8s2	152	4529	76	0.39	2
ring_n7p1s3	84	1330	42	0.38	2		ring_n10p0.8s3	144	4043	72	0.39	2
ring_n7p1s4	84	1330	42	0.38	2		ring_n10p0.8s4	146	4160	73	0.39	2
ring_n7p1s5	84	1330	42	0.38	2		ring_n10p0.8s5	138	3716	69	0.39	2
ring_n10p0.1s1	8	11	4	0.35	2		ring_n10p0.9s1	162	5144	81	0.39	2
ring_n10p0.1s2	16	45	8	0.35	2		ring_n10p0.9s2	164	5278	82	0.39	2
ring_n10p0.1s3	16	44	8	0.35	2		ring_n10p0.9s3	166	5396	83	0.39	2
ring_n10p0.1s4	28	149	14	0.38	2		ring_n10p0.9s4	164	5255	82	0.39	2
ring_n10p0.1s5	20	69	10	0.35	2		ring_n10p0.9s5	166	5414	83	0.39	2
ring_n10p0.2s1	26	124	13	0.37	2		ring_n10p1s1	180	6360	90	0.39	2
ring_n10p0.2s2	36	230	18	0.36	2		ring_n10p1s2	180	6360	90	0.39	2
ring_n10p0.2s3	30	168	15	0.37	2		ring_n10p1s3	180	6360	90	0.39	2
ring_n10p0.2s4	42	337	21	0.38	2		ring_n10p1s4	180	6360	90	0.39	2
ring_n10p0.2s5	44	355	22	0.37	2		ring_n10p1s5	180	6360	90	0.39	2
ring_n10p0.3s1	38	270	19	0.37	2		ring_n15p0.1s1	30	164	15	0.37	2
ring_n10p0.3s2	54	540	27	0.37	2		ring_n15p0.1s2	38	268	19	0.37	2
ring_n10p0.3s3	42	330	21	0.37	2		ring_n15p0.1s3	52	543	26	0.4	2
ring_n10p0.3s4	52	519	26	0.38	2		ring_n15p0.1s4	62	757	31	0.39	2
ring_n10p0.3s5	62	727	31	0.38	2		ring_n15p0.1s5	54	557	27	0.38	2
ring_n10p0.4s1	56	593	28	0.38	2		ring_n15p0.2s1	74	1081	37	0.39	2
ring_n10p0.4s2	72	979	36	0.38	2		ring_n15p0.2s2	100	1954	50	0.39	2
ring_n10p0.4s3	60	688	30	0.38	2		ring_n15p0.2s3	90	1620	45	0.4	2
ring_n10p0.4s4	68	888	34	0.38	2		ring_n15p0.2s4	94	1764	47	0.4	2
ring_n10p0.4s5	82	1283	41	0.38	2		ring_n15p0.2s5	106	2198	53	0.39	2

Tabela 9.4: Instâncias RINGS (parte 2).

instancia	**n**	**m**	**C**	**d**	**c**		instancia	**n**	**m**	**C**	**d**	**c**
ring_n15p0.3s1	116	2681	58	0.4	2		ring_n20p0.2s1	128	3281	64	0.4	2
ring_n15p0.3s2	146	4251	73	0.4	2		ring_n20p0.2s2	172	5954	86	0.4	2
ring_n15p0.3s3	130	3365	65	0.4	2		ring_n20p0.2s3	162	5212	81	0.4	2
ring_n15p0.3s4	140	3900	70	0.4	2		ring_n20p0.2s4	152	4651	76	0.4	2
ring_n15p0.3s5	150	4450	75	0.4	2		ring_n20p0.2s5	162	5368	81	0.41	2
ring_n15p0.4s1	162	5262	81	0.4	2		ring_n20p0.3s1	218	9509	109	0.4	2
ring_n15p0.4s2	188	7005	94	0.4	2		ring_n20p0.3s2	246	12308	123	0.41	2
ring_n15p0.4s3	164	5402	82	0.4	2		ring_n20p0.3s3	234	10951	117	0.4	2
ring_n15p0.4s4	180	6459	90	0.4	2		ring_n20p0.3s4	222	10058	111	0.41	2
ring_n15p0.4s5	196	7663	98	0.4	2		ring_n20p0.3s5	240	11802	120	0.41	2
ring_n15p0.5s1	208	8656	104	0.4	2		ring_n20p0.4s1	306	18772	153	0.4	2
ring_n15p0.5s2	226	10167	113	0.4	2		ring_n20p0.4s2	318	20693	159	0.41	2
ring_n15p0.5s3	208	8724	104	0.4	2		ring_n20p0.4s3	296	17570	148	0.4	2
ring_n15p0.5s4	210	8869	105	0.4	2		ring_n20p0.4s4	292	17305	146	0.41	2
ring_n15p0.5s5	226	10168	113	0.4	2		ring_n20p0.4s5	330	22257	165	0.41	2
ring_n15p0.6s1	250	12442	125	0.4	2		ring_n20p0.5s1	392	30924	196	0.4	2
ring_n15p0.6s2	258	13266	129	0.4	2		ring_n20p0.5s2	396	32151	198	0.41	2
ring_n15p0.6s3	258	13400	129	0.4	2		ring_n20p0.5s3	380	29061	190	0.4	2
ring_n15p0.6s4	260	13594	130	0.4	2		ring_n20p0.5s4	358	26018	179	0.41	2
ring_n15p0.6s5	252	12682	126	0.4	2		ring_n20p0.5s5	390	31090	195	0.41	2
ring_n15p0.7s1	288	16601	144	0.4	2		ring_n20p0.6s1	458	42316	229	0.4	2
ring_n15p0.7s2	330	21756	165	0.4	2		ring_n20p0.6s2	464	44107	232	0.41	2
ring_n15p0.7s3	306	18821	153	0.4	2		ring_n20p0.6s3	456	41906	228	0.4	2
ring_n15p0.7s4	310	19300	155	0.4	2		ring_n20p0.6s4	452	41530	226	0.41	2
ring_n15p0.7s5	282	15985	141	0.4	2		ring_n20p0.6s5	440	39445	220	0.41	2
ring_n15p0.8s1	336	22658	168	0.4	2		ring_n20p0.7s1	536	58142	268	0.4	2
ring_n15p0.8s2	352	24773	176	0.4	2		ring_n20p0.7s2	580	68658	290	0.41	2
ring_n15p0.8s3	338	22942	169	0.4	2		ring_n20p0.7s3	534	57579	267	0.4	2
ring_n15p0.8s4	344	23812	172	0.4	2		ring_n20p0.7s4	536	58373	268	0.41	2
ring_n15p0.8s5	328	21572	164	0.4	2		ring_n20p0.7s5	518	54784	259	0.41	2
ring_n15p0.9s1	370	27469	185	0.4	2		ring_n20p0.8s1	614	76568	307	0.41	2
ring_n15p0.9s2	386	29922	193	0.4	2		ring_n20p0.8s2	624	79501	312	0.41	2
ring_n15p0.9s3	380	29067	190	0.4	2		ring_n20p0.8s3	614	76290	307	0.4	2
ring_n15p0.9s4	380	29043	190	0.4	2		ring_n20p0.8s4	610	75741	305	0.41	2
ring_n15p0.9s5	392	30925	196	0.4	2		ring_n20p0.8s5	602	73890	301	0.41	2
ring_n15p1s1	420	35490	210	0.4	2		ring_n20p0.9s1	672	91739	336	0.41	2
ring_n15p1s2	420	35490	210	0.4	2		ring_n20p0.9s2	696	98814	348	0.41	2
ring_n15p1s3	420	35490	210	0.4	2		ring_n20p0.9s3	686	95572	343	0.41	2
ring_n15p1s4	420	35490	210	0.4	2		ring_n20p0.9s4	686	95789	343	0.41	2
ring_n15p1s5	420	35490	210	0.4	2		ring_n20p0.9s5	706	101600	353	0.41	2
ring_n20p0.1s1	56	626	28	0.4	2		ring_n20p1s1	760	117420	380	0.41	2
ring_n20p0.1s2	80	1248	40	0.39	2		ring_n20p1s2	760	117420	380	0.41	2
ring_n20p0.1s3	90	1590	45	0.39	2		ring_n20p1s3	760	117420	380	0.41	2
ring_n20p0.1s4	92	1696	46	0.4	2		ring_n20p1s4	760	117420	380	0.41	2
ring_n20p0.1s5	74	1095	37	0.4	2		ring_n20p1s5	760	117420	380	0.41	2

Tabela 9.5: Instâncias RINGS (parte 3).

- ADAPTIVE - uma das soluções anteriores é escolhida dinamicamente para resolver o *pricing*, dependendo da densidade do subgrafo de entrada do problema de *pricing*. Se a densidade do grafo por maior que 40% utiliza-se o TRICK, caso contrário o SOLVER é selecionado.

Também avaliamos quatro heurísticas de *pricing* parcial para encontrar boas colunas de custo reduzido negativo a um custo menor de processamento: TRICK$_p$, GRASP , GRAP e ADAPTIVE$_p$. A primeira refere-se ao algoritmo TRICK definido anteriormente, mas aqui é usado como uma heurística, ou seja, o algoritmo pára quando colunas de custo reduzido negativo são encontradas ou um limite de tempo é atingido. A segunda é um algoritmo *Greedy randomized Adaptative Search Procedure* [20] proposto para o PCP em [22]. A terceira heurística consiste no algoritmo GRASP sem a rotina de busca local. A última utiliza a mesma idéia que ADAPTIVE e escolhe TRICK$_p$ ou GRASP como heurística, dependendo da densidade do subgrafo de entrada. Todas as heurísticas terminam a procura por colunas de custo reduzido negativo quando encontram uma quantidade pré-determinada de colunas de custo reduzido negativo ou ultrapassam um limite de tempo de processamento. Fixamos a quantidade de colunas em 20% do número de vértices do grafo de entrada e o limite de tempo em 10 segundos.

Neste experimento foram utilizadas 45 instâncias com 90 vértices, divididas em grupos de 5 instâncias com densidades variando de 0.1 a 0.9. Os códigos foram executados apenas para resolver a relaxação linear na raiz da árvore de enumeração e, assim, permitir uma comparação dos diferentes algoritmos de geração de colunas para o PCP. O desempenho de cada esquema de combinação de algoritmo de *pricing* parcial e exato é apresentado na Tabela 9.6. Uma linha da tabela é dedicada para cada esquema. A soma dos tempos de processamentos de cada esquema para resolver a geração de colunas em cada grupo de instância é apresentada nas colunas `tempo`. O número de instâncias resolvidas mais rapidamente por cada esquema é dado pela coluna `fast`, enquanto a coluna `opt` mostra o total de instâncias resolvidas na otimalidade. A média da taxa de speed-up, que é calculado dividindo-se o tempo de processamento do esquema mais lento com o tempo de processamento do esquema, é apresentada na coluna `speed-up`.

Como pode ser observado na Tabela 9.6, os esquemas que usam o algoritmo de TRICK (esquemas `S1`, `S2`, `S3`, `S4`, `S7` e `S11`) não foram capazes de resolver a relaxação linear de instâncias com densidade baixa (inferior a 30%) enquanto os esquemas `S5`, `S6` e `S8` resolveram todas as instâncias de testes na otimalidade, mas são mais lentos. Para permitir uma comparação dos diferentes esquemas, calculamos uma média ponderada de desempenho para cada esquema da seguinte forma: o valor para cada esquema nas colunas `fast`, `opt` e `speed-up` é dividido pelo valor máximo na coluna correspondente e multiplicado, respectivamente, pelos pesos 0.05, 0.75 e 0.2. Estes pesos foram escolhidos para refletir a importância que damos para cada uma dessas informações. A Tabela 9.7 apresenta as

S1	GRASP+TRICK	S5	GRASP+SOLVER	S9	GRASP+ADAPTIVE
S2	GRAP+TRICK	S6	GRAP+SOLVER	S10	GRAP+ADAPTIVE
S3	TRICK$_p$TRICK	S7	TRICK$_p$+SOLVER	S11	TRICK$_p$+ADAPTIVE
S4	ADAPTIVE$_p$+TRICK	S8	ADAPTIVE$_p$+SOLVER	S12	ADAPTIVE$_p$+ADAPTIVE

	tempo(s)/densidade											
sch.	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	fast	opt	speed-up
S1	*	1536	96.0	27.6	13.4	8.6	6.1	4.8	3.5	0	36	0.7
S2	*	*	119	31.6	21.2	14.7	9.2	7.2	4.1	0	33	1.2
S3	*	*	70.7	12.5	6.4	4.9	3.4	3.3	3.1	0	33	3.0
S4	*	*	66.2	7.1	1.8	1.1	0.9	1.0	1.0	17	33	9.0
S5	419	188	81.1	41.4	20.5	7.9	2.3	1.6	1.3	1	45	3.3
S6	154	150	98	57.5	31.9	10.6	3.8	1.9	1.3	9	45	3.1
S7	*	945	61.2	13.5	5.9	2.6	1.4	1.2	1.2	1	38	4.5
S8	157	983	61.1	13.3	6.0	2.7	1.4	1.3	1.2	2	45	4.2
S9	437	1078	72.3	14.8	5.6	2.5	1.5	1.3	1.2	0	44	4.0
S10	155	1073	77.3	8.7	2.7	1.6	1.2	1.2	1.1	1	42	6.0
S11	*	1659	66.1	7.2	1.8	1.1	0.9	1.0	1.1	10	35	8.7
S12	156	1141	66.3	7.1	1.8	1.1	0.9	1.1	1.1	4	41	7.9

Tabela 9.6: Geração de colunas nas instâncias da classe RAND.

médias ponderadas de desempenho para cada um dos esquemas propostos.

ordem	1	2	3	3	4	5	6	7	8	9	10	11
média	0.87	0.85	0.84	0.84	0.83	0.82	0.81	0.8	0.74	0.62	0.61	0.58
esquema	S12	S8	S6	S10	S5	S9	S11	S4	S7	S3	S1	S2

Tabela 9.7: Média ponderada de desempenho.

Usando esta medida, o esquema S12 teve a melhor média ponderada, seguido pelo esquema S8. De fato, ambos os esquemas apresentaram melhor desempenho geral, no sentido em que manteve um bom compromisso entre otimalidade e speed-up. Isso se justifica pelo fato de que instâncias esparsas são resolvidas usando-se o SOLVER e instâncias densas tiram proveito da eficiência do TRICK.

9.3 Base Inicial

A base inicial para o problema mestre restrito referente ao modelo (PCPr) consiste de colunas relativas às variáveis de folga do modelo e um conjunto de q colunas, uma para

cada parte da partição Q. Seja Q_k uma parte da partição Q e v um vértice em Q_k. A coluna relativa a Q_k que compõe a base inicial contém coeficientes não-nulos nas entradas correspondentes ao vértice v, nas restrições (8.16) e (8.17), e ao Q_k nas restrições (8.18). Isso garante que as colunas iniciais formam uma base e uma solução primal viável.

9.3.1 Heurística Primal

Nós avaliamos o impacto no desempenho do algoritmo *branch-and-price* ao incluir uma heurística primal para gerar soluções primais em cada nó da árvore de enumeração. Adotamos a heurística de busca Tabu proposta por Frota et al. [22] e utilizada no código *branch-and-cut*.

Ao habilitar a heurística primal em cada nó da árvore de enumeração, obtivemos uma melhora no desempenho do algoritmo *branch-and-price*, como pode ser constatada na Tabela 9.8. O campo GAP representa a redução média do gap de otimalidade, dentre as instâncias não resolvidas na otimalidade, ao habilitar a heurística primal. O TEMPO refere-se ao speedup médio, definido pela razão do tempo médio para resolver as instâncias na otimalidade sem habilitar a heurística e o tempo médio ao habilitá-la. O campo OTIMALIDADE mostra o aumento no número de instâncias resolvidas na otimalidade ao habilitar a heurística.

INSTANCIA	GAP	TEMPO	OTIMALIDADE
n20p5	*	1	0
n40p5	*	1.3	0
n60p5	*	2.83	0
n70p5	*	1.34	0
n80p5	*		1
n90p5	0		2
n90p3	12.95		0
n90p4	0		0
n90p5	7.5		2
n90p6	*		1
n90p7	*	77.43	0
n90p8	*	3.23	0
n90p9	*	0.31	0
total	5.11	12.49	6

Tabela 9.8: Melhora no desempenho ao habilitar uma heurística primal.

Houve um aumento no total de instâncias resolvidas na otimalidade, assim como uma redução considerável no tempo de processamento e no gap final, os quais mostram a importância da heurística primal no desempenho do algoritmo *branch-and-price*.

9.3.2 Colunas Iniciais

Ao adicionar as colunas relativas às soluções geradas pela heurística primal obtivemos um aumento de soluções degeneradas, que tornaram lenta a convergência da geração colunas em alguns casos. Por outro lado, a inclusão destas colunas permitiram o *branch-and-price* resolver um número maior de instâncias.

Uma vez que o número de colunas relativas às soluções primais é reduzido, iniciamos um estudo para povoar a base inicial e, desta forma, tentar melhorar a convergência da geração de colunas.

Sabe-se que a adição de "boas" colunas na base inicial podem ajudar na convergência do método. A dificuldade está em determinar colunas "boas". Para este fim, realizamos um estudo da base ótima da relaxação linear do problema mestre obtida na raiz da árvore de enumeração. Este estudo nos permitiu identificar um conjunto de operações de inserção e remoção de vértices que aplicadas às colunas da base inicial podem ser transformadas em colunas da base ótima e, portanto, são fortes candidatas a entrar na base e assim ajudar na convergência do método da geração de colunas. Note que a base ótima da relaxação pode ser alterada dependendo das colunas adicionadas na base inicial. Isso implica que não há garantias de que as colunas adicionadas por meio desta técnica estarão na base ótima.

O algoritmo utilizado para povoar a base consiste em realizar operações de remoção e inclusão de vértices nos conjuntos independentes da solução primal para gerar outros conjuntos independentes. O pseudocódigo do algoritmo é apresentado no Algoritmo 17.

Dado um conjunto independente S em G, o algoritmo gera novos conjuntos independentes removendo um vértice k de S e tentando incluir vizinhos de k em S, quando possível.

A inclusão das colunas geradas pela heurística POVOAMENTO (Algoritmo 17) na base inicial de cada nó da árvore de enumeração contribuiu para diminuir drasticamente o número de soluções degeneradas e consequentemente contribuindo para a convergência da geração de colunas. A Tabela 9.9 resume os ganhos no desempenho do *branch-and-price* ao povoar a base. A coluna SEM POVOAMENTO refere-se ao *branch-and-price* sem a inclusão de nenhuma coluna na base inicial, seja da heurística primal ou da rotina de povoar a base. A coluna COM POVOAMENTO refere-se aos resultados do algoritmo *branch-and-price* após incluir colunas relativas às soluções primais e daquelas geradas pela rotina para povoar a base inicial. A linha DEGERESCÊNCIA representa o percentual de soluções degeneradas do total de iterações da geração de colunas. As linhas OTIMALIDADE e TEMPO MÉDIO apresentam o total de instâncias resolvidas na otimalidade por cada algoritmo e o tempo médio de processamento para resolve-los, respectivamente.

Para avaliar a qualidade das colunas geradas pela rotina para povoar a base inicial, utilizamos dois valores. Ambos referem-se a comparações entre as colunas da base ótima

Algoritmo 17: Heurística POVOAMENTO.

Input: grafo $G = (V, E)$, um conjunto independente S em G
Output: conjuntos independentes em G

```
1  L = ∅; /* Constrói novos conjuntos independentes removendo um vértice de S */
2  for k ∈ S do
3      Candidatos= ∅
4      for i adjacente a k do
5          if ij ∉ E, ∀j ∈ S then
6              Candidatos=Candidatos +{i}
7      for i ∈ Candidatos do
8          S̄ = S − {k} + {i}
9          Candidatos=Candidatos −{i}
10         for j ∈ Candidatos do
11             if lj ∉ E, ∀l ∈ S̄ then
12                 S̄ = S̄ + {j}
13                 Candidatos=Candidatos −{j}
14         L = L + {S̄}
15 devolva L
```

	SEM POVOAMENTO	COM POVOAMENTO
DEGERESCÊNCIA (%)	41.18	13.84
OTIMALIDADE	279	343
TEMPO MÉDIO (S)	62.29	42.97

Tabela 9.9: Média de degenerescência.

da relaxação linear e aquelas da base inicial. O primeiro refere-se ao total de colunas da base ótima da relaxação linear que eram colunas da base inicial. A outra medida refere-se à distância média das colunas da base ótima da relaxação linear em relação à colunas da base inicial. A distância entre duas colunas é calculada pelo total de alterações (inclusão ou remoção de vértices) necessárias em uma coluna para torná-las idênticas. A Tabela 9.10 apresenta ambas as medidas comparando o algoritmo *branch-and-price* usando ou não a heurística POVOAMENTO para povoar a base inicial. A primeira linha refere-se ao percentual de colunas da base ótima que são colunas da base inicial. A segunda linha apresenta a distância média das colunas da base ótima e da base inicial.

Os resultados apresentados nas tabelas anteriores, deixam claro a melhora na convergência ao povoar a base inicial usando a heurística POVOAMENTO.

	sem Povoamento	com Povoamento
colunas iniciais (%)	7.7	76.4
distância média	5.5	3.3

Tabela 9.10: Colunas da base ótima da relaxação linear.

9.4 Detalhes de Implementação

Nesta seção apresentamos alguns detalhes de implementação de nosso algoritmo *branch-and-price*.

9.4.1 Limitantes Duais

Avaliamos três maneiras diferentes de calcular um limitante dual inicial em cada nó da árvore de enumeração:

- LB_PAI = limitante dual do nó pai;
- ELB = estimativa para o limitante dual. De fato, não se trata de um limitante dual. O valor ótimo da relaxação linear do problema mestre restrito inicial do nó é usado como estimativa;
- LASDON = limitante de Lasdon [32] calculado a partir da solução ótima do problema de *pricing*. Para calcular este limitante nós incluímos a restrição $\sum_{p \in P} \lambda_p \leq UB$ na formulação (PCPr), onde UB é o melhor limitante primal conhecido naquele nó da árvore de enumeração.

Utilizamos o algoritmo trick no passo de *pricing* dentro de nosso algoritmo *branch-and-price* para avaliar os limitantes duais. Um total de 30 instâncias com 90 vértices e diferentes densidades foi avaliado.

A Tabela 9.11 sumariza o desempenho do algoritmo *branch-and-price* usando cada uma das três maneiras de calcular um limitante dual inicial. A primeira linha da tabela mostra o total de instâncias resolvidas mais rapidamente por cada um dos métodos. Na segunda, é apresentado o total de instâncias resolvidas na otimalidade em cada caso. Por último, na terceira linha é dado o speed-up médio.

	ELB	LASDON	LB_PAI
mais rapidamente	14	11	5
otimalidade	29	30	29
Speed-up	0.85	0.78	0.84

Tabela 9.11: Comparativo dos cálculos de limitantes duais.

Embora usando o limitante ELB, o algoritmo tenha resolvido uma quantidade maior de instâncias mais rapidamente que os demais, o ganho de desempenho com respeito ao speed-up foi menor do que com o limitante LASDON. O limitante de Lasdon tem uma característica adicional em relação aos demais limitantes, que o possibilitou resolver uma instância a mais na otimalidade. A característica adicional do LASDON é que ele também é um limitante dual que pode ser obtido ao longo das iterações do método da geração de colunas, o que permite o término mais cedo da otimização de um nó.

Com base nesse experimento, o limitante LASDON foi adotado para ser incorporado dentro do nosso algoritmo *branch-and-price*.

9.4.2 Regra de Branching

Para realizar o branching utilizamos a mesma regra descrita em [22], que descrevemos brevemente nesta subseção.

Considere Q_i e Q_j duas partes da partição Q que não estão completamente conectados, ou seja, existem vértices $u \in Q_i$ e $v \in Q_j$ tais que $(u, v) \notin E$. A estratégia de branching consiste em criar dois subproblemas: no primeiro, vértices em Q_i e Q_j são forçados a terem o mesmo representante, ou seja, coloridos pela mesma cor, enquanto no segundo subproblema representantes distintos são requeridos para cada parte. O primeiro branching equivale a unir as partes Q_i e Q_j para formar uma nova parte Q_{ij}, onde pares de vértices $u \in Q_i$ e $v \in Q_j$ não adjacentes são unidos para formar um novo vértice uv. A vizinhança do novo vértice uv é definida pela vizinhança de u e de v. O segundo branching pode ser implementado inserindo arestas entre todo par de vértices não adjacentes de Q_1 e Q_2. Os grafos resultantes do processo de branching sobre as partes Q_1 e Q_2 são mostrados na Figura 9.1. O grafo resultado do primeiro branching é mostrado em (a) e do segundo branching em (b).

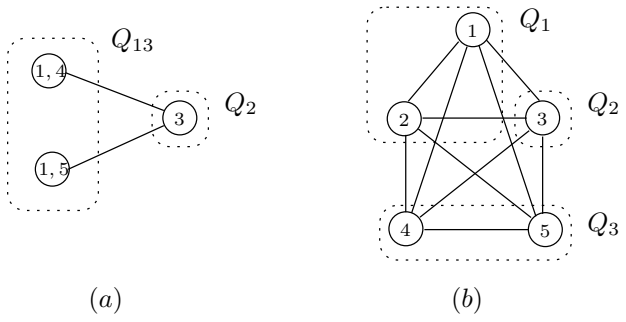


Figura 9.1: Branching (a) e (b) das partes Q_1 and Q_3.

9.5 Resultados Computacionais

Nesta seção apresentamos o desempenho do algoritmo BP usando os algoritmos de *pricing*, heurísticas e limitantes descritos nas seções anteriores e comparando-o com o desempenho do algoritmo *branch-and-cut* (chamaremos de código BC) proposto anteriormente em [22], cujo código original foi disponibilizado pelos autores para a realização dos testes. Desta forma, foi possível executar ambos os códigos na mesma máquina, usando as mesmas linguagens e bibliotecas. Para evitar distorção nos resultados, foram desconsideradas das análises, todas as instâncias resolvidas em menos de 1 segundo por algum dos códigos.

As Tabelas 9.12, 9.13, 9.14 e 9.15 apresentam os tempos de processamento para resolver cada instância nos diferentes esquemas de *pricing* do algoritmo BP. As colunas RAIZ(BC) e RAIZ(BP) mostram os valores das relaxações lineares na raiz da árvore de enumeração obtida pelo BC e BP, respectivamente. As colunas UB e LB referem-se aos melhores limitantes primais e duais obtidas pelos algoritmos. As demais colunas referem-se ao tempo de processamento gasto em cada algoritmo para resolver cada instância. As variantes dos algoritmos BP estão explicadas no título da tabela. O símbolo "*" é usado para indicar que o algoritmo não resolveu a instância dentro do limite de 1800 segundos.

Na Tabela 9.16 resumimos o total de instâncias resolvidas na otimalidade e o tempo médio gastos por cada um dos algoritmos.

Um resumo da comparação entre BC e BP é dado na Tabela 9.17. Apresentamos o desempenho dos algoritmos em cada classe de instância. O total de instâncias consideradas em cada classe é mostrada entre parênteses logo após o nome da classe. Colunas `opt`, `fast` e `speed-up` têm os mesmos significados que as da Tabela 9.6. Coluna `dual` representa o total de instâncias em que o algoritmo gerou limitante dual no nó raiz da árvore de enumeração melhor do que seu oponente. O melhor desempenho entre BC e BP em cada coluna é destacado em negrito nas entradas dessa tabela.

O algoritmo BP apresentou um desempenho geral superior em relação ao *branch-and-cut* em muitas instâncias. Inspecionando a coluna OPT na Tabela 9.17, podemos notar que BP resolveu na otimalidade 28 instâncias a mais que BC, que representa 15% do total. Como esperado, o limitante dual na raiz da árvore de enumeração usando a nova formulação proposta nesta tese foi mais apertado que o do modelo usado pelo algoritmo BC. A superioridade do BP sobre BC também é significativa quando analisamos o tempo de processamento (BP resolveu cerca de 64% das instâncias mais rapidamente que BC, reduzindo o tempo médio em 40%). Restringindo nossa análise apenas para as instâncias resolvidas na otimalidade por ambos os algoritmos, o número de vezes que o BP foi mais rápido é quase o dobro que BC e, consequentemente, BP teve *speed-up* maiores.

No entanto, o BC resolveu mais instâncias na otimalidade do que o BP na classe RINGS. Uma vez que tais instâncias são de interesse prático, decidimos fazer uma análise

instância	raiz(BC)	raiz(BP)	LB	UB	BC	BP(1)	BP(2)	BP(3)	BP(4)
n60p5t2s1	4.41	4.62	5	5	3.74	1.08	0.93	2.03	5.86
n70p5t2s2	4.45	4.96	6	6	815.78	11.8	9.66	26.14	70.88
n70p5t2s3	4.49	4.81	6	6	682.56	59.38	59.56	146.08	240.62
n70p5t2s4	4.55	4.93	6	6	566.74	6.48	16.32	43.35	122.29
n80p5t2s1	4.98	5.45	6	6	8.56	1.35	1.29	3.54	12.58
n80p5t2s2	4.92	5.39	6	6	37.93	1.12	1.18	3.36	4.4
n80p5t2s3	5.07	5.54	6	6	17.54	114.11	10.85	37.44	120.13
n80p5t2s4	5.04	5.64	6	6	89.2	235.81	150.79	810.59	*
n80p5t2s5	5.06	5.51	6	6	2.86	15.58	14.22	49.12	124.85
n90p5t2s1	5.34	5.88	7	7	*	72.12	71.29	224.78	597.82
n90p5t2s2	5.12	5.67	7	7	*	568.11	324.02	1058.83	1355.72
n90p5t2s3	5.3	5.96	7	7	*	23.82	21.62	80.47	487.45
n90p5t2s4	5.32	6.04	7	7	*	2.14	1.82	6.3	12.39
n90p5t2s5	5.53	6.27	7	7	*	1.62	1.5	4.95	10.98
n100p5t2s1	5.57	6.32	7	7	*	2.87	3.05	8.3	22.81
n100p5t2s2	5.44	6.28	7	7	*	3.39	3.56	9.88	48.89
n100p5t2s3	5.6	6.38	7	7	*	14.5	13.7	44.36	202.66
n100p5t2s4	5.4	6.08	7	7	*	2.58	2.97	8.04	31.47
n100p5t2s5	5.63	6.28	7	7	*	2.78	2.57	7.26	17.74
n120p5t2s1	6.06	7.29	8	8	*	7.08	7.15	18.73	38.52
n120p5t2s2	5.9	6.97	8	8	*	148.7	64.52	190.54	244.83
n120p5t2s3	6.06	7.29	8	8	*	6.32	7.96	30.31	65.93
n120p5t2s4	5.85	7	8	8	*	20.78	19.08	54.13	183.82
n120p5t2s5	6.28	7.35	8	8	*	12.16	11.81	32.09	116.49
n90p1t2s1	1.61	1.89	2	3	*	*	*	*	*
n90p1t2s2	1.63	1.87	2	3	*	*	*	*	*
n90p1t2s3	1.67	1.95	3	3	*	*	881.61	992.31	*
n90p1t2s4	1.59	1.85	2	2	40.08	*	*	*	*
n90p1t2s5	1.69	1.98	3	3	1069.24	1123.73	537.52	526.22	1076.02
n90p2t2s1	2.23	2.77	3	4	*	*	*	*	*
n90p2t2s2	2.24	2.69	3	3	280.53	*	*	*	*
n90p2t2s3	2.32	2.91	3	4	*	*	*	*	*
n90p2t2s4	2.22	2.75	3	4	*	*	*	*	*
n90p2t2s5	2.34	2.85	3	4	*	*	*	*	*
n90p3t2s1	3.05	3.75	4	5	*	*	*	*	*
n90p3t2s2	3.16	3.78	4	5	*	*	*	*	*
n90p3t2s3	3.2	3.9	5	5	*	*	1253.39	1345.2	*
n90p3t2s4	3.02	3.66	4	5	*	*	*	*	*
n90p3t2s5	3.17	3.84	5	5	*	*	*	*	1409.86
n90p4t2s1	4.02	4.69	5	6	*	*	*	*	*
n90p4t2s2	3.96	4.69	5	6	*	*	*	*	*
n90p4t2s3	4.1	4.81	6	6	*	518.36	337.74	774.61	*
n90p4t2s4	4.11	4.78	6	6	*	1254.56	165.76	334.79	630.99
n90p4t2s5	4.15	4.91	6	6	*	159.38	233.11	529.11	630.99
n90p6t2s1	6.93	7.4	8	8	19.33	1.25	1.06	2.71	3.28
n90p6t2s2	6.73	7.2	8	8	224.15	1.19	1.02	2.94	8.51
n90p6t2s3	6.91	7.38	8	8	20.66	1.23	1.23	3.09	5.92
n90p6t2s5	7.21	7.73	9	9	*	147.26	345.35	723.84	706.27
n90p8t2s1	11.19	11.41	12	12	1.89	1.17	1.09	1.24	1.45
n90p8t2s2	10.8	10.87	12	12	34.17	37.54	47.25	60.22	39.56
n90p8t2s3	11.51	11.9	12	12	2.28	1.2	1.13	1.31	1.51
n90p8t2s5	11.3	11.57	12	12	2.51	1.15	1.11	1.25	1.65
n90p9t2s1	15.12	15.67	16	16	2.27	1.32	1.3	1.35	1.64
n90p9t2s2	14.86	15.07	16	16	20.03	1.08	1.1	1.15	1.19
n90p9t2s3	15.06	15.56	16	16	1.82	1.1	1.12	1.23	1.19
n90p9t2s5	14.73	14.84	16	16	38.69	91.54	69.33	78.15	34.54

Tabela 9.12: Algoritmos BC e BP (1) esquema `S12` sem povoar base inicial, (2) esquema `S12`, (3) `S8` e (4) `S5` povoando a base inicial nas instâncias RAND.

instância	raiz(BC)	raiz(BP)	LB	UB	BC	BP(1)	BP(2)	BP(3)	BP(4)
nsf_p0.4_s3	5.75	6	6	6	2.45	38.44	6.28	6.55	3.1
nsf_p0.4_s4	6.25	6.25	7	7	2.36	4.5	5.56	5.29	8.66
nsf_p0.4_s5	5.5	5.6	6	6	1.25	8.51	13.4	9.78	13.38
nsf_p0.5_s1	7.25	7.5	8	8	8.5	4.42	2.68	2.59	4.86
nsf_p0.5_s2	7.25	7.5	8	8	1.55	12.09	19.9	15.17	14.81
nsf_p0.5_s4	6.75	7	7	7	14.38	16	12.24	28.1	29.7
nsf_p0.5_s5	6.5	6.8	7	7	1.48	9.69	13.95	12.23	15.84
nsf_p0.6_s1	8.05	8.75	9	9	65.45	9.41	6.31	5.18	6.51
nsf_p0.6_s2	8.33	9	9	9	3.11	15.97	2.01	1.69	2.3
nsf_p0.6_s3	8	8.33	9	9	*	11.78	19.03	17.56	11.38
nsf_p0.6_s4	7.88	8.25	9	9	*	21.84	86.54	107.83	30.06
nsf_p0.6_s5	7.75	8.5	9	9	86.68	6.76	2.07	2	2.35
nsf_p0.7_s1	9	9.14	10	10	*	31.18	115.85	82	32.69
nsf_p0.7_s2	10.75	10.75	11	11	104.1	43.62	41.63	34.8	27.61
nsf_p0.7_s3	8.6	9.5	10	10	*	30.15	3.96	3.37	9.08
nsf_p0.7_s4	10.33	10.33	11	11	4.27	79.55	53.45	51.39	26.27
nsf_p0.7_s5	8.25	8.5	9	9	3.53	101.98	165.44	84.17	36.3
nsf_p0.8_s1	10.17	10.75	11	11	266.9	31.37	238.16	62.82	23.74
nsf_p0.8_s2	10.5	10.75	11	11	154.41	98.59	74.56	81.93	26.08
nsf_p0.8_s3	9.5	10.5	11	11	*	130.9	185.95	110.03	43.17
nsf_p0.8_s4	10	10.33	11	11	*	75.86	96.47	132.93	55.7
nsf_p0.8_s5	9.75	10.5	11	11	*	23.8	11.05	10.29	9.23
nsf_p0.9_s1	11	11.5	12	12	*	55.58	194.96	66.34	68.5
nsf_p0.9_s2	11.25	11.75	12	12	14.93	68.87	186.92	360.26	45.42
nsf_p0.9_s3	11.25	11.75	12	12	11.06	56.85	13.74	12.01	34.87
nsf_p0.9_s4	11	11.29	12	12	*	251.05	274.88	304.99	132.76
nsf_p0.9_s5	11	11.5	12	12	*	74.99	298.85	186.92	35.97
nsf_p1.0_s1	*	12.67	13	13	*	126.28	184.92	84.52	54.97
nsf_p1.0_s2	12.13	12.5	13	13	1461.06	790.8	397.46	182.28	123.52
nsf_p1.0_s3	12.25	12.75	13	13	16.56	96.71	29.8	29.44	12.44
nsf_p1.0_s4	*	12.5	13	13	*	261.13	274.95	151.3	108.79
nsf_p1.0_s5	12.5	12.67	13	13	15.07	80.56	12.39	12.24	31.82

Tabela 9.13: Algoritmos BC e BP (1) esquema `S12` sem povoar base inicial, (2) esquema `S12`, (3) `S8` e (4) `S5` povoando a base inicial nas instâncias Nsf.

instância	raiz(BC)	raiz(BP)	LB	UB	BC	BP(1)	BP(2)	BP(3)	BP(4)
ring_n10p0.7s3	10	10	10	10	1,84	582	1,12	1,5	1,34
ring_n10p0.8s1	10,75	10,75	11	11	2,2	1542,75	4,57	5,08	3,8
ring_n10p0.8s2	12	12	12	12	6,21	1020,77	1,36	1,74	2,12
ring_n10p0.8s3	11	11	11	11	1,48	*	1,32	1,84	2,27
ring_n10p0.8s4	11	11	11	11	2,59	1403,82	1,46	1,75	1,74
ring_n10p0.8s5	10,5	10,5	11	11	1,35	412,19	2,53	2,95	1,84
ring_n10p0.9s1	12	12	12	12	8,38	*	2,5	3,08	2,87
ring_n10p0.9s2	12	12	12	12	6,03	*	2,79	3,46	3,22
ring_n10p0.9s3	12,5	12,5	13	13	2,66	*	3,26	4,09	4,33
ring_n10p0.9s4	11,5	11,5	12	12	3,83	*	13,4	14,79	7,91
ring_n10p0.9s5	12,5	12,5	13	13	2,17	*	6,93	7,8	3,8
ring_n10p1s1	12,5	12,5	13	13	7,2	*	73,36	34,64	9,66
ring_n10p1s2	12,5	12,5	13	13	7,03	*	72,35	34,93	9,62
ring_n10p1s3	12,5	12,5	13	13	7	*	73,95	34,91	9,6
ring_n10p1s4	12,5	12,5	13	13	7,07	*	73,57	34,74	9,61
ring_n10p1s5	12,5	12,5	13	13	7,02	*	74,5	34,86	10,86
ring_n15p0.3s3	13	13	13	13	1,37	13,26	1,22	1,59	1,87
ring_n15p0.3s4	11,5	11,5	12	12	1,73	33,57	2,56	3,12	2,4
ring_n15p0.3s5	11,5	11,5	12	12	2,19	332,12	3,44	3,78	2,48
ring_n15p0.4s1	15	15	15	15	7,19	1254,37	1,55	2,28	2,83
ring_n15p0.4s2	14	14	14	14	142,43	*	3,65	4,29	4,57
ring_n15p0.4s3	15	15	15	15	36,73	352,86	1,28	1,95	2,38
ring_n15p0.4s4	14	14	14	14	13,19	*	3,52	4,39	6,49
ring_n15p0.4s5	15,5	15,5	16	16	4,83	*	39,71	41,09	10,1
ring_n15p0.5s1	17,5	17,5	18	18	4,14	*	1,94	3,59	4,47
ring_n15p0.5s2	17,5	17,5	18	18	5,78	*	5,91	7,19	7,61
ring_n15p0.5s3	18,5	18,5	19	19	4,03	*	2,41	3,77	4,59
ring_n15p0.5s4	16,25	16,24	17	17	6,49	*	57,91	55,48	13,4
ring_n15p0.5s5	18	18	18	18	26,12	*	4,67	6,06	10,05
ring_n15p0.6s1	18,5	18,5	19	19	15,1	*	7,57	10,25	17,68
ring_n15p0.6s2	19	19	19	19	1034,78	*	12,2	13,75	13,67
ring_n15p0.6s3	22	22	22	22	10,91	*	5,61	8,34	10,54
ring_n15p0.6s4	19,25	19,25	20	20	13,32	*	935,33	418,82	20,99
ring_n15p0.6s5	20,5	20,5	21	21	9,08	*	10,04	12,2	15,38
ring_n15p0.7s1	20,5	20,5	21	21	27,67	*	558,14	275,41	45,58
ring_n15p0.7s2	24,5	24,5	25	25	35,64	*	10,54	13,25	26,25
ring_n15p0.7s3	22,5	23	23	23	42,97	*	15,86	20,86	22,29
ring_n15p0.7s4	23	23	23	23	33,8	*	17,83	21,74	24,53
ring_n15p0.7s5	22,5	22,5	23	23	25,33	*	9,28	14,57	15,77
ring_n15p0.8s1	24	24	24	24	622,95	*	16,64	18,83	21,37
ring_n15p0.8s2	25	25	25	25	66,66	*	15,11	18,62	30,64
ring_n15p0.8s3	25,5	25,5	26	26	27,05	*	29	32	28,6
ring_n15p0.8s4	25	25	25	25	64,83	*	39,81	44,57	31,84
ring_n15p0.8s5	24,5	24,5	25	25	31,68	*	26,83	32,12	55,12
ring_n15p0.9s1	26	26	26	26	379,69	*	51,21	57,28	37,36
ring_n15p0.9s2	26,75	26,75	27	27	126,3	*	*	*	87,24
ring_n15p0.9s3	27,5	27,5	28	28	58,03	*	*	254,62	70,26
ring_n15p0.9s4	27	27	27	27	186,47	*	41,59	44,13	46,38
ring_n15p0.9s5	27	27	27	27	*	*	*	*	180,02
ring_n15p1s1	28	28	28	29	*	*	*	*	*
ring_n15p1s2	28	28	28	29	*	*	*	*	*
ring_n15p1s3	28	28	28	29	*	*	*	*	*
ring_n15p1s4	28	28	28	29	*	*	*	*	*
ring_n15p1s5	28	28	28	29	*	*	*	*	*

Tabela 9.14: Algoritmos BC e BP (1) esquema `S12` sem povoar base inicial, (2) esquema `S12`, (3) `S8` e (4) `S5` povoando a base inicial nas instâncias RING (parte 1).

instância	raiz(BC)	raiz(BP)	LB	UB	BC	BP(1)	BP(2)	BP(3)	BP(4)
ring_n20p0.2s1	9,5	10	10	10	14,28	20,16	1,12	1,52	1,87
ring_n20p0.2s2	14,5	14,5	15	15	2,92	79,51	7,04	7,81	5,41
ring_n20p0.2s3	12,5	13	13	13	2,2	*	1,32	1,97	2,75
ring_n20p0.2s4	12,5	13	13	13	2,48	784,1	1,56	1,97	2
ring_n20p0.2s5	15,5	15,5	16	16	53,59	44,23	20,19	20,63	7,04
ring_n20p0.3s1	17,5	17,5	18	18	6,74	*	30,91	32,34	9,91
ring_n20p0.3s2	24	24	24	24	9,76	*	6,55	8,32	8,9
ring_n20p0.3s3	17	17,5	18	18	*	*	8,44	10,41	12,09
ring_n20p0.3s4	19,5	20	20	20	7,35	*	4,01	5,46	6,81
ring_n20p0.3s5	20	20,5	21	21	*	*	15,77	17,3	13,64
ring_n20p0.4s1	24	24	24	24	32,81	*	9,95	14,59	24,86
ring_n20p0.4s2	27,5	27,5	28	28	18,13	*	14,19	15,57	16,09
ring_n20p0.4s3	20,5	20,5	21	21	22,25	*	1556,27	223,32	48,91
ring_n20p0.4s4	24,5	24,5	25	25	12,6	*	7,44	10,82	15,22
ring_n20p0.4s5	28	28,5	29	29	*	*	12,93	15,46	28,69
ring_n20p0.5s1	32	32	32	32	53,9	*	22,39	26	43,11
ring_n20p0.5s2	32	32	32	32	100,26	*	34,37	39,69	34,53
ring_n20p0.5s3	27	27	27	27	240,88	*	*	712,15	119,57
ring_n20p0.5s4	27,5	27,5	28	28	28,18	*	64,15	70,88	34,34
ring_n20p0.5s5	31,5	32	32	32	54,73	*	207,39	141,44	71,21
ring_n20p0.6s1	35,5	35,5	36	36	85,25	*	36,03	38,76	76,71
ring_n20p0.6s2	36	36	36	36	210,98	*	50,53	54,78	104,14
ring_n20p0.6s3	*	32	32	32	*	*	84,89	90,39	101,57
ring_n20p0.6s4	32	32	32	32	206,73	*	118,94	131,12	117,37
ring_n20p0.6s5	33,5	34	34	34	122,44	*	37,09	39,28	82,02
ring_n20p0.7s1	*	39	39	39	*	*	94,96	98,78	151,44
ring_n20p0.7s2	43	43	43	43	402,79	*	73,35	76,64	98,04
ring_n20p0.7s3	36,5	36,75	37	37	539,55	*	*	938,87	157,86
ring_n20p0.7s4	37,5	37,5	38	38	551,39	*	*	*	309,93
ring_n20p0.7s5	37,5	37,5	38	38	334,9	*	*	*	419,85
ring_n20p0.8s1	*	44	44	44	*	*	174,1	178,22	432,9
ring_n20p0.8s2	46	46	46	46	752,8	*	93,38	96,98	132,57
ring_n20p0.8s3	42,5	42,5	43	43	821,8	*	*	424,78	448,46
ring_n20p0.8s4	43	43	43	43	923,23	*	*	*	1391,85
ring_n20p0.8s5	43	43	43	43	831,63	*	181,66	192,88	331,78
ring_n20p0.9s1	47,5	47,5	48	48	1425,82	*	1533,57	1309,8	813,9
ring_n20p0.9s2	48,5	48,5	49	49	1600,56	*	*	*	487,79
ring_n20p0.9s3	*	*	47	48	*	*	*	*	*
ring_n20p0.9s4	*	*	47	48	*	*	*	*	*
ring_n20p0.9s5	*	48,5	49	49	*	*	*	*	1194,8
ring_n20p1s1	*	*			*	*	*	*	*
ring_n20p1s2	*	*			*	*	*	*	*
ring_n20p1s3	*	*			*	*	*	*	*
ring_n20p1s4	*	*			*	*	*	*	*
ring_n20p1s5	*	*			*	*	*	*	*

Tabela 9.15: Algoritmos BC e BP (1) esquema `S12` sem povoar base inicial, (2) esquema `S12`, (3) `S8` e (4) `S5` povoando a base inicial nas instâncias RING (parte 2).

instância	BC	BP(1)	BP(2)	BP(3)	BP(4)
tempo	154.4	177.02	97.29	112.08	114.52
otimalidade(187)	122	86	150	154	158

Tabela 9.16: Resumo comparativo dos algoritmos BC e BP (1) esquema `S12` sem povoar base inicial, (2) esquema `S12`, (3) `S8` e (4) `S5` povoando a base inicial.

	BC				BP			
classe	opt	fast	dual	speed-up	opt	fast	dual	speed-up
RAND(56)	23	4	0	2.46	**42**	**17**	**53**	**27.02**
NSF (32)	20	**10**	1	**11.28**	**32**	**10**	**27**	6.87
RING(99)	**79**	25	1	**10.47**	76	**45**	**10**	7.12
TOTAL(187)	122	39	2	9.86	**150**	**72**	**90**	**11.79**

Tabela 9.17: Resumo dos desempenhos dos códigos BP e BC.

mais apurada nessa classe. Inicialmente, notamos que o algoritmo TRICK, quando aplicado no método ADAPTIVE, não foi capaz de resolver sequer a primeira iteração da geração de colunas em algumas instâncias. Como observamos nos experimentos anteriores, esse algoritmo tem dificuldade muito grande para lidar com instâncias esparsas. Como a densidade de arestas das instâncias da classe RINGS é 0.35, percebemos que esta poderia ser a explicação para a perda de desempenho do BP nesta classe de instância. Repetimos, então, o experimento usando o esquema `S8` no lugar do esquema `S12` para avaliar o comportamento do BP naquelas instâncias, uma vez que este esquema apresentou a segunda melhor performance, perdendo apenas para o esquema `S12` conforme a média ponderada de desempenho considerada na Tabela 9.6. Os novos resultados estão sumarizados na Tabela 9.18.

	BC				BP			
classe	opt	fast	dual	speed-up	opt	fast	dual	speed-up
RAND(56)	23	5	0	6.44	**42**	**16**	**56**	**11.4**
NSF (32)	20	**11**	1	8.69	**32**	9	**27**	**8.83**
RING(99)	79	30	0	4.63	**80**	**44**	**11**	**6.08**
TOTAL(187)	122	46	1	5.63	**154**	**69**	**94**	**7.68**

Tabela 9.18: Comparativo de desempenho do BP usando esquema `S8` e BC.

Conforme os testes reportados na Tabela 9.6, o desempenho do BP nas instâncias

RAND diminuiu quando o método TRICK foi substituído pelo SOLVER na solução do *pricing*. Por outro lado, para as classes NSF e RINGS, esta substituição reverteu a situação observada na Tabela 9.18. Como esperávamos o número de instâncias resolvidas na otimalidade pelo BP na classe RINGS superou o do BC. No entanto, houve uma pequena queda no total de instâncias resolvidas mais rapidamente pelo BP. Isso é uma indicação que o tempo gasto com o resolvedor de PLI na formulação do conjunto independente máximo é sensível à estrutura do grafo. Esta é uma questão que merece uma investigação mais profunda, uma vez que não estamos convencidos que a densidade do grafo seja a única propriedade que afeta a performance do algoritmo.

Capítulo 10

Conclusões

Nós estudamos um novo modelo de PLI para o problema da coloração particionada e propusemos um algoritmo *branch-and-price* para resolvê-lo de forma exata. Destacamos que parte do algoritmo *branch-and-price* proposto nesta tese é devido a trabalhos anteriores. As heurísticas primal de busca Tabu e de *pricing* parcial GRASP propostas em [22] foram utilizadas no algoritmo *branch-and-price*. Nossa contribuição foi unificar dois modelos existentes e propor um algoritmo *branch-and-price* para resolver o problema.

O modelo é baseado em um trabalho similar que foi aplicado ao problema da coloração clássica e consiste em unificar duas formulações: uma usando conjuntos independentes e outra usando vértices representantes. Experimentos computacionais foram realizados sob um total de 372 instâncias para comparar o desempenho do algoritmo *branch-and-price* em relação ao único algoritmo exato, um algoritmo *branch-and-cut* proposto em [22, 23], que é conhecido na literatura até esta data para o problema. Como esperado, os limitantes duais gerados pelo algoritmo foram muito mais apertados que aqueles obtidos pelo algoritmo *branch-and-cut*, que usa o modelo por representantes.

O algoritmo *branch-and-price* apresentou um desempenho geral superior ao *branch-and-cut* em todas as classes de instâncias de teste. Mais especificamente, o *branch-and-price* resolveu 82% das instâncias de teste (17% a mais do que *branch-and-cut*), e 60% das instâncias resolvidas por ambos foi processada mais rapidamente pelo *branch-and-price*, o que implicou em um tempo médio 27% menor em relação ao *branch-and-cut*. O passo crucial para o algoritmo *branch-and-price* atingir estes resultados foi a escolha das colunas iniciais para povoar a base inicial. Um estudo detalhado das colunas da base ótima da relaxação linear do problema mestre nos ajudaram no desenvolvimento de um método para povoar a base que substancialmente melhoraram o desempenho do algoritmo *branch-and-price*. Observamos que as colunas geradas pelo método reduziram a degenerescência e aceleraram a convergência da geração de colunas, os quais contribuíram para reduzir o tempo de processamento global do algoritmo.

Capítulo 11

Conclusões Finais

Nesta tese propusemos novas formulações por programação linear inteira e novos algoritmos exatos para dois problemas de otimização em grafos: o problema dos anéis-estrelas capacitados e o problema da coloração particionada. As novas formulações baseiam-se no modelo de cobertura de conjuntos e os algoritmos exatos foram desenvolvidos usando-se o método da geração de colunas. A qualidade dos limitantes obtidos pela relaxação linear dos novos modelos e o desempenho dos algoritmos exatos propostos (*branch-and-price* e *branch-and-cut-and-price*) foram avaliados por experimentos computacionais. A avaliação consistiu na comparação destes limitantes e desempenhos com aqueles obtidos por outros algoritmos exatos conhecidos na literatura, os quais se baseiam no método de *branch-and-cut*. Os resultados desta análise mostraram que o método da geração de colunas é adequado para a resolução de ambos os problemas.

Desenvolvemos dois algoritmos exatos, um algoritmo *branch-and-price* e outro *branch-and-cut-and-price*, a partir do modelo de cobertura de conjuntos para o problema dos anéis-estrelas capacitados. Os desempenhos destes algoritmos foram comparados com aquele gerado por um algoritmo *branch-and-cut* que implementamos com base no único algoritmo exato que está descrito na literatura para o problema. Como esperado, os limitantes duais gerados pelo modelo na raiz da árvore de *branch-and-price* foram, quase que na totalidade, mais apertados que aqueles gerados pelo algoritmo de *branch-and-cut*. Os experimentos mostraram que os desempenhos dos algoritmos *branch-and-price* e *branch-and-cut* eram similares, ou seja, em algumas instâncias, o *branch-and-price* superou o desempenho do *branch-and-cut* e vice-versa. Por outro lado, o desempenho do algoritmo *branch-and-cut-and-price* foi muito superior àquele do *branch-and-cut*. O *branch-and-cut-and-price* resolveu 64% das instâncias de teste de um total de 273, 23% a mais que o *branch-and-cut*. Também notamos que os desempenhos dos algoritmos são afetados de forma diferente quando o número de anéis-estrelas e a capacidade dos veículos são alterados. Em instâncias onde a frota é maior, o algoritmo de *branch-and-cut-and-price*

apresentou um desempenho bem superior àquele do algoritmo *branch-and-cut*.

No problema da coloração particionada, um total de 187 instâncias de teste foram consideradas nos experimentos para comparar o algoritmo *branch-and-price* proposto nesta tese com um algoritmo *branch-and-cut*, o único reportado na literatura para o problema. O algoritmo *branch-and-price* resolveu aproximadamente 82% destas instâncias, o que representa 17% de instâncias a mais que o algoritmo *branch-and-cut*. Com relação às instâncias resolvidas por ambos os algoritmos, *branch-and-price* resolveu a maioria das instâncias mais rapidamente que o *branch-and-cut* e o tempo médio foi cerca de 27% menor que em relação ao *branch-and-cut*.

Uma das contribuições para melhorar o desempenho dos algoritmos *branch-and-price* e *branch-and-cut-and-price* para o problema dos anéis-estrelas capacitados, foi o uso de autômatos finitos determinísticos para identificar caminhos úteis (rotina usada dentro do algoritmo para geração de colunas). Ele diminuiu, em média, $2,7$ vezes o tempo total gasto com a geração de colunas. No problema da coloração particionada, observamos que a heurística que desenvolvemos para povoar a base inicial teve um papel importante para melhorar o desempenho geral do algoritmo *branch-and-price*. Ela ajudou a reduzir a degenerescência em mais de 27% e a aumentar a convergência do algoritmo de geração de colunas, que permitiram o algoritmo resolver 64 instâncias a mais na otimalidade. O método empregado para a construção da heurística consiste no estudo da base ótima da relaxação linear do problema mestre obtida na raiz da árvore de enumeração de um algoritmo. Este estudo nos permitiu identificar um conjunto de operações de inserção e remoção de vértices que aplicadas às colunas da base inicial podem ser transformadas em colunas da base ótima e, portanto, são fortes candidatas a entrar na base e, assim, ajudar na convergência do método da geração de colunas. Pelo nosso conhecimento, não há outros trabalhos que utilizam esta idéia para povoamento da base inicial, que potencialmente podem diminuir o número de iterações em um algoritmo de geração de colunas, como constatado no problema da coloração particionada.

Resumidamente, as principais contribuições da tese foram:

- novas formulações de programação linear inteira para o problema dos anéis-estrelas capacitados e para o problema da coloração particionada;
- novos algoritmos exatos, um *branch-and-cut* e outro *branch-and-cut-and-price*, para o problema dos anéis-estrelas capacitados;
- um novo algoritmo exato, um *branch-and-price*, para o problema da coloração particionada;
- um novo algoritmo para identificação de caminhos úteis baseado em autômatos de estados finitos;

- uma heurística para povoamento de bases iniciais, que pode ser usada em algoritmos de geração de colunas para a resolução de outros problemas de otimização combinatória.

Algumas melhorias podem ser incluídas nos algoritmos de *branch-and-price* e *branch-and-cut-and-price* propostos nesta tese como, por exemplo, incluir heurísticas primais para o algoritmo *branch-and-cut-and-price* do problema dos anéis-estrelas capacitados, aplicar a heurística para povoamento da base ao algoritmo *branch-and-cut-and-price* do problema dos anéis-estrelas capacitados, entre outros. No entanto, elas apenas contribuiriam para melhorar os desempenhos destes algoritmos, que atualmente já superam expressivamente os dos algoritmos *branch-and-cut* existentes. Além destas melhorias, damos como direções futuras, estender os algoritmos para o problema dos anéis-estrelas capacitados com demanda não unitária.

Referências Bibliográficas

[1] V. Aho, R. Sethi, e J.D. Ullman. *Compilers, Principles,. Techniques, and Tools*. Addison-Wesley, 1986.

[2] R.K. Ahuja, T.L. Magnanti, e J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[3] H.B. Amor, J. Desrosiers, e J.M. V. de Carvalho. Dual-optimal inequalities for stabilizes column generation. *Operations Research*, 54(3):454–463, 2006.

[4] R. Baldacci, M. Dell'Amico, e J.J. Salazar. The capacitated m-ring star problem. *Operations Research*, 55:1147–1162, 2007.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, e P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[6] M. Bazaraa, J.J. Jarvis, e H.D. Sherali. *Linear Programming and Networks Flows*. John Wiley & Sons, 2nd edition, 2003.

[7] M. Campelo e V. Campos. On the asymmetric representatives formulation for the vertex coloring problem. In *Proceedings of the 2nd Brazilian Symposium on Graphs, Algorithms and Combinatorics*, volume 19 of *Electronic Notes in Discrete Mathematics*, pages 337–343, Rio de Janeiro, april 2005.

[8] M. Campêlo, V. Campos, R. Corrêa, e C. Rodrigues. On fractional and integral chromatic numbers of a graph via cutting and pricing. In *Proceedings of Fifth ALIO/EURO Conference on Combinatorial Optimization*, pages 42–42, Paris, october 2005.

[9] Y. Corrêa e M. Campelo. Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, 89:159–164, 2004.

[10] G. Dantzig e P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[11] G. B. Dantzig e J.H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[12] M. Dell'Amico, F. Maffioli, e P. Varbrand. On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3):297–308, 1995.

[13] M. Desrochers e F. Soumis. A generalized permanent labelling algorithm for the shortest path problems with time windows. *Information Systems and Operations Research*, 26:191–212, 1988.

[14] J. Desrosiers e M. E. Lübbecke. *Column Generation*, chapter A primer in Column Generation. Springer, 2005.

[15] J. Desrosiers, F. Soumis, e M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[16] M. Dror. Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978, 1994.

[17] O. du Merle, D. Villeneuve, J. Desrosiers, e P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

[18] I. Elhallaoui, A. Metrane, G. Desaulniers, e F. Soumis. An improved primal simplex algorithm for degenerate linear programs. Technical Report G-2007-66, Les Cahiers du GERAD, HEC Montréal, 2007.

[19] D. Feillet, P. Dejax, e M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.

[20] T.A. Feo e M.G.C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.

[21] R. Figueiredo, V. Barbosa, N. Maculan, e C. C. de Souza. New 0-1 integer formulations of the graph coloring problem. In *Proceedings of the XI Congresso Latino Iberoamericano de Investigacion de Operaciones*, Concepción, Chile, october 2002.

[22] Y. Frota, N. Maculan, T. Noronha, e C. Ribeiro. A branch-and-cut algorithm for partition coloring. In *Proceedings of the International Network Optimization Conference*, Spa, Belgium, april 2007.

[23] Y. Frota, N. Maculan, T. Noronha, e C.C. Ribeiro. A branch-and-cut algorithm for partition coloring. *Networks*, 2009. (to appear).

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

[25] A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

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

[27] J. Gondzio e R. Sarkissian. Column generation with a primal-dual method. Technical Report 96.6, University of Geneva, 102 Bd Carl Vogt, CH-1211 Geneva 4, Switzerland, 1996.

[28] E. A. Hoshino e C. C. de Souza. Um estudo do método da geração de colunas em programação inteira: uma aplicação ao problema do separador de vértices. Technical Report IC-07-03, Institute of Computing, University of Campinas, 2007.

[29] D.J. Houck, J.C. Picard, M. Queyranne, e R.R Vemuganti. The travelling salesman problem as a constrained shortest path problem: theory and computational experience. *Opsearch*, 17:93–109, 1980.

[30] S. Irnich e G. Desaulniers. Shortest path problems with resource constraints. Technical Report G-2004-11, Les Cahiers du GERAD, HEC Montréal, Montréal, Quebec, Canada, 2004.

[31] S. Irnich e D. Villeneuve. The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *Informs Journal on Computing*, 18(3):391–406, 2006.

[32] L.S. Lasdon. *Optimization Theory for Large Systems*. MacMillan, 1970.

[33] G. Li e R. Simha. The partition coloring problem and its application to wavelength routing and assignment. In *Proceedings of the First Workshop on Optical Networks*, 2000.

[34] J. Lysgaard, A. Letchford, e R. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

[35] M. E. Lübbecke e J. Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2002.

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

[37] A. Mehrotra e M.A. Trick. *Extending the Horizons: Advances in Computing, Optimization and Decision Technologies*, volume 37 of *Operations Research/Computer Science Interface*, chapter A branch-and-price approach for Graph Multi-Coloring, pages 15–30. Springer, 2007.

[38] I. Méndez-Díaz e P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006.

[39] G. L. Nemhauser e G. Sigismondi. A strong cutting plane/branch-and-bound algorithn for node packing. *The Journal of the Operational Research Society*, 43:443–457, 1992.

[40] G. L. Nemhauser e L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[41] T.F. Noronha e C.C. Ribeiro. Routing and wavelength assign by partition coloring. *European Journal of Operational Research*, 171(3):797–810, 2006.

[42] L. M. Rousseau, M. Gendreau, e D. Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668, 2007.

[43] M. W. P. Savelsbergh. A branch and price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

[44] P. Toth e D. Vigo. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2001.

[45] TSPLIB: a library of instances for the TSP and other related problems. `www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`.

[46] E. Uchoa e M. P. de Aragão. Integer program reformulation for robust branch-and-cut-and-price algorithms. In *Conference Mathematical Program in Rio: A Conference in Honour of Nelson Maculan*, pages 56–61, Búzios, november 2003.

[47] E. Uchoa, R. Fukasawa, J. Lysgaard, A. Pessoa, M.P. de Aragão, e D. Andrade. Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Mathematical Programming*, 112(2):443–472, 2007.

[48] F. Vanderbeck e L. A. Wolsey. An exact algorithm for ip column generation. *Operations Research Letters*, 19:151–159, 1996.

[49] L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.

Índice Remissivo