Instituto de Computação
Universidade Estadual de Campinas

Relaxações Lagrangianas e Planos de Corte Faciais na resolução de Problemas de Particionamento de Conjuntos

Andrei de A. S. Braga

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Andrei de A. S. Braga e aprovada pela Banca Examinadora.

Campinas, 2 de setembro de 2011.

Prof. Dr. Cid Carvalho de Souza (Orientador)

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

FICHA CATALOGRÁFICA ELABORADA POR ANA REGINA MACHADO – CRB8/5467
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA – UNICAMP

B73r

Braga, Andrei de Almeida Sampaio, 1986-
Relaxações Lagrangianas e planos de corte faciais na resolução de problemas de particionamento de conjuntos / Andrei de Almeida Sampaio Braga. – Campinas, SP : [s.n.], 2011.

Orientador: Cid Carvalho de Souza.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Programação inteira. 3. Algoritmos. 4. Particionamento de conjuntos (Matemática). I. Souza, Cid Carvalho de, 1963-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em inglês: Lagrangian relaxations and cutting planes in solving set partitioning problemas
Palavras-chave em inglês:
Combinatorial optimization
Integer programming
Algorithms
Set partitioning (Mathematics)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Cid Carvalho de Souza [Orientador]
Antônio Carlos Moretti
Flávio Keidi Miyazawa
Data da defesa: 02-09-2011
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 02 de setembro de 2011, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Antônio Carlos Moretti
IMECC / UNICAMP

Prof. Dr. . Flávio Keidi Miyazawa
IC / UNICAMP

Prof. Dr. Cid Carvalho de Souza
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Relaxações Lagrangianas e Planos de Corte Faciais na resolução de Problemas de Particionamento de Conjuntos

Andrei de A. S. Braga[1]

Setembro de 2011

Banca Examinadora:

- Prof. Dr. Cid Carvalho de Souza (Orientador)
- Prof. Dr. Antônio Carlos Moretti - IMECC/UNICAMP
- Prof. Dr. Flávio Keidi Miyazawa - IC/UNICAMP
- Prof. Dr. Vinícius Amaral Armentano (Suplente) - FEEC/UNICAMP
- Prof. Dr. Orlando Lee (Suplente) - IC/UNICAMP

[1]Suporte financeiro de: Bolsa FAPESP (processo 2008/03285-8) 2008–2011.

Resumo

O *problema de particionamento de conjuntos* (SPP, do inglês *set partitioning problem*) é considerado um dos problemas de otimização combinatória com mais vasta gama de aplicações. Para solucioná-lo, utilizam-se comumente métodos tradicionais para a resolução de problemas $\mathcal{NP}$-Difíceis. Nesta dissertação, estuda-se o uso da combinação de relaxação Lagrangiana com planos de corte.

Relaxação Lagrangiana é uma técnica que tem sido usada com bastante sucesso para atacar vários problemas $\mathcal{NP}$-Difíceis. Os algoritmos *relax-and-cut*, em especial, onde se adicionam dinamicamente planos de corte a relaxações Lagrangianas, têm ganhado bastante destaque nas últimas décadas. Em [15], Cavalcante *et al.* aplicam um algoritmo *relax-and-cut* ao SPP e obtêm ótimos resultados. No entanto, tal algoritmo, bem como implementações em geral da citada combinação, são ainda passíveis de refinamentos e extensões. O estudo proposto aqui é realizado por meio das seguintes extensões do referido algoritmo: a implementação de uma *partida quente* para o multiplicador de uma inequação adicionada; a incorporação do algoritmo a uma enumeração, gerando, assim, um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP; a implementação do citado *branch-and-cut* com o emprego de relaxações alternativas e a implementação de uma versão distribuída do algoritmo.

Abstract

The *set partitioning problem* (SPP) is considered one of the combinatorial optimization problems with the widest range of applications. To solve the SPP, one commonly uses traditional methods for $\mathcal{NP}$-Hard problem solving. In this dissertation, we study the use of the combination of Lagrangean relaxation with cutting planes.

Lagrangean relaxation is a technique that has been used quite successfully to tackle several $\mathcal{NP}$-Hard problems. In particular, *relax-and-cut* algorithms, in which cutting planes are added dynamically to Lagrangean relaxations, have gained much importance in the last decades. In [15], Cavalcante *et al.* applied a *relax-and-cut* algorithm to the SPP and obtained promising results. However, that algorithm, as well as implementations of the mentioned combination in general, are still subject to refinements and extensions. The study proposed here is carried out through the following extensions of that algorithm: the implementation of a *warm start* to the multiplier of an added inequality; the incorporation of the algorithm to an enumeration, thus generating a Lagrangean relaxation based *branch-and-cut* for the SPP; the implementation of that *branch-and-cut* with the use of alternative relaxations and the implementation of a distributed version of the algorithm.

Agradecimentos

À FAPESP, pelo suporte financeiro ao meu projeto de pesquisa.

Aos funcionários do Instituto de Computação da UNICAMP, pela disposição em ajudar e prontidão em atender solicitações, por mais difíceis ou incomuns que sejam.

Ao meus colegas de laboratório, pelas contribuições valiosas e conversas descontraídas.

Ao meu orientador, Prof. Dr. Cid Carvalho de Souza, pela disposição, paciência e confiança cruciais.

Aos meus amigos, fundamentais para as diversas superações que tive a alegria de viver.

A Alice Suéllen da Silva Cordovil, pela importância demonstrada em tão pouco tempo.

A meus pais, José Ueleres Braga e Erika de Almeida Sampaio Braga, pelo apoio simplesmente incondicional.

A Deus, por tudo.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Acrônimos

COP	*combinatorial optimization problem*, 17
DRC	*delayed relax-and-cut*, 36
GUB	*generalized upper bound*, 48
IP	*integer linear programming problem*, 7
LP	*linear programming problem*, 6
MP	*mathematical programming problem*, 5
MRP	*matriz de rede pura*, 79
MRRP	*matriz de rede refletida pura*, 79
MS	*método do subgradiente*, 37
MTU	*matriz totalmente unimodular*, 78
NDRC	*non-delayed relax-and-cut*, 36
NPP	*node packing problem*, 15
PDL	*problema dual Lagrangiano*, 11
PRL	*problema primal Lagrangiano*, 10
SCP	*set covering problem*, 1
SPP	*set partitioning problem*, 1
SSP	*stable set problem*, 2
TSP	*traveling salesman problem*, 1

Capítulo 1

Introdução

Dentre os problemas e estruturas especiais de programação inteira, três são os citados como os de mais vasta gama de aplicações: o *problema de particionamento de conjuntos* (SPP, do inglês *set partitioning problem*), o *problema de cobertura de conjuntos* (SCP, do inglês *set covering problem*) e o *problema do caixeiro viajante* (TSP, do inglês *traveling salesman problem*). Ainda, se o primeiro colocado fosse escolhido, este provavelmente seria o SPP [7].

Algumas aplicações do SPP são: escalonamento de frotas de veículos e de frotas e tripulações de aviões [13, 36, 39, 46], localização de facilidades [55], investimento financeiro [58], projeto de circuitos integrados [57], recuperação de informação [19], divisão de distritos políticos [25] e problemas de corte e empacotamento [53] (para uma bibliografia mais extensa, veja [7].) A mais recorrente dessas parece ser o escalonamento de tripulações de voo [7].

A grande aplicabilidade do SPP é alimentada pelos seguintes fatos. Para uma grande variedade de problemas de escalonamento, a modelagem abaixo é adequada.

(i) Um conjunto finito S,
(ii) um conjunto de restrições definindo uma família F de subconjuntos *válidos* de S e
(iii) um custo associado a cada subconjunto em F,

onde se busca escolher subconjuntos em F que constituam uma partição de S e que tenham custo mínimo. Em geral, problemas definidos desta maneira podem ser resolvidos com satisfatório grau de aproximação com os seguintes passos.

1. Usando as restrições em (ii), gere explicitamente uma família $\bar{F} \subset F$ com probabilidade suficientemente alta de conter uma solução ótima (uma solução é uma coleção de subconjuntos em F);
2. Substitua o conjunto de restrições em (ii) pelos elementos de $\bar{F}$ e resolva o problema resultante.

O modelo obtido é exatamente a caracterização do SPP apresentada a seguir.

1.1 O SPP, o SCP e o SSP

O SPP pode ser enunciado como abaixo. Sejam S um conjunto finito e $F = \{S_1, S_2, \ldots, S_k\}$ uma família de subconjuntos de S (F aqui equivale a $\bar{F}$ definida acima). Suponha que a cada subconjunto S_j em F esteja associado um custo c_j. Seja C uma coleção de subconjuntos em F. O custo de C é a soma dos custos associados aos subconjuntos presentes em C. Diz-se que C é uma partição de S se as seguintes condições são satisfeitas.

$$\bigcup_{j|S_j \in C} S_j = S \tag{1.1}$$

$$S_j \cap S_l = \emptyset, \ \forall S_j, S_l \in C, \ j \neq l \tag{1.2}$$

O SPP é o problema de encontrar C tal que C seja uma partição de S e tenha custo mínimo.

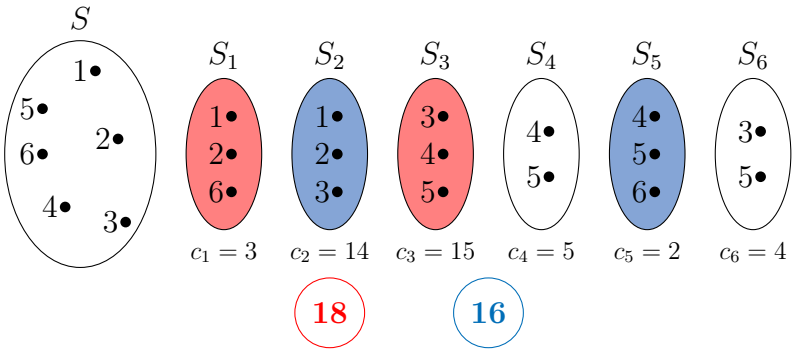


Figura 1.1: Instância do SPP.

Na Figura 1.1, ilustra-se uma instância do SPP. A coleção $C_1 = \{S_1, S_3\}$ é uma solução viável para o problema, pois constitui uma partição de S. Observe que $S_1 \cup S_3 = S$ e que $S_1 \cap S_3 = \emptyset$. Tal solução, entretanto, não é ótima. O motivo é que existe uma outra coleção, a saber, $C_2 = \{S_2, S_5\}$, que também é uma partição de S e tem menor custo. Os custos das coleções C_1 e C_2 são, respectivamente, $18 = 3 + 15$ e $16 = 14 + 2$. A última, por sua vez, é uma solução ótima para o problema, pois tem custo mínimo.

No escalonamento de tripulações de voo, uma das citadas aplicações do SPP, S representa um conjunto de viagens – da cidade X para a cidade Y no tempo t. Cada S_j em F simboliza uma jornada – sequência de viagens com pontos inicial e final iguais ($X \rightarrow Y$, $Y \rightarrow Z$, $Z \rightarrow X$) – a ser cumprida por uma tripulação. Uma solução viável para o problema é uma coleção C de jornadas onde cada viagem está presente em exatamente uma jornada.

O SPP está, ainda, intimamente relacionado a dois outros problemas: o SCP e o *problema de empacotamento de conjuntos* (SSP, do inglês *stable set problem* – esta nomenclatura será esclarecida posteriormente). O SCP e o SSP podem ser definidos de maneira semelhante à do SPP. Tomando a descrição do SPP dada acima, a diferença é que, no SCP, exige-se, quanto à coleção C, somente que a mesma constitua uma cobertura de S, isto é, que a condição (1.1) seja satisfeita. Para o SSP, as mudanças são duas. Primeiramente, requere-se apenas que C represente um empacotamento de S, isto é, que a condição (1.2) seja contemplada. Em segundo lugar, uma solução ótima deve ter custo máximo, ao invés de mínimo. Pode-se ver, então, que uma partição de um conjunto ocorre quando se tem uma cobertura e um empacotamento do referido conjunto ao mesmo tempo.

Na ilustração da Figura 1.1, se o problema considerado fosse o SCP, a coleção $C_3 = \{S_1, S_5, S_6\}$, com custo 9, seria uma solução ótima (observe que esse custo é menor que o de uma solução ótima para o SPP). Já se o SSP fosse abordado, a coleção $C_4 = \{S_2, S_4\}$, com custo 19, resolveria a questão.

1.2 Combinação de relaxação Lagrangiana com planos de corte na resolução do SPP

Relaxação Lagrangiana é uma técnica usada com bastante sucesso para atacar vários problemas difíceis. Como posto por Beasley [9], há duas razões básicas para esse êxito. A primeira é que problemas difíceis são comumente constituídos por outros fáceis acrescidos de restrições *complicadoras*. Essa situação é adequada para a referida técnica. Com a mesma, pode-se *dualizar* tais restrições e obter um problema de pouca complexidade computacional para ser resolvido (ainda que por várias vezes). A segunda razão é que os métodos utilizados para solucionar uma relaxação Lagrangiana mostram-se muito eficientes na prática.

Em [15], Cavalcante *et al.* aplicam um algoritmo *relax-and-cut* ao SPP e obtêm ótimos resultados. Tal algoritmo adiciona dinamicamente inequações válidas (ou planos de corte) a uma relaxação Lagrangiana visando à melhoria de qualidade dos limitantes atingidos. Uma constatação importante é que o mesmo é, apesar de bastante efetivo, passível de extensões. Por exemplo, outras opções de relaxação Lagrangiana podem ser implementadas. A gerada em [15] é a tradicionalmente empregada para o SPP, onde todas as restrições são dualizadas. É possível identificar um subconjunto de restrições que represente um problema fácil e proceder como na primeira razão de êxito comentada acima. Outra modificação interessante é tornar o algoritmo desenvolvido um método *exato*; e não mais heurístico.

Nesta dissertação, estuda-se a combinação de relaxação Lagrangiana com planos de corte na resolução do SPP. Isto é realizado por meio de extensões, como as supracitadas, do mencionado algoritmo *relax-and-cut*. Para tanto, tais extensões são propostas, implementadas e avaliadas e os resultados obtidos são discutidos.

As principais atividades realizadas neste trabalho são:

1. um estudo sobre o impacto de uma *partida quente* para o multiplicador de uma inequação adicionada em um algoritmo *relax-and-cut*;
2. a implementação e a avaliação de um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP;
3. a implementação e a avaliação de um *branch-and-cut* para o SPP baseado em relaxações alternativas e
4. um estudo sobre a cooperação entre algoritmos *relax-and-cut* para o SPP.

O texto a seguir está organizado em seis capítulos. No Capítulo 2, são abordados os conceitos básicos necessários para o entendimento deste trabalho. No Capítulo 3, são descritos algoritmos para atacar um problema de programação linear inteira. Tal problema é a principal forma de modelar o SPP, bem como problemas $\mathcal{NP}$-Difíceis em geral. Os algoritmos discutidos nesse capítulo compõem os métodos desenvolvidos na presente dissertação. No Capítulo 4, apresenta-se o algoritmo proposto por Cavalcante *et al.* [15], a referência principal para este trabalho. Tal capítulo compreende, também, as duas primeiras atividades listadas acima. Os Capítulos 5 e 6 abrangem, respectivamente, as atividades 3 e 4. Por fim, no Capítulo 7, trazem-se as conclusões obtidas.

Capítulo 2

Conceitos básicos

Neste capítulo, apresentam-se conceitos básicos necessários para o entendimento do trabalho desenvolvido nesta dissertação. Para um boa compreensão do mesmo, espera-se adicionalmente, contudo, que o leitor seja familiar aos temas principais da Ciência da Computação, mais especificamente aos relativos ao campo da Otimização. Teoria de Complexidade Computacional e Grafos são, em especial, dois assuntos dos quais se supõe um conhecimento prévio do leitor.

Recomenda-se que a leitura deste capítulo seja realizada da seguinte forma. Ao leitor que já domina os conceitos discutidos, que esta seja rápida e superficial (sendo mais cuidadosa em possíveis assuntos a relembrar). Ao leitor que não o faz, que esta seja mais atenciosa e que sirva de um guia para futuras consultas. A ambos, que esta tenha o papel de introduzir a notação usada na presente dissertação.

A apresentação feita a seguir é baseada em um conjunto de livros-texto, artigos e teses que representam referências consolidadas para os temas presentes neste trabalho. Tal apresentação é considerada suficiente, no sentido de abranger todos os temas. A mesma certamente não substitui, porém, uma exposição completa e aprofundada sobre os assuntos tratados. Indica-se, portanto, de imediato, as referências que a originaram, separadas por seções e suas divisões. Desta forma, dá-se ao leitor os apontamentos necessários. A Seção 2.1 é baseada nos livros-texto [26] e [49]. A Subseção 2.2.1 fundamenta-se nos artigos [32] e [9]. Para a Subseção 2.2.2, as referências são [59], [49] e [7]. Finalmente, veja [13] e [49], [13] e [7] para um tratamento mais aprofundado sobre as Subseções 2.3.1, 2.3.2 e 2.3.3 respectivamente. Sempre que adequado, são ainda citadas referências específicas ao longo do texto.

As proposições estabelecidas neste capítulo são todas enunciadas sem prova. Tais proposições são, quando não bastante básicas, amplamente conhecidas e discutidas na literatura. Para conhecer provas para as mesmas, o leitor é convidado a explorar as já mencionadas referências.

2.1 Programação linear e programação linear inteira

Um *problema de programação matemática* (MP, do inglês *mathematical programming problem*) pode ser definido como

$$\begin{aligned} z = \min\ & f(x) \\ & x \in S \subseteq \mathbb{R}^n. \end{aligned}$$

Neste modelo, a função $f : S \mapsto \mathbb{R}$ é denominada *função objetivo* e o conjunto S é dito *conjunto de soluções viáveis* ou, por simplicidade, *conjunto de soluções*. Uma *solução viável* para o problema é um vetor $x \in \mathbb{R}^n$ tal que $x \in S$. Uma solução viável x^* tal que $f(x^*) \leq f(x), \forall x \in S$, é uma *solução ótima*. O objetivo, em um MP, é encontrar uma ou mais soluções ótimas, determinando, consequentemente, o *valor ótimo* z.

Um *problema de programação linear* (LP, do inglês *linear programming problem*) é uma instância de um MP que pode ser descrita como

$$\begin{aligned} z = \min\ & cx \\ & Ax \leq b \\ & x \in \mathbb{R}^n_+. \end{aligned} \tag{2.1}$$

Neste modelo, A é uma matriz $m \times n$, b é um vetor $m \times 1$ e c é um vetor $1 \times n$, todos com entradas racionais. Por consequência, x é um vetor $n \times 1$. A matriz A é denominada *matriz de restrições*. A entrada c_j do vetor c representa o *custo da variável* x_j. O *custo* ou *valor* de uma solução x é dado por cx ou $\sum_j c_j x_j$. O conjunto $S = \{x \in \mathbb{R}^n_+, Ax \leq b\}$ é um conjunto em $\mathbb{R}^n$ determinado por um número finito de inequações lineares. Logo, S constitui um *poliedro*. Ainda, por ser um poliedro, S é um *conjunto convexo*, ou seja, $x_1, x_2 \in S \Rightarrow \alpha x_1 + (1-\alpha)x_2 \in S,\ \forall\ \alpha \in \mathbb{R},\ 0 \leq \alpha \leq 1$.

Dado um LP, uma das três seguintes situações ocorre.

1. Não há solução viável para o problema, ou seja, $\nexists\ x \in \mathbb{R}^n_+$ tal que $Ax \leq b$. O problema é *inviável*.
2. Para todo valor $k \in \mathbb{R}$, existe $x \in \mathbb{R}^n_+$ tal que $Ax \leq b$ e $cx < k$. Logo, $z = -\infty$. O problema é *viável* e *ilimitado*.
3. Existe uma solução ótima (e viável) x^* para o problema com $cx^* > -\infty$. O problema é *viável* e *limitado*.

Para solucionar um LP, o algoritmo mais utilizado é o *algoritmo do simplex*, pois este se mostra muito eficiente na prática. Tal algoritmo, contudo, possui um limitante superior exponencial no tamanho da entrada para o número máximo de iterações executadas. No entanto, para resolver um LP, outros algoritmos, de tempo polinomial, também são conhecidos. O *algoritmo de projeção* e o *algoritmo do elipsóide* são exemplos (para uma descrição dos mesmos, veja [49]). Pela existência dos últimos, tem-se o seguinte resultado.

Proposição 2.1. *Um LP é resolvível em tempo polinomial em m, n e $\log \theta_A$, sendo θ_A o maior valor absoluto de uma entrada da matriz A.*

Um *problema de programação linear inteira* (IP, do inglês *integer linear programming problem*) é uma instância de um MP que pode ser definida de modo semelhante ao de um LP. Um IP pode ser modelado como em (2.1) com uma exceção: as variáveis são inteiras. O modelo segue.

$$\begin{aligned} z = \min\ & cx \\ & Ax \leq b \\ & x \in \mathbb{Z}^n_+ \end{aligned} \tag{2.2}$$

Como para um LP, três situações são possíveis para um IP. De forma equivalente, o problema é inviável, viável e ilimitado ou viável e limitado.

Um IP é, em geral, um problema mais difícil que um LP. Em termos da Teoria de Complexidade Computacional [24], um LP genérico pertence à classe de problemas $\mathcal{P}$, enquanto um IP genérico é um problema $\mathcal{NP}$-Completo. Acredita-se que a classe $\mathcal{P}$ seja um subconjunto próprio da classe $\mathcal{NP}$, principalmente por ser provável que os problemas considerados os mais difíceis presentes na última, os problemas $\mathcal{NP}$-Completos, não pertençam à primeira. Para os problemas $\mathcal{NP}$-Completos, não se conhece um algoritmo de tempo polinomial no tamanho da entrada. Assim, para um IP genérico, tem-se o seguinte resultado.

Proposição 2.2. *Não se conhece, para um IP genérico, um algoritmo de tempo polinomial em m, n e $\log \theta_A$, sendo θ_A o maior valor absoluto de uma entrada da matriz A.*

Um caso especial bastante importante de um IP é um problema de programação linear inteira $0-1$ (IP $0-1$). Um IP $0-1$ é uma instância de um IP onde as variáveis são binárias. Apesar de mais restrito, esse problema é, em geral, tão difícil quanto o anterior: um IP $0-1$ genérico é um problema $\mathcal{NP}$-Completo. Logo, tem-se, também, o seguinte resultado.

Proposição 2.3. *Não se conhece, para um IP $0-1$ genérico, um algoritmo de tempo polinomial em m, n e $\log \theta_A$, sendo θ_A o maior valor absoluto de uma entrada da matriz A.*

Em teoria, pode-se resolver um IP por meio de um LP. O conceito de envoltória convexa de um conjunto, apresentado a seguir, é o fundamento para tal estratégia.

Definição 2.1. O ponto x é uma *combinação convexa* dos pontos $x_1, x_2, \ldots, x_k \in \mathbb{R}^n$ se existem $\alpha_i \in \mathbb{R}_+,\ i = 1, \ldots, k$, com $\sum_{i=1}^k \alpha_i = 1$ e tais que $x = \sum_{i=1}^k \alpha_i x_i$. Tem-se uma *combinação convexa estrita* se $\alpha_i > 0,\ i = 1, \ldots, k$.

Definição 2.2. A *envoltória convexa* de um conjunto $S \subseteq \mathbb{R}^n$, denotada por $conv(S)$, é o conjunto de todos os pontos que são combinação convexa de pontos em S, ou seja, $conv(S) = \{x : x = \sum_{i=1}^{k} \alpha_i x_i,\ x_i \in S,\ \alpha_i \in \mathbb{R}_+,\ i = 1, \ldots, k,\ \sum_{i=1}^{k} \alpha_i = 1\}$.

Proposição 2.4. *Dado um conjunto $S \subseteq \mathbb{R}^n$, $conv(S)$ é um poliedro.*

Considere um IP como definido em (2.2) e denote por S o conjunto de soluções do mesmo. O conjunto $conv(S)$ é um poliedro e pode, portanto, ser determinado por um número finito de inequações lineares. Suponha, então, que $conv(S) = \{x \in \mathbb{R}^n_+, A'x \leq b'\}$. Ao substituir, em (2.2), $(Ax \leq b, x \in \mathbb{Z}^n_+)$ por $(A'x \leq b', x \in \mathbb{R}^n_+)$, obtém-se um LP por meio do qual o IP pode ser resolvido. O mesmo segue.

$$\begin{aligned} z = \min\ & cx \\ & A'x \leq b' \\ & x \in \mathbb{R}^n_+ \end{aligned} \tag{2.3}$$

O resultado enunciado acima é formalizado na proposição a seguir.

Proposição 2.5. *Seja $S = \{x \in \mathbb{Z}^n_+, Ax \leq b\}$ e $conv(S) = \{x \in \mathbb{R}^n_+, A'x \leq b'\}$. Então:*

1. *Se o LP (2.3) é inviável, o IP (2.2) é inviável.*
2. *Se o LP (2.3) é viável e ilimitado, o IP (2.2) é viável e ilimitado.*
3. *Se existe uma solução ótima para o LP (2.3), existe uma solução ótima para o mesmo que também é uma solução ótima para o IP (2.2) (sendo os valores ótimos iguais para os dois problemas).*

O principal embasamento da Proposição 2.5 é a propriedade do LP (2.3) posta abaixo.

Definição 2.3. Dado um poliedro S, um ponto $x \in S$ é um *ponto extremo* de S se não existem dois pontos $x_1, x_2 \in S$ com $x_1 \neq x_2$ tais que x seja combinação convexa estrita de x_1 e x_2.

Proposição 2.6. *Dados um poliedro S e um LP definido por $\min\{cx : x \in S\}$, se existe uma solução ótima para o LP, existe uma solução ótima para o LP que é um ponto extremo de S.*

Proposição 2.7. *Dado um conjunto $S \subseteq \mathbb{R}^n$, os pontos extremos de $conv(S)$ pertencem a S.*

Pela Proposição 2.6, se há uma solução ótima para o LP (2.3), há uma solução ótima para o mesmo que é um ponto extremo de $conv(S)$ – sendo S o conjunto de soluções do IP (2.2). Pela Proposição 2.7, esta solução ótima pertence a S. Assim, se existe uma solução

ótima para o LP (2.3), existe uma solução ótima x^* para o mesmo que é viável para o IP (2.2). Ao analisar os custos de soluções para ambos os problemas, é fácil concluir que x^* é também ótima para o IP (2.2).

A estratégia de resolver um IP por meio do LP (2.3) é, no entanto, em geral, impraticável. Isto, pois ou um conjunto $A'x \leq b'$ que determine $conv(S)$ não é conhecido ou este conjunto possui um número exponencialmente grande de inequações.

2.2 Relaxações

Pela Proposição 2.1, apresentada na Seção 2.1, sabe-se que um LP é um problema de fácil resolução, ou seja, para o qual se tem um algoritmo de tempo polinomial no tamanho da entrada. Ainda, pela Proposição 2.2, estabelecida na mesma seção, sabe-se que, para solucionar um IP, não se conhece, em geral, um algoritmo com tal propriedade. No entanto, uma vasta gama de problemas de grandes interesse prático e relevância teórica são descritos como IPs. Assim, é bastante importante dispor, também, de boas estratégias para resolver esses modelos.

Uma estratégia comumente usada para atacar um IP é a seguinte: não se resolve o problema diretamente, e sim uma relaxação do mesmo. Resolvendo uma relaxação de um IP, obtém-se um *limitante dual* $\underline{z}$ para o valor ótimo z. Como, nesta apresentação, um IP é definido por uma minimização, um limitante dual representa, aqui, um limitante inferior. Quando um limitante dual aplica-se a um problema descrito por uma maximização, esse representa um limitante superior. O intuito é que, por meio de tal limitante, consiga-se provar que uma solução viável x' conhecida para o IP é também ótima.

O resultado acima é conseguido quando $cx' = \underline{z}$. De fato, se isso ocorre, então, para toda solução viável x para o IP, tem-se que $cx \geq z \geq \underline{z} = cx'$. Logo, x' é uma solução ótima para o IP.

Uma relaxação de um IP pode ser formalmente definida como abaixo. Considere que o problema P é um IP modelado como

$$\begin{aligned} z = \min\ & cx \\ & x \in S \subseteq \mathbb{Z}^n_+. \end{aligned} \tag{2.4}$$

Uma *relaxação* de P é um problema dado por

$$\begin{aligned} z_R = \min\ & f(x) \\ & x \in T \subseteq \mathbb{R}^n_+ \end{aligned} \tag{2.5}$$

onde $T \supseteq S$ e $f(x) \leq cx, \forall x \in S$.

Proposição 2.8. *Tem-se que* $z_R \leq z$.

Duas relaxações são bastante conhecidas e utilizadas na resolução de IPs. Essas são as relaxações presentes nas análises feitas nesta dissertação. As mesmas são descritas a seguir.

2.2.1 Relaxação linear e relaxação Lagrangiana

Considere que P é um IP como definido em (2.2). O LP dado por (2.1) é a *relaxação linear* de P. A relaxação linear de um IP consiste, então, em desprezar a restrição de que as variáveis sejam inteiras.

É possível ver que a relaxação linear é, de fato, ou seja, nos termos dos modelos (2.4) e (2.5), uma relaxação. Para os problemas (2.2) e (2.1), tem-se que $S = \{x \in \mathbb{Z}^n_+, Ax \leq b\}$, $f(x) = cx$ e $T = \{x \in \mathbb{R}^n_+, Ax \leq b\}$. Claramente, ocorre $T \supseteq S$ e $f(x) \leq cx, \forall x \in S$.

Considere, agora, que as restrições de P são dadas por dois grupos de inequações: $Ax \leq b$ e $Cx \leq d$. O problema é, portanto, definido como

$$\begin{aligned} z = \min\ & cx \\ & Ax \leq b \\ & Cx \leq d \\ & x \in \mathbb{Z}^n_+. \end{aligned} \tag{2.6}$$

Suponha, ainda, que, sem as restrições $Ax \leq b$, P possa ser resolvido facilmente, ou seja, que haja, para o mesmo, um algoritmo de tempo polinomial. Assim, o IP

$$\begin{aligned} z(\lambda) = \min\ & cx + \lambda(Ax - b) \\ & Cx \leq d \\ & x \in \mathbb{Z}^n_+ \end{aligned} \tag{2.7}$$

onde $\lambda \geq 0$ constitui uma relaxação de P e representa um problema fácil. Esta é uma *relaxação Lagrangiana* de P. Note que, no IP acima, λ é constante. Apenas x é variável.

Uma relaxação Lagrangiana de um IP consiste, então, em retirar desigualdades do conjunto de restrições e adicioná-las à função objetivo, com respectivos *multiplicadores* (de Lagrange). As desigualdades retiradas do conjunto de restrições são ditas *dualizadas*. O problema resultante é denominado *problema primal Lagrangiano* (PRL). No PRL, os custos das variáveis são chamados de *custos Lagrangianos*. No PRL (2.7), os custos Lagrangianos são dados pelo vetor $(c + \lambda A)$. O mesmo pode ser reescrito como

$$\begin{aligned} z(\lambda) = \min\ & (c + \lambda A)x - \lambda b \\ & Cx \leq d \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

onde $\lambda \geq 0$.

Como para a relaxação linear, pode-se ver que uma relaxação Lagrangiana satisfaz à definição de relaxação dada anteriormente. Associando os problemas (2.6) e (2.7) aos modelos (2.4) e (2.5), tem-se que $S = \{x \in \mathbb{Z}_+^n, Ax \leq b, Cx \leq d\}$, $f(x) = cx + \lambda(Ax - b)$ e $T = \{x \in \mathbb{Z}_+^n, Cx \leq d\}$. De imediato, conclui-se que $T \supseteq S$. Observando que $\lambda \geq 0$ e que $(Ax - b) \leq 0, \forall x \in S$, nota-se, também, que $f(x) \leq cx, \forall x \in S$.

Em uma relaxação Lagrangiana de um IP, a ideia é que as desigualdades *complicadoras* sejam retiradas do conjunto de restrições, mas não totalmente desprezadas. No PRL, essas desigualdades não precisam mais ser satisfeitas. Violá-las, porém, tem um custo. Observe que, em (2.7), se $\lambda > 0$ e uma solução x' não satisfaz $Ax \leq b$, ou seja, $Ax' > b$ ou ainda $(Ax' - b) > 0$, um termo positivo, a saber, $\lambda(Ax' - b)$, é adicionado à função objetivo, o que é ruim para a minimização. Já se x' respeita $Ax \leq b$, soma-se uma parcela não positiva, a saber, $\lambda(Ax' - b) \leq 0$, o que não aumenta o valor da solução.

Para qualquer vetor de multiplicadores $\lambda \geq 0$, o valor ótimo $z(\lambda)$ de uma relaxação Lagrangiana é um limitante inferior para z. O melhor limitante inferior possível é, então, obtido ao resolver o problema

$$z_D = \max_{\lambda \geq 0} \{z(\lambda)\}.$$

Este é denominado *problema dual Lagrangiano* (PDL).

Uma questão importante é determinar o quão preciso é o limitante produzido pelo PDL. Em [28], Geoffrion mostra que, para a relaxação (2.7) de P,

$$z_D = \min\{cx : Ax \leq b, x \in conv(\{x \in \mathbb{Z}_+^n, Cx \leq d\})\}. \tag{2.8}$$

Comparando o conjunto de restrições da relaxação linear de P com o estabelecido acima, nota-se que o último é mais restrito. Isto, pois

$$\{Ax \leq b, x \in conv(\{x \in \mathbb{Z}_+^n, Cx \leq d\})\} \subseteq \{Ax \leq b, Cx \leq d, x \in \mathbb{R}_+^n\}.$$

Como consequência, tem-se que

$$\begin{aligned} \min \{cx : Ax \leq b, x \in conv(\{x \in \mathbb{Z}_+^n, Cx \leq d\})\} \geq \\ \min\{cx : Ax \leq b, Cx \leq d, x \in \mathbb{R}_+^n\}. \end{aligned} \tag{2.9}$$

Por este resultado, sabe-se que, para um dado IP, o melhor limitante inferior atingido com uma relaxação Lagrangiana é maior ou igual ao alcançado com a relaxação linear. Quando a relaxação Lagrangiana possui a *propriedade da integralidade* [32], os limitantes conseguidos são iguais. Sendo a mesma dada por (2.7), essa propriedade implica que $conv(\{x \in \mathbb{Z}_+^n, Cx \leq d\}) = \{x \in \mathbb{R}_+^n, Cx \leq d\}$.

2.2.2 Refinamento de uma relaxação

Com a Proposição 2.5, viu-se que um IP pode, em teoria, ser resolvido por meio de um LP. Este LP tem a função objetivo dada pela mesma função objetivo do IP e o conjunto de soluções viáveis dado pela envoltória convexa do conjunto das soluções viáveis do IP. Tal LP é, portanto, uma relaxação do IP. Mais ainda, esse problema é uma relaxação "ideal" do IP. Isto, no sentido em que o mesmo produz garantidamente uma solução ótima para o problema original.

Para atacar um dado IP, usam-se, frequentemente, várias opções de relaxação. Dentre essas, pode-se estabelecer que, em um dado sentido, umas são mais vantajosas que outras. Uma relaxação é mais vantajosa quando, por exemplo, produz um limitante mais preciso ou representa um problema de mais fácil resolução. Na subseção anterior, a primeira situação foi ilustrada. Na mesma, foi visto, que, para um IP, o melhor limitante produzido por uma relaxação Lagrangiana é maior ou igual ao gerado pela relaxação linear.

A citada vantagem da relaxação Lagrangiana advém de a formulação presente no lado esquerdo da inequação (2.9), onde se tem um valor ótimo igual ao do PDL associado à relaxação (2.7), ser melhor que a formulação presente no lado direito da mesma. Apesar de posto em (2.9), esse resultado, bem como o conceito de melhor formulação, ainda não foram formalizados. Os mesmos são apresentados abaixo.

Definição 2.4. Um poliedro $T \subseteq \mathbb{R}^n$ é uma *formulação* para um conjunto $S \subseteq \mathbb{Z}^n$ se $S = T \cap \mathbb{Z}^n$.

Definição 2.5. Dadas duas formulações T_1 e T_2 para um conjunto $S \subseteq \mathbb{Z}^n$, tem-se que T_1 é uma *formulação melhor* ou *mais forte* que T_2 se $T_1 \subset T_2$.

Proposição 2.9. *Sejam T_1 e T_2 duas formulações para um conjunto $S \subseteq \mathbb{Z}^n$ e RP_1 dada por $z_{RP_1} = \min\{f(x) : x \in T_1\}$ e RP_2 dada por $z_{RP_2} = \min\{f(x) : x \in T_2\}$ duas relaxações de um IP dado por $\min\{cx : x \in S\}$. Se T_1 é uma formulação melhor ou mais forte que T_2, então $z_{RP_1} \geq z_{RP_2}$.*

O conceito apresentado na Definição 2.5 vai ao encontro da afirmação posta acima: o LP por meio do qual se pode resolver um IP é uma relaxação "ideal" do mesmo. O motivo é que a formulação usada nessa relaxação é a melhor possível. Isto, pois, sendo S o conjunto de soluções viáveis do IP, tem-se que $conv(S) \subseteq T$, para toda formulação T para S.

Refinar uma relaxação de um IP significa torná-la uma relaxação mais vantajosa em algum sentido. Isto, é claro, sem implicar desvantagens. Dada uma relaxação de um IP, é possível, então, em geral, refiná-la. Um modo de tornar o limitante de uma relaxação mais preciso é fortificar a formulação presente na mesma. Um mecanismo usado para fortificar uma formulação é a adição de inequações válidas. Tal mecanismo é descrito adiante. Para tanto, é preciso, contudo, apresentar primeiramente alguns conceitos de Teoria Poliédrica.

Conceitos de Teoria Poliédrica

Definição 2.6. Os pontos $x_1, x_2, \ldots, x_k \in \mathbb{R}^n$ são *afim independentes* se a única solução para a equação $\sum_{i=1}^{k} \alpha_i x_i = 0$ com $\sum_{i=1}^{k} \alpha_i = 0$ é $\alpha_i = 0, i = 1, \ldots, k$.

Definição 2.7. A *dimensão* de um poliedro S, denotada por $dim(S)$, é igual a k, ou seja, $dim(S) = k$, se o número máximo de pontos afim independentes em S é igual a $k + 1$.

Em um poliedro $S \subseteq \mathbb{R}^n$, há no máximo $n+1$ pontos afim independentes. A definição a seguir caracteriza tal situação.

Definição 2.8. Um poliedro $S \subseteq \mathbb{R}^n$ é de *dimensão cheia* se $dim(S) = n$.

Definição 2.9. Uma inequação $\pi x \leq \pi_0$ é uma *inequação válida* para um poliedro S se $\forall x \in S, \pi x \leq \pi_0$.

Uma inequação válida para um poliedro determina um conjunto de pontos do mesmo que a satisfazem na igualdade. Esse conjunto é definido abaixo.

Definição 2.10. Uma *face* de um poliedro S é um conjunto de pontos $F = \{x \in S, \pi x = \pi_0\}$ onde $\pi x \leq \pi_0$ é uma inequação válida para S. Uma face F de um poliedro S é uma *face própria* se $F \neq \emptyset$ e $F \neq S$.

Note que uma equação pode ser reescrita como duas inequações. Com esta observação, vê-se que uma face de um poliedro é um conjunto de pontos em $\mathbb{R}^n$ determinado por um número finito de inequações lineares. Logo, uma face é também, por sua vez, um poliedro. Uma classe de faces, em especial, é de grande importância. A mesma segue.

Definição 2.11. Uma *faceta* de um poliedro S é uma face própria F de S tal que $dim(F) = dim(S) - 1$.

Dado um poliedro S, há infinitas maneiras de representá-lo por um conjunto de inequações lineares. Em uma descrição do mesmo, podem-se sempre adicionar novas desigualdades válidas e é possível, comumente, eliminar restrições redundantes. Tomando, então, uma representação $S = \{x \in \mathbb{R}^n, Ax \leq b\}$, uma questão interessante é: Dentre as inequações em $Ax \leq b$, quais são, de fato, necessárias para a descrição de S? O resultado abaixo mostra que as inequações que definem facetas de S o são.

Proposição 2.10. *Para cada faceta F de um poliedro S, uma inequação que defina F é necessária para a descrição de S.*

Além de necessárias, as desigualdades definidoras de facetas são suficientes para a descrição de S. Isto é explicitado na seguinte proposição.

Proposição 2.11. *Toda inequação que define uma face F de um poliedro S tal que $dim(F) < dim(S) - 1$ é redundante para a descrição de S.*

Adição de inequações válidas

Acima, foi definido o conceito de inequação válida para um poliedro. Para aplicar esse conceito ao conjunto de soluções de um IP, é necessária uma definição mais abrangente. A mesma segue.

Definição 2.12. Uma inequação $\pi x \leq \pi_0$ é uma *inequação válida* para um conjunto $S \subseteq \mathbb{R}^n$ se $\forall x \in S, \pi x \leq \pi_0$.

Ainda, para fortificar uma formulação para um conjunto de soluções de um IP, a seguinte relação é essencial.

Proposição 2.12. *Se uma inequação é válida para um conjunto $S \subseteq \mathbb{R}^n$, a mesma é também válida para $conv(S)$.*

Ao longo deste texto, uma inequação é, por simplicidade, comumente dita válida para um dado IP. Tal inequação deve ser entendida como válida para o conjunto de soluções S do IP. Por consequência, válida também para $conv(S)$.

Dadas inequações válidas para um conjunto $S \subseteq \mathbb{R}^n$, é possível estabelecer relações de dominância e equivalência entre as mesmas. Tais relações estão descritas a seguir.

Definição 2.13. Uma inequação $\pi x \leq \pi_0$ *equivale* a uma inequação $\gamma x \leq \gamma_0$, ambas válidas para um conjunto $S \subseteq \mathbb{R}^n$, se existe $\alpha > 0$ tal que $\pi = \alpha\gamma$ e $\pi_0 = \alpha\gamma_0$.

Definição 2.14. Uma inequação $\pi x \leq \pi_0$ *domina* ou é *mais forte* que uma inequação $\gamma x \leq \gamma_0$, ambas válidas para um conjunto $S \subseteq \mathbb{R}^n$, se as mesmas não são equivalentes e se existe $\alpha > 0$ tal que $\pi \geq \alpha\gamma$ e $\pi_0 \leq \alpha\gamma_0$. A inequação $\gamma x \leq \gamma_0$ é dita *mais fraca* ou *dominada* pela inequação $\pi x \leq \pi_0$.

Proposição 2.13. *Dado um conjunto $S \subseteq \mathbb{R}^n_+$, se uma inequação $\pi x \leq \pi_0$ domina ou é mais forte que uma inequação $\gamma x \leq \gamma_0$, ambas válidas para S, tem-se que $\{x \in S, \pi x \leq \pi_0\} \subset \{x \in S, \gamma x \leq \gamma_0\}$.*

Dada uma formulação presente em uma relaxação de um IP, é possível fortificar a mesma através da adição de inequações válidas. Tal processo ocorre tipicamente da seguinte maneira. Abaixo, considere o IP definido pelo problema P (2.4) e a relaxação do mesmo definida por RP (2.5). Suponha que $S = \{x \in \mathbb{Z}^n_+, Ax \leq b\}$ e que T é uma formulação para S tal que $T = \{x \in \mathbb{R}^n_+, Cx \leq d\}$.

1. Sendo x' um dado ponto em $T \setminus S$, tome uma inequação $\pi x \leq \pi_0$ válida para S tal que $\pi x' > \pi_0$.

2. Adicione $\pi x \leq \pi_0$ às restrições de RP e obtenha RP: $\min\{f(x) : Cx \leq d, \pi x \leq \pi_0, x \in \mathbb{R}^n_+\}$.

Note que $\{x \in \mathbb{R}^n_+, Cx \leq d, \pi x \leq \pi_0\} \subset \{x \in \mathbb{R}^n_+, Cx \leq d\}$.

Como posto antes, a melhor formulação possível para S é dada por $conv(S)$. Fortificar a formulação de uma relaxação de P significa, então, no fundo, tornar a mesma mais próxima de $conv(S)$. Observe que, pela Proposição 2.12, ao adicionar uma inequação válida para S no item 2, adiciona-se uma inequação válida para $conv(S)$, o que é coerente com a afirmação acima.

Idealmente, a inequação $\pi x \leq \pi_0$ adicionada às restrições de RP deve definir uma faceta de $conv(S)$. O motivo é que, pelas Proposições 2.10 e 2.11, as desigualdades desse tipo são necessárias e suficientes para a descrição de $conv(S)$. Em geral, quanto mais forte é a inequação adicionada, melhor. Isto, pois, pela Proposição 2.13, mais forte é a formulação resultante. Para a descrição de $conv(S)$, uma inequação dominante torna todas as outras dominadas redundantes. Uma inequação válida que define uma faceta de $conv(S)$ não é dominada por nenhuma outra inequação válida para esse conjunto.

O exemplo a seguir ilustra a fortificação da formulação de uma relaxação com a adição de inequações válidas. Como consequência, um limitante mais preciso é obtido com a relaxação refinada.

Exemplo 2.1. O *problema de empacotamento de vértices* (NPP, do inglês *node packing problem*) em um grafo é o problema de encontrar um conjunto de vértices do grafo que constitua um empacotamento e tenha cardinalidade máxima. O NPP pode ser modelado como um IP da seguinte maneira. Seja A_E a matriz de incidência aresta-vértice do grafo. O NPP é dado por

$$\begin{aligned} z_{NPP} = \max\ & \mathbf{1}x \\ & A_E x \leq \mathbf{1} \\ & x \in \mathbb{B}^n \end{aligned} \tag{2.10}$$

onde $\mathbf{1}$ é um vetor de 1's.

No modelo acima, uma variável está associada a um vértice do grafo. Com as restrições $A_E x \leq 1$, diz-se que, para cada aresta, no máximo um vértice incidente à mesma pode estar em uma solução viável. Logo, uma solução viável constitui exatamente um empacotamento de vértices. Também, quanto maior é o número de vértices presentes em uma solução, maior é o seu custo. Assim, uma solução ótima é um empacotamento de cardinalidade máxima.

Mostra-se, a seguir, como a formulação da relaxação linear do NPP pode ser fortificada com a adição de inequações válidas. Antes, contudo, algumas definições de estruturas em grafos são necessárias.

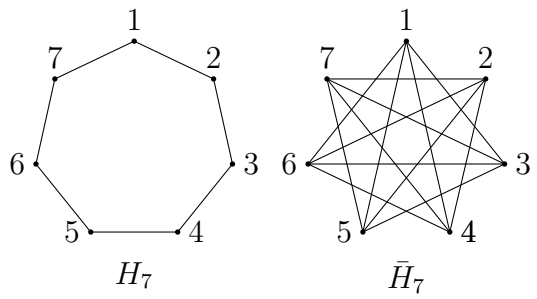
Definição 2.15. Um *ciclo* em um grafo é um conjunto de vértices $v_0, v_1, \ldots, v_k$ onde $v_0 = v_k$ e não há mais vértices repetidos e tal que existem as arestas $(v_{i-1}, v_i), i = 1, \ldots, k$. O *tamanho* de um ciclo é dado por k. Um ciclo é *par* ou *ímpar* de acordo com a paridade do seu tamanho.

Definição 2.16. Um *ciclo sem corda* em um grafo é um ciclo tal que existem somente as arestas $(v_{i-1}, v_i), i = 1, \ldots, k$. Em outras palavras, um ciclo sem corda é um ciclo onde cada vértice é adjacente a exatamente dois outros vértices do mesmo.

Definição 2.17. Um ciclo sem corda de tamanho maior ou igual a 4 é um *buraco*. O complemento de um buraco é um *antiburaco*.

Definição 2.18. Uma *clique* em um grafo é um conjunto de vértices onde todos os vértices são adjacentes dois a dois.

Considere a relaxação linear do NPP definido para o antiburaco $\bar{H}_7$ de tamanho 7 apresentado na Figura 2.1. A solução $x_j = \frac{1}{2}$, $\forall j$, é ótima para a mesma e tem custo $\frac{7}{2} = 3{,}5$. O NPP, por sua vez, tem valor ótimo 2.



$$\text{NPP}(\bar{H}_7)\text{: } A_E x \leq \mathbf{1} \longrightarrow \begin{array}{l} x_1 + x_3 \leq 1 \\ x_1 + x_4 \leq 1 \\ x_1 + x_5 \leq 1 \\ x_1 + x_6 \leq 1 \\ x_2 + x_4 \leq 1 \\ x_2 + x_5 \leq 1 \\ x_2 + x_6 \leq 1 \\ x_2 + x_7 \leq 1 \\ x_3 + x_5 \leq 1 \\ x_3 + x_6 \leq 1 \\ x_3 + x_7 \leq 1 \\ x_4 + x_6 \leq 1 \\ x_4 + x_7 \leq 1 \\ x_5 + x_7 \leq 1 \end{array} \quad \begin{array}{l} x_1 + x_4 + x_6 \leq 1 \\ x_2 + x_5 + x_7 \leq 1 \\ x_1 + x_3 + x_6 \leq 1 \\ x_2 + x_4 + x_7 \leq 1 \\ x_1 + x_3 + x_5 \leq 1 \\ x_2 + x_4 + x_6 \leq 1 \\ x_3 + x_5 + x_7 \leq 1 \\ \uparrow \\ \textit{Inequações clique} \end{array}$$

Figura 2.1: O NPP definido para o antiburaco $\bar{H}_7$ e inequações clique.

Suponha, agora, que um conjunto de inequações válidas para o NPP é adicionado às restrições da relaxação. Esse conjunto é o das desigualdades associadas às cliques de cardinalidade máxima em um antiburaco. Com tais inequações, diz-se que no máximo um vértice presente em uma clique pode pertencer a um empacotamento. As mesmas têm a forma

$$\sum_{j \in Q} x_j \leq 1,$$

onde Q é a clique em questão. As inequações adicionadas são mostradas na Figura 2.1. Essas desigualdades definem facetas para $conv(S)$, onde S é o conjunto de soluções do NPP.

Denote, então, por T_E a formulação presente na relaxação linear inicial e por T_C a formulação presente na relaxação linear refinada, cujas restrições foram adicionadas das inequações associadas às cliques. Observe que toda inequação associada a uma aresta é dominada ou mais fraca que uma inequação associada a uma clique. Como $S \subseteq \mathbb{R}^n_+$, tem-se, pela Proposição 2.13, que $T_C \subset T_E$. A formulação da relaxação linear do NPP foi, portanto, fortificada.

Agora, a solução $x_j = \frac{1}{2}$, $\forall j$, já não é viável, nem sequer ótima, para a referida relaxação. Uma solução ótima é dada por $x_j = \frac{1}{3}$, $\forall j$, valendo $\frac{7}{3} \approx 2{,}3$. Este valor representa um limitante superior para o valor ótimo do NPP (o NPP foi definido por uma maximização) e, como no NPP os custos das variáveis são inteiros, tal limitante ainda pode ser arredondado para 2. Assim, o limitante obtido com a relaxação linear do NPP tornou-se, além de mais preciso, o melhor possível.

2.3 Aplicações ao SPP

Na Seção 1.1, o SPP foi descrito nos moldes de um *problema de otimização combinatória* (COP, do inglês *combinatorial optimization problem*). Como um IP, o SPP pode ser modelado por

$$\begin{aligned} z_{SPP} = \min\ & cx \\ & Ax = \mathbf{1} \\ & x \in \mathbb{B}^n, \end{aligned} \tag{2.11}$$

onde A é uma matriz de 0's e 1's. Este modelo relaciona-se com a descrição anterior do SPP da seguinte forma. Cada linha da matriz A corresponde a um elemento de S e cada coluna a um subconjunto S_j em F. Os elementos de S_j correspondem às linhas i para as quais $a_{ij} \neq 0$ (linhas cobertas pela coluna j). Cada variável binária x_j representa, então, a inclusão ou não do subconjunto S_j, de custo c_j, na coleção C (solução). Em uma solução viável para o problema, uma linha é coberta por exatamente uma coluna incluída, o que equivale a um elemento de S estar presente em exatamente um subconjunto S_j escolhido.

Sendo o SPP um problema $\mathcal{NP}$-Difícil modelado como um IP, uma estratégia tipicamente usada para atacá-lo é, então, resolver uma relaxação do mesmo. Duas relaxações são comumente utilizadas. A primeira é a relaxação linear. A mesma é dada por

$$\begin{aligned} \min\ & cx \\ & Ax = \mathbf{1} \\ & \mathbf{0} \leq x \leq \mathbf{1}. \end{aligned}$$

A segunda é a relaxação Lagrangiana obtida com a dualização de todas as restrições em

$Ax = \mathbf{1}$. A mesma é dada por

$$\begin{aligned} \min\ & cx + \lambda(\mathbf{1} - Ax) \\ & x \in \mathbb{B}^n \end{aligned} \tag{2.12}$$

ou

$$\begin{aligned} \min\ & (c - \lambda A)x + \lambda\mathbf{1} \\ & x \in \mathbb{B}^n. \end{aligned}$$

Observe que o PRL gerado nessa relaxação é um problema trivial de inspeção. Para resolvê-lo, basta fazer $x_j \Leftarrow 1$ se x_j é uma variável de custo Lagrangiano negativo e $x_j \Leftarrow 0$ se x_j é uma variável de custo Lagrangiano não negativo.

Ainda, uma relaxação do SPP pode ser refinada com a adição de inequações válidas. Para isto, a relação estabelecida adiante é de grande importância.

Na Seção 1.1, foram definidos, além do SPP, dois outros problemas $\mathcal{NP}$-Difíceis bastante íntimos ao mesmo: o SCP e o SSP. Como IPs, estes problemas podem ser modelados por

$$\begin{aligned} z_{SCP} = \min\ & cx \\ & Ax \geq \mathbf{1} \\ & x \in \mathbb{B}^n \end{aligned} \tag{2.13}$$

e

$$\begin{aligned} z_{SSP} = \max\ & cx \\ & Ax \leq \mathbf{1} \\ & x \in \mathbb{B}^n \end{aligned} \tag{2.14}$$

respectivamente. Os objetos destes modelos são idênticos aos presentes em (2.11).

Na referida seção, o SCP e o SSP foram definidos por meio de diferenças em relação à descrição do SPP. Aqui, poder-se-ia ter feito o mesmo para os IPs (2.13) e (2.14). No entanto, ainda é possível relacioná-los ao IP (2.11) apontando as distinções entre os modelos. A diferença entre o IP (2.13) e o IP (2.11) é que, em uma solução viável para o primeiro, uma linha i da matriz A é coberta por pelo menos (e não por exatamente) uma coluna j tal que $x_j = 1$. O IP (2.14), por sua vez, distingue-se do IP (2.11) por dois motivos. Primeiramente, o IP (2.14) é um problema de maximização (e não de minimização). Em segundo lugar, em uma solução viável para o mesmo, uma linha i da matriz A é coberta por no máximo (e não por exatamente) uma coluna j tal que $x_j = 1$.

Outra correspondência possível é a feita entre o modelo (2.11) e a descrição do SPP em termos de um COP, que também é válida dos modelos (2.13) e (2.14) para com as descrições anteriores do SCP e do SSP. Com a mesma, vê-se que, em uma solução viável para o SCP, um elemento de S está presente em pelo menos um subconjunto S_j incluído na coleção C. Já em uma solução viável para o SSP, um elemento de S está presente em no máximo um subconjunto S_j escolhido.

Aqui, é oportuno ressaltar uma importante distinção. Pode-se notar que é trivial dizer se o SCP ou o SSP são viáveis. Para o SSP, o vetor nulo é uma solução viável. No SCP, se a matriz de restrições não possuir linhas nulas, dar valor 1 a todas as variáveis também origina uma solução viável. Já para o SPP isso não é verdade. Somente saber se o mesmo é viável ou não já caracteriza uma tarefa bastante difícil.

O SCP e o SSP são problemas bastante úteis na resolução do SPP. Isto, primeiramente, pois inequações válidas para o conjunto de soluções do SCP ou do SSP são também válidas para o conjunto de soluções do SPP e, através das mesmas, é possível refinar uma relaxação do último. Mais ainda, pois pode-se estabelecer a seguinte relação entre os três problemas.

Proposição 2.14. *Sejam S_{SPP}, S_{SCP} e S_{SSP} os conjuntos de soluções do SPP, do SCP e do SSP respectivamente. Tem-se que*

$$conv(S_{SPP}) = conv(S_{SCP}) \cap conv(S_{SSP}). \tag{2.15}$$

Por esta relação, sabe-se que, para conhecer as inequações que definem facetas de $conv(S_{SPP})$, é suficiente conhecer as inequações que definem facetas de $conv(S_{SCP})$ e de $conv(S_{SSP})$.

Com o intuito de documentar recursos para a resolução do problema-alvo desta dissertação, o SPP, apresenta-se, a seguir, uma lista de inequações válidas para o SCP e para o SSP. Um grupo dessas desigualdades constitui as inequações usadas para refinar relaxações do SPP neste trabalho. Ao final da seção, são descritas, também, algumas desigualdades válidas específicas para o SPP.

2.3.1 Inequações válidas para o SSP

São conhecidas algumas técnicas para produzir inequações válidas para o SSP. Uma dessas é o estudo de *grafos definidores de faceta* [13]. O fundamento de tal técnica é a associação do SSP a um grafo, descrito abaixo.

Seja A a matriz de restrições do SSP. Associe a A o seguinte *grafo de interseção* $G(A)$: os vértices de $G(A)$ correspondem às colunas de A e existe uma aresta entre os vértices v_i e v_j se a^i e a^j são colunas de A que têm interseção, ou seja, $a^i . a^j \neq 0$. A importância desta construção é que o conjunto das soluções viáveis para o SSP é exatamente igual ao conjunto dos possíveis empacotamentos de vértices em $G(A)$. Assim, uma inequação válida para o último conjunto é também válida para o SSP.

Aqui, é oportuno estabelecer uma interessante equivalência. Considere a versão ponderada do NPP (definido no Exemplo 2.1). Nesta versão, consideram-se pesos para os vértices e busca-se um empacotamento de vértices de peso total máximo. Sendo o peso

de um vértice v_j em $G(A)$ dado pelo custo c_j da variável correspondente do SSP, resolver o SSP é equivalente a solucionar o NPP ponderado para $G(A)$. Uma consequência desta equivalência é a nomenclatura *stable set problem* dada ao SSP. Um empacotamento de vértices é também denominado de conjunto independente de vértices, ou, usualmente em inglês, *stable set*. Tem-se, portanto, tal nomenclatura.

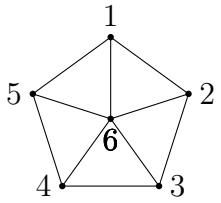
O uso de grafos definidores de faceta para derivar inequações válidas para o SSP baseia-se na seguinte relação. Tome o SSP associado a um subgrafo de $G(A)$. Se uma inequação é válida para o mesmo, essa também é válida para o SSP vinculado a $G(A)$. Assim, subestruturas do grafo $G(A)$ dão origem a inequações válidas para o SSP. Se uma inequação define uma faceta para o SSP correspondente a um subgrafo de $G(A)$, tal subgrafo é dito definidor dessa faceta.

É importante ressaltar que uma inequação que define uma faceta para o SSP associado a um subgrafo de $G(A)$ não necessariamente define uma faceta para o SSP vinculado a $G(A)$. É interessante, portanto, estudar extensões/operações que possam ser aplicadas a inequações provenientes de subgrafos de $G(A)$ para gerar desigualdades mais fortes para o SSP geral. Esta ideia também é útil para produzir inequações que não sejam obteníveis por meio de algum subgrafo isoladamente. Comentam-se, a seguir, as duas maneiras mais comumente empregadas para fazer isto: *lifting* e operações com grafos.

O *lifting* é um procedimento que visa introduzir variáveis em uma inequação. Variáveis com coeficiente 0 podem ser interpretadas como ausentes em uma inequação. Ao introduzir uma variável em uma desigualdade, busca-se determinar o maior coeficiente possível para a mesma. Dado um conjunto de variáveis a serem incorporadas, a ordem seguida no processo define a *sequência de lifting*. Este procedimento é ilustrado abaixo.

$$
\begin{aligned}
x_1 + x_2 + x_6 &\leq 1 \\
x_2 + x_3 + x_6 &\leq 1 \\
x_3 + x_4 + x_6 &\leq 1 \\
x_4 + x_5 + x_6 &\leq 1 \\
x_1 + x_5 + x_6 &\leq 1
\end{aligned}
$$

Restrições do SSP



$G(A)$

Figura 2.2: SSP e o grafo associado $G(A)$.

Exemplo 2.2. Considere o SSP e o grafo associado $G(A)$ mostrados na Figura 2.2. Tome, agora, o buraco (Definição 2.17) ímpar induzido pelos vértices 1, 2, 3, 4 e 5. A inequação

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 2 \tag{2.16}$$

é válida para o SSP vinculado a esse subgrafo de $G(A)$. Mais ainda, tal inequação define uma faceta para o mesmo. Isto, porém, não é verdade para o SSP geral. A desigualdade

é válida, mas não define faceta. Pode-se questionar, então: É possível torná-la uma inequação válida mais forte para o SSP associado a $G(A)$ introduzindo a variável x_6? Se sim, qual o maior coeficiente que x_6 pode ter? (Atualmente, o coeficiente de x_6 é 0.)

Suponha que o coeficiente de x_6 na inequação (2.16) é dado por α. Assim, tem-se que

$$x_1 + x_2 + x_3 + x_4 + x_5 + \alpha x_6 \leq 2.$$

A variável x_6 pode assumir os valores 0 ou 1. Se $x_6 = 0$, a inequação acima é válida para o SSP para qualquer valor de α. Se $x_6 = 1$, o mesmo ocorre quando

$$\alpha \leq 2 - (x_1 + x_2 + x_3 + x_4 + x_5).$$

Ainda, pelas restrições do SSP, se $x_6 = 1$, todas as outras variáveis têm de valer 0. Logo, a inequação é válida se $\alpha \leq 2$. É possível, portanto, introduzir x_6 na desigualdade (2.16) com coeficiente 2 e torná-la uma inequação válida mais forte para o SSP geral dada por

$$x_1 + x_2 + x_3 + x_4 + x_5 + 2x_6 \leq 2.$$

Agora, esta inequação define uma faceta para o mesmo.

Outra maneira de estender desigualdades obtidas de subgrafos de $G(A)$ é através de operações com grafos. A ideia é compor subgrafos para gerar um novo subgrafo. Pretende-se, com isto, construir uma nova inequação a partir de inequações menores. Operações possíveis são: extensão, adicionando vértices (e arestas) ao grafo; substituição, trocando parte do grafo (como um vértice ou um subgrafo) por outro grafo; e junção, unindo grafos com a identificação de algumas partes (essas partes passando a ser comuns aos mesmos).

Apresenta-se, agora, uma lista de inequações válidas para o SSP. Estas inequações são derivadas da técnica comentada acima: o estudo de grafos definidores de facetas. As mesmas são inequações válidas para o conjunto dos possíveis empacotamentos de vértices em um subgrafo do grafo $G(A)$ associado ao SSP. Para uma referência mais completa sobre inequações assim produzidas para o SSP, veja [13]. Abaixo, o grafo $G(A)$ é denotado simplesmente por G.

(**Inequações aresta**). Associada a uma aresta (v_i, v_j) do grafo G está a inequação

$$x_i + x_j \leq 1.$$

Com esta desigualdade, diz-se que apenas um dos vértices incidentes a uma aresta pode estar presente em um empacotamento. A inequação aresta também pode ser vista como um caso particular das inequações clique (cliques de tamanho 2), citadas a seguir.

Uma observação importante é que o sistema composto pelas inequações aresta possíveis para G e pelas restrições de não negatividade para as variáveis origina uma relaxação do SSP. Tipicamente, esta relaxação difere da linear, pois, em geral, o conjunto de restrições da última contém propriamente o conjunto de restrições da primeira. Tal sistema adicionado das restrições de integralidade e de limitantes superiores para as variáveis foi usado para descrever o NPP em (2.10).

(**Inequações clique**)**.** Associada a uma clique Q (Definição 2.18) do grafo G está a inequação

$$\sum_{j \in Q} x_j \leq 1.$$

Veja a clique de tamanho 7 na Figura 2.3. Com a desigualdade acima, diz-se que, dentre os vértices de Q, apenas um pode pertencer a um empacotamento. Um importante resultado conhecido é que uma inequação clique define uma faceta para o SSP associado a G se, e somente se, a clique em questão for maximal com respeito a inclusão de vértices de G.

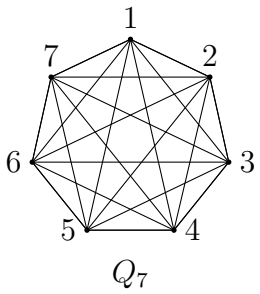


Figura 2.3: Clique de tamanho 7.

(**Inequações ciclo ímpar**)**.** Associada a um ciclo ímpar C (Definição 2.15) do grafo G está a inequação

$$\sum_{j \in C} x_j \leq \frac{(|C| - 1)}{2}.$$

Veja um ciclo ímpar de tamanho 5 na Figura 2.4. Com esta desigualdade, revela-se o número máximo de vértices não adjacentes dois a dois em tal estrutura.

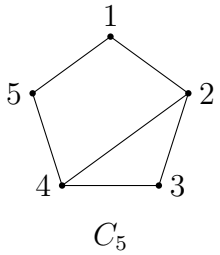


Figura 2.4: Ciclo ímpar (com corda) de tamanho 5.

(**Inequações antiburaco ímpar**). Associada a um antiburaco ímpar $\bar{H}$ (Definição 2.17) do grafo G está a inequação

$$\sum_{j \in \bar{H}} x_j \leq 2.$$

Veja o antiburaco ímpar de tamanho 7 na Figura 2.1. Observe que, em um antiburaco ímpar $\bar{H}$, um vértice somente não está ligado a outros dois vértices. Além disso, existe uma aresta entre os últimos. Assim, há no máximo dois vértices não adjacentes dois a dois em $\bar{H}$. Com a desigualdade acima, expressa-se exatamente isso.

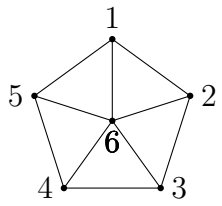
Definição 2.19. Uma *roda* em um grafo é constituída por um ciclo ímpar sem corda adicionado de um vértice que está conectado a todos os outros vértices.

(**Inequações roda**). Na Figura 2.5, mostra-se a roda definida pelo ciclo ímpar sem corda de tamanho 5. Seja, então, j^* o índice do vértice destacado em uma roda R do grafo G e seja $k = \frac{|C|-1}{2}$ onde C é o ciclo ímpar induzido pelos demais vértices de R. Associada à roda R está a inequação

$$kx_{j^*} + \sum_{j \in C} x_j \leq k.$$

A parte $\sum_{j \in C} x_j \leq k$ desta desigualdade é uma inequação ciclo ímpar. Com o termo kx_{j^*}, impõe-se que, se o vértice destacado for incluído em um empacotamento, nem mais um outro vértice o será.

Pelo desmembramento feito acima, pode-se ver que uma inequação roda é um *lifting* de uma inequação ciclo ímpar. Ainda, para uma roda com 6 ou mais vértices, a desigualdade correspondente é um *lifting* de uma inequação buraco ímpar. No Exemplo 2.2, é ilustrada tal situação. As inequações buraco ímpar não foram definidas explicitamente nesta subseção. Estas nada mais são que inequações ciclo ímpar.



R_5

Figura 2.5: Roda definida pelo ciclo ímpar sem corda de tamanho 5.

2.3.2 Inequações válidas para o SCP

Uma técnica utilizada para produzir inequações válidas para o SCP é o estudo de *matrizes definidoras de faceta* [13]. O uso de tais estruturas para o SCP é análogo ao uso de grafos definidores de faceta para o SSP e baseia-se na relação estabelecida abaixo.

Seja A a matriz de restrições do SCP. Considere que são conhecidas inequações válidas para o SCP vinculado a uma submatriz de A. Essas inequações são também válidas para o SCP maior. Se uma inequação define uma faceta para o SCP correspondente a uma submatriz de A, tal submatriz é dita definidora dessa faceta.

Na Subseção 2.3.1, quando se discutiram os grafos definidores de faceta, uma observação importante foi feita. Aqui, uma semelhante cabe. Uma inequação que define uma faceta para o SCP associado a uma submatriz de A não necessariamente define uma faceta para o SCP vinculado a A. O *lifting*, descrito na referida subseção, é um mecanismo usado para tornar a mesma uma desigualdade mais forte para o SCP maior. Uma ressalva é que o *lifting* de desigualdades válidas para o SCP é comumente considerado mais complexo [13] que o correspondente para o SSP. O princípio, porém, é o mesmo.

Com o estudo de inequações válidas para o SCP, trilha-se um caminho parecido com o percorrido para o SSP. Estruturas como cliques e ciclos também são geradas. Nos dois casos, porém, esse estudo não tem obtido o mesmo sucesso ao longo dos anos. À abrangência da revisão bibliográfica feita nesta dissertação, constatou-se que, ao contrário do que ocorre para o SSP, não se sabe de classes de inequações separáveis polinomialmente[1] para o SCP. Além disso, SSPs especiais, para os quais a envoltória convexa do conjunto de soluções é descrita completamente por desigualdades de um certo tipo, são conhecidos. Isto não acontece para o SCP. Börndorfer [13] cita, como aparente causa desta diferença, que abordagens via teoria de grafos são menos eficientes para o SCP.

As inequações válidas para o SCP listadas abaixo são provenientes da técnica mencionada acima: o estudo de matrizes definidoras de faceta. Essas desigualdades são mais complexas, no sentido descritivo, que as apresentadas na Subseção 2.3.1. Por este motivo, as mesmas são citadas com poucos detalhes técnicos. Para uma relação mais extensa de inequações, veja [13].

(**Inequações antirrede generalizada**). Uma *antirrede generalizada* $AW(n, t, q)$ é uma matriz $0-1$ de estrutura especial definida pelos parâmetros inteiros n, t e q. Veja um exemplo na Figura 2.6 (obtida de [13]). Associada a uma antirrede generalizada $AW(n, t, q)$ está a inequação

$$\sum_j x_j \geq \lceil n\frac{(t-q+1)}{t} \rceil.$$

Tal desigualdade sumariza várias outras, a saber, inequações clique generalizada, inequações ciclo generalizado, inequações antiburaco generalizado e inequações buraco ímpar.

[1] Esse conceito será exposto no próximo capítulo.

$$\begin{pmatrix} 1 & 1 & . & . & . \\ 1 & . & 1 & . & . \\ . & 1 & 1 & . & . \\ . & 1 & . & 1 & . \\ . & . & 1 & 1 & . \\ . & . & 1 & . & 1 \\ . & . & . & 1 & 1 \\ 1 & . & . & 1 & . \\ . & 1 & . & . & 1 \end{pmatrix}$$

Figura 2.6: Antirrede generalizada $AW(5, 3, 2)$ [13].

(**Inequações rede generalizada**). Uma *rede generalizada* $W(n, t, q)$ é o complemento de uma antirrede generalizada (também uma matriz $0 - 1$). Veja um exemplo na Figura 2.7 [13]. Associada a uma rede generalizada $W(n, t, q)$ está a inequação

$$\sum_j x_j \geq (n - t).$$

$$\begin{pmatrix} 1 & . & . & 1 & . & . & . \\ 1 & . & . & . & 1 & . & . \\ . & 1 & . & . & 1 & . & . \\ . & 1 & . & . & . & 1 & . \\ . & . & 1 & . & . & 1 & . \\ . & . & 1 & . & . & . & 1 \\ . & . & . & 1 & . & . & 1 \end{pmatrix}$$

Figura 2.7: Rede generalizada $AW(7, 3, 2)$ [13].

Por fim, algumas menções são válidas. As inequações listadas acima têm apenas coeficientes 0 ou 1 no lado esquerdo. Essas desigualdades são do tipo *inequações posto*. Nobili e Sassano [50] estudaram composições de inequações posto. Balas e Ng [5, 6] caracterizaram completamente as desigualdades definidoras de facetas do SCP com apenas coeficientes 0, 1 ou 2 no lado esquerdo.

2.3.3 Inequações válidas para o SPP

Pela relação (2.15), sabe-se que o SPP herda inequações válidas dos problemas "irmãos" SCP e SSP. Ainda, várias classes de inequações válidas para os últimos são bem conhecidas e utilizadas há muitos anos na resolução do SPP. Entretanto, não são somente dessa

natureza as desigualdades sabidas para o SPP. Há outras, obtidas por vias diferentes. Abaixo, comentam-se algumas.

Primeiramente, o SPP, bem como IPs em geral ou ainda problemas de outras classes, podem ser modelados por meio de *programação disjuntiva*. Isto significa descrever um problema usando programação linear e restrições disjuntivas, que são definidas por conectivos lógicos $\vee$ (ou). Com esta modelagem, permite-se a produção de desigualdades com coeficientes negativos no lado esquerdo, ao contrário de todas as inequações vistas até agora nesta seção. Balas e Padberg [7] citam tais inequações como computacionalmente baratas de se obter.

Em segundo lugar, há uma classe de inequações válidas para o SPP introduzida por Balas [3]. Essas inequações são baseadas em implicações lógicas das restrições do SPP e geralmente não são satisfeitas por soluções viáveis para o SSP. As mesmas podem ser descritas como abaixo.

Seja A a matriz de restrições do SPP. Sejam k e i coluna e linha de A. Lembre que a coluna k cobre a linha i ou a linha i é coberta pela coluna k se $a_{ik} \neq 0$. Defina, então, $J(k,i)$ como o conjunto das colunas de A que cobrem i e são ortogonais a k (colunas j de A tais que $a^j . a^k = 0$).

Considere, agora, que k e i são coluna e linha de A tais que i não é coberta por k. A inequação

$$x_k - \sum_{j \in J(k,i)} x_j \leq 0 \tag{2.17}$$

é válida para o SPP.

A validade da inequação acima é justificada a seguir. Tome uma solução viável $x_j = b_j, b_j \in \{0,1\}, \forall j$, para o SPP. Como $x_j \geq 0, \forall j$, tem-se que $\sum_{j \in J(k,i)} x_j \geq 0$ e $-\sum_{j \in J(k,i)} x_j \leq 0$. Logo, se $x_k = 0$, a inequação é satisfeita pela solução. Suponha, então, que $x_k = 1$. Observe que existe uma única variável x_l tal que a coluna l cobre a linha i e $x_l = 1$. Como x_k vale 1, l tem de ser ortogonal a k e, portanto, pertencer a $J(k,i)$. Assim, $\sum_{j \in J(k,i)} x_j = 1$ e o lado esquerdo da inequação tem valor 0.

Como dito anteriormente, a desigualdade acima não é válida para o SSP. Isto, pois, em uma solução viável para o SSP, não necessariamente existe uma variável x_l tal que a coluna l cubra a linha i e $x_l = 1$. Assim, pode-se ter $x_k = 1$ e $\sum_{j \in J(k,i)} x_j = 0$, com o lado esquerdo da inequação resultando em 1.

É possível, ainda, substituir, em (2.17), $J(k,i)$ por $Q(k,i) \subseteq J(k,i)$. A inequação resultante será válida se, e somente se, $x_k = 1$ implicar $x_j = 0, \forall j \in J(k,i) \setminus Q(k,i)$. Note que esta modificação torna a desigualdade mais forte. Em geral, pode-se, a partir de (2.17), aplicar procedimentos como os descritos em [3] para obter novas inequações válidas.

Por fim, inequações válidas para o SPP podem ser derivadas por meio da associação

do mesmo a um grafo, sendo esta associação análoga à feita entre o SSP e o grafo de interseção $G(A)$ na Subseção 2.3.1. Como para a construção realizada para o SSP, ocorre que o conjunto das soluções viáveis para o SPP é exatamente igual ao conjunto dos possíveis empacotamentos de vértices no grafo criado. Assim, uma inequação válida para o último conjunto é também válida para o SPP.

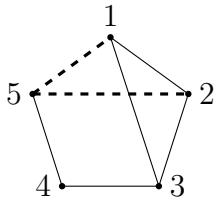
Antes de apresentar o grafo usado para o SPP, é ainda conveniente definir $G(A)$ de uma maneira levemente diferente. Com esta alteração, os dois grafos passam a ter definições uniformes. As mesmas seguem.

Seja A a matriz de restrições do SSP. Associe a A o seguinte *grafo de interseção* $G(A)$: os vértices de $G(A)$ correspondem às colunas de A e existe uma aresta entre os vértices v_i e v_j se não pode ocorrer, em uma solução do SSP, $x_i = 1$ e $x_j = 1$. Observe que esta definição de $G(A)$ é equivalente à apresentada na Subseção 2.3.1.

Seja A a matriz de restrições do SPP. Associe a A o seguinte *grafo forte de interseção* $G(A)'$: os vértices de $G(A)'$ correspondem às colunas de A e existe uma aresta entre os vértices v_i e v_j se não pode ocorrer, em uma solução do SPP, $x_i = 1$ e $x_j = 1$. É válido ressaltar que a condição dada acima para a existência de uma aresta entre os vértices v_i e v_j não é equivalente às colunas a^i e a^j terem interseção ($a^i . a^j \neq 0$).

O exemplo abaixo ilustra como inequações válidas específicas para o SPP podem ser geradas com uso do grafo $G(A)'$.

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$



$G(A)$ e $G(A)'$

Figura 2.8: Matriz A, $G(A)$ e $G(A)'$.

Exemplo 2.3. Seja A a matriz $0-1$ da Figura 2.8. Nesta figura, são mostrados o grafo de interseção $G(A)$, considerando A a matriz de restrições de um SSP, e o grafo forte de interseção $G(A)'$, sendo A associada a um SPP. O grafo $G(A)$ é composto apenas pelas arestas contínuas e o grafo $G(A)'$, pelas contínuas e pontilhadas. As arestas pontilhadas são derivadas de implicações lógicas das restrições (de igualdade) do SPP. Note que $G(A)$ é sempre um subgrafo de $G(A)'$.

O importante, aqui, é que o estudo de grafos definidores de faceta pode ser usado para gerar, em relação às inequações obtidas via $G(A)$, novas e mais fortes desigualdades para o SPP via $G(A)'$. Por exemplo, o grafo $G(A)$ mostrado na Figura 2.8 tem apenas uma

clique (de tamanho maior ou igual a 3): $\{1, 2, 3\}$. Já o grafo $G(A)'$ possui uma outra clique e um buraco ímpar: $\{1, 2, 5\}$ e $\{1, 2, 3, 4, 5\}$.

Capítulo 3

Algoritmos para atacar um IP

Neste capítulo, são descritos métodos bem sucedidos e bem conhecidos para atacar um IP. Tais métodos são fundamentados em uma estratégia mencionada no capítulo anterior: não resolver o problema diretamente, e sim uma relaxação do mesmo. Assim, antes de descrevê-los, apresenta-se um algoritmo geral à luz do qual os mesmos podem ser definidos: um algoritmo geral baseado em relaxações. Os métodos discutidos aqui compõem os algoritmos implementados nesta dissertação.

Como na Seção 2.2, tome por P um IP modelado como

$$\begin{aligned} z = \min \quad & cx \\ & x \in S \subseteq \mathbb{Z}^n_+. \end{aligned} \tag{3.1}$$

Suponha que P é viável e limitado, ou seja, que $S \neq \emptyset$ e $z > -\infty$. Sejam $\overline{z}$ e $\underline{z}$ limitantes superior e inferior para z respectivamente. Algoritmos para solucionar P, buscam, de uma forma geral, a cada iteração k, obter menores $\overline{z}^k$ e maiores $\underline{z}^k$. Isto, até que $\overline{z}^k = \underline{z}^k$ ou $(\overline{z}^k - \underline{z}^k) < \epsilon$. Comumente, os limitantes inferiores são produzidos por relaxações e os superiores por soluções viáveis. Assim, ao final do algoritmo, duas situações podem ocorrer: ou uma solução ótima x^k com $\overline{z}^k = cx^k = \underline{z}^k$ é encontrada ou uma solução ϵ-aproximada com $cx^k = \overline{z}^k$ e $(\overline{z}^k - \underline{z}^k) < \epsilon$ é obtida.

Na referida seção, foi definida uma relaxação de P, que é um problema RP dado por

$$\begin{aligned} z_R = \min \quad & f(x) \\ & x \in T \subseteq \mathbb{R}^n_+, \end{aligned} \tag{3.2}$$

onde $T \supseteq S$ e $f(x) \leq cx, \forall x \in S$. Resolvendo RP, obtém-se um limitante inferior para z, a saber, $\underline{z} = z_R = f(x_R)$, onde x_R é solução ótima para RP. Se x_R é viável para P ($x_R \in S$), tem-se, também, um limitante superior para z, dado por $\overline{z} = cx_R$. E se $f(x_R) = cx_R$, x_R é solução ótima para P, pois $\overline{z} = cx_R = f(x_R) = \underline{z}$.

Considerando as observações acima, um algoritmo geral baseado em relaxações para resolver P é o seguinte.

Algoritmo 3.1 Algoritmo geral baseado em relaxações

Inicialização: Escolha uma relaxação inicial RP^0, ou seja, determine $(f^0(x), T^0)$.
Faça $\overline{z}^0 \Leftarrow \infty$, $\underline{z}^0 \Leftarrow -\infty$.

Iteração k:

Passo 1: Resolva a relaxação RP^k, obtendo $z_R^k = f(x_R^k)$.

Passo 2: Teste se a solução x_R^k obtida é ótima para P. Isto ocorre se $x_R^k \in S$ e $f(x_R^k) = cx_R^k$, pois, por consequência, tem-se que $\underline{z}^k = z_R^k = f(x_R^k) = cx_R^k = \overline{z}^k$. Se sim, pare.

Passo 3: Se $z_R^k > \underline{z}^k$, faça $\underline{z}^k \Leftarrow z_R^k$.
Se $x_R^k \in S$ e $cx_R^k < \overline{z}^k$, faça $\overline{z}^k \Leftarrow cx_R^k$.
Refine a relaxação. Escolha T^{k+1} tal que $T^k \supseteq T^{k+1} \supseteq S$ e $f^{k+1}(x)$ tal que $f^k(x) \leq f^{k+1}(x) \leq cx, \forall x \in S$, ocorrendo $T^{k+1} \neq T^k$ ou $f^{k+1}(x) \neq f^k(x)$.

No Passo 3 de uma iteração deste algoritmo, procura-se refinar a relaxação. Com isto, busca-se um limitante z_R^{k+1} maior que z_R^k ou uma solução x_R^{k+1} mais próxima de pertencer a S que x_R^k. Para tanto, deve-se modificar a relaxação de fato, com T^{k+1} ou $f^{k+1}(x)$ sendo diferente do anterior. É comum que, em um algoritmo visto como uma instância deste, se tenha $f^k(x) = cx, \forall k$. Neste caso, refinar a relaxação significa ter $T^{k+1} \subset T^k$. É fundamental, ainda, que $x_R^k \notin T^{k+1}$. Em caso contrário, o limitante obtido será o mesmo que o anterior.

Tal como a redação do capítulo anterior, a apresentação feita a seguir é sucinta e baseada em um conjunto de livros-texto e artigos que constituem bibliografia padrão para os temas aqui discutidos. Para um tratamento mais aprofundado dos mesmos, dá-se, ao leitor, os apontamentos necessários. Para as Seções 3.1, 3.3 e 3.4, as referências são [26], [49] e [59]. A Seção 3.2 fundamenta-se nos artigos [43, 44], [38], [9] e [32]. Novamente, ao leitor familiar aos assuntos apresentados, recomenda-se uma leitura rápida e superficial.

3.1 Algoritmos de planos de corte

Um *algoritmo de planos de corte* para atacar um IP consiste na resolução de relaxações do mesmo que, iterativamente, são refinadas por meio da adição de uma ou mais inequações válidas. A estratégia presente em um algoritmo de planos de corte fundamenta-se em uma observação feita a seguir.

Seja P (3.1) o IP em questão. Considere que $conv(S)$ possa ser descrita pelo sistema de inequações $A'x \leq b'$, ou seja, que

$$conv(S) = \{x \in \mathbb{R}_+^n, A'x \leq b'\}.$$

Como posto na Proposição 2.5, com isso, pode-se, a princípio, resolver P otimizando o LP

$$\begin{aligned} \min \quad & cx \\ & A'x \leq b' \\ & x \in \mathbb{R}^n_+. \end{aligned}$$

Por esse motivo, tal LP representa uma relaxação "ideal" de P.

De um modo geral, é possível solucionar P de uma maneira potencialmente fácil resolvendo uma relaxação tal que a formulação presente na mesma contenha o sistema $A'x \leq b'$. Para o problema

$$\begin{aligned} \min \quad & f(x) \\ & A'x \leq b' \\ & x \in T', \end{aligned}$$

onde $S \subseteq T' \subseteq \mathbb{R}^n_+$, há uma solução ótima x_R que é viável para P. A solução x_R também é ótima para P se $f(x_R) = cx_R$.

Como comentado antes, a ideia de resolver P mediante um LP é, entretanto, raramente concretizável. Isto, pois ou não se conhece $A'x \leq b'$ ou este sistema possui um número exponencialmente grande de inequações, o que torna seu uso impraticável.

Uma observação fundamental é que, embora a descrição da envoltória convexa de S seja suficiente para que P seja solucionado através de uma relaxação, a mesma não é necessária. Se somente as inequações que determinam um ponto extremo de $conv(S)$ ótimo para P estão presentes nas restrições de um LP, já é possível que P seja resolvido por meio do mesmo. Para tanto, basta que o referido ponto seja a solução ótima encontrada na otimização de tal problema. Esta constatação sugere a seguinte estratégia. Adicione iterativamente inequações válidas a uma relaxação até que a solução ótima x_R obtida pertença a S. Se $f(x_R) = cx_R$, x_R será também ótima para P. É isto que faz um algoritmo de planos de corte.

Apresenta-se, abaixo, um algoritmo de planos de corte para atacar P. O mesmo é descrito como uma instância do algoritmo geral baseado em relaxações. Observe que, para realizar tal descrição, é suficiente definir como uma relaxação RP^k é refinada. Neste algoritmo, F é uma família de inequações válidas para S.

Aqui, uma ressalva é necessária. Note que, caso a família F não contenha um conjunto de inequações que descreva completamente $conv(S)$, o Algoritmo 3.2 pode terminar com uma solução x_R que não é viável e, consequentemente, não é ótima para P.

É importante ressaltar também que, no Algoritmo 3.2, a relaxação empregada não é necessariamente a linear. Ainda, adicionar uma inequação a uma relaxação não significa obrigatoriamente inserí-la no conjunto de restrições. Tais opções, contudo, são as tipica-

Algoritmo 3.2 Algoritmo de planos de corte

Refinamento da relaxação: Procure em F uma ou mais inequações $\pi x \leq \pi_0$ que não sejam satisfeitas por x_R^k, ou seja, com $\pi x_R^k > \pi_0$.

Se as inequações forem encontradas, adicione-as à relaxação RP^k e obtenha RP^{k+1} (f^{k+1}, T^{k+1}).

Senão, pare: a solução x_R^k satisfaz a todas as inequações em F.

mente usadas para algoritmos de planos de corte. É das mesmas, inclusive, que advém essa nomenclatura. Para o algoritmo resultante, o *algoritmo de planos de corte fracionário*, ocorre o seguinte. Ao longo das iterações, as inequações (planos) adicionadas *cortam* do conjunto de soluções da relaxação os pontos x_R que não são viáveis para P. As inequações adicionadas são denominadas, então, *planos de corte* ou simplesmente *cortes*. Em um exemplo adiante, ilustra-se tal situação.

O estudo da adição de inequações válidas a um problema foi iniciado por Gomory [29] nos anos 50. Gomory propôs inequações que podem ser consideradas gerais, por não dependerem da estrutura específica de um problema. Para o algoritmo de planos de corte fracionário, o uso das mesmas garante que, em teoria, as soluções ótimas das relaxações convirjam, em um número finito de passos, para uma solução ótima do problema original. As inequações de Gomory, porém, mostraram-se pouco eficientes na prática. Isto se deve ao fato de essas inequações raramente definirem facetas ou pelo menos faces de dimensão alta da envoltória convexa das soluções do problema.

No entanto, nas décadas mais recentes, o estudo poliédrico de vários problemas $\mathcal{NP}$-Difíceis propiciou a descoberta de muitas desigualdades com a propriedade acima. No capítulo anterior, por exemplo, foram vistas várias classes de inequações com tal característica para o SSP, o SCP e o SPP. Desde então, vêm-se obtendo ótimos resultados práticos com algoritmos de planos de corte.

Um outro ponto a destacar é a complexidade do problema que se tem que atacar no refinamento de uma relaxação no Algoritmo 3.2. Tal problema pode ser enunciado da seguinte maneira. Dado $x' \in \mathbb{R}^n$, encontre, dentre uma família F de inequações, uma inequação $\pi x \leq \pi_0$ que seja violada por x', ou seja, com $\pi x' > \pi_0$, ou mostre que x' satisfaz a todas essas. O mesmo é um *problema de separação* e é, em geral, um problema $\mathcal{NP}$-Difícil. Ainda, um resultado fundamental de Grötschel, Lovász e Schrijver [31] mostra que, para um dado problema, as tarefas de otimizar e separar são equivalentemente difíceis. Isto implica que, se a família F contém um conjunto de inequações que descreve completamente $conv(S)$ e se existe um algoritmo de tempo polinomial para o problema de separação em F, um algoritmo de planos de corte fracionário soluciona P em tempo polinomial. O passo de separar uma solução x_R é estratégico para um bom desempenho de um algoritmo de planos de corte.

Especifica-se, agora, o algoritmo de planos de corte fracionário para atacar P. Na sequência, é dado um exemplo de aplicação para ilustrar o funcionamento do mesmo. Novamente, este algoritmo de planos de corte é descrito como uma instância do algoritmo geral baseado em relaxações. Para a descrição, contudo, é necessário, desta vez, determinar, além do refinamento de uma relaxação RP^k, qual a relaxação inicial. Considere que, para o algoritmo a seguir, $S = \{x \in \mathbb{Z}^n_+, Ax \leq b\}$.

Algoritmo 3.3 Algoritmo de planos de corte fracionário

Relaxação inicial: Escolha $T^0 = \{x \in \mathbb{R}^n_+, Ax \leq b\}$ e $f^0(x) = cx, \forall x \in \mathbb{R}^n_+$.

Refinamento da relaxação: Tome $f^{k+1}(x) = f^k(x)$.

Procure em F uma ou mais inequações $\pi x \leq \pi_0$ que não sejam satisfeitas por x^k_R, ou seja, com $\pi x^k_R > \pi_0$.

Se as inequações forem encontradas, a saber, $\pi^1 x \leq \pi^1_0, \ldots, \pi^l x \leq \pi^l_0$, escolha $T^{k+1} = T^k \cap \{x \in \mathbb{R}^n_+, \pi^1 x \leq \pi^1_0\} \cap \ldots \cap \{x \in \mathbb{R}^n_+, \pi^l x \leq \pi^l_0\}$.

Senão, pare: a solução x^k_R satisfaz a todas as inequações em F.

Exemplo 3.1. Considere o IP mostrado na Figura 3.1. As soluções viáveis do IP são os pontos pretos. O conjunto de soluções viáveis da relaxação linear do mesmo é dado pela região delimitada pelas linhas contínuas. Essa região está colorida de cinza.

Um algoritmo de planos de corte fracionário para atacar o IP funciona como a seguir.

$$
\begin{aligned}
z = \max 4x_1 - \ & x_2 \\
7x_1 - 2x_2 &\leq 14 \\
x_2 &\leq 3 \\
2x_1 - 2x_2 &\leq 3 \\
x_1, \quad x_2 &\geq 0, \text{ e inteiras}
\end{aligned}
$$

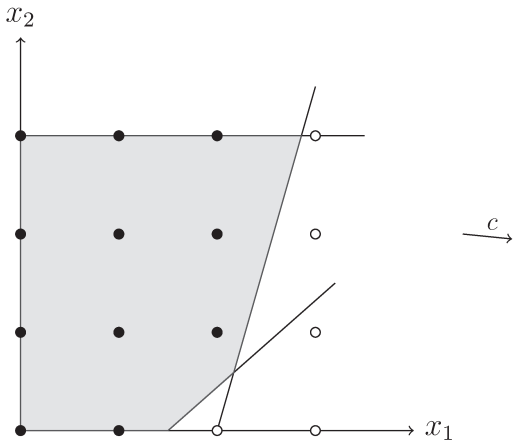


Figura 3.1: IP, suas soluções e sua relaxação linear.

Inicialização: RP^0 é dada pela relaxação linear do IP.

Iteração 1: Resolvendo RP^0, obtém-se a solução $x^1_R = (\frac{20}{7}, 3)$, mostrada na Figura 3.2. Suponha que o corte $x_1 \leq 2$ violado por x^1_R é encontrado. Note que esse corte não é, de

fato, satisfeito por x_R^1. Isto, pois, $x_{R_1}^1 = \frac{20}{7} > 2$. Tal corte é, então, inserido no conjunto de restrições da relaxação, o que é representado pela linha pontilhada azul na referida figura. Assim, a relaxação mais refinada RP^1 é produzida. Agora, o ponto x_R^1 não é mais viável. Esse ponto foi cortado do conjunto de soluções.

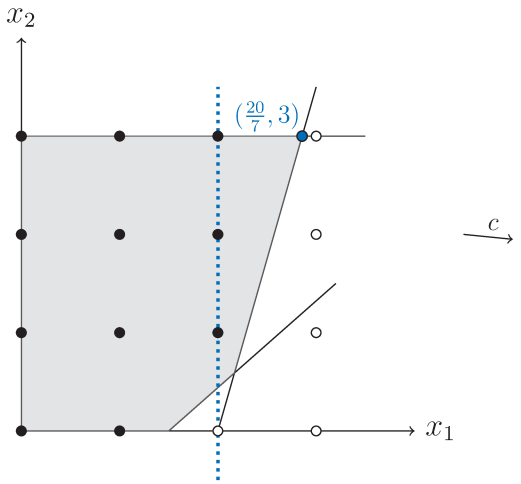


Figura 3.2: Algoritmo de planos de corte fracionário: iteração 1.

Iteração 2: A solução ótima obtida da relaxação RP^1 é $x_R^2 = (2, \frac{1}{2})$. Veja a Figura 3.3. Considere que o corte $x_1 - x_2 \leq 1$ não satisfeito pela solução x_R^2 é descoberto. Esse corte é adicionado à formulação da relaxação. Novamente, o corte corresponde à linha pontilhada azul na figura citada. A relaxação RP^2 é obtida.

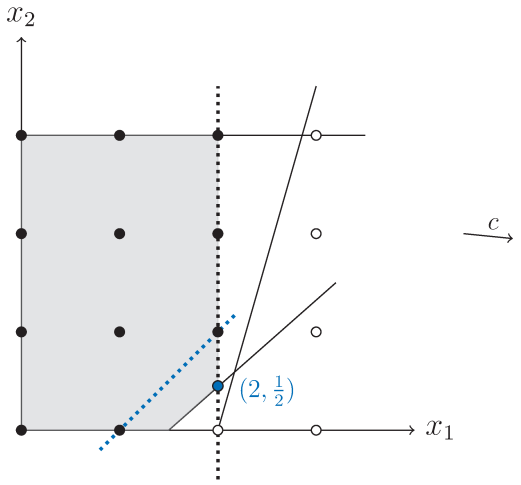


Figura 3.3: Algoritmo de planos de corte fracionário: iteração 2.

Iteração 3: A solução ótima obtida agora é $x_R^3 = (2, 1)$. Veja a Figura 3.4. Essa solução é inteira e, como a função objetivo da relaxação é idêntica à original, é também ótima para o IP. O algoritmo para.

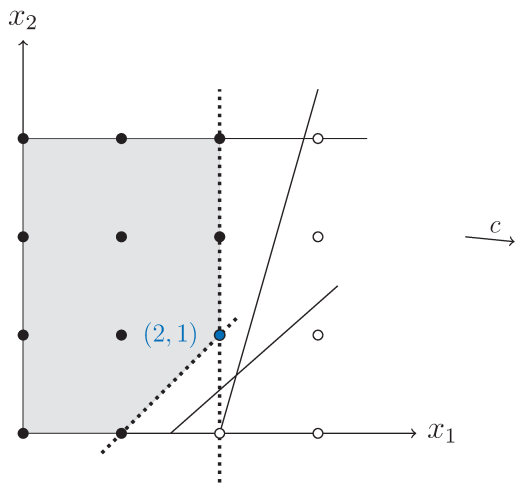


Figura 3.4: Algoritmo de planos de corte fracionário: iteração 3.

3.2 Algoritmos *relax-and-cut*

Um algoritmo *relax-and-cut* ataca um IP resolvendo, iterativamente, relaxações Lagrangianas do mesmo. Também, tais relaxações são, a cada iteração, refinadas. Uma relaxação Lagrangiana é refinada por meio da atualização dos multiplicadores e da adição de inequações válidas. Com essas operações, promovem-se modificações somente na função objetivo ou na função objetivo e no conjunto de restrições. Em um algoritmo *relax-and-cut*, os objetivos são: dar aos multiplicadores valores com os quais o melhor limitante alcançável com a relaxação Lagrangiana seja obtido, ou seja, resolver o PDL, e alterar a mesma de modo a tornar tal limitante mais preciso. Pela descrição acima, pode-se ver um algoritmo *relax-and-cut* como um algoritmo de planos de corte definido sobre relaxações Lagrangianas.

O termo *relax-and-cut*, em português, relaxe e corte, surge em [20], quando Escudero, Guignard e Malik assim denominam o algoritmo proposto para o *problema de ordem sequencial com restrições de precedência*. Este trabalho e o de Lucena [43, 44], com o *problema de Steiner em grafos*, foram os responsáveis pelo grande destaque ganho por essa classe de algoritmos. A ideia de combinar relaxação Lagrangiana com planos de corte, porém, já tinha sido testada anteriormente. Lucena [43] cita Fisher [21] como um

de seus precurssores. A referência mais importante para esta dissertação é a aplicação com sucesso de um algoritmo *relax-and-cut* ao SPP, por Cavalcante, de Souza e Lucena [15].

Apesar de o termo *relax-and-cut* comumente abranger toda a classe de algoritmos que combinam relaxação Lagrangiana com planos de corte [15], há uma clara distinção interna. Algoritmos como o proposto em [20] resolvem o PDL para depois adicionar inequações válidas. As desigualdades podem até ser encontradas durante a resolução do PDL, mas somente são adicionadas no fim. Em seguida, repete-se esse processo até que um critério de parada seja atingido. Outros como o apresentado em [43] solucionam o PRL, separam a solução ótima da relaxação e já fazem uso dos planos de corte identificados. Assim, inequações são adicionadas ao longo da resolução do PDL. Por essa diferença, os primeiros são chamados *delayed relax-and-cut* (DRC), os algoritmos que atrasam a adição de planos de corte, e os últimos são denominados *non-delayed relax-and-cut* (NDRC), os que não atrasam.

Nesta dissertação, um NDRC é empregado nos métodos aplicados ao SPP. No Algoritmo 3.4, exposto a seguir, mostra-se como um método desse tipo é comumente implementado. Quando uma inequação violada é identificada, esta é adicionada como um termo da função objetivo e não inserida no conjunto de restrições da relaxação. Isto é feito com o intuito de manter o PRL um problema de fácil resolução.

Abaixo, considere as três seguintes suposições. O problema P (3.1) é o IP a ser resolvido. O conjunto de soluções do mesmo é dado por $S = \{x \in \mathbb{Z}^n_+, Ax \leq b, Cx \leq d\}$. A relaxação Lagrangiana utilizada no algoritmo deriva-se da dualização do conjunto de restrições $Ax \leq b$. Também, note que a função objetivo da relaxação RP^k, resolvida na iteração k, contém, além do termo relativo às restrições $Ax \leq b$, a parcela $\mu^k(A'x^k_R - b')$. Esta parcela é gerada pelos planos de corte identificados no algoritmo. Os mesmos são adicionados à relaxação como inequações dualizadas e estão encapsulados no conjunto $A'x \leq b'$. Por esse motivo, usa-se a notação $z(\lambda^k, \mu^k)$, onde μ^k é o vetor de multiplicadores associados a tais inequações, para denotar o valor ótimo da relaxação.

É importante destacar uma vantagem que algoritmos *relax-and-cut* levam em relação a algoritmos de planos de corte fracionário: o problema de separação com o qual se lida em um dos primeiros é geralmente mais fácil que o atacado em um dos últimos. Isto, pois é comum que, enquanto em um algoritmo de planos de corte fracionário tal problema consista em separar uma solução $x_R \in \mathbb{R}^n_+$, em um algoritmo *relax-and-cut* o mesmo resuma-se a separar uma solução $x_R \in \mathbb{Z}^n_+$. Como comentado antes, o passo de separar uma solução x_R, onde inequações violadas são identificadas, é estratégico para um bom desempenho de um algoritmo de planos de corte.

A função $z(\lambda)$, que representa o valor ótimo de uma relaxação Lagrangiana de P, vista como uma função implícita de λ ($z(\lambda)$ é uma função de x), é côncava e linear por partes.

Algoritmo 3.4 NDRC

Inicialização: Escolha uma relaxação Lagrangiana RP^0, ou seja, determine

$$\begin{aligned} z(\lambda^0) = \min \quad & cx + \lambda^0(Ax - b) \\ & Cx \leq d \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

onde $\lambda^0 \geq 0$. Faça $\overline{z}^0 \Leftarrow \infty$, $\underline{z}^0 \Leftarrow -\infty$.

Iteração k:

Passo 1: Resolva a relaxação RP^k, obtendo $z(\lambda^k, \mu^k) = cx^k_R + \lambda^k(Ax^k_R - b) + \mu^k(A'x^k_R - b')$.

Passo 2: Teste se a solução x^k_R obtida é ótima para P. Isto ocorre se $x^k_R \in S$ e $cx^k_R + \lambda^k(Ax^k_R - b) + \mu^k(A'x^k_R - b') = cx^k_R$, pois, por consequência, tem-se que $\underline{z}^k = z(\lambda^k, \mu^k) = cx^k_R = \overline{z}^k$. Se sim, pare.

Passo 3: Se $z(\lambda^k, \mu^k) > \underline{z}^k$, faça $\underline{z}^k \Leftarrow z(\lambda^k, \mu^k)$.

Se $x^k_R \in S$ e $cx^k_R < \overline{z}^k$, faça $\overline{z}^k \Leftarrow cx^k_R$.

Refine a relaxação. Procure em F uma ou mais inequações $\pi x \leq \pi_0$ que não sejam satisfeitas por x^k_R, ou seja, com $\pi x^k_R > \pi_0$.

Se as inequações forem encontradas, a saber, $\pi^1 x \leq \pi^1_0, \ldots, \pi^l x \leq \pi^l_0$, determine valores iniciais para os multiplicadores $\alpha^1, \ldots, \alpha^l$ a serem associados às mesmas e atualize as estruturas que representam as inequações adicionadas à função objetivo da relaxação: $\mu = \begin{bmatrix} \mu \\ \alpha^1 \\ \vdots \\ \alpha^l \end{bmatrix}$ $A' = \begin{bmatrix} A' \\ \pi^1 \\ \vdots \\ \pi^l \end{bmatrix}$ $b' = \begin{bmatrix} b' \\ \pi^1_0 \\ \vdots \\ \pi^l_0 \end{bmatrix}$

Atualize os valores dos multiplicadores λ^k e μ^k, obtendo λ^{k+1} e μ^{k+1}.

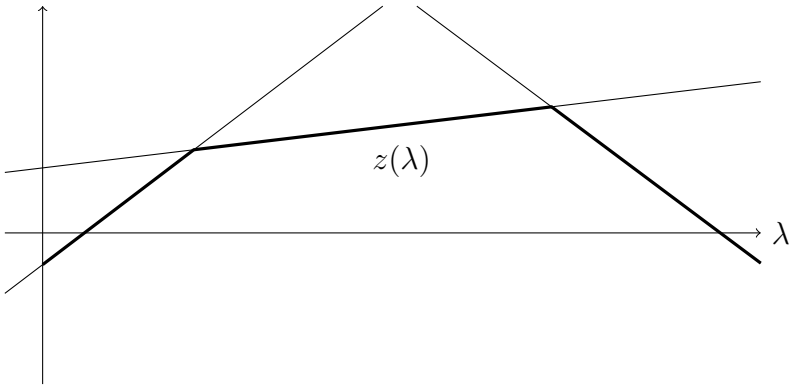


Figura 3.5: Esboço da função $z(\lambda)$.

A mesma é esboçada na Figura 3.5 (recomenda-se a publicação [32] de Guignard para mais detalhes sobre as características dessa função). Assim, para resolver o PDL, um dos objetivos de um algoritmo *relax-and-cut* para atacar P, métodos gerais para otimização convexa não-linear podem ser empregados. Neste trabalho, utiliza-se o *método do subgradiente* (MS). O mesmo é descrito a seguir.

3.2.1 Método do Subgradiente

É comum aplicar-se o *método de subida do gradiente* para maximizar funções côncavas. Esse método resume-se na seguinte estratégia: partindo de um ponto inicial, iterativamente dê passos na direção do *gradiente* da função para obter pontos de maior valor. O gradiente de uma função em um ponto determina a direção de maior crescimento da função a partir do ponto. A sequência de pontos assim obtida pode ser descrita por

$$x^{k+1} = x^k + \theta^k \nabla f(x^k),\ k = 0, 1, \ldots, \tag{3.3}$$

onde $f(x)$ é a função em questão e $\nabla f(x^k)$ é o gradiente de $f(x)$ em x^k. Para um passo θ^k suficientemente pequeno, tem-se que $f(x^{k+1}) \geq f(x^k)$. A tarefa mais complexa desse método é determinar o melhor tamanho de passo possível.

Uma função côncava, porém, pode não possuir gradiente em todos os pontos de seu domínio. A função $z(\lambda)$, que determina o valor ótimo de uma relaxação Lagrangiana de P (3.1), vista como uma função de λ, é um exemplo. Uma função com tal característica apresenta "bicos", onde não se pode obter a derivada. São nesses pontos, em que não é diferenciável, que a função não possui gradiente. Para maximizar funções não completamente diferenciáveis, não se pode aplicar o método de subida do gradiente. É possível, no entanto, empregar um método semelhante.

Um conceito menos restritivo que um gradiente é um *subgradiente*. Em todo ponto para o qual está definida, uma função côncava possui um conjunto não vazio de subgradientes. Um subgradiente de uma função é um vetor, contudo, que não representa necessariamente uma direção de crescimento da função. De posse desse conceito, pode-se empregar um método semelhante ao de subida do gradiente para maximizar funções côncavas que não são completamente diferenciáveis. Agora, usando a direção de um subgradiente.

No método acima, escolhe-se comumente, dentre o conjunto de subgradientes da função em um ponto, o subgradiente que represente a melhor direção segundo algum critério. Held *et al.* [38], contudo, enxergam uma seleção desse tipo como um processo muito custoso computacionalmente e propõem o uso, de modo corajoso, de um subgradiente arbitrário, independendo se há outros melhores. Os autores sugerem, também, que não se faça muito esforço no cálculo do tamanho de passo. Isto, por dois motivos. O primeiro é que tal cálculo representa, ao método, um considerável ônus em tempo de execução. O segundo

é que a direção seguida não é garantidamente favorável. Na configuração proposta pelos autores, o tamanho de passo é determinado por valores prontamente disponíveis (com ressalva em relação a um) e sofre ajustes simples de acordo com o comportamento da função. O método resultante é o MS [37], que teve sua eficiência comprovada em [38].

Aplicado à função $z(\lambda)$, o MS consiste em determinar, para os multiplicadores, uma sequência de valores atualizados segundo a fórmula

$$\lambda^{k+1} = \max\{0, \lambda^k + \theta^k s^k\},\ k = 0, 1, \ldots. \tag{3.4}$$

Isto, de modo que $z(\lambda^k)$ convirja para z_D. Aqui, assim como na fórmula (3.3), θ^k representa um tamanho de passo. O vetor s^k é um subgradiente de $z(\lambda)$ em λ^k. A cada iteração do MS, então, um passo é dado na direção de um subgradiente, sempre respeitando $\lambda \geq 0$.

Suponha que o conjunto de soluções de P (3.1) é dado por $S = \{x \in \mathbb{Z}^n_+, Ax \leq b, Cx \leq d\}$ e que a relaxação Lagrangiana de P associada à função $z(\lambda)$ seja derivada da dualização das inequações $Ax \leq b$. O subgradiente s^k acima é, então, dado pelo vetor $(Ax^k_R - b)$, onde x^k_R é uma solução ótima para o PRL definido para λ^k. Esse vetor é, de fato, um subgradiente da função $z(\lambda)$ em λ^k. A validade do mesmo é justificada a seguir.

Definição 3.1. Dada uma função côncava $f(x)$ definida em $\mathbb{R}^n$, um vetor $u \in \mathbb{R}^n$ é um subgradiente de $f(x)$ no ponto $x' \in \mathbb{R}^n$ se, $\forall x \in \mathbb{R}^n, f(x) \leq f(x') + u(x - x')$.

Para o vetor $(Ax^k_R - b)$, tem-se que, $\forall \lambda \geq 0$,

$$\begin{aligned}
z(\lambda) \leq\ & cx + \lambda(Ax - b),\ \forall x \in \{x \in \mathbb{Z}^n_+, Cx \leq d\},\\
z(\lambda) \leq\ & cx^k_R + \lambda(Ax^k_R - b),\\
z(\lambda) \leq\ & cx^k_R + \lambda^k(Ax^k_R - b) - \lambda^k(Ax^k_R - b) + \lambda(Ax^k_R - b),\\
z(\lambda) \leq\ & z(\lambda^k) + (Ax^k_R - b)(\lambda - \lambda^k).
\end{aligned}$$

Pela Definição 3.1, esse vetor é um subgradiente de $z(\lambda)$ no ponto λ^k.

A convergência de $z(\lambda^k)$ para z_D depende de uma escolha adequada de tamanhos de passo. Held *et al.* [38] sumarizam que, para que a mesma ocorra, é necessário que

$$\lim_{k\to\infty} \theta^k = 0. \tag{3.5}$$

Ainda, esta condição e a de que

$$\lim_{n\to\infty} \sum_{k=0}^{n} \theta^k = \infty \tag{3.6}$$

são suficientes para assegurar tal feito. O principal resultado para a garantia de convergência é o seguinte.

Proposição 3.1. *Se o tamanho de passo θ^k é suficientemente pequeno, a atualização feita na fórmula (3.4) garante que a distância entre λ^{k+1} e λ^* é menor que a distância entre λ^k e λ^*, sendo λ^* uma solução ótima para o PDL.*

O resultado acima é válido ainda que o subgradiente s^k não represente uma direção de crescimento de $z(\lambda)$. O mesmo é justificado pela observação a seguir.

Sendo s^k um subgradiente de $z(\lambda)$ em λ^k, tem-se que

$$s^k(\lambda^* - \lambda^k) \geq z(\lambda^*) - z(\lambda^k),$$

onde λ^* pertence ao conjunto das soluções ótimas para o PDL. Se λ^k não está também nesse conjunto, então

$$s^k(\lambda^* - \lambda^k) \geq z(\lambda^*) - z(\lambda^k) = z_D - z(\lambda^k) > 0.$$

Assim, o subgradiente s^k faz um ângulo agudo com o vetor que une λ^k a λ^*. Logo, se θ^k é suficientemente pequeno, o ponto λ^{k+1} está mais próximo do ponto λ^* do que o ponto λ^k. Veja a ilustração na Figura 3.6.

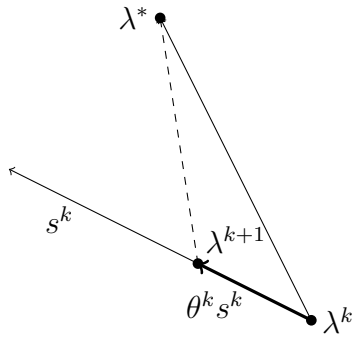


Figura 3.6: MS aplicado a $z(\lambda)$: convergência de λ^k para um ponto ótimo para o PDL.

A dificuldade do MS está, então, na escolha dos tamanhos de passo. Uma opção tipicamente utilizada é o tamanho de passo

$$\theta^k = \epsilon^k \, \frac{\hat{z} - z(\lambda^k)}{|s^k|^2} \tag{3.7}$$

a cada iteração k, onde $0 < \epsilon^k \leq 2$ e $\hat{z} < z_D$, sendo $z(\lambda)$ a função em questão. Com isso, garante-se que $z(\lambda^k)$ convirja para $\hat{z}$ ou que um ponto λ^k tal que $z(\lambda^k) \geq \hat{z}$ seja obtido. Poljak [54] mostrou, ainda, que, se z_D é usado no lugar de $\hat{z}$, a convergência de λ^k para o conjunto de soluções ótimas do PDL é assegurada.

À opção apresentada acima, porém, está intrínseco um grande desafio: escolher o valor de $\hat{z}$. Obviamente, não se conhece z_D. Também, é improvável que se saiba um valor próximo e menor que o mesmo. O motivo é que, se isso ocorre, tem-se, logo de início, um bom limitante inferior para o valor ótimo z de P. Como valor de $\hat{z}$, então, utiliza-se

geralmente o melhor limitante superior $\overline{z}$ conhecido para z, embora se tenha $\overline{z} \geq z_D$. No entanto, com o uso de tal limitante, a condição (3.5) pode não ser satisfeita. O parâmetro ϵ^k é usado para corrigir isso. Os seus valores vão sendo diminuídos ao longo do processo, de forma que tendam a 0. Consequentemente, $\theta^k \to 0$. Agora, porém, a condição (3.6) é violada. É possível, portanto, que a convergência para o conjunto de soluções ótimas do PDL seja comprometida. A correção é feita mesmo assim. Na prática, essa violação não parece ser crítica, pois o desempenho do método é muito bom.

A aplicação do MS à função $z(\lambda)$ descrita acima está sintetizada no algoritmo abaixo.

Algoritmo 3.5 MS aplicado à função $z(\lambda)$

Inicialização: Determine valores iniciais para os multiplicadores, ou seja, escolha λ^0.
Iteração k:
Passo 1:
Resolva o PRL associado a λ^k, obtendo $z(\lambda^k) = cx_R^k + \lambda^k(Ax_R^k - b)$.
Passo 2:
Atualize os valores dos multiplicadores. Faça $\lambda^{k+1} = \max\{0, \lambda^k + \theta^k(Ax_R^k - b)\}$.

Mais uma vez, determinar os tamanhos de passo é a tarefa mais complexa do MS. Vários autores apontam a mesma como um assunto ainda não perfeitamente compreendido.

3.3 *Branch-and-bound*

Um *branch-and-bound* é uma técnica utilizada para resolver (garantidamente) um IP. Essa técnica representa um *algoritmo exato*. Isto significa que, com o uso de um *branch-and-bound*, uma solução ótima do IP é garantidamente encontrada ou é provado que o IP é inviável ou ilimitado.

Um *branch-and-bound* é uma abordagem enumerativa que se baseia no conceito de *dividir para conquistar*. Neste conceito, a ideia é dividir o problema em partes menores (e mais fáceis), resolvê-las e combinar os resultados, gerando a solução final. No entanto, as partes podem, ainda, por sua vez, representar problemas difíceis. O procedimento é, então, repetido. Quando se consegue resolver um subproblema diretamente, não é mais necessário dividí-lo.

Considere que um *branch-and-bound* é aplicado ao problema P (3.1). Em uma divisão de P, o conjunto de soluções S é decomposto em conjuntos S_i, $i = 1, \ldots, k$, tais que $\bigcup_{i=1}^{k} S_i = S$. Associado a um conjunto S_i, está o subproblema P_i, dado por

$$\begin{aligned} z_i = \min \quad & cx \\ & x \in S_i. \end{aligned} \tag{3.8}$$

Observe que $z = \min_{i=1,\ldots,k} z_i$. Desta forma, pode-se combinar os resultados das partes para gerar a solução final.

A técnica faz uso, então, da divisão dos subproblemas P_i, até que, por exemplo, estes sejam fáceis de resolver. Tal procedimento origina o que se pode interpretar como uma árvore (veja a Figura 3.7). Em geral, não é factível proceder com a divisão até que os conjuntos S_i sejam muito pequenos. Isto, pois, para um conjunto S de tamanho moderado, a quantidade de subproblemas criados é muito grande. É importante, portanto, saber quando não é mais necessário ramificar um nó da árvore. Assim, revela-se o conceito de *poda* na árvore.

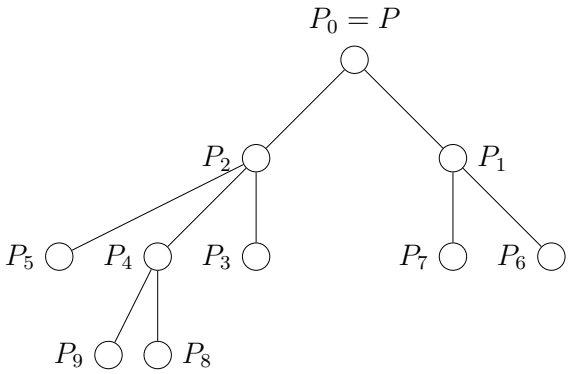


Figura 3.7: Árvore de divisões.

A finitude da árvore produzida em um *branch-and-bound* é garantida pelo modo como é feita a divisão de um problema. O conjunto de soluções de um subproblema criado tem que representar, de fato, uma redução do conjunto de soluções do problema que o originou. Embora não precise (reveja, acima, a descrição da divisão de um problema), a divisão de um problema geralmente define uma partição do conjunto de soluções do mesmo.

Em um *branch-and-bound*, usam-se relaxações para atacar os subproblemas P_i, pois ainda estes são comumente problemas difíceis. Pode-se descrever um *branch-and-bound* para P como uma instância do algoritmo geral baseado em relaxações. Isto é feito no Algoritmo 3.6. No mesmo, L é uma lista de subproblemas P_i.

Com as correspondências estabelecidas a seguir, pode-se ver o Algoritmo 3.6 como uma instância do algoritmo geral baseado em relaxações. O Passo 1 de uma iteração do mesmo está vinculado ao Passo 2 de uma iteração do algoritmo geral. Isto no sentido em que é nesses onde se determina se uma solução ótima de P foi obtida. O Passo 3, por sua vez, está em correspondência ao Passo 1 do algoritmo geral. Note que, no primeiro, é resolvida uma relaxação de P^k, não de P (é uma relaxação de P somente quando $k = 0$). No entanto, para os RP^k associados aos P^k que são folhas da árvore de divisões, tem-se

que $\bigcup_k T^k \supseteq \bigcup_k S^k = S$. Portanto, resolver RP^k é parte do processo de solucionar uma relaxação de P, o que é feito no referido passo do algoritmo geral. Ainda, no Passo 5, promove-se um refinamento dessa subjacente relaxação de P. Os subproblemas $P_{i_m}, m = 1, \ldots, n$, criados devem estar associados a relaxações RP_{i_m} tais que $\bigcup_{m=1}^{n} T_{i_m} \subset T^k$. Assim, estão relacionados, também, os passos 4 e 5 e o Passo 3 do algoritmo geral.

Algoritmo 3.6 *Branch-and-bound*

Inicialização: Faça $\mathrm{P}_0 \Leftarrow \mathrm{P}$, $L \Leftarrow \{\mathrm{P}_0\}$ e $\overline{z} \Leftarrow \infty$.

Iteração k:

Passo 1: Teste se L está vazia. Se sim, pare; a solução que deu origem a $\overline{z}$ é ótima para P.

Passo 2: Escolha um subproblema em L. Denote-o por P^k, com conjunto de soluções S^k. Retire P^k de L.

Passo 3: Resolva uma relaxação RP^k (f^k,T^k) de P^k e obtenha $z_R^k = f(x_R^k)$.

Passo 4: Se P^k é inviável, vá para a próxima iteração.

Se $z_R^k \geq \overline{z}$, vá para a próxima iteração. (Aqui, z_R^k pode ser substituído por $\underline{z}^k$, onde $\underline{z}^k$ é um limitante inferior para z^k, o valor ótimo de P^k.)

Se $x_R^k \in S^k$ e $cx_R^k < \overline{z}$, faça $\overline{z} \Leftarrow cx_R^k$.

Se a solução ótima x^k de P^k é encontrada, vá para a próxima iteração.

Passo 5: Divida P^k em $\mathrm{P}_{i_m}, m = 1, \ldots, n$, e adicione esses problemas a L.

A cada iteração k do algoritmo acima, são feitos, no Passo 4, três testes que determinam se o Passo 5 deixa de ser executado. Isto implica não haver divisão para o subproblema P^k e existir, na árvore de divisões gerada, uma poda no nó correspondente. Tais testes, no entanto, dependem dos limitantes conhecidos para z e para z^k. Assim, pode-se dizer que um *branch-and-bound*, em português, ramificar e limitar, é uma técnica que gera uma árvore de divisões e calcula limitantes para prover podas na mesma.

Uma poda na árvore de divisões acontece, então, se uma das três seguintes condições é satisfeita.

(i) Se P^k é inviável, ocorre uma poda *por inviabilidade*.

(ii) Se é conhecido um limitante inferior $\underline{z}^k$ para z^k tal que $\underline{z}^k \geq \overline{z}$, onde $\overline{z}$ é um limitante superior para z, ocorre uma pode *por limitante*.

(iii) Se o valor ótimo de P^k é encontrado, ocorre uma poda *por otimalidade*.

Observe que, sendo $\underline{z}^k = \infty$ em caso de P^k inviável, a condição (i) é englobada pela condição (ii).

Quanto aos limitantes produzidos em um *branch-and-bound*, há duas consequências importantes. Quando um subproblema P_i é considerado em uma iteração (o problema P

é o primeiro), são gerados limitantes inferior e superior para seu valor ótimo. O limitante inferior é obtido através de uma relaxação do subproblema. A consequência importante desse limitante é a seguinte.

Proposição 3.2. *Dado um subproblema P_i, um limitante inferior $\underline{z_i}$ para z_i é válido para todo z_l tal que P_l é um subproblema derivado de P_i.*

O limitante superior, por sua vez, é dado por uma solução viável para o subproblema. Essa solução é produzida por uma heurística primal ou por uma transformação de uma solução ótima de uma relaxação. Analogamente, a consequência útil desse limitante é a seguinte.

Proposição 3.3. *Dado um subproblema P_i, um limitante superior $\overline{z_i}$ para z_i é válido para todo z_l tal que P_l é um subproblema do qual P_i é derivado.*

Corolário 3.4. *Dado um subproblema P_i, um limitante superior $\overline{z_i}$ para z_i é válido para z.*

A versão mais comumente utilizada de um *branch-and-bound* tem as seguintes especificações: emprega relaxações lineares e aplica-se a um IP $0-1$. Apresenta-se, então, abaixo, essa versão do algoritmo. Para ilustrar o seu funcionamento, mostra-se, também, um exemplo de aplicação a um problema de três variáveis. Considere, para a exposição a seguir, que as variáveis do problema P (3.1), o problema a ser resolvido, são binárias.

Algoritmo 3.7 *Branch-and-bound* baseado em relaxação linear para um IP $0-1$

Inicialização: Faça $\mathrm{P}_0 \Leftarrow \mathrm{P}$, $L \Leftarrow \{\mathrm{P}_0\}$ e $\overline{z} \Leftarrow \infty$.

Iteração k:

Passo 1: Teste se L está vazia. Se sim, pare; a solução que deu origem a $\overline{z}$ é ótima para P.

Passo 2: Escolha o primeiro subproblema em L. Denote-o por P^k, com conjunto de soluções S^k. Retire P^k de L.

Passo 3: Resolva a relaxação linear de P^k e obtenha $z_R^k = cx_R^k$.

Passo 4: Se P^k é inviável, vá para a próxima iteração.

Se $z_R^k \geq \overline{z}$, vá para a próxima iteração. (Aqui, z_R^k pode ser substituído por $\underline{z}^k$, onde $\underline{z}^k$ é um limitante inferior para z^k, o valor ótimo de P^k.)

Se $x_R^k \in \mathbb{B}^n$, faça o teste a seguir e vá para a próxima iteração; a solução x_R^k é ótima para P^k. Se $cx_R^k < \overline{z}$, faça $\overline{z} \Leftarrow cx_R^k$.

Passo 5: Considere as variáveis cujos valores não são inteiros em x_R^k. Seja x_j a variável com valor mais fracionário dentre essas. (O valor de x_j é o mais distante de ser inteiro, 0 ou 1.) Divida P^k em P_{i_1} e P_{i_2}, onde $S_{i_1} = S^k \cap \{x \in \mathbb{B}^n, x_j = 1\}$ e $S_{i_2} = S^k \cap \{x \in \mathbb{B}^n, x_j = 0\}$. Adicione esses problemas a L de forma que P_{i_1} e P_{i_2} passem a ser os primeiros da lista, nessa ordem.

Para ratificar a corretude do algoritmo apresentado, observe o seguinte. As condições testadas no Passo 4 – condições (i), (ii) e (iii) descritas acima – implicam que, para todo nó em que há uma poda, o subproblema P^k correspondente não pode ter solução melhor (de menor custo) que a originária do limitante superior $\overline{z}$ estabelecido ao término da iteração. Assim, ao fim do processo, $\overline{z}$ está vinculado a uma solução de custo mínimo e, portanto, ótima para P.

Exemplo 3.2. Considere que P, o problema a ser resolvido no Algoritmo 3.7, é dado pelo IP $0-1$ abaixo.

$$\begin{aligned} z = \min \quad & 2x_1 + x_2 - 2x_3 \\ & 0{,}7x_1 + 0{,}5x_2 + x_3 \geq 1{,}8 \\ & x \in \mathbb{B}^n \end{aligned}$$

A execução do algoritmo é como a seguir.

Inicialização: A lista L tem inicialmente o subproblema P_0 =P. Também, $\overline{z} = \infty$.

Iteração 1: O subproblema P_0 é escolhido e a relaxação linear do mesmo (relaxação linear de P) é resolvida. A solução obtida é $x_1 = 0{,}428\,5$, $x_2 = 1$, $x_3 = 1$, com valor $-0{,}143$. Nenhuma das condições testadas no Passo 4 é satisfeita. Assim, não é possível realizar uma poda no nó. Logo, P_0 tem de ser dividido. A variável selecionada no Passo 5 é x_1, pois é a única com valor fracionário. São gerados dois subproblemas: P_1, associado à restrição $x_1 = 1$, e P_2, vinculado à restrição $x_1 = 0$. A Figura 3.8 ilustra a situação. A lista de subproblemas é dada, agora, por $L = \{\mathrm{P}_1, \mathrm{P}_2\}$.

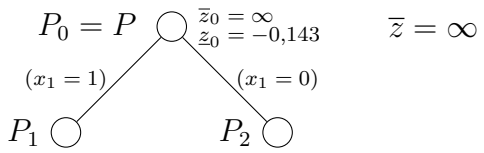


Figura 3.8: *Branch-and-bound*: iteração 1.

Iteração 2: Aqui, P_1 é o subproblema selecionado. Resolvendo a relaxação linear do mesmo, obtém-se a solução $x_1 = 1$, $x_2 = 0{,}2$, $x_3 = 1$ (o valor da variável x_1 está fixado). Esta solução tem valor 0,2. Novamente, não é possível realizar poda no nó. Assim, procede-se com a divisão de P_1. Usando a variável x_2, produzem-se dois subproblemas: P_3, com $x_2 = 1$, e P_4, com $x_2 = 0$. Tem-se, ao fim desta iteração, $L = \{\mathrm{P}_3, \mathrm{P}_4, \mathrm{P}_2\}$. A árvore gerada até então está representada na Figura 3.9.

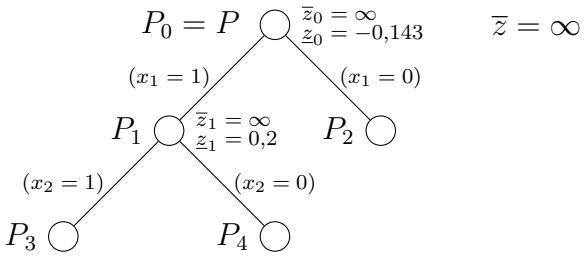


Figura 3.9: *Branch-and-bound*: iteração 2.

Iteração 3: Nesta iteração, P_3, o primeiro subproblema em L, é escolhido. A solução $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, com valor 1, é ótima para a relaxação linear do mesmo. Além disso, tal solução é inteira. Assim, a terceira condição testada no Passo 4 é satisfeita e há uma poda no nó correspondente. Também, tem-se, agora, pelo Corolário 3.4, $\overline{z} = 1$. Os novos limitantes conhecidos são mostrados na Figura 3.10.

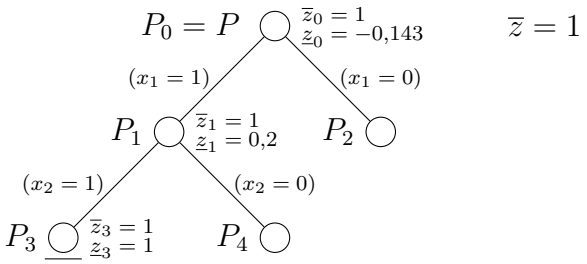


Figura 3.10: *Branch-and-bound*: iteração 3.

Iterações 4, 5 e 6: Nas duas próximas iterações, são considerados os subproblemas P_4 e P_2 respectivamente. Nesses subproblemas, uma dentre as variáveis x_1 e x_2 está fixada em 0. Com isso, a (única) restrição presente em ambos não é satisfeita. Assim, os mesmos são inviáveis. O primeiro teste realizado no Passo 4 promove podas nos nós correspondentes.

Na sexta e última iteração, a lista L está vazia. O algoritmo, então, para. A solução $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, que deu origem ao limitante $\overline{z} = 1$, é ótima para P. A configuração final da árvore de divisões produzida é apresentada na Figura 3.11.

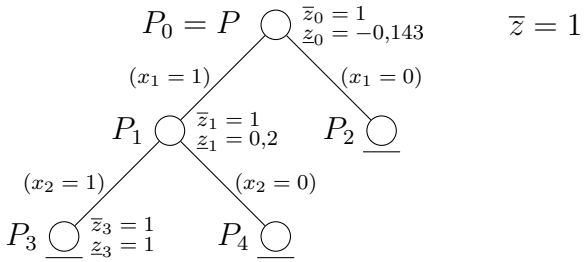


Figura 3.11: *Branch-and-bound*: iteração 6.

O exemplo dado acima é de fim meramente didático. O motivo é que os valores de todas as três variáveis do IP considerado podem ser facilmente fixados. Para ilustrar, tome a variável x_1. Note que essa variável não pode valer 0. Se isso ocorre, a (única) restrição do IP não pode ser respeitada. Veja que o mesmo vale para as variáveis x_2 e x_3. Assim, a solução $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, é a única viável.

Neste exemplo, contudo, são vistas duas situações de poda em um nó da árvore de divisões. Nas iterações 4 e 5, a condição (i) é satisfeita e ocorre uma poda por inviabilidade. Na iteração 3, uma solução ótima de P_3 é encontrada. Pela condição (iii), acontece uma poda por otimalidade.

Em um *branch-and-bound*, existem passos estratégicos. No Algoritmo 3.7, esses são implementados de uma maneira. Há, porém, outras alternativas. Tais passos são explorados abaixo.

A cada iteração de um *branch-and-bound*, ocorre a *escolha de um subproblema*. No Algoritmo 3.7, seleciona-se sempre o subproblema mais recentemente criado. Quando existem dois, o associado à restrição $x_j = 1$, para alguma variável x_j, é preferido. Esta é a estratégia *profundidade primeiro*. É através de um bom limitante superior $\overline{z}$ que se pode realizar podas na árvore de divisões. Aprofundar-se rapidamente na mesma para descobrir soluções viáveis é, então, uma boa iniciativa. É isto que prega a estratégia.

Outra conduta possível é escolher sempre o subproblema com menor limitante inferior. Com isso, nunca um subproblema P_i para o qual $\underline{z}_i > z$ é dividido. Em outras palavras, nunca ocorre ramificação em um nó onde a árvore de divisões poderia ser podada. Observe que $\underline{z} = \min_i \{\underline{z}_i\}$.

A primeira das opções de escolha acima possui uma vantagem no tocante à próxima tarefa ressaltada. No Algoritmo 3.7, o uso da mesma permite que o *cálculo de limitantes* (*bounding*) seja, em muitos casos, fácil. Considere que um subproblema vinculado a uma restrição $x_j = 1$ foi selecionado no Passo 2. Empregando a relaxação linear, é possível utilizar a base ótima do subproblema processado anteriormente e lidar com a simples mudança de limitante de uma variável na reotimização.

O passo principal em um *branch-and-bound* é a obtenção de limitantes para o valor ótimo de um subproblema. Para cumprir essa tarefa, outras relaxações que não a linear também podem ser empregadas. Nesta dissertação, por exemplo, avalia-se um algoritmo baseado em relaxação Lagrangiana. Ainda, pode-se refinar uma relaxação para gerar melhores limitantes. A adição de cortes em um *branch-and-bound* é discutida na próxima seção. Quanto melhores são os limitantes produzidos nos nós da árvore de divisões, mais podas são realizadas e menos subproblemas são criados.

Na Passo 5 do Algoritmo 3.7, é determinada a *divisão de um subproblema*, ou seja, a ramificação da árvore de divisões (*branching*). A partir dos possíveis valores de uma variável, 0 ou 1, são gerados dois novos subproblemas. Há, porém, outras maneiras de fazer a ramificação. Quando existem restrições do tipo $\sum_{j \in J'} x_j \leq 1$ (conhecidas por GUB, do inglês *generalized upper bound*), pode-se, por exemplo, definir dois subconjuntos distintos de variáveis com índice em J' para serem anuladas. Criam-se, então, dois subproblemas, cada um associado a um subconjunto. É possível, ainda, produzir mais de dois ramos a partir de um nó.

A primeira opção de divisão acima possui a desvantagem de tender a gerar ramos de tamanhos bastante diferentes na árvore de divisões. Com a segunda, os ramos produzidos são, geralmente, de tamanhos equilibrados.

Existem outros aspectos importantes na implementação de um *branch-and-bound*. Um *pré-processamento*, onde eliminam-se restrições e variáveis redundantes e derivam-se limitantes mais precisos para as últimas, tem bastante influência no resultado final. A *fixação de variáveis* também costuma ser relevante. Isto consiste em detectar que, em qualquer solução ótima, $x_j = u_j$ ou $x_j = l_j$, onde $l_j \leq x_j \leq u_j$.

Por fim, é importante comentar que o *branch-and-bound* é também nomeado como *enumeração implícita*. (Alguns autores, porém, tratam enumeração implícita como uma classe especial de algoritmos de *branch-and-bound* definida para IPs $0-1$ [26].) Observe que o que ocorreria na divisão de subproblemas se esta fosse levada à exaustão seria uma enumeração de todas as soluções do IP inicial. A que acontece, contudo, é implícita, pois apenas parte (na verdade, uma muito pequena parte) das soluções são avaliadas explicitamente.

3.4 *Branch-and-cut*

Para solucionar um IP, um *branch-and-bound* resolve relaxações de IPs mais restritos, que representam subproblemas do mesmo. Uma relaxação de um IP, por sua vez, pode ser refinada por meio da adição de inequações válidas. Uma integração de um *branch-and-bound* com planos de corte, portanto, apresenta-se bastante interessante. Afinal, um *branch-and-bound* usa relaxações para obter limitantes e, possivelmente, soluções viáveis

para um IP. A adição de inequações válidas pode melhorar a qualidade de ambos.

Uma primeira abordagem que faz a integração citada já foi bastante utilizada em décadas anteriores. Na verdade, esta abordagem não integra, mas sim combina as ideias. A distinção feita aqui é esclarecida adiante. A abordagem pode ser descrita como a seguir. Aplique um algoritmo de planos de corte ao IP. Se uma solução ótima do IP é obtida, não há mais nada a fazer. Em caso contrário, execute um *branch-and-bound*. Na execução do último, contudo, aproveitando o progresso feito na etapa anterior.

A abordagem acima é conhecida como *cut-and-branch*, em português, cortar e ramificar, e está formalizada no Algoritmo 3.8. Considere que P (3.1) é o IP a ser resolvido.

Algoritmo 3.8 *Cut-and-branch*

Etapa 1: Execute um algoritmo de planos de corte para P. Se uma solução ótima do mesmo é obtida, pare. Senão, adicione ao conjunto de restrições de P os planos de corte identificados.

Etapa 2: Execute um *branch-and-bound* para P. Como entrada, passe a melhor solução viável encontrada e o melhor limitante inferior obtido.

Observe que, neste algoritmo, o impacto dos planos de corte adicionados é repassado ao *branch-and-bound*. Durante a execução do último, porém, as relaxações de P não são mais refinadas. Assim, as duas ideias estão combinadas, mas atuando isoladamente. Por isso, não há, de fato, uma integração. Esta abordagem é usada, por exemplo, por Nemhauser e Sigismondi [48] para o NPP.

Para alguns problemas, atacar IPs que os modelam é uma tarefa que inclui uma dificuldade adicional. Um exemplo ocorre para o TSP. O modelo usual do TSP como um IP tem um número exponencial de restrições. Logo, é impraticável usá-las todas explicitamente. Para atacar tal IP, um algoritmo de planos de corte onde as inequações sejam adicionadas dinamicamente é, então, imprescindível. Ainda, como consequência, em um *branch-and-bound*, não se tem acesso ao modelo completo do IP. Assim, ao final do mesmo, a solução obtida pode ser, apesar de inteira, inviável para o problema.

Em tais situações, é aplicado um método semelhante ao mostrado acima, no sentido que combina um *branch-and-bound* com planos de corte. Crowder e Padberg [16] propõem uma instância desse método para o TSP. A mesma está exposta no Algoritmo 3.9 (para mais detalhes, veja [16]).

Uma outra abordagem realiza, de fato, a integração de um *branch-and-bound* com planos de corte. A mesma pode ser descrita como a seguir. Execute um *branch-and-bound* para o IP da seguinte maneira. A cada subproblema, aplique um algoritmo de planos de corte.

Em [52], Padberg e Rinaldi utilizam essa abordagem para atacar o TSP. Em [51], os mesmos autores a denominam *branch-and-cut*. A abordagem é formalizada no Algoritmo

3.10. Considere que P (3.1) é o IP em questão.

Algoritmo 3.9 Combinação de *branch-and-bound* com planos de corte para o TSP

Inicialização: O TSP é considerado em seu modelo usual como um IP, mas não completo.

Iteração k:

Etapa 1: Execute um algoritmo de planos de corte baseado em relaxação linear para o TSP. Se uma solução ótima do mesmo é obtida, pare. Senão, adicione ao conjunto de restrições do TSP os planos de corte identificados.

Etapa 2: Execute um *branch-and-bound* baseado em relaxação linear para o TSP. Como entrada, passe a melhor solução viável encontrada e o melhor limitante inferior obtido. Se uma solução ótima do TSP é descoberta, pare.

Algoritmo 3.10 *Branch-and-cut*

Inicialização: Faça $\mathrm{P}_0 \Leftarrow \mathrm{P}$, $L \Leftarrow \{\mathrm{P}_0\}$ e $\overline{z} \Leftarrow \infty$.

Iteração k:

Passo 1: Teste se L está vazia. Se sim, pare; a solução que deu origem a $\overline{z}$ é ótima para P.

Passo 2: Escolha um subproblema em L. Denote-o por P^k, com conjunto de soluções S^k. Retire P^k de L.

Passo 3: Resolva uma relaxação RP^k (f^k,T^k) de P^k e obtenha $z_R^k = f(x_R^k)$.

Passo 4: Se P^k é inviável, vá para a próxima iteração.

Se $z_R^k \geq \overline{z}$, vá para a próxima iteração. (Aqui, z_R^k pode ser substituído por $\underline{z}^k$, onde $\underline{z}^k$ é um limitante inferior para z^k, o valor ótimo de P^k.)

Se $x_R^k \in S^k$ e $cx_R^k < \overline{z}$, faça $\overline{z} \Leftarrow cx_R^k$.

Se a solução ótima x^k de P^k é encontrada, vá para a próxima iteração.

Passo 5: Procure em F uma ou mais inequações $\pi x \leq \pi_0$ que não sejam satisfeitas por x_R^k, ou seja, com $\pi x_R^k > \pi_0$.

Se as inequações foram encontradas, adicione-as à relaxação RP^k, obtendo novos T^k e f^k, e vá para o Passo 3.

Passo 6: Divida P^k em $\mathrm{P}_{i_m}, m = 1, \ldots, n$, e adicione esses problemas a L.

Há um aspecto fundamental para a implementação de um *branch-and-cut*. Devem-se adicionar, no algoritmo, somente *planos de corte globalmente válidos*, ou seja, válidos para todos os subproblemas. Assim, é possível manter um *pool* de inequações que é compartilhado por todos os nós da árvore de divisões. O mesmo não ocorreria se cortes não globalmente válidos fossem utilizados. Teria que se guardar, para cada subproblema, as inequações identificadas. Isto representaria requisitos gigantescos de memória. Um

exemplo são os cortes de Gomory (citados na Seção 3.1). Esses cortes são combinações lineares de, dentre outras, restrições que limitam valores de variáveis em um subproblema específico. Por exemplo, $x_j = 0$ ou $x_j = 1$. Obviamente, as mesmas não são válidas para toda a árvore. O uso de cortes globais ainda garante a seguinte interessante propriedade: enquanto um subproblema é processado, as relaxações de todos os outros são refinadas.

Como em um *branch-and-bound*, existem passos estratégicos em um *branch-and-cut*. Há diferentes opções para a escolha e a divisão de um subproblema e para o cálculo de limitantes. Aqui, porém, o último é ainda mais importante. Em geral, em comparação com um *branch-and-bound*, é feito um esforço bem maior no processamento de um subproblema em um *branch-and-cut*. Busca-se, com mais esmero, limitantes inferiores precisos e, possivelmente, soluções viáveis de baixo custo.

Em um *branch-and-bound*, a resolução das relaxações é o maior responsável pelo tempo de execução do algoritmo. Em um *branch-and-cut*, essa responsabilidade é mais bem dividida com a geração de cortes. Para um bom desempenho do último, portanto, deve-se ter, no Passo 5, um procedimento eficiente para a identificação de inequações violadas.

Capítulo 4

Algoritmos baseados em relaxação Lagrangiana para o SPP

Algoritmos baseados em relaxação Lagrangiana têm se mostrado uma forma eficiente de atacar vários problemas $\mathcal{NP}$-Difíceis. Como dito antes, uma razão para o sucesso do emprego da relaxação Lagrangiana é que muitos problemas difíceis são compostos por outros fáceis adicionados de restrições complicadoras. Alguns exemplos são os seguintes. Lucena [45] e Andrade, Lucena e Maculan [1] evidenciam uma subestrutura de *árvore geradora mínima* que compõe, respectivamente, os problemas de *árvore de Steiner* e *árvore geradora mínima com restrição de grau*. Em [22], Fisher relaxa o *problema de roteamento de veículos* para passar a lidar com o *problema da k-árvore mínima com uma restrição de grau no depósito*. Uma referência clássica é a do TSP [37], que também tem escondida uma estrutura de árvore geradora mínima (mais especificamente uma 1-árvore).

Os algoritmos *relax-and-cut*, em especial, ganharam bastante destaque com as contribuições de Lucena [43, 44] e Escudero, Guignard e Malik [20]. A combinação de relaxação Lagrangiana com planos de corte, contudo, já tinha sido empregada em trabalhos anteriores [4, 27, 21]. Como mencionado antes, nas contribuições [43, 44] e [20], tais algoritmos foram aplicados, respectivamente, ao problema de Steiner em grafos e ao problema de ordem sequencial com restrições de precedência. Desde então, algoritmos *relax-and-cut* têm sido usados com muito êxito para atacar vários problemas $\mathcal{NP}$-Difíceis: o *problema da clique ponderada*, por Hunting, Faigle e Kern [42]; o SPP, por Cavalcante, de Souza e Lucena [15]; e muitos outros [18, 14, 47, 10].

Considerando o algoritmo desenvolvido em [15] para o SPP, é possível apontar uma interessante modificação. Tome uma inequação válida adicionada no algoritmo. O valor inicial dado ao multiplicador da mesma é sempre 0. Entretanto, na execução do MS para a resolução do PDL, multiplicadores de inequações adicionadas tardiamente têm menos iterações para convergir para seu valor ótimo. Pode-se, então, implementar um cálculo

para gerar uma inicialização mais adequada. Com esperança, essa inicialização propiciará melhores resultados.

Sobre este algoritmo *relax-and-cut* para o SPP, é possível, ainda, ressaltar uma importante questão. O mesmo não resolve (garantidamente) o SPP. Para obter um método exato, os autores sugerem, em [15], o seguinte esquema. O algoritmo *relax-and-cut* para o SPP é executado. Se uma solução ótima não é determinada, o XPRESS, um *software* comercial de otimização, é chamado, recebendo, da execução anterior, a melhor solução encontrada e as inequações válidas adicionadas. Tal esquema não é, porém, baseado completamente em relaxação Lagrangiana. Isto, pois, na segunda etapa, um *branch-and-cut* baseado em relaxação linear é utilizado.

Neste capítulo, são propostas e avaliadas as supra-anunciadas sugestões: uma *partida quente* para o multiplicador de uma inequação adicionada em um algoritmo *relax-and-cut* para o SPP e um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP. Para tanto, são feitas respectivas extensões do algoritmo desenvolvido em [15].

4.1 Extensão de um algoritmo *relax-and-cut* para o SPP

Quando um algoritmo baseado em relaxação Lagrangiana é aplicado ao SPP, a relaxação (2.12), onde todas as restrições são dualizadas, é comumente empregada. No entanto, no algoritmo desenvolvido em [15], por Cavalcante, de Souza e Lucena, os melhores resultados são obtidos com uma opção um pouco diferente. A mesma segue. Adicione ao conjunto de restrições do SPP a inequação $\sum_j x_j \leq m$, onde m é o número de restrições do mesmo. Essa é uma inequação válida, pois, em uma solução viável para o SPP, o número de variáveis valendo 1 é igual a m. Em seguida, dualize apenas as igualdades $Ax = \mathbf{1}$. A relaxação produzida é

$$\begin{aligned} z_{SPP}(\lambda) = \min \quad & cx + \lambda(\mathbf{1} - Ax) \\ & \mathbf{1}^T x \leq m \\ & x \in \mathbb{B}^n. \end{aligned} \tag{4.1}$$

Observe que a formulação presente nesta relaxação é mais forte que a encontrada na relaxação (2.12). Entretanto, ter uma formulação mais forte não faz de uma relaxação a melhor opção garantidamente em um algoritmo baseado em relaxação Lagrangiana. O motivo é que a resolução do PDL pode se tornar muito custosa computacionalmente. Em (4.1), tem-se um PRL não completamente trivial: determinar as variáveis correspondentes aos m menores custos Lagrangianos. Em [15], conclui-se, contudo, que esse PRL ainda é fácil o suficiente para propiciar um bom desempenho ao método desenvolvido.

O algoritmo proposto em [15] está descrito abaixo.

Algoritmo 4.1 Algoritmo baseado em relaxação Lagrangiana para atacar o SPP [15]

Etapa 1 (Opcional): Realize um pré-processamento. Execute um número pequeno de iterações do MS aplicado à função $z_{SPP}(\lambda)$. Nesta etapa, o objetivo é fixar variáveis através dos testes executados ao longo das iterações do MS.

Etapa 2: Execute um NDRC como implementado no Algoritmo 3.4, empregando o MS para resolver o PDL.

Etapa 3: Execute um procedimento de *lifting* para tentar fortificar as inequações adicionadas na Etapa 2.

Etapa 4: Execute o MS aplicado à função $z_{SPP}(\lambda)$ considerando a relaxação (4.1) adicionada das inequações identificadas na Etapa 2. Nesta etapa, somente resolvem-se relaxações. Não se adicionam mais inequações.

Na Etapa 1, são executadas poucas iterações do MS da seguinte maneira. O tamanho de passo θ^k, usado em uma iteração k, é calculado segundo a fórmula (3.7), apresentada na Subseção 3.2.1. O parâmetro ϵ é inicializado com valor 2, ou seja, $\epsilon^0 = 2$, e seu valor é multiplicado pelo fator 0,75 a cada 20 iterações consecutivas sem melhoria no limitante inferior $\underline{z}^k$. O MS para quando se tem 800 iterações executadas ou $\epsilon^k \leq 0{,}00001$, o que ocorrer primeiro. Nas Etapas 2 e 4, a execução do MS distingue-se por valores diferentes de parâmetros. O valor inicial do parâmetro ϵ é 2, ou seja, $\epsilon^0 = 2$, e o mesmo também é multiplicado pelo fator 0,75, mas, agora, a cada 100 iterações consecutivas sem melhoria no limitante inferior $\underline{z}^k$. Quando o pré-processamento (Etapa 1) não é realizado, o número máximo de iterações é 4000. Em caso contrário, esse número é 3600. No algoritmo como um todo, são executadas, no máximo, 8000 iterações do MS. Em toda iteração do MS, é executada uma heurística primal que tenta obter uma solução viável para o SPP fazendo uso da solução ótima do PRL corrente. Por fim, no NDRC (Etapa 2), as inequações válidas para o SPP que constituem a família F são as inequações clique (Subseção 2.3.1).

Em [15], foi feita uma comparação entre o método proposto e o *branch-and-cut* baseado em relaxação linear implementado no resolvedor comercial XPRESS versão 16.05.01. É preciso ressaltar que o último resolve o SPP, ou seja, encontra garantidamente uma solução ótima para o mesmo, enquanto o primeiro não o faz. Apesar desta distinção, é possível confrontar os limitantes alcançados pelos algoritmos considerando o tempo despendido para obtê-los. É isto que se quer avaliar com a comparação. Os resultados encontrados estão sumarizados a seguir.

No restante desta seção, o algoritmo *relax-and-cut* desenvolvido em [15] por Cavalcante, de Souza e Lucena é denotado por RC-CDSL e o *branch-and-cut* baseado em relaxação linear implementado no XPRESS é tratado por BCLIN-XPS.

Para um primeiro conjunto de instâncias, o RC-CDSL obteve um desempenho comparável ao do BCLIN-XPS em metade e substancialmente inferior no restante dos casos.

Para um segundo conjunto, porém, o RC-CDSL alcançou resultados melhores que os do BCLIN-XPS. Neste conjunto, as intâncias tinham como características comuns grandes *gaps* de integralidade e baixa densidade. Os autores examinaram, então, a combinação dos dois métodos: o RC-CDSL foi executado e os cortes e o melhor limitante superior encontrados passados como entrada ao BCLIN-XPS. O esquema híbrido superou o BCLIN-XPS.

É possível apontar uma questão ainda não avaliada para o RC-CDSL. A mesma é evidenciada pelas observações abaixo.

Considere o momento em que uma inequação é adicionada em um algoritmo *relax-and-cut* que empregue o MS para resolver o PDL. Se esse momento é tardio, ou seja, próximo de o algoritmo parar por algum critério, pode não haver iterações do MS suficientes para que o multiplicador da inequação convirja para seu valor ótimo (ou, pelo menos, para um bom valor). A tal multiplicador, é interessante, então, dar um valor inicial adequado. Com isso, almeja-se simular as iterações do MS "perdidas" pela inequação.

Lembre-se que o problema de identificar inequações violadas em uma iteração de um algoritmo *relax-and-cut*, um problema de separação, é, em geral, $\mathcal{NP}$-Difícil. Assim, usa-se comumente, para a atacar o mesmo, um algoritmo que não encontra garantidamente uma inequação violada, caso esta desigualdade exista. Ainda, quando várias inequações são encontradas, não obrigatoriamente todas são utilizadas. Logo, uma inequação pode ser adicionada somente momentos depois da sua primeira oportunidade. Faz sentido, então, tentar recuperar iterações do MS "perdidas".

No RC-CDSL, quando uma inequação é adicionada, é dado arbitrariamente o valor 0 ao seu multiplicador. Como dito, é possível que se encontre um melhor valor inicial para o mesmo. Neste trabalho, é realizado um estudo sobre o impacto de uma *partida quente* para o multiplicador de uma inequação adicionada em um algoritmo *relax-and-cut*.

Aqui, duas ressalvas são necessárias. A primeira é que vários pesquisadores relatam a não importância dos valores dados aos multiplicadores no início do MS. Por exemplo, Beasley [9] e Escudero, Guignard e Malik [20] citam essa inicialização como pouco influente no resultado final. Para inequações adicionadas durante o processo, porém, o valor inicial pode ser relevante. É isto que se quer avaliar com o referido estudo. A segunda é que um cálculo realizado para tal fim não pode ser muito custoso computacionalmente. Isto, pois o mesmo será repetido muitas vezes, a saber, em toda iteração em que uma inequação violada é identificada.

No Apêndice A, apresenta-se um relatório onde se descreve o estudo acima citado. Comentam-se, adiante, os aspectos principais do mesmo, finalizados pelas conclusões.

Antes, porém, é necessário discutir uma dificuldade adicional com a qual se lida no RC-CDSL; e com a qual bem como se lida comumente em um algoritmo *relax-and-cut* aplicado a um problema $\mathcal{NP}$-Difícil. Ocorre que, em geral, o conjunto formado pelas inequações adicionadas no algoritmo é de tamanho muito grande. É preciso, então, tratá-

lo de maneira eficiente. Para empregar o MS na resolução do PDL, uma modificação é adequada. Esta modificação é detalhada na subseção a seguir.

4.1.1 Modificação no MS

Considere o MS aplicado à função $z(\lambda)$, que determina o valor ótimo de uma relaxação Lagrangiana do problema P (3.1), descrito na Subseção 3.2.1. Suponha que, no mesmo, os tamanhos de passo são calculados segundo a fórmula (3.7). Em uma iteração k, então, têm-se as componentes $s_i^k = (a_i x^k - b_i)$ do subgradiente s^k de $z(\lambda)$ em λ^k e os multiplicadores λ_i^k, para $i = 1, \ldots, m$, onde m é o número de restrições dualizadas. Os últimos são atualizados de acordo com a fórmula

$$\lambda_i^{k+1} = \max\{0; \lambda_i^k + \theta^k s_i^k\},\ i = 1, \ldots, m. \tag{4.2}$$

O tamanho de passo θ^k é dado por

$$\theta^k = \epsilon^k \frac{(\hat{z} - z(\lambda^k))}{\sum_{i=1}^m {s_i^k}^2}. \tag{4.3}$$

Estas equações são reescritas das equações (3.4) e (3.7) respectivamente.

Para um m muito grande, lidar com as fórmulas acima não é trivial. Neste contexto, a seguinte modificação sugerida por Beasley [9] é bem interessante. Para as restrições $a_i x \leq b_i$ tais que $\lambda_i^k = 0$ e $s_i^k < 0$, faça $s_i^k \Leftarrow 0$ antes de proceder com a atualização.

Há algumas razões que fundamentam tal mudança. Considere uma restrição $a_i x \leq b_i$ para a qual as condições citadas sejam satisfeitas. Primeiramente, sem alteração sugerida, λ_i permaneceria nulo ao final da iteração. Logo, tem-se $\lambda_i^{k+1} = 0$ com ou sem a mesma. Em segundo lugar, apesar de s_i^k não contribuir para o valor de λ_i^{k+1}, essa componente influencia o tamanho de passo θ^k. E se há um número demasiado de restrições para as quais o subgradiente não é nulo, θ^k tende a ser muito pequeno e os multiplicadores a ficarem praticamente inalterados. Pode-se, então, deparar-se com problemas de convergência. Por fim, s_i^k não parece ter importância para essa iteração. Isto, pois uma restrição $a_i x \leq b_i$ tal que $\lambda_i^k = 0$ não é relevante para a obtenção de x_R^k, a solução ótima do PRL definido para λ^k. De fato, a mesma origina uma parcela nula na soma que determina os custos Lagrangianos para tal PRL.

Tipicamente, o número de restrições que se qualificam para a mudança acima é muito grande. É possível, portanto, com essa modificação, reduzir significativamente o número de restrições tratadas explicitamente. Neste trabalho, usa-se essa sugestão.

4.1.2 Implementação, resultados e conclusão

O algoritmo *relax-and-cut* (NDRC) presente no RC-CDSL é estendido com a implementação de uma partida quente para o multiplicador de uma inequação adicionada.

A mesma é feita primeiramente da seguinte maneira. Quando uma inequação $\pi^i x \leq \pi_0^i$ é adicionada, é executado um procedimento para calcular o valor inicial para o seu multiplicador μ_i. Esse procedimento nada mais é que um MS. Assim, pode-se entender o mesmo como o MS interno (MSINT), em contrapartida ao outro MS aplicado no algoritmo, o MS externo (MSEXT). O MSINT é executado considerando todas as inequações adicionadas até então e tendo os valores atuais dos multiplicadores das mesmas como os seus iniciais; 0 para μ_i. O valor de μ_i ao final dessa execução é usado, então, para inicializar tal multiplicador no MSEXT. Observe que, nesse ponto, os multiplicadores das outras inequações têm novos valores. Tais valores, a princípio, também podem ser utilizados para alimentar o MSEXT. E é o que acontece. Em testes preliminares, a última opção mostrou-se mais vantajosa. Note que, desta maneira, o MSINT pode ser visto como iterações adicionais do MSEXT.

Como o cálculo do valor inicial não pode ser custoso computacionalmente, a execução do MSINT tem de ser rápida. Isto é alcançado com duas restrições. A primeira é que não há adição de novos cortes. A segunda é que impõe-se um limite de iterações para o decréscimo do parâmetro ϵ^k menor que o usado no MSEXT. Esse é dado por $\lfloor \frac{(\frac{k}{2500})}{30} \rfloor$, onde k é a iteração corrente do MSEXT. O número acima foi obtido empiricamente e tenta capturar a ideia de que quanto mais ao final do MSEXT (quanto mais passam-se 2500 iterações), mais pode-se esforçar para encontrar um bom valor inicial para μ_i. O limite utilizado no MSEXT é de 30 iterações.

Foi testada, ainda, uma segunda versão do algoritmo estendido. Nesta, o valor usado para o limite tratado acima é igual ao do MSEXT: 30 iterações. O valor inicial encontrado é potencialmente melhor. Entretanto, o tempo gasto tende a ser maior.

Considerando a primeira das versões citadas, os resultados obtidos são os seguintes. Este algoritmo estendido será denotado por RCEST-1.

Para 32,4% das instâncias, o RCEST-1 obteve um limitante inferior maior que o do RC-CDSL. Houve, ainda, 56,8% de empates e 10,8% de derrotas. Aqui, uma ressalva é importante. Em 17 dos 37 casos, o limitante encontrado pelo RC-CDSL já era igual ao valor ótimo. Nos mesmos, portanto, não poderia haver vitórias do RCEST-1. Observe que isso não invalida o uso de tais instâncias. O motivo é que os resultados do RCEST-1 poderiam ser piores. Sem incluir tais testes, o RCEST-1 superou o RC-CDSL em 63,1% das vezes.

A melhoria nos limitantes é, contudo, pequena. Essa é, em média, de 0,58%. No entanto, tal constatação não é surpreendente. Isto, pois os limitantes obtidos pelo RC-CDSL já são bastante próximos ao valor ótimo.

Também houve um impacto positivo nos limitantes superiores. Com melhores valores para os multiplicadores, as soluções ótimas obtidas para os PRLs tendem a ser mais próximas de serem viáveis para o SPP. A heurística primal empregada usa as informações

desses pontos. Assim, tende-se a encontrar, com a mesma, soluções de melhor valor. Em 21,6% das instâncias, o RCEST-1 obteve um limitante superior menor que o do RC-CDSL. Sem contabilizar os casos para os quais o RC-CDSL já havia descoberto o valor ótimo, esse número sobe para 53,3%.

Quanto ao tempo de execução, ocorreu uma piora significativa. O RCEST-1 gasta, em média, 3 vezes o tempo despendido pelo RC-CDSL.

Para a segunda das versões descritas, também houve melhorias nos dois limitantes. Essas, inclusive, foram mais acentuadas. Como era esperado, porém, o tempo de execução foi bem maior. O mesmo foi, em média, 10 vezes o tempo despendido pelo algoritmo original.

Um valor inicial mais adequado para o multiplicador de uma inequação adicionada impacta, então, positiva e negativamente nos resultados do RC-CDSL. Os algoritmos estendidos obtém limitantes mais precisos que os do original. Os mesmos, porém, alcançam tais limitantes em um tempo substancialmente maior que o último. Assim, a extensão provê melhores limitantes a um custo alto, em tempo de execução, a ser pago.

4.2 Um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP

Um *branch-and-cut* baseado em relaxação linear é um método consolidado na resolução de um IP. Empregar a relaxação linear não é, no entanto, o único modo de implementar essa técnica. Lucena [45], entre outros pesquisadores, apontam um *branch-and-cut* baseado em relaxação Lagrangiana como uma interessante proposição. Observa-se, contudo, na literatura, o pouco uso de tal combinação. Em relação à opção tradicional, o uso de uma relaxação Lagrangiana propicia pelo menos três vantagens.

Primeiramente, considerando uma relaxação Lagrangiana e a relaxação linear de um IP, é comum que a execução de um método aplicado ao PDL associado à primeira seja mais rápida que a execução de um método para otimizar o LP determinado pela última, além de os limitantes obtidos no primeiro caso serem bastante precisos. Logo, com o uso de uma relaxação Lagrangiana, resolve-se, geralmente, a relaxação de um subproblema mais rapidamente. Solucionar os LPs é o gargalo para o desempenho de um *branch-and-bound* baseado em relaxação linear. O impacto também é grande em um respectivo *branch-and-cut*. Em segundo lugar, têm-se, tipicamente, em uma relaxação Lagrangiana, somente variáveis inteiras. Um problema de separação definido para uma solução inteira tende a ser mais fácil que um associado a uma solução fracionária. Assim, ao utilizar uma relaxação Lagrangiana, tende-se a separar uma solução mais eficientemente. O passo de separar uma solução é estratégico em um *branch-and-cut*. A terceira vantagem é relativa

ao armazenamento de informações. Quando se emprega uma relaxação Lagrangiana, a prática de guardar bases ótimas não precisa ser adotada. Os subproblemas são, então, representados de forma mais simples.

Obviamente, também existem desvantagens. Uma que se pode citar de imediato é ter a difícil tarefa de determinar os melhores valores para os parâmetros do MS (supondo que esse é o método usado para solucionar o PDL). Há, ainda, um desafio adicional: para nós distintos da árvore de divisões, diferentes configurações são provavelmente mais adequadas. Por exemplo, é razoável que as aplicações do MS em um subproblema recentemente criado e associado a uma restrição $x_j = 1$ e em um subproblema antigo e associado a uma restrição $x_j = 0$ requeram valores de parâmetros diferenciados. O que é determinado por tais valores são os tamanhos de passo. As dificuldades inerentes a esse assunto foram discutidas na Subseção 3.2.1.

Para atacar o SPP, a ideia de desenvolver um algoritmo que use a combinação de relaxação Lagrangiana com planos de corte é especialmente promissora: em [15], o algoritmo *relax-and-cut* aplicado por Cavalcante *et al.* ao SPP mostrou-se muito eficiente. Dadas as vantagens e recomendações citadas, é certamente interessante avaliar, principalmente para o SPP, o desempenho de um método exato que faça tal uso.

Assim, propõe-se, a seguir, uma extensão do algoritmo desenvolvido em [15]. O algoritmo *relax-and-cut* é incorporado a uma enumeração, sendo gerado, desta forma, um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP. Os resultados são apresentados.

4.2.1 Implementação

Descreve-se, nesta subseção, em um bom nível de detalhes, como foi implementado o *branch-and-cut* baseado em relaxação Lagrangiana para o SPP proposto aqui. O mesmo é uma extensão do algoritmo desenvolvido em [15].

É preciso dizer, contudo, que o algoritmo *relax-and-cut* componente do *branch-and-cut* proposto nesta dissertação foi codificado de forma independente à do algoritmo aplicado em [15]. Logo, apesar de serem equivalentes, os respectivos programas de computador possuem códigos-fonte naturalmente diferentes. É necessário relatar, também, que, para este trabalho, o código-fonte produzido em [15] foi disponibilizado. Isto foi de grande valia, por prover o acesso a todas as nuances da programação do algoritmo *relax-and-cut* desenvolvido em tal referência.

O *branch-and-cut* descrito a seguir é uma instância do Algoritmo 3.10. Assim, ao longo desta subseção, as estruturas presentes no mesmo são por várias vezes referenciadas.

Estruturas de dados para os subproblemas

Adotando notação semelhante (mas não igual) à usada em [52], um subproblema pode ser representado por $\mathrm{P}(F', V_0, V_1)$. O mesmo é dado pelo problema P acrescido do conjunto $F' \subseteq F$ de inequações adicionadas e restrito pela fixação dos valores das variáveis em V_0 e V_1 em 0 e 1 respectivamente. Os conjuntos F', V_0 e V_1 são armazenados em listas encadeadas simples. Além disso, para cada subproblema, são guardados uma lista com as restrições ainda não eliminadas (por consequência de fixações de variáveis) e o melhor limitante inferior conhecido. Por fim, a estrutura de um subproblema também contém os valores iniciais para os multiplicadores das restrições dualizadas (relaxação Lagrangiana).

A lista L (de subproblemas correspondentes aos nós para os quais ainda não houve poda ou divisão) é implementada por uma lista encadeada simples.

Pré-processamento

Em uma iteração do *branch-and-cut*, antes do processamento do subproblema, são aplicados testes para tentar reduzir o tamanho do mesmo. Principalmente em instâncias do SPP derivadas de situações reais, é comum que muitas variáveis possam ser fixadas, ou seja, tenham seus valores fixados, e que muitas restrições possam ser eliminadas. As reduções são consequências de implicações lógicas. O pré-processamento, aqui, implementa essencialmente os mesmos testes usados em [15]. Esses testes são descritos abaixo. (Somente são avaliadas as variáveis ainda não fixadas e as restrições ainda não eliminadas.)

1. Se x_j e x_k são variáveis que estão presentes exatamente nas mesmas restrições e $c_j \leq c_k$, x_k (a variável duplicada de maior custo) é fixada em 0.

2. Se x_j é a única variável presente em uma restrição, x_j é fixada em 1 e a restrição é eliminada. Além disso, todas as outras restrições que contém x_j são agora satisfeitas e, portanto, excluídas. Por consequência, ainda, todas as outras variáveis presentes nessas restrições são fixadas em 0.

3. Se uma restrição está contida em outra, todas as variáveis presentes na última e não na primeira são fixadas em 0.

Divisão de um subproblema (*branching*)

Aqui, a divisão de um subproblema (*branching*) é feita de duas maneiras. Considere a solução x_R^k e o vetor λ^k vinculados ao melhor limitante inferior obtido com o processamento do subproblema, ou seja, com a execução do algoritmo *relax-and-cut*. A primeira maneira ocorre quando existem restrições dualizadas do tipo $\sum_{j \in J'} x_j \leq 1$ violadas por x_R^k, ou seja, com $\sum_{j \in J'} x_{R_j}^k > 1$ (as igualdades do SPP podem ser vistas como duas desigualdades,

uma sendo desse tipo). A mesma é mencionada na Seção 3.3 e é conhecida como GUB *branching*. Tome, dentre as restrições citadas, uma cuja violação é máxima ($\sum_{j \in J'} x^k_{R_j}$ maior possível). Nesta divisão, são gerados dois novos subproblemas: um com o conjunto S_1 e outro com o conjunto S_2 de variáveis fixadas em 0. Seja p o número de variáveis presentes na referida restrição que valem 1 em x^k_R (há pelo menos duas). Tem-se que $x_j \in S_1$ se $j \leq l$ e $x_j \in S_2$ se $j > l$, sendo l o índice da $\lfloor \frac{p}{2} \rfloor$-ésima variável valendo 1.

A segunda maneira acontece quando não há violações como as descritas acima. Nesta divisão, uma variável x_j é escolhida para determinar dois novos subproblemas: um satisfazendo $x_j = 1$ e outro respeitando $x_j = 0$. A escolha é guiada pela seguinte sugestão de Beasley [9]. Considere o termo da função objetivo da relaxação Lagrangiana gerado pelas restrições dualizadas (inclusive cortes). Seja o mesmo dado por $\lambda(A'x - b')$. Tome uma restrição $a'_i x \leq b'_i$ para a qual o valor absoluto de $\lambda^k_i(a'_i x^k_R - b'_i)$ é máximo. Das variáveis presentes nesta restrição, selecione a de menor custo Lagrangiano. Se existem variáveis valendo 1 em x^k_R, a seleção deve ser feita somente entre essas.

Com tal divisão, busca-se satisfazer a restrição que mais influencia o valor da função objetivo da relaxação Lagrageana. Ao fazê-lo, tenta-se diminuir o valor absoluto da parcela $\lambda^k_i(a'_i x^k_R - b'_i)$ e, consequentemente, tornar o termo $\lambda^k(A'x^k_R - b')$ mais próximo de ser nulo. Isto, pois, quando uma solução x^k_R é viável para um subproblema e ocorre $\lambda^k(A'x^k_R - b') = 0$, a mesma é também ótima. Ainda, neste *branching*, a variável escolhida para determinar os dois novos subproblemas é, por ter o menor custo Lagrangiano dentre as selecionadas, a mais provável de valer 1 em uma solução ótima para o subproblema em questão. Combinada à estratégia profundidade primeiro empregada para a escolha de um subproblema, esta parece uma boa conduta.

A primeira opção de *branching* é sempre preferida, pois, como comentado na Seção 3.3, a mesma tende a gerar ramos de tamanhos balanceados na árvore de divisões. Para a segunda, o ramo relativo a $x_j = 1$ é geralmente promissor, enquanto o relativo a $x_j = 0$ não fornece comumente boas soluções.

Escolha de um subproblema

Para implementar o Passo 2 do Algoritmo 3.10, adota-se uma sugestão de Beasley [9]: estratégia profundidade primeiro (abordada na Seção 3.3). Assim, sempre o subproblema disponível mais recentemente criado é escolhido. Quando existem dois, há dois casos. Se a divisão que os derivou é baseada em uma variável, o subproblema associado à restrição $x_j = 1$ é preferido. Isto, pois é no ramo determinado pelo mesmo que provavelmente estão as soluções ótimas para o subproblema que os originou. Se a divisão é baseada em uma inequação, o subproblema com maior número de variáveis fixadas no processo é selecionado.

Cálculo de limitantes (*bounding*)

Em um *branch-and-cut*, escolhido um subproblema, calculam-se limitantes para seu valor ótimo (*bounding*) resolvendo uma relaxação. Aqui, usa-se uma relaxação Lagrangiana. Ainda, inequações válidas são adicionadas gradativamente para refinar a mesma. Aplica-se, então, um algoritmo *relax-and-cut* ao subproblema. Nesta implementação, emprega-se o algoritmo desenvolvido em [15].

Para solucionar o PDL aplica-se, então, o MS. Os valores dos parâmetros do MS são iguais para as duas aplicações efetuadas do mesmo. Tais valores, no entanto, variam conforme os nós considerados. Antes de comentá-los, então, é preciso definir os tipos de subproblema.

Suponha que é feita a divisão em um nó. Se essa é baseada em uma variável, o subproblema associado à restrição $x_j = 1$ é considerado *bom*, enquanto o vinculado à restrição $x_j = 0$ é tido como *ruim*. Isto reflete o fato de que é no ramo determinado pelo primeiro que provavelmente encontram-se as soluções ótimas. Se a divisão é baseada em uma inequação, os dois subproblemas são julgados bons. A razão é que ambos modificam a solução da relaxação Lagrangiana gerando, assim, algum progresso.

Em todos os nós, o parâmetro ϵ do tamanho de passo é inicializado com valor 2. Para os subproblemas bons, adota-se uma estratégia semelhante à utilizada em [15]: a cada 100 iterações consecutivas do MS sem melhoria no limitante inferior, o valor de ϵ é multiplicado por 0,75. Além disso, são executadas no máximo 4000 iterações. Para os ruins, segue-se uma sugestão de Beasley [9]. Como observado antes, tais subproblemas determinam ramos que não são, provavelmente, proveitosos. Ao considerar um desses, então, é válido fazer um maior esforço, a fim de encontrar limitantes que possibilitem a poda no nó correspondente. Assim, usam-se, para os mesmos, o fator 0,75 e 200 iterações. Também, o número máximo de iterações é elevado para 8000.

O subproblema inicial (problema original) é considerado bom. Nesse subproblema, porém, também é válido fazer um maior esforço para encontrar bons limitantes. As informações geradas são úteis em toda a árvore e o tamanho da mesma pode ser drasticamente reduzido em função dos produtos do primeiro nó. Assim, a configuração empregada é igual à dos outros subproblemas bons, com uma exceção: o número máximo de iterações. São permitidas 14000.

A mais importante das estratégias implementadas aqui é relativa ao parâmetro $\hat{z}$. Como discutido na Subseção 3.2.1, com o tamanho de passo no MS determinado pela fórmula (3.7), $z(\lambda)$ converge para $\hat{z}$ ou um ponto λ' é encontrado tal que $z(\lambda') > \hat{z}$. Comumente, $\hat{z}$ é incializado com o valor de um limitante superior qualquer e é atualizado ao passo que soluções viáveis de menor custo são descobertas. Nesta implementação, $\hat{z}$ é tratado da seguinte forma. Para um subproblema P_i, $\hat{z}_i$ tem inicialmente o valor do melhor limitante superior conhecido para o problema original: $\overline{z}$. Se $z_i \leq \overline{z}$, tudo ocorre

como descrito acima. Se $z_i > \overline{z}$, $\hat{z}_i$ não é um limitante superior para z_i. Assim, a sequência de limitantes inferiores produzida provavelmente ultrapassa $\hat{z}_i$ rapidamente. Quando isso acontece, o MS termina. O motivo é que o tamanho de passo torna-se muito pequeno e este é um dos critérios de parada. Observe que tal situação representa a condição (ii) de poda em um nó. E é o que ocorre para o subproblema. Logo, $\overline{z}$ é um valor inicial adequado para $\hat{z}$, pois, quando não é um bom limitante superior, provavelmente propicia, a pouco custo (com poucas iterações do MS), uma poda da árvore.

Por fim, para um subproblema, os valores iniciais dos multiplicadores são dados pelos melhores encontrados no outro que o originou.

Geração de cortes

Ao longo das iterações do MS, são adicionados planos de corte, para refinar a relaxação. Este processo, aqui, guia-se pela implementação feita em [15].

Os cortes utilizados são as inequações clique (Subseção 2.3.1). Estas são desigualdades de comprovada eficiência para o SPP. Em [15], tais cortes obtêm ótimos resultados na melhoria dos limitantes. Quando uma inequação violada é encontrada, a mesma é armazenada em um *pool*. Este é implementado por um vetor e tem, portanto, tamanho fixo. Há, no mesmo, 50000 posições. Diferentemente do que acontece em [15], o *pool* de restrições é compartilhado por vários problemas. Por isso, o processo de geração de cortes, aqui, é um pouco diferente. Tal processo ocorre em dois passos e é como descrito adiante.

Quando um subproblema é criado, esse recebe uma lista com os cortes presentes no nó pai. Quando o mesmo é escolhido para ser processado, é possível que tenham ocorrido mudanças no *pool*: novas inequações (desconhecidas) tenham sido adicionadas e outras conhecidas tenham sido excluídas (a exclusão de uma inequação é comentada no próximo tópico abordado nesta subseção). Antes de mais nada, então, a lista citada tem de ser atualizada.

Em uma iteração do MS, para adicionar planos de corte, o primeiro passo é buscar por inequações violadas desconhecidas no *pool*. Se a busca tem sucesso, essas passam a compor a lista de desigualdades conhecidas e a geração de cortes termina. Se a busca falha, é preciso encontrar, de fato, novas inequações violadas. Para tanto, usa-se o algoritmo descrito em [48].

A fim de identificar as cliques que originam as desigualdades, é preciso construir o grafo de interseção $G(A)$. Este é determinado pelo SSP associado ao SPP. Uma clique em $G(A)$ define uma faceta para o SSP se, e somente se, a mesma é maximal. O algoritmo utilizado tem por objetivo encontrar tais cliques em $G(A)$.

No grafo de interseção implementado, somente há vértices para variáveis que valem 1 na solução relaxada. Observe que, desta forma, qualquer clique (três ou mais vértices)

encontrada corresponde a uma inequação violada. Mesmo com tal restrição, o grafo gerado pode ser muito grande. Para assegurar a eficiência do algoritmo citado acima, que depende do tamanho do mesmo, $G(A)$ tem o número máximo de vértices pré-definido.

A principal diferença para o *relax-and-cut* proposto em [15] é que, nesta extensão, não é implementado o *lifting* de cortes (Etapa 3 do Algoritmo 4.1). Observou-se, contudo, em execuções do *relax-and-cut*, que o impacto do *lifting* está relacionado ao número máximo de vértices do grafo de interseção $G(A)$. Em geral, esse procedimento tem influência positiva nos limitantes obtidos quando tal número é pequeno. Isto é razoável, pois, com poucos vértices em $G(A)$, são provavelmente produzidos cortes com densidade pequena e passível de aumento. Com grafos de interseção grandes, porém, o processo citado tem comumente pouca relevância. É claro que, também, neste contexto, a identificação de inequações violadas tende a ser mais custosa computacionalmente. Em testes preliminares, constatou-se um bom desempenho do *branch-and-cut* com valores para $G(A)$ maiores que os indicados em [15]. Assim, adotam-se, por padrão, no mesmo, os máximos de 1000, para o subproblema inicial, e de 500 vértices, para os demais. No algoritmo desenvolvido em [15], usa-se o máximo de 200 vértices para $G(A)$. O valor maior empregado para o primeiro subproblema do *branch-and-cut* advém do senso de que é válido fazer um maior esforço computacional em tal nó.

Quando uma nova inequação violada é encontrada, seja por qualquer uma das alternativas acima, a mesma é adicionada ao subproblema, não no conjunto de restrições, mas sim como um termo da função objetivo. Assim, há um novo multiplicador associado a tal desigualdade. E esse é inicializado com valor 0. Infelizmente, a estratégia estudada na Seção 4.1, que calcula um melhor valor inicial, não é claramente proveitosa.

Gerenciamento do *pool* de restrições

A nova estratégia que talvez mais chame atenção no algoritmo *relax-and-cut* desenvolvido em [15] é a política de gerenciamento do *pool* de restrições. A importância de um bom gerenciamento do *pool* pode ser evidenciada com as seguintes observações. No contexto de relaxação Lagrangiana, um fenômeno recorrente na adição de planos de corte é que uma mesma desigualdade é gerada várias vezes, em iterações diferentes. Há, também, frequentemente, cortes adicionados que são descartáveis, dos quais o impacto está sendo representado por outras inequações. Se a política de excluir do *pool* de restrições cortes como os citados é adotada, o número de desigualdades consideradas pode ser reduzido substancialmente e sem perda de qualidade. É comumente aceito que quanto maior é o número de restrições dualizadas maior é o tempo para o MS convergir. Com tal prática, então, problemas de convergência do MS podem ser mitigados. Por consequência, um algoritmo *relax-and-cut* baseado nesse método pode ter seu desempenho melhorado.

A política de gerenciamento empregada em [15] abrange os dois pontos comentados

acima. Primeiramente, não é permitida a adição de uma desigualdade idêntica a outra já adicionada. Em segundo lugar, são eliminados cortes descartáveis, identificados através da relação de dominância entre inequações. Isto se dá da seguinte forma. Suponha que várias inequações adicionadas são dominadas por uma única outra. Apenas a dominante é suficiente para causar o impacto provocado por todas juntas. Os cortes dominados são considerados descartáveis e, portanto, excluídos.

Para gerenciar o *pool* de inequações compartilhado no *branch-and-cut*, usa-se, aqui, a mesma política. São, assim, implementadas as seguintes regras.

1. Uma nova desigualdade é adicionada ao *pool* se
   (a) não é igual a nenhuma outra existente (inequações no *pool* e no conjunto de restrições do subproblema);
   (b) não é dominada por nenhuma outra existente (inequações no *pool* e no conjunto de restrições do subproblema).
2. Uma desigualdade já adicionada é excluída do *pool* se
   (a) é dominada por outra recentemente adicionada.

Os resultados apresentados em [15] ratificam a importância dessa política. Com tal estratégia, a geração de cortes no *relax-and-cut* torna-se mais cara computacionalmente. No entanto, as melhorias proporcionadas aos limitantes e à convergência do MS são compensadoras.

Heurística Primal

Para gerar soluções viáveis e, consequentemente, limitantes superiores ao longo das iterações do MS, uma heurística essencialmente idêntica à desenvolvida em [15] é utilizada. Heurísticas primais, no contexto de relaxação Lagrangiana, geralmente usam informações de multiplicadores ou soluções relaxadas para construir pontos viáveis. A implementada aqui baseia-se nos valores das variáveis na solução do PRL: 0 ou 1.

O procedimento usado pode ser descrito, de maneira simplificada, como abaixo. Considere a notação $S = \{x_j : j \in J'\}$ para a solução $x_j = 1, j \in J'$ e $x_j = 0, j \notin J'$. Tente construir uma solução viável S para o subproblema inserindo, iterativamente, variáveis na mesma. Para a inserção, tome a seguinte ordem. Primeiro as variáveis que valem 1 na solução do PRL e depois as que valem 0. Uma variável é adicionada somente se não está presente em nenhuma restrição já satisfeita por S. Se, ao final das tentativas, todas as restrições são respeitadas, S é viável. Senão, a heurística falha.

Já que uma heurística para o SPP pode não conseguir produzir soluções viáveis, em [15], os autores propõem um cálculo para gerar um (muito pouco preciso) limitante superior para o valor ótimo do mesmo. Esse também é adotado aqui. O cálculo é o seguinte.

Tome o número mínimo necessário de variáveis incluídas em uma solução viável. Seja k esse número. Some os k maiores custos e obtenha o limitante.

Para mais detalhes sobre os procedimentos mencionados acima, veja [15].

Fixação de variáveis

Para um programa inteiro, é possível aplicar testes que permitam fixar os valores das variáveis. Os implementados aqui para o SPP são os mesmos propostos em [15], que, por sua vez, são descritos em [9]. Considerando um problema $0-1$, o processo de fixação de variáveis pode ser explicado da seguinte maneira. Obrigue uma variável x_j a ter valor $k \in \{0,1\}$, ou seja, adicione a restrição $x_j = k$ ao problema. Se as soluções obteníveis a partir daí são garantidamente sub-ótimas, o valor de x_j pode ser fixado no oposto, $1-k$.

Beasley [9] diz que a fixação de variáveis representa comumente ganhos para o desempenho de um algoritmo. Essa pode, porém, em várias situações, ser muito ineficiente. O sucesso da mesma depende fortemente dos limitantes para o valor ótimo do problema disponíveis. Com ciência disso, adota-se a seguinte estratégia no algoritmo desenvolvido em [15]. Somente tenta-se fixar variáveis quando uma distância mínima entre os limitantes inferior e superior é atingida. Esta estratégia é replicada aqui.

No *branch-and-cut*, o valor da distância mínima diferencia-se com os tipos de nó – veja a classificação dos subproblemas no tópico *Cálculo de limitantes (bounding)*. Para os subproblemas bons, a distância é de 3%. Para os ruins e para o nó inicial, o valor usado é 6%.

4.2.2 Resultados

O objetivo traçado aqui foi, por meio de uma extensão do algoritmo *relax-and-cut* apresentado em [15], gerar um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP e avaliar o desempenho do método exato produzido. Para tanto, tal método é implementado e executado para o mesmo conjunto de instâncias utilizado em [15]. Desta forma, é possível analisar o quão eficiente é a técnica *relax-and-cut* quando se trata de encontrar garantidamente soluções ótimas para as referidas instâncias.

Um aspecto importante é que a codificação do *branch-and-cut* foi feita de modo independente. Assim, os códigos-fonte dos programas de computador respectivos ao algoritmo usado em [15] e à subrotina do *branch-and-cut* correspondente ao último são distintos. Consequentemente, os resultados alcançados com os mesmos podem diferir. Para realizar a citada análise de forma justa, é imprescindível, contudo, que os referidos programas sejam igualmente efetivos.

É preciso, então, antes de mais nada, confrontar o algoritmo *relax-and-cut* empregado em [15] com a componente do método exato correspondente ao mesmo. Para isto, pode-se

comparar execuções do primeiro algoritmo e do *branch-and-cut* para o subproblema inicial (P^0 =P).

A fim de realizar tal comparação, aplicam-se os dois algoritmos *relax-and-cut* ao já mencionado conjuntos de testes em um mesmo ambiente computacional: sistema operacional GNU/Linux, distribuição Kubuntu 8.04, processador Intel Corel 2 Quad e memória RAM de 4GB. Os programas de computador produzidos em [15] e nesta dissertação foram escritos nas linguagens `C++` e `C` respectivamente. Utilizou-se, para os mesmos, nessa ordem, os compiladores `g++` (`gcc` versão 4.0.2) e `gcc` versão 4.2.4, ambos com a opção `-O3` habilitada. Deve-se salientar o importante fato de ambas as alternativas serem executadas em um mesmo ambiente. Isto somente foi possível com a disponibilização do código-fonte do algoritmo desenvolvido em [15].

Também, para as duas implementações, foram usados valores iguais de parâmetros (os comuns a ambos) com uma exceção. Esses valores são os mesmos empregados em [15] e estão descritos na subseção anterior. A exceção é por conta do número máximo de vértices para o grafo de interseção $G(A)$. Tal distinção é abordada, na subseção anterior, quando se descreve a geração de cortes. Salvo em algumas instâncias do tipo cyc, são permitidos 1000 vértices para o *branch-and-cut* aplicado ao subproblema inicial e 200 vértices para o algoritmo *relax-and-cut* proposto em [15]. Os casos destacados são os cyc09, cyc10 e cyc11 para o primeiro e cyc11 para o último algoritmo. Por representarem problemas de dimensões bastante grandes, tais casos requerem valores maiores. Estes são, respectivamente, 10000 e 500.

Abaixo, RC-CDSL denota o algoritmo *relax-and-cut* desenvolvido em [15] por Cavalcante, de Souza e Lucena e RC-BC designa o algoritmo *relax-and-cut* componente do *branch-and-cut* aqui proposto aplicado ao subproblema inicial.

Na Tabela 4.1, apresentam-se os resultados das execuções dos algoritmos. A mesma contém as seguintes informações. No primeiro grupo de colunas, estão as dimensões das instâncias – o número de restrições, NRes, o número de variáveis, NVar, e a densidade (em porcentagem) Den – e os respectivos valores ótimos, Óti. No segundo grupo, revelam-se dados sobre as execuções do RC-CDSL – o número de restrições e variáveis remanescentes após o pré-processamento, NRes e NVar sob o título *Pós-pré-p*, os limitantes superiores e inferiores atingidos, LSup e LInf, o número de cortes ao final do processo, NCor, a porcentagem de variáveis fixadas, VarF, e o tempo de otimização (em segundos), Tem. Nas colunas restantes, trata-se, de forma correspondente, do RC-BC.

Ao observar as duplas de colunas NRes e NVar sob os títulos Pós-pré-p, nota-se que o pré-processamento é igualmente efetivo nos dois algoritmos. E tal comportamento é esperado, já que as regras implementadas são essencialmente as mesmas. Apenas em 7 dos 45 casos, há uma redução maior para o RC-BC. Além disso, somente para a instância us04, a diferença é significativa.

Instância					RC-CDSL							RC-BC							
					Pós-pré-p							Pós-pré-p							
Nome	NRes	NVar	Den	Óti	NRes	NVar	LSup	LInf	NCor	VarF	Tem	NRes	NVar	LSup	LInf	NCor	VarF	Tem	⌈ÓtiLR⌉
nw03	59	43749	14,10	24492	59	38964	24492	24490	1051	99,88	301,05	59	38964	24492	**24492**	355	99,98	61,86	24492
nw04	36	87482	20,22	16862	36	46190	20328	16312	2345	0,00	470,95	36	46190	16868	16309	1942	94,39	191,29	16320
nw06	50	6774	18,17	7810	50	5977	7810	**7810**	829	99,98	2,81	50	5977	7810	**7810**	429	100,00	2,84	7810
nw11	39	8820	16,64	116256	39	6488	116256	**116256**	177	99,92	1,29	39	6488	116256	**116256**	297	99,95	3,38	116256
nw13	51	16043	12,78	50146	51	10905	50146	**50146**	250	99,62	39,49	51	10905	50146	**50146**	295	99,75	10,01	50146
nw17	61	118607	13,96	11115	61	78186	37938	10890	4076	0,00	512,70	61	78186	11115	10893	1641	99,65	474,59	10920
nw18	124	10757	6,82	340160	124	8460	340160	339481	1782	91,74	57,32	124	8460	340160	340029	1869	94,52	23,76	340121
nw20	22	685	24,70	16812	22	566	16812	**16812**	405	99,29	0,25	22	566	16812	**16812**	106	100,00	0,07	16812
nw21	25	577	24,89	7408	25	426	7408	**7408**	27	100,00	0,02	25	426	7408	**7408**	12	100,00	0,05	7408
nw22	23	619	23,87	6984	23	531	6984	**6984**	53	99,81	0,03	23	531	6984	**6984**	58	100,00	0,06	6984
nw23	19	711	24,80	12534	18	473	12534	**12534**	96	90,27	1,88	18	473	12534	**12534**	83	98,73	0,26	12534
nw24	19	1366	33,20	6314	19	926	6314	**6314**	48	93,52	1,88	19	926	6314	**6314**	138	100,00	0,16	6314
nw25	20	1217	30,16	5960	20	844	5960	**5960**	186	99,88	0,15	20	844	5960	**5960**	24	100,00	0,04	5960
nw26	23	771	23,77	6796	23	542	6796	**6796**	84	99,82	0,06	23	542	6796	**6796**	48	100,00	0,08	6796
nw27	22	1355	31,52	9933	22	926	9933	**9933**	68	100,00	0,15	22	926	9933	**9933**	31	100,00	0,07	9933
nw28	18	1210	39,27	8298	18	825	8298	**8298**	71	100,00	0,06	18	825	8298	**8298**	31	100,00	0,05	8298
nw29	18	2540	31,04	4274	18	2034	4430	4257	320	0,00	7,98	18	2034	4324	4241	676	94,15	1,19	4249
nw30	26	2653	29,63	3942	26	1884	3942	**3942**	246	99,95	0,59	26	1884	3942	**3942**	61	100,00	0,19	3942
nw31	26	2662	28,86	8038	26	1823	8038	**8038**	61	99,89	0,99	26	1823	8038	**8038**	45	100,00	0,30	8038
nw32	19	294	24,29	14877	18	251	14877	**14877**	327	97,61	0,23	18	251	14877	**14877**	139	100,00	0,15	14877
nw33	23	3068	30,76	6678	23	2415	7840	6536	101	0,00	4,77	23	2415	6678	6536	57	99,13	1,01	6536
nw34	20	899	28,06	10488	20	750	10488	**10488**	94	100,00	0,07	20	750	10488	**10488**	5	100,00	0,02	10488
nw35	23	1709	26,70	7216	23	1403	7216	**7216**	110	100,00	0,10	23	1403	7216	**7216**	41	100,00	0,12	7216
nw36	20	1783	36,90	7314	20	1408	7328	7281	400	91,83	7,17	20	1408	7324	6635	806	0,00	6,41	7265
nw37	19	770	25,82	10068	19	639	10068	**10068**	28	100,00	0,02	19	639	10068	**10068**	6	100,00	0,01	10068
nw38	23	911	31,44	5558	23	911	5558	**5558**	109	100,00	0,07	23	911	5558	**5558**	60	100,00	0,04	5558
nw39	25	677	26,55	10080	25	567	10080	**10080**	65	100,00	0,09	25	567	10080	**10080**	16	100,00	0,03	10080
nw40	19	404	26,95	10809	19	336	10809	**10809**	72	100,00	0,04	19	336	10809	**10809**	46	97,92	0,02	10809
nw41	17	197	22,10	11307	17	177	11307	**11307**	13	98,87	0,09	17	177	11307	**11307**	7	100,00	0,01	11307
nw42	23	1079	26,32	7656	23	895	7666	7629	133	90,39	5,08	23	895	7656	**7656**	49	99,66	0,07	7656
nw43	18	1072	25,18	8904	17	982	8904	**8904**	94	99,08	0,08	17	982	8904	**8904**	24	100,00	0,06	8904
kl01	55	7479	13,67	1086	47	5970	1089	1085	463	90,99	29,85	47	5957	1087	1085	454	94,98	3,77	1085
kl02	71	36699	8,16	219	69	16734	220	216	629	69,22	62,79	69	16734	220	216	91	72,06	32,79	216
us04	163	28016	6,52	17854	105	12169	18269	17497	456	0,00	78,44	101	8900	17854	17732	141	98,83	19,71	17732
aa01	823	8904	1,00	56138	624	7767	57528	55405	875	0,00	52,08	620	7687	57528	55464	2076	0,08	39,77	55601
aa03	825	8627	0,99	49649	569	7208	50747	49623	331	0,97	31,45	563	7047	50747	49610	670	0,87	25,13	49638
aa04	426	7195	1,70	26374	343	6215	29888	25961	904	0,00	29,80	343	6200	29888	25954	1524	0,00	26,04	25964
aa05	801	8308	0,99	53839	555	6585	54014	53701	493	58,44	28,53	542	6428	53935	53745	819	74,50	23,51	53776
aa06	646	7292	1,10	27040	516	6276	27293	27000	251	18,75	22,45	511	6153	27293	27002	516	19,06	17,38	27006
cyc06	240	432	1,16	112	240	432	120	112	64	0,00	5,26	240	432	112	**112**	64	0,00	0,16	112
cyc07	672	1120	0,45	352	672	1120	352	**352**	139	0,00	0,66	672	1120	352	**352**	128	0,00	0,28	352
cyc08	1792	2816	0,18	1024	1792	2816	1048	1024	256	0,00	30,44	1792	2816	1036	1024	256	0,00	10,69	1024
cyc09	4608	6912	0,07	[2844,2816]	4608	6912	2844	2816	512	0,00	102,26	4608	6912	2830	2815	512	0,00	87,26	2816
cyc10	11520	16640	0,03	7424	11520	16640	7424	**7424**	1024	0,00	235,03	11520	16640	7424	7421	1030	0,00	413,80	7424
cyc11	28160	39424	0,01	[19016,18944]	28160	39424	19016	18944	2048	0,00	1687,92	28160	39424	18998	18929	2076	0,00	641,85	18944

Tabela 4.1: Resultados para o RC-CDSL e para o RC-BC.

Em tempo de otimização, o RC-BC é melhor que o RC-CDSL na grande maioria dos testes. Os casos seguintes são destacáveis. Nas instâncias nw03, nw42 e us04, o RC-BC supera o RC-CDSL. Na primeira, há uma grande redução de tempo. Nas duas últimas, além de ser mais rápido, o RC-BC atinge limitantes superior e inferior melhores. O cenário inverso ocorre no teste cyc10. Neste teste, o RC-CDSL obtém melhores valores relativos ao tempo e ao limitante inferior.

Os limitantes alcançados pelas duas alternativas, contudo, são, em geral, bastante parecidos. Para os limitantes superiores, dentre todos os casos, há apenas 13 desempates. Estes, no entanto, são todos a favor do RC-BC. Uma diferença expressiva, de 14,82%, acontece para a instância nw33, quando os valores obtidos pelo RC-BC e pelo RC-CDSL são 6678 e 7840 respectivamente.

Em limitantes inferiores, cada algoritmo supera o outro em 8 testes. Nos demais, os resultados alcançados são idênticos. As diferenças são poucos expressivas, a maior ocorrendo para a instância nw36. Nesta situação, há uma variação em favor do RC-CDSL. Os valores atingidos são 7281 e 6635.

Quanto aos cortes adicionados nos algoritmos, nota-se que a qualidade é mais importante que a quantidade. Em 28 dos 45 casos, há menos cortes adicionados ao final do RC-BC. Entretanto, em 27 destes, o mesmo produz um limitante inferior melhor ou igual ao do seu concorrente. Um exemplo acontece para a instância us04. Para o RC-BC e para o RC-CDSL, os valores correspondentes são 17732 e 141 e 17497 e 456. Tal cenário também ocorre no sentido inverso.

A fixação de variáveis provoca uma certa discrepância entre os dois algoritmos. Na maioria dos testes, o RC-BC tem mais sucesso nesse quesito e os números obtidos são bastante próximos. Em alguns casos, contudo, vê-se que um dos valores é 0, enquanto o outro é próximo de 100. Uma provável razão para tal desnível é a ausência de melhores limitantes para a alternativa perdedora. Para a instância nw36, por exemplo, o *gap* atingido pelo RC-BC está aquém do alcançado pelo RC-CDSL.

Um questionamento esperado do leitor é sobre o que representa a coluna ⌈ÓtiLR⌉, a última da Tabela 4.1. A função da mesma é mostrar o potencial do RC-BC. Lembre-se que, como discutido na Subseção 2.2.1 (e decorrente da Proposição 2.9 da Subseção 2.2.2), o limitante inferior teórico obtido com uma relaxação Lagrangiana é maior ou igual ao atingido com a relaxação linear. Na citada coluna, trazem-se os valores ótimos arredondados (para um inteiro maior ou igual) dos LPs constituídos pelas relaxações lineares das intâncias acrescidas dos cortes presentes no *pool* ao final do RC-BC. Assim, são apresentados limitantes inferiores plenamente alcançáveis pelo RC-BC.

É importante esclarecer, ainda, que todos os limitantes inferiores apresentados na Tabela 4.1 estão arredondados. Isto, pois, para todas as instâncias utilizadas, têm-se somente custos inteiros.

Nota-se, então, que, para vários casos, o limitante inferior atingido pelo RC-BC poderia ser mais preciso. É razoável supor, portanto, que se pode elevá-lo. E isto é, de fato, concretizável. Como discutido antes, com o uso do MS, a dificuldade maior está em determinar os tamanhos de passo. Com configurações diferentes para os parâmetros do mesmo, podem-se conseguir tamanhos mais adequados e, consequentemente, melhores limitantes.

Com as constatações acima, vê-se que os resultados obtidos com o algoritmo *relax-and-cut* desenvolvido em [15] são compatíveis com os produzidos pela correspondente componente do *branch-and-cut* implementado. Pode-se dizer, ainda, que o desempenho da última alternativa é superior. Assim, é possível proceder, sem qualquer débito, com a avaliação pretendida.

O método exato proposto é avaliado por meio de duas comparações: com o algoritmo *relax-and-cut* empregado em [15] e com o *branch-and-cut* baseado em relaxação linear implementado no *software* comercial de otimização CPLEX versão 12.1.0. Na primeira comparação, pesquisa-se se a extensão é mais precisa que o algoritmo original em encontrar uma solução ótima do SPP em um tempo razoável. Na segunda, desvenda-se se tal solução é obtida eficientemente.

Na sequência desta subseção, o algoritmo *relax-and-cut* proposto em [15], o *branch-and-cut* baseado em relaxação Lagrangiana avaliado aqui e o *branch-and-cut* baseado em relaxação linear implementado no CPLEX são tratados, respectivamente, por RC-CDSL, BCLAG e BCLIN-CPX.

Na Tabela 4.2, trazem-se os resultados produzidos pelos dois métodos exatos. Nas cinco primeiras colunas da mesma, mostram-se as informações igualmente intituladas na Tabela 4.1. Nas seis seguintes, relatam-se as execuções do BCLAG – o valor da melhor solução inteira encontrada, LSup, o *gap* de limitantes alcançado (em porcentagem), Gap, o número de subproblemas criados, NSubp, as quantidades de cortes adicionados, NCor e NCorG e o tempo de otimização (em segundos), Tem. Nas demais colunas, trata-se de forma semelhante o BCLIN-CPX.

Aqui, algumas explicações são necessárias. Primeiramente, os métodos exatos são executados com um limite de tempo. O mesmo é de 7200 segundos ou 2 horas. Assim, tais métodos não obtêm garantidamente soluções ótimas para as instâncias. Por este motivo, são mostrados, na Tabela 4.2, o valor da melhor solução viável encontrada e o *gap* atingido. Caso a otimalidade seja provada, nenhum percentual é especificado na coluna Gap. Em segundo lugar, a fim de enriquecer o relato apresentado, é revelada, para o BCLAG, além do número de cortes presentes no *pool* ao final do processo, a quantidade de inequações geradas durante a execução.

Para a análise dos resultados, é útil classificar as intâncias utilizadas quanto às suas dimensões. A classificação sugerida abaixo provê, a grosso modo, uma hierarquia de difi-

culdade entre as mesmas. São gerados, nesta classificação, três grupos: *grupo 1*, formado pelas instâncias dos tipos nw, kl e us; *grupo 2*, constituído dos casos do tipo aa; e *grupo 3*, composto pelos testes do tipo cyc.

O grupo 1 contém majoritariamente instâncias pequenas, com poucas restrições e variáveis. As instâncias de maiores dimensões, as sete primeiras e as três últimas, possuem quantidades razoavelmente grandes de colunas em suas matrizes de restrições. As mesmas, contudo, ainda têm proporcionalmente um número bastante pequeno de linhas. Uma instância do SPP com tais características é, em geral, um problema fácil. Assim, espera-se que os testes deste grupo sejam os resolvidos mais rapidamente.

Comparadas às do grupo 1, as instâncias do grupo 2 apresentam um número substancialmente maior de restrições. Além disso, nas últimas, a diferença entre as quantidades de linhas e de colunas na matriz de restrições é bem menor. Resolver uma instância do SPP com tal formato tende a ser uma tarefa mais difícil. Logo, é natural que os tempos de execução obtidos para o grupo 2 sejam maiores que os conseguidos para o grupo 1.

Os testes comentados até agora são instâncias reais do *problema de escalonamento de tripulações de voo* apresentadas em [39] por Hoffman e Padberg. Os mesmos são bastante conhecidos e amplamente utilizados na literatura, podendo ser considerados como mandatórios para a avaliação de um método exato para o SPP.

Os testes contidos no grupo 3 têm origem diferente da citada acima. Cavalcante *et al.* designam instâncias do SCP abordadas em [30] e consideradas bastante difíceis. Para serem usadas como instâncias do SPP em [15], essas têm, então, as inequações substituídas por igualdades. As mesmas caracterizam-se pelas duas seguintes propriedades: quantidades de restrições e de variáveis muito próximas e número bastante grande de restrições, enorme para as quatro últimas. É sabido que uma instância do SPP com tal desenho tende a ser de alta complexidade computacional. É verossímil, portanto, que os referidos testes sejam, dentre todos, os mais custosos computacionalmente.

Com as definições acima, pode-se, então, analisar as comparações descritas anteriormente guiando-se pelos grupos de instâncias determinados. Os resultados das execuções são discutidos a seguir.

Vê-se, por meio da Tabela 4.1, que a execução mais demorada do RC-CDSL dura 1 687,92 segundos ou 28,13 minutos. Visando à primeira comparação, parece razoável confrontar, então, para um limite de tempo baseado em tal dado, a quantidade de instâncias para as quais este algoritmo e o BCLAG conseguem obter uma solução ótima. Toma-se, aqui, esse limite como 29 minutos ou 1740 segundos. A comparação segue.

Ao observar novamente a Tabela 4.1, nota-se o bom desempenho do RC-CDSL para as instâncias do grupo 1. O mesmo encontra uma solução ótima para 23 dessas, 67,65% do total. Por meio da Tabela 4.2, contudo, vê-se, também, que, com o BCLAG, é possível resolver 32 das mesmas, 94,12% do total. Ainda, para os únicos dois casos nos quais o úl-

Instância					BCLAG						BCLIN-CPX				
Nom	NRes	NVar	Den	Óti	LSup	Gap	NSubp	NCor	NCorG	Tem	LSup	Gap	NSubp	NCor	Tem
nw03	59	43749	14,10	24492	24492	-	1	355	554	100,88	24492	-	1	4	0,81
nw04	36	87482	20,22	16862	16868	3,41	475	5248	136911	7200,00	16862	-	515	32	17,57
nw06	50	6774	18,17	7810	7810	-	1	429	515	3,30	7810	-	1	3	0,20
nw11	39	8820	16,64	116256	116256	-	1	297	449	4,01	116256	-	1	0,07	0,00
nw13	51	16043	12,78	50146	50146	-	1	295	390	11,79	50146	-	1	0	0,16
nw17	61	118607	13,96	11115	11115	-	13	1992	8287	1065,77	11115	-	1	22	7,05
nw18	124	10757	6,82	340160	340160	-	15	2931	5892	117,39	340160	-	1	11	0,36
nw20	22	685	24,70	16812	16812	-	1	106	171	0,08	16812	-	1	7	0,01
nw21	25	577	24,89	7408	7408	-	1	12	29	0,05	7408	-	1	2	0,02
nw22	23	619	23,87	6984	6984	-	1	58	93	0,06	6984	-	1	0	0,03
nw23	19	711	24,80	12534	12534	-	1	83	464	0,26	12534	-	1	0	0,02
nw24	19	1366	33,20	6314	6314	-	1	138	310	0,18	6314	-	1	3	0,03
nw25	20	1217	30,16	5960	5960	-	1	24	63	0,05	5960	-	1	2	0,03
nw26	23	771	23,77	6796	6796	-	1	48	82	0,08	6796	-	1	0	0,02
nw27	22	1355	31,52	9933	9933	-	1	31	68	0,10	9933	-	1	1	0,03
nw28	18	1210	39,27	8298	8298	-	1	31	40	0,07	8298	-	1	2	0,03
nw29	18	2540	31,04	4274	4274	-	9	534	935	5,02	4274	-	1	11	0,07
nw30	26	2653	29,63	3942	3942	-	1	61	111	0,28	3942	-	1	3	0,07
nw31	26	2662	28,86	8038	8038	-	1	45	82	0,39	8038	-	1	4	0,08
nw32	19	294	24,29	14877	14877	-	1	139	337	0,15	14877	-	1	0	0,02
nw33	23	3068	30,76	6678	6678	-	5	57	69	1,53	6678	-	1	0	0,06
nw34	20	899	28,06	10488	10488	-	1	5	5	0,03	10488	-	1	6	0,03
nw35	23	1709	26,70	7216	7216	-	1	41	71	0,16	7216	-	1	0	0,02
nw36	20	1783	36,90	7314	7314	-	25	328	5965	25,96	7314	-	1	8	0,10
nw37	19	770	25,82	10068	10068	-	1	6	6	0,02	10068	-	1	1	0,02
nw38	23	911	31,44	5558	5558	-	1	60	101	0,04	5558	-	1	1	0,03
nw39	25	677	26,55	10080	10080	-	1	16	37	0,04	10080	-	1	3	0,03
nw40	19	404	26,95	10809	10809	-	1	46	55	0,03	10809	-	1	0	0,02
nw41	17	197	22,10	11307	11307	-	1	7	14	0,01	11307	-	1	0	0,01
nw42	23	1079	26,32	7656	7656	-	1	49	108	0,09	7656	-	1	1	0,03
nw43	18	1072	25,18	8904	8904	-	1	24	53	0,07	8904	-	1	1	0,02
kl01	55	7479	13,67	1086	1086	-	19	685	2229	23,84	1086	-	5	11	0,41
kl02	71	36699	8,16	219	219	1,39	2313	200	11282	7200,00	219	-	6	0	0,66
us04	163	28016	6,52	17854	17854	-	5	141	159	31,48	17854	-	1	0	0,32
aa01	823	8904	1,00	56138	57453	3,59	225	13139	50431	7200,00	56138	-	179	27	9,51
aa03	825	8627	0,99	49649	49649	-	7	1105	1617	100,02	49649	-	1	2	0,79
aa04	426	7195	1,70	26374	27949	7,69	947	5013	59579	7200,00	26374	-	504	10	9,29
aa05	801	8308	0,99	53839	53839	-	1745	1668	55142	6382,62	53839	-	13	14	1,37
aa06	646	7292	1,10	27040	27040	-	93	2896	9541	954,66	27040	-	3	10	0,68
cyc06	240	432	1,16	112	112	-	1	64	145	0,16	112	-	1	68	0,24
cyc07	672	1120	0,45	352	352	-	1	128	328	0,32	352	-	151	122	4,82
cyc08	1792	2816	0,18	1024	1024	-	3	256	837	11,66	1024	-	9	257	19,38
cyc09	4608	6912	0,07	[2844,2816]	2830	0,53	281	512	512	7200,00	2816	-	1	526	122,88
cyc10	11520	16640	0,03	7424	7424	0,04	133	1030	3895	7200,00	7424	-	20	1026	2901,42
cyc11	28160	39424	0,01	[19016,18944]	18989	0,32	69	2048	8561	7200,00	19160	*	1	6080	7200,00

Tabela 4.2: Resultados para o BCLAG e para o BCLIN-CPX.

timo método não determina o valor ótimo, a saber, nw04 e kl02, podem-se apontar justificativas para o insucesso.

Como relatam Hoffman e Padberg, a instância nw04 é intrinsecamente difícil. Os autores não conseguem, em [39], identificar aspectos dessa instância, como, por exemplo, a distribuição dos elementos da matriz de restrições, que possam ser responsabilizados por tal dificuldade; apesar de a observarem nos resultados obtidos. Os mesmos mensuram, que, no referido trabalho, um grande esforço foi feito para que esse teste se tornasse tratável pelo algoritmo em questão.

Na execução do BCLAG para o caso kl02, nota-se o seguinte comportamento. O melhor limitante superior possível, igual ao valor ótimo do problema, é encontrado rapidamente. No processamento de um subproblema, em geral, tal cenário repete-se: uma solução ótima é obtida em pouco tempo. Os limitantes inferiores produzidos, porém, não conseguem, tipicamente, superar um certo nível de precisão. Assim, são efetuadas muitas divisões. Aparentemente, para esse teste, a relaxação Lagrangiana usada e as inequações clique não conseguem prover uma melhor qualidade, comparada à fornecida pela relaxação linear, aos referidos limitantes. No entanto, é válido revelar que, ao longo da enumeração, muitas podas também são realizadas. Em um intervalo razoavelmente longo de iterações, podem-se observar, aproximadamente, as mesmas quantidades de podas e de divisões. Durante todo o processo, é pequena a quantidade de subproblemas na lista L (Algoritmo 3.10). Quando a execução é interrompida, apenas 6 dos 2313 subproblemas criados permanecem não-processados.

Para o grupo 2, o RC-CDSL não se mostra tão eficiente. Apesar dos excelentes limitantes obtidos, para nenhuma das 5 instâncias, uma solução ótima é encontrada. O BCLAG, por sua vez, resolve 2 dos testes, 40% do total. Como posto antes, as instâncias do grupo 2 tendem a ser de maior complexidade computacional. Ainda, dificuldades adicionais com os casos aa01 e aa04 são esperadas. Como o teste nw04, os últimos são frequentemente ditos muito difíceis. Börndorfer [13], por exemplo, relata que esses são, dentre os problemas englobadas nos grupos 1 e 2, os de maior complexidade computacional.

As instâncias formadoras do grupo 3, são as de maiores dimensões dentre todas. Dadas as complicações já mencionadas para lidar com esses problemas, o desempenho do RC-CDSL é considerado bom. Para 2 dos 6 testes, um de menor e outro de maior porte, uma solução ótima é encontrada. Os resultados do BCLAG também são bons. Apesar de o mesmo não resolver um dos casos citados, para um número maior de testes, 3, o valor ótimo é determinado.

No cômputo geral, nota-se que, nesta comparação, o RC-CDSL é superado pelo BCLAG. Com o RC-CDSL, 25 das 45 instâncias, 55,55% do total, são resolvidas. Com o BCLAG, atinge-se a otimalidade em 37 casos, 82,22% do total.

Com a incorporação do *relax-and-cut* a uma enumeração, é possível, então, solucionar,

em um tempo razoável, um número substancialmente maior de instâncias representativas do SPP. É válido, porém, questionar: Os problemas são resolvidos de maneira eficiente? Pode-se responder a esta pergunta mediante uma segunda comparação: entre as execuções do *branch-and-cut* proposto e as de outro método habitualmente usado para tratar o SPP. O escolhido aqui é o *branch-and-cut* baseado em relaxação linear implementado no *software* comercial CPLEX versão 12.1.0.

Em [15], Cavalcante *et al.* realizam uma comparação que se relaciona à supracitada. Nessa comparação, o algoritmo *relax-and-cut* apresentado é confrontado com o *branch-and-cut* baseado em relaxação linear implementado no *software* XPRESS versão 16.01.05. Observa-se que, para as instâncias do grupo 3, os resultados da primeira alternativa são comparáveis aos da segunda. Para os testes dos grupos 1 e 2, contudo, o método exato leva larga vantagem. Tais constatações, porém, não desestimulam o uso dos algoritmos *relax-and-cut*. Como posto em [15], enquanto o algoritmo proposto pelos autores é o primeiro desse tipo para o SPP, algoritmos baseados em relaxação linear aplicados ao mesmo são frutos de vários anos de melhoramentos e refinamentos.

A análise feita em [15] difere da sugerida nesta dissertação por não ser completa. Isto, no sentido em que os algoritmos confrontados são distintos, apenas um sendo exato. Ao estabelecer um paralelo entre o método desenvolvido aqui e o método implementado no CPLEX, avalia-se, de fato, a eficiência de um algoritmo baseado em relaxação Lagrangiana na resolução do SPP. A comparação segue.

As alternativas estudadas são executadas em um mesmo ambiente computacional: o descrito no início da seção. Na já mencionada Tabela 4.2, apresentam-se os resultados obtidos (para a instância cyc11, o *gap* final não está disponível no *log* do CPLEX). Vê-se, na mesma, um comportamento semelhante ao ocorrido em [15]. Para as instâncias dos grupos 1 e 2, o BCLIN-CPX tem um desempenho incontestavelmente melhor que o do BCLAG. Somente para os testes do grupo 3, tem-se um equilíbrio: nos 3 primeiros casos, o BCLAG é superior, nos 3 demais, o cenário é invertido. Em favor do BCLAG, é possível destacar a instância cyc07. Para essa instância, o mesmo necessita de menos de 1 segundo e de 1 subproblema para encontrar uma solução ótima. A execução do BCLIN-CPX, por outro lado, leva 4,82 segundos, com a criação 151 subproblemas.

4.2.3 Conclusão

Nesta seção, foi proposto e avaliado um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP. Para tanto, foi feita uma extensão do algoritmo *relax-and-cut* desenvolvido em [15], o mesmo sendo incorporado a uma enumeração.

Com o método exato produzido, é possível solucionar, de forma prática, ou seja, em um limite de tempo bastante razoável, um número substancialmente maior de instâncias

representativas do SPP. No entanto, tal método mostra-se pouco competitivo quando comparado ao *branch-and-cut* baseado em relaxação linear implementado no resolvedor comercial CPLEX versão 12.1.0.

O revés obtido nessa comparação, contudo, não reflete o potencial da técnica empregada aqui. A incorporação do *relax-and-cut* a uma enumeração ainda não foi suficientemente testada a ponto de ser refinada de modo satisfatório. O trabalho realizado nesta dissertação apresenta-se como um passo em tal direção.

Capítulo 5

Um *branch-and-cut* para o SPP baseado em relaxações alternativas

No capítulo passado, foi proposto um *branch-and-cut* baseado em relaxação Lagrangiana para o SPP. O mesmo representou um avanço na resolução de instâncias do SPP por um algoritmo fundamentado em relaxação Lagrangiana. No entanto, o *branch-and-cut* desenvolvido não apresentou um bom desempenho quando comparado a outro algoritmo largamente utilizado para solucionar o SPP: o *branch-and-cut* baseado em relaxação linear implementado no resolvedor comercial de otimização CPLEX.

É válido notar, contudo, que, no *branch-and-cut* proposto, a relaxação Lagrangiana aplicada ao SPP é a usual, onde todas as restrições são dualizadas, e que, como mencionado anteriormente, outras opções podem ser empregadas. Neste contexto, dois aspectos são importantes: a força da formulação presente na relaxação e a quantidade de multiplicadores criados. Tais aspectos têm grande influência nos limitantes produzidos e na dificuldade encontrada na resolução do PDL e, portanto, no desempenho de um *branch-and-cut* baseado em relaxação Lagrangiana. Sabe-se que, em se tratando de uma relaxação Lagrangiana, há, no fundo, um compromisso entre a complexidade computacional do PRL e a qualidade dos limitantes obtidos na resolução do PDL. Assim, é interessante investigar, também, como o referido *branch-and-cut* comporta-se com o emprego de relaxações Lagrangianas alternativas. Espera-se que, com o uso das mesmas, melhores resultados possam ser alcançados.

Para o SPP, é comumente possível identificar subconjuntos de igualdades que representem problemas fáceis. Ao construir uma relaxação Lagrangiana para o SPP, é factível, então, manter tais restrições no PRL, ou seja, não dualizá-las. O motivo é que, constituído dessas restrições, o PRL permanece um problema de pouca complexidade computacional. Com isso, é derivada uma relaxação Lagrangiana alternativa para o SPP. Há vários subconjuntos desses bem conhecidos. Alguns exemplos são apresentados na próxima seção.

Também, existem, tipicamente, dois (ou até mais) grupos de restrições do SPP com a referida propriedade. Para este caso, uma técnica é adequada. Com a aplicação dessa técnica, ambos os grupos de restrições são mantidos no PRL. Isto, contudo, de maneira que o PRL mantenha-se de fácil resolução. Trata-se da decomposição Lagrangiana, descrita na Seção 5.2.

A fim de realizar a anunciada investigação, são avaliadas, neste capítulo, duas novas situações. O algoritmo proposto no capítulo anterior é testado com o emprego das seguintes relaxações Lagrangianas para o SPP:

1. a derivada com a manutenção de um subconjunto de igualdades que represente um problema fácil no PRL e
2. a decomposição Lagrangiana, construída com dois subconjuntos de igualdades que representem problemas fáceis não sendo dualizados.

Os principais aspectos de implementação das novas versões do *branch-and-cut* e os resultados e conclusões obtidos são apresentados nas Seções 5.4, 5.5 e 5.6 respectivamente.

5.1 Identificação de problemas fáceis

Como comentado antes nesta dissertação, muitos problemas difíceis possuem outros fáceis como componentes. Por exemplo, encontrar uma árvore geradora mínima é uma tarefa a ser cumprida em muitos problemas difíceis. Ainda, de um modo geral, quando a estrutura de um problema fácil componente existe, mas não é visível, é possível aplicar uma transformação que a evidencie.

Existem várias técnicas eficientes para atacar um problema difícil que se baseiam na identificação de problemas fáceis componentes. Relaxação Lagrangiana é uma dessas. Como consequência, tal identificação designa comumente um importante passo dado para a resolução de um problema difícil. Também, quanto maiores, em proporção, são as estruturas associadas aos problemas fáceis componentes, melhor desempenham-se essas técnicas.

Considere um problema difícil modelado como um IP. Uma maneira direta de identificar problemas fáceis componentes do mesmo é detectar partes da matriz de restrições do IP que constituam um conjunto de restrições que os representem. Há, tipicamente, nessa matriz, estruturas especiais que, quando consideradas isoladamente, expressam tal situação. Para ilustrá-las, apresentam-se, a seguir, duas bastante recorrentes. A última é a estrutura procurada na matriz de restrições do SPP nas implementações feitas neste capítulo.

Definição 5.1. Tem-se uma *matriz totalmente unimodular* (MTU) se o determinante de qualquer submatriz quadrada da mesma é igual a 0, a 1, ou a -1.

Seja A uma MTU. Se um IP tem o conjunto de restrições dado por $Ax \leq b$ onde b possui somente entradas inteiras, o mesmo é um problema de fácil resolução. O motivo é que uma solução ótima da relaxação linear do mesmo é garantidamente inteira. Assim, é possível resolvê-lo em tempo polinomial.

Definição 5.2. Tem-se uma *matriz de rede pura* (MRP) se, para a mesma, as duas seguintes condições são satisfeitas:

1. todas as entradas pertencem ao conjunto $\{0, 1, -1\}$ e,

2. em uma coluna, há no máximo uma entrada 1 e no máximo uma entrada -1.

Definição 5.3. Em uma matriz, uma *reflexão de linha* é uma operação que inverte os sinais de todos os coeficientes não nulos de uma linha.

Definição 5.4. Tem-se uma *matriz de rede refletida pura* (MRRP) se existe uma sequência de reflexões de linha que torna a mesma uma MRP.

É possível mostrar que uma MRRP é também uma MTU. Logo, se A é uma MRRP e um IP tem o conjunto de restrições como definido acima, o mesmo é também um problema fácil. A novidade, aqui, é que tal IP ainda pode ser solucionado em tempo polinomial através de um algoritmo para um *problema de fluxo em redes*.

É importante destacar que, comparada a uma grande MTU genérica, uma grande MRRP tende a ser mais facilmente identificada como uma submatriz da matriz de restrições de um IP. Uma ressalva é que determinar a maior MRRP possível é um problema $\mathcal{NP}$-Difícil [8]. Há, contudo, na literatura, várias heurísticas com esse propósito [35, 2, 11, 12, 34].

5.2 Decomposição Lagrangiana

Para um problema difícil modelado como um IP, existem comumente duas ou mais opções interessantes de relaxação Lagrangiana. Como apontado por Beasley [9], escolher entre essas opções é uma questão estratégica. Em especial, examine o caso em que, para o IP tratado, existam dois subconjuntos de restrições que representem problemas fáceis (exemplos de subconjuntos como esses foram dados na seção anterior). Neste caso, é preciso, a princípio, tomar uma decisão. Tais subconjuntos representam problemas fáceis apenas se considerados isoladamente (supõe-se que, em caso contrário, os mesmos são somente um conjunto). Não é praticável, portanto, mantê-los, de forma direta, ambos no PRL. Surgem naturalmente duas opções de relaxação Lagrangiana: dualizar as restrições relativas a um ou a outro.

Há uma relaxação alternativa, contudo, com a qual foge-se de tal decisão. O intuito da mesma é aproveitar ambas as estruturas conhecidas. Com notação semelhante à empregada em [32], esta relaxação pode ser descrita como a seguir.

Como na discussão sobre relaxação Lagrangiana feita na Subseção 2.2.1, considere que o IP a ser relaxado é o problema P dado por

$$\begin{array}{rll} z = \min & cx & \\ & Ax \leq b & \\ & Cx \leq d & \\ & x \in \mathbb{Z}^n_+. & \end{array} \tag{5.1}$$

Suponha, também, que, se P contivesse apenas um dos conjuntos de restrições $Ax \leq b$ ou $Cx \leq d$, o mesmo seria um problema de fácil resolução.

Observe que, a partir de P, é possível definir o seguinte problema P$'$ equivalente.

$$\begin{array}{rll} z = \min & cx & \\ & Ax \leq b & \\ & Cy \leq d & \\ & x = y & \\ & x \in \mathbb{Z}^n_+ & \\ & y \in \mathbb{Z}^n_+ & \end{array} \tag{5.2}$$

Os problemas P e P$'$ são equivalentes no sentido de terem conjuntos de soluções entre os quais se pode fazer uma correspondência bijetiva. Além disso, o valor ótimo é o mesmo para os dois problemas. Note que, se a solução $x = x'$ é válida para P, a solução $x = x'$, $y = x'$ é válida para P$'$. E ambas têm o mesmo custo. Analogamente, se a solução $x = x'$, $y = y'$ satisfaz as restrições de P', a solução $x = x'$ o faz para P. Isto, pois $x' = y'$. Ainda, os custos dessas soluções são iguais.

Relaxe, então, P$'$ dualizando as retrições $x = y$, que igualam as cópias de variáveis. O resultado é

$$\begin{array}{rll} z(\lambda) = \min & cx + \lambda(y - x) & \\ & Ax \leq b & \\ & Cy \leq d & \\ & x \in \mathbb{Z}^n_+ & \\ & y \in \mathbb{Z}^n_+ & \end{array} \tag{5.3}$$

ou ainda

$$\begin{array}{rllll} z(\lambda) = \min & (c - \lambda)x + \min & \lambda y & & \\ & Ax \leq b & & Cy \leq d & \\ & x \in \mathbb{Z}^n_+ & & y \in \mathbb{Z}^n_+. & \end{array}$$

Vê-se que, com isso, P' é decomposto em dois outros problemas. Estes problemas são fáceis e independentes: o *problema* x, associado ao par $(Ax \leq b, x \in \mathbb{Z}^n_+)$,

e o *problema* y, vinculado ao par $(Cy \leq d, y \in \mathbb{Z}^n_+)$. Aqui está a origem do nome dado a tal relaxação: decomposição Lagrangiana [33]. Pode-se notar, ainda, que uma decomposição Lagrangiana respeita a definição de relaxação dada na Seção 2.2. Tomando $S = \{x \in \mathbb{Z}^n_+, y \in \mathbb{Z}^n_+, Ax \leq b, Cy \leq d, x = y\}$, $f(x) = cx + \lambda(y - x)$ e $T = \{x \in \mathbb{Z}^n_+, y \in \mathbb{Z}^n_+, Ax \leq b, Cy \leq d\}$, é possível associar os modelos (5.2) e (5.3) aos modelos (2.4) e (2.5), ocorrendo claramente $T \supseteq S$ e $f(x) \leq x, \forall x \in S$.

A introdução de cópias de variáveis é um "truque" usado há algum tempo no contexto de relaxação Lagrangiana. Em [33], Guignard e Kim mostram e reunem várias propriedades de uma relaxação derivada de tal maneira: a descrita acima. Abaixo, comentam-se algumas dessas propriedades.

O resultado mais importante a ser destacado é que, para P (5.1), o melhor limitante obtido com a decomposição Lagrangiana (5.3) é maior ou igual ao melhor limitante alcançado com as relaxações Lagrangianas RP_1 e RP_2, onde, respectivamente, as restrições $Cx \leq d$ e $Ax \leq b$ são mantidas no PRL. Guignard e Kim mostram que o primeiro melhor limitante é dado por

$$\min\{cx : x \in \{conv(\{x \in \mathbb{Z}^n_+, Ax \leq b\}) \cap conv(\{x \in \mathbb{Z}^n_+, Cx \leq d\})\}\}.$$

Lembrando que o melhor limitante conseguido com uma relaxação Lagrangiana é expresso pela equação (2.8), vê-se que o resultado destacado acima decorre da Proposição 2.9.

Como acontece para a relação entre limitantes obtidos com uma relaxação Lagrangiana e com a relaxação linear correspondente (estabelecida na Subseção 2.2.1), há casos em que, no resultado supracitado, a superioridade não se concretiza. Se a relaxação RP_1 tem a propriedade da integralidade (mencionada na referida subseção), o melhor limitante gerado pela decomposição Lagrangiana é igual ao produzido pela relaxação RP_2. Isto, pois

$$\begin{aligned} \min \quad & \{cx : x \in \{conv(\{x \in \mathbb{Z}^n_+, Ax \leq b\}) \cap conv(\{x \in \mathbb{Z}^n_+, Cx \leq d\})\}\} = \\ & \min\{cx : Cx \leq d, x \in conv(\{x \in \mathbb{Z}^n_+, Ax \leq b\})\}. \end{aligned}$$

Caso a relaxação RP_2 goze de tal propriedade, os melhores limitantes alcançados com a decomposição Lagrangiana e com a relaxação RP_1 coincidem. A razão é que

$$\begin{aligned} \min \quad & \{cx : x \in \{conv(\{x \in \mathbb{Z}^n_+, Ax \leq b\}) \cap conv(\{x \in \mathbb{Z}^n_+, Cx \leq d\})\}\} = \\ & \min\{cx : Ax \leq b, x \in conv(\{x \in \mathbb{Z}^n_+, Cx \leq d\})\}. \end{aligned}$$

Se ocorrem as duas situações, os melhores limitantes conseguidos com as três relaxações são idênticos ao atingido com a linear.

Uma decomposição Lagrangiana ainda pode ser vista como uma generalização de uma relaxação Lagrangiana. De fato, é possível reproduzir, através da primeira técnica, qualquer limitante gerado pela última. Uma vantagem de uma decomposição Lagrangiana é que a reformulação promovida na mesma pode revelar estruturas de problemas fáceis

componentes ocultas. Em [33], Guignard e Kim ilustram tal situação.

Por fim, é possível que haja, em P, um terceiro conjunto de restrições: desassociado aos dois primeiros e sem que P, constituído somente deste conjunto, torne-se um problema fácil. Sendo $Ex \leq f$ tal conjunto, P é dado por

$$\begin{aligned} z = \min \quad & cx \\ & Ax \leq b \\ & Cx \leq d \\ & Ex \leq f \\ & x \in \mathbb{Z}^n_+. \end{aligned}$$

Ao construir uma decomposição Lagrangiana, pode-se, então, dualizar as restrições $Ex \leq f$ ou mantê-las no problema x e no problema y ou em apenas um desses. Com as últimas opções, tem-se um PRL potencialmente mais difícil. Com a dualização, contudo, produz-se, provavelmente, um limitante menos preciso.

5.3 O uso das relaxações alternativas

Como detalhado na seção a seguir, nas novas versões do *branch-and-cut* baseado em relaxação Lagrangiana implementadas neste capítulo, adotam-se as seguintes práticas. Primeiramente, ao contrário do que ocorre no capítulo anterior, a restrição $\sum_j x_j \leq m$ não está presente no PRL resolvido em um subproblema – tal restrição está presente no PRL (4.1). Em segundo lugar, busca-se identificar e não dualizar, dentre o conjunto de restrições do SPP, subconjuntos de restrições que contenham do lado esquerdo uma estrutura de MRRP. Uma relaxação Lagrangiana onde somente um subconjunto de restrições como esse é mantido no PRL possui a propriedade da integralidade (citada na Subseção 2.2.1). Logo, pode-se ver que os melhores limitantes gerados pelas relaxações alternativas empregadas aqui não são melhores que o atingido com a relaxação linear do SPP. Mesmo assim, as primeiras relaxações ainda podem ser mais vantajosas que a última. É possível que tais relaxações sejam, na prática, mais facilmente resolvidas.

Em relação à relaxação Lagrangiana tradicionalmente aplicada ao SPP, onde todas as restrições são dualizadas, as relaxações alternativas utilizadas aqui apresentam vantagens. Com a relaxação onde um subconjunto de restrições do SPP é mantido no PRL, criam-se menos multiplicadores. Por consequência, o PDL tende a ser resolvido mais rapidamente. Com a decomposição Lagrangiana, dois subconjuntos de restrições que representam problemas fáceis são aproveitados: ambos são mantidos no PRL. Isto é adequado, pois nenhum desses é, de fato, um conjunto de restrições complicadoras.

Outro aspecto positivo a favor da decomposição Lagrangiana é relativo a divisão de um subproblema no *branch-and-cut*. Como comentado no Capítulo 3, este é um passo

estratégico. Se a decomposição Lagrangiana é a relaxação utilizada, surge, comumente, uma maneira direta de fazer a divisão. Quando uma solução ótima x_R obtida com tal relaxação não é também ótima para o subproblema, há, geralmente, pelo menos uma restrição (dualizada) $x_i = y_i$ violada por x_R, ou seja, com $x_{R_i} \neq y_{R_i}$. Seja i' o índice de uma igualdade dentre essas. É possível, então, gerar dois novos subproblemas: um onde $x_{i'} = y_{i'} = 0$ e outro para o qual $x_{i'} = y_{i'} = 1$. Guignard e Kim [33] ressaltam que esta divisão é mais natural que as usualmente empregadas com relaxação Lagrangiana. Isto, pois, na mesma, o foco é, na verdade, sobre uma variável, e não sobre uma restrição.

As observações feitas acima ratificam, portanto, o interesse em investigar como o *branch-and-cut* proposto anteriormente comporta-se com o uso das referidas relaxações. Com esperança, um melhor desempenho pode ser obtido.

5.4 Implementação

O objetivo, aqui, é apresentar as implementações de duas novas versões do *branch-and-cut* para o SPP proposto anteriormente. Estas implementações são realizadas com as relaxações Lagrangianas indicadas no início deste capítulo.

A primeira nova versão do *branch-and-cut* pode ser definida de forma idêntica à descrita na Seção 4.2 com apenas uma exceção: o PRL a ser resolvido em um subproblema é diferente. Tal PRL, no entanto, é um problema componente do PRL solucionado na segunda nova versão. Ainda, no último PRL há somente uma tarefa adicional a ser cumprida: otimizar o problema y. Assim, é suficiente, na apresentação a seguir, mostrar unicamente como a segunda nova versão é implementada. Abaixo, os tópicos omitidos devem ser considerados como postos na Subseção 4.2.1.

Identificação de problemas fáceis

O problema fácil componente do SPP que se quer encontrar aqui é formado por um subconjunto de restrições cujo lado esquerdo é uma MRRP (descrita na Seção 5.1). Há, na literatura, vários métodos para identificar uma MRRP em uma matriz. Nesta dissertação, usa-se o método proposto em [35]. Em tal referência, o método citado mostra-se bastante competitivo quando comparado a outras alternativas. No mesmo, a ideia principal é associar a uma MRRP um conjunto independente em um certo grafo sinalizado.

Como mencionado antes, a partir de uma matriz, extrair uma MRRP com tamanho máximo é um problema $\mathcal{NP}$-Difícil. Assim, métodos com este propósito são, em geral, heurísticos, no sentido que tal feito não é garantido. Ainda, tais métodos são comumente executados como subrotinas várias vezes em um algoritmo. Para tanto, os mesmos têm de ser muito eficientes.

Na implementação de um *branch-and-cut*, podem-se adotar duas estratégias. Uma estratégia é, em cada subproblema, extrair uma MRRP das restrições correntes (ainda não eliminadas). Outra conduta é fazer a identificação de uma MRRP somente para o problema inicial, antes do desencadeamento das divisões. Com a última conduta, em cada subproblema, o problema fácil é formado, então, pelas restrições associadas à MRRP identificada ainda presentes. Note que, com a primeira opção, o número de restrições destacadas nos subproblemas tende a ser maior. Desta forma, porém, o processo como um todo é mais custoso computacionalmente. Aqui, a detecção de uma MRRP é feita antes do desencadeamento das divisões.

Para implementar o método apresentado em [35], usa-se o programa de computador desenvolvido por Furushima [23]. O código-fonte do mesmo, gentilmente cedido pela autora, foi incorporado ao código-fonte do *branch-and-cut* desenvolvido neste trabalho.

Cálculo de limitantes (*bounding*)

Como na implementação apresentada no capítulo anterior, aplica-se, aqui, a cada iteração do *branch-and-cut*, um algoritmo NDRC ao subproblema considerado. A novidade é que, agora, a decomposição Lagrangiana é a relaxação empregada. Para construir tal relaxação, usam-se dois subconjuntos de igualdades do SPP que representam problemas fáceis. Cada um desses subconjuntos de igualdades contém do lado esquerdo uma estrutura de MRRP.

Mais especificamente, a decomposição Lagrangiana é derivada da seguinte maneira. Como descrito no tópico anterior, os subconjuntos supracitados são constituídos das restrições identificadas inicialmente, ou seja, antes do desencadeamento das divisões de subproblemas, e ainda presentes no subproblema em questão, isto é, ainda não eliminadas por consequência de fixação de variáveis. O vetor y e as restrições $x = y$, que igualam as cópias de variáveis, são introduzidos. Dualiza-se, então, as restrições $x = y$ e as restrições que não pertencem a nenhum dos dois subconjuntos destacados. A construção acima está representada pela sequência de modelos abaixo, onde A_1 e A_2 são MRRPs. Observe que esta é a situação ilustrada no fim da Seção 5.2.

$$
\begin{aligned}
z_{SPP} = \min \quad & cx \\
& A_1 x = \mathbf{1} \\
& A_2 x = \mathbf{1} \\
& A_3 x = \mathbf{1} \\
& x \in \mathbb{B}^n
\end{aligned}
\tag{5.4}
$$

$$
\begin{aligned}
z_{SPP} = \min \quad & cx \\
& A_1 x = \mathbf{1} \\
& A_2 y = \mathbf{1} \\
& A_3 x = \mathbf{1} \\
& x = y \\
& x \in \mathbb{B}^n \\
& y \in \mathbb{B}^n
\end{aligned}
\tag{5.5}
$$

$$
\begin{aligned}
z_{SPP}(\lambda, \mu) = \min \quad & cx + \lambda(y - x) + \mu(\mathbf{1} - A_3 x) \\
& A_1 x = \mathbf{1} \\
& A_2 y = \mathbf{1} \\
& x \in \mathbb{B}^n \\
& y \in \mathbb{B}^n
\end{aligned}
$$

$$
\begin{aligned}
z_{SPP}(\lambda, \mu) = \min \quad & (c - \lambda - \mu A_3)x + \min \quad \lambda y \qquad\qquad + \mu \mathbf{1} \\
& A_1 x = \mathbf{1} \qquad\qquad\qquad\quad A_2 y = \mathbf{1} \\
& x \in \mathbb{B}^n \qquad\qquad\qquad\quad y \in \mathbb{B}^n.
\end{aligned}
\tag{5.6}
$$

Para resolver o PRL, é preciso, então, realizar as seguintes três tarefas. Solucionar o problema x, otimizar o problema y e dar valor às variáveis *livres*: 1 em caso de custo Lagrangiano negativo e 0 em caso contrário. As variáveis livres são as variáveis que não estão presentes no problema x e não estão presentes no problema y. Tais variáveis são componentes do vetor x e estão associadas apenas às restrições $A_3 x = \mathbf{1}$.

Aqui, ao contrário do que ocorre na versão anterior do *branch-and-cut*, a inequação $\sum_j x_j \leq m$ não está contida no conjunto de restrições do PRL. O motivo é que, com a inclusão desta inequação, impediria-se que o PRL fosse resolvido da maneira eficiente descrita acima. Observando a situação a seguir, vê-se que a mesma não seria correta.

Considere a seguinte instância do PRL (5.6).

$$
m = 3, \quad A_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad (c - \lambda - \mu A_3) = \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \end{bmatrix}
$$

Tem-se que $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}$, sendo x_1, x_2 e x_3 variáveis presentes no problema x e x_4 e x_5 variáveis livres. Suponha, então, que a solução ótima encontrada para o problema x seja $x_1 = 1$, $x_2 = 1$, $x_3 = 0$. Assim, ao dar valor às variáveis x_4 e x_5, obtém-se, pela restrição $\sum_j x_j \leq m$, $x_4 = 1$ e $x_5 = 0$ ou $x_4 = 0$ e $x_5 = 1$. Suponha, também, que a solução ótima alcançada para o problema y tenha custo k. Consegue-se, com isso, uma solução de custo $(2 + k) = 1 + 2 - 1 + k$ para o PRL. Esta solução não é ótima para o PRL. Isto, pois a atribuição de valores $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = 1$ combinada

com a solução ótima para o problema y citada acima gera uma solução para o PRL de menor custo, a saber, $(1 + k) = 3 - 1 - 1 + k$.

Para solucionar o problema x, aplica-se a função `CPXNETprimopt` (codificada na linguagem de programação `C`), parte integrante do *software* comercial CPLEX. A mesma implementa eficientemente um algoritmo para problemas de fluxo em redes. Como mencionado na Seção 5.1, esse é um modo de tratar um IP cujo conjunto de restrições é dado por $Ax \leq b$ onde A é uma MRRP. Tal função também é empregada para otimizar o problema y.

Como no capítulo anterior, o PDL é resolvido através do MS. Os valores dos parâmetros do MS são definidos de forma idêntica.

Divisão de um subproblema (*branching*)

A divisão de um subproblema, aqui, pode ocorrer de três maneiras. Duas dessas maneiras são descritas na Subseção 4.2.1. A terceira é a divisão mencionada na Seção 5.3. A implementação da última é detalhada abaixo. Seja x_R^k a solução associada ao melhor limitante inferior obtido com o processamento do subproblema. Tome as igualdades dualizadas $x_i = y_i$ violadas por x_R^k, ou seja, com $x_{R_i}^k \neq y_{R_i}^k$. Se existem igualdades dualizadas $x_i = y_i$ violadas por x_R^k onde $x_{R_i}^k = 1$, selecione apenas as mesmas. Dentre as restrições consideradas, escolha uma cuja variável valendo 1 tenha custo mínimo. Sendo i' o índice dessa restrição, gere dois novos subproblemas: um associado a $x_{i'} = y_{i'} = 1$ e outro vinculado a $x_{i'} = y_{i'} = 0$.

Lembre-se que, nesta versão do *branch-and-cut*, as igualdades $x_i = y_i$ não são as únicas restrições dualizadas. Assim, é possível que a solução x_R^k satisfaça a todas essas igualdades e não seja ótima para o subproblema. Em tal situação, o *branching* detalhado acima não pode ser empregado.

Os tipos de divisão são, então, considerados na seguinte ordem de preferência. Quando o terceiro é factível, o mesmo é preferido. Quando não, a escolha entre os outros dois é feita como na Subseção 4.2.1.

Escolha de um subproblema

Nesta versão do *branch-and-cut*, adota-se, também, para a escolha de um subproblema, a estratégia profundidade primeiro (mencionada na Seção 3.3). Assim, sempre o subproblema mais recentemente criado é o próximo a ser processado. Quando existem dois, a seleção depende da divisão que os originou. Se tal divisão é um das empregadas no capítulo anterior, tudo ocorre como lá descrito. Se o *branching* em questão é o novo apresentado nesta seção, o subproblema associado a $x_j = y_j = 1$ é preferido. Como no outro *branching* baseado em uma variável, tal preferência reflete qual dos ramos determinados

na árvore de divisões é provavelmente o mais proveitoso.

Estruturas de dados para os subproblemas

A representação de um subproblema, aqui, é muito semelhante à descrita na Subseção 4.2.1. A mesma contém todas as estruturas mencionadas na referida subseção. Há, no entanto, distinções e novidades. As distinções são decorrentes dos elementos adicionais presentes no IP que modela o subproblema. Agora, a lista de variáveis é composta de x e y. Também, a lista de restrições inclui igualdades do SPP e do tipo $x_i = y_i$. As novidades são por conta do armazenamento dos problema x e problema y. Esse problemas são guardados em estruturas `CPXNET` (*structs* da linguagem de programação `C`), providas com o *software* comercial CPLEX.

Outros aspectos

É válido, ainda, comentar outros aspectos da implementação desta versão do *branch-and-cut*. Ao fixar uma variável x_j, pode-se fazer o mesmo para a variável y_j correspondente (pela relação $x_j = y_j$) e vice-versa. Quando a heurística primal e o algoritmo para encontrar cliques no grafo de interseção $G(A)$ (no processo de geração de cortes) são aplicados, o subproblema é tratado na forma *não decomposta*. Isto significa que o mesmo é considerado como no modelo (5.4) e não como no modelo (5.5). Note que, com o subproblema visto como no modelo (5.5), tem-se a estrutura especial do SPP perdida. O motivo é que no conjunto de restrições não há somente igualdades do tipo $a_i x = 1$. Ao subproblema na forma decomposta, então, não seria possível aplicar os algoritmos citados, sendo os mesmos algoritmos dependentes da estrutura especial do SPP.

5.5 Resultados

Pretende-se, aqui, avaliar o desempenho do *branch-and-cut* proposto no Capítulo 4 quando este se baseia em relaxações Lagrangianas alternativas. Tais relaxações são derivadas com a identificação de problemas fáceis componentes do SPP e com a manutenção dos mesmos no PRL.

Assim, são comparadas três versões do referido *branch-and-cut*. A primeira, denotada por BCLAG, é a versão testada anteriormente com uma modificação, apontada a seguir. A segunda, designada BCLAG1MRRP, utiliza a relaxação Lagrangiana construída com a manutenção no PRL de um subconjunto de igualdades do SPP que tenha do lado esquerdo uma estrutura de MRRP. A terceira, referenciada como BCLAGDEC, emprega a decomposição Lagrangiana, com o problema x e o problema y tendo os conjuntos de restrições constituídos de subconjuntos de igualdades do SPP com a citada propriedade.

Nesta seção, utiliza-se uma implementação modificada do *branch-and-cut* proposto no Capítulo 4. Como discutido no tópico *Cálculo de limitantes (bounding)* da seção anterior, não se inclui, nas implementações do BCLAG1MRRP e do BCLAGDEC, a inequação $\sum_j x_j \leq m$ no conjunto de restrições do PRL resolvido em um subproblema. A fim de tornar a comparação entre os três algoritmos justa, é preciso, portanto, que se faça o mesmo para a versão do *branch-and-cut* aplicada anteriormente. Esta é a modificação realizada.

O BCLAG1MRRP e o BCLAGDEC são, então, testados com o mesmo conjunto de instâncias usado no capítulo anterior. Também, o mesmo ambiente computacional é utilizado.

Nas Tabelas 5.1 e 5.2, apresentam-se os resultados obtidos. As primeiras cinco colunas das mesmas trazem as informações sobre as instâncias: nome, Nom, dimensões – número de restrições e variáveis, NRes e NVar, e densidade (em porcentagem), Den – e valor ótimo, Óti. Os dois próximos grupos de oito colunas relatam as execuções, respectivamente, nas Tabelas 5.1 e 5.2, do BCLAG e do BCLAG1MRRP e do BCLAG e do BCLAGDEC. Em cada grupo, são mostrados, primeiramente, os limitantes atingidos no processamento do primeiro subproblema – limitante superior, LSup, e limitante inferior, LInf, sob o título *Prim subp*. Em seguida, revelam-se os dados globais: o valor da melhor solução inteira encontrada, LSup, o *gap* de limitantes alcançado (em porcentagem), Gap, o número de subprolemas criados, NSubp, a quantidade de cortes ao fim da enumeração, NCor, a quantidade de cortes gerados durante a enumeração, NCorG, e o tempo total de computação, Tem. Quando um algoritmo é interrompido pelo limite de 2 horas de execução, são expostos os valores tidos no momento da parada.

É possível ver, por meio da citada tabela, que os desempenhos do BCLAG1MRRP e do BCLAGDEC são, em geral, inferiores ao do BCLAG. Os números de nós criados e os tempos despendidos são comumente maiores. Ainda, os limitantes conseguidos com o processamento do primeiro subproblema são piores em sua grande maioria. Tais resultados são inesperados. É importante, então, prover argumentos para justificá-los.

Um primeiro aspecto prejudicial ao desempenho das duas novas versões do *branch-and-cut* é a ineficiência da heurística primal em ambas. Tal ineficiência é revelada pelos muito pouco precisos limitantes superiores obtidos. Também, é sabido o impacto negativo que um limitante superior ruim provoca à qualidade do inferior atingido no MS. É através do mesmo que se determinam os tamanhos de passo, sendo esses tamanhos definidos segundo a fórmula (3.7), discutida na Subseção 3.2.1. Assim, a má atuação da heurística influencia, ainda, os limitantes inferiores produzidos. Com a falta de bons limitantes, são fatalmente necessários um maior número de subproblemas e um maior tempo para completar o processo de enumeração.

Instância					BCLAG								BCLAG1MRRP							
					Prim subp								Prim subp							
Nom	NRes	NVar	Den	Óti	LSup	LInf	LSup	Gap	NSubp	NCor	NCorG	Tem	LSup	LInf	LSup	Gap	NSubp	NCor	NCorG	Tem
nw03	59	43749	14,10	24492	24492	24492	24492	-	1	325	513	101,08	24492	24447	24492	-	7	210	225	122,28
nw04	36	87482	20,22	16862	16868	16309	16868	3,41	475	7200	5248	80,00	16968	16312	16896	4,83	131	632	5673	7200,00
nw06	50	6774	18,17	7810	7968	7793	7810	-	3	188	796	7,72	7968	7656	7810	-	7	144	317	12,22
nw11	39	8820	16,64	116256	116256	116256	116256	-	1	721	782	10,57	116256	116255	116256	-	3	211	327	10,59
nw13	51	16043	12,78	50146	50146	50146	50146	-	1	432	565	10,41	50146	50132	50146	-	5	206	224	26,21
nw17	61	118607	13,96	11115	11115	10897	11115	-	5	739	2877	961,90	11115	10890	11115	-	19	1222	5963	3748,16
nw18	124	10757	6,82	340160	340160	340160	340160	-	1	1618	2073	19,31	342742	339076	340160	-	505	2086	26003	1298,00
nw20	22	685	24,70	16812	16812	16812	16812	-	1	88	173	0,11	16965	16641	16812	-	5	54	66	1,06
nw21	25	577	24,89	7408	7408	7408	7408	-	1	33	53	0,09	7408	7380	7408	-	3	12	15	0,36
nw22	23	619	23,87	6984	6984	6984	6984	-	1	83	155	0,09	6984	6942	6984	-	3	30	46	0,38
nw23	19	711	24,80	12534	12534	12534	12534	-	1	132	397	0,66	12534	12321	12534	-	5	28	30	0,74
nw24	19	1366	33,20	6314	6314	6314	6314	-	1	25	38	0,28	6314	5843	6314	-	7	28	40	2,76
nw25	20	1217	30,16	5960	5960	5960	5960	-	1	42	75	0,22	5960	5852	5960	-	3	52	61	1,25
nw26	23	771	23,77	6796	6796	6796	6796	-	1	43	69	0,12	6796	6743	6796	-	3	15	38	0,49
nw27	22	1355	31,52	9933	9933	9933	9933	-	1	44	45	0,43	9933	9933	9933	-	1	10	20	0,20
nw28	18	1210	39,27	8298	8298	8298	8298	-	1	14	36	0,21	8298	8298	8298	-	1	27	59	0,14
nw29	18	2540	31,04	4274	4324	4217	4274	-	17	479	2115	10,27	4324	4240	4274	-	5	129	659	4,19
nw30	26	2653	29,63	3942	3942	3942	3942	-	1	148	195	0,90	3942	3942	3942	-	1	38	66	0,66
nw31	26	2662	28,86	8038	8038	8038	8038	-	1	47	144	0,87	8046	8022	8038	-	3	37	55	1,37
nw32	19	294	24,29	14877	14877	14877	14877	-	1	49	78	0,08	14877	14570	14877	-	5	23	36	0,49
nw33	23	3068	30,76	6678	6682	6536	6678	-	3	60	81	1,58	6678	6536	6678	-	3	31	56	1,79
nw34	20	899	28,06	10488	10488	10488	10488	-	1	33	35	0,19	10488	10488	10488	-	1	11	11	0,11
nw35	23	1709	26,70	7216	7216	7216	7216	-	1	63	89	0,48	7216	7216	7216	-	1	21	36	0,23
nw36	20	1783	36,90	7314	7322	7260	7314	-	17	367	2228	16,59	7328	7268	7314	-	13	66	507	7,23
nw37	19	770	25,82	10068	10068	10068	10068	-	1	28	30	0,13	10068	10068	10068	-	1	12	12	0,04
nw38	23	911	31,44	5558	5558	5355	5558	-	3	1080	1509	8,47	5558	5558	5558	-	1	33	58	0,10
nw39	25	677	26,55	10080	10080	10080	10080	-	1	36	56	0,14	10080	9869	10080	-	5	13	13	0,42
nw40	19	404	26,95	10809	10809	10809	10809	-	1	39	53	0,04	10809	10659	10809	-	7	8	9	0,47
nw41	17	197	22,10	11307	11307	11307	11307	-	1	10	14	0,02	11307	10973	11307	-	3	5	6	0,18
nw42	23	1079	26,32	7656	7656	7234	7656	-	3	217	1123	4,27	7656	7511	7656	-	3	23	41	0,86
nw43	18	1072	25,18	8904	8904	8904	8904	-	1	68	127	0,33	8904	8897	8904	-	3	18	31	0,53
us04	163	28016	6,52	17854	17854	17732	17854	-	3	483	504	44,71	17862	17732	17854	-	7	443	448	53,85
kl01	55	7479	13,67	1086	1087	1085	1086	-	17	1364	3178	32,89	1088	1084	1086	-	73	146	469	49,52
kl02	71	36699	8,16	219	220	216	219	1,39	1853	182	15381	7200,00	220	216	219	1,39	1859	85	378	7200,00
aa01	823	8904	1,00	56138	57528	55367	57528	3,90	251	11136	38033	7200,00	385514	54522	168901	209,79	411	10171	44892	7200,00
aa03	825	8627	0,99	49649	50747	49616	49649	-	19	1764	2970	250,21	317506	49211	49649	-	143	6085	17326	2868,41
aa04	426	7195	1,70	26374	29888	25952	26801	3,27	927	6027	78575	7200,00	216238	25881	26733	3,29	495	4704	37526	7200,00
aa05	801	8308	0,99	53839	53885	53740	53839	-	831	2418	27736	3362,65	333944	52600	53839	-	387	4461	38271	3359,04
aa06	646	7292	1,10	27040	27293	27001	27040	-	199	2592	11134	1280,74	27305	26884	27040	5,34	479	9209	40307	7200,00
cyc06	240	432	1,16	112	112	112	112	-	1	64	186	0,37	116	48	116	141,67	13891	220	243	7200,00
cyc07	672	1120	0,45	352	352	352	352	-	1	133	390	1,20	367	117	367	213,68	5189	773	1080	7200,00
cyc08	1792	2816	0,18	1024	1036	1023	1036	1,27	1983	256	939	7200,00	1108	328,07	1108	236,78	1351	2126	3904	7200,00
cyc09	4608	6912	0,07	2816	2830	2814	2830	0,57	269	512	2054	7200,00	3061	1077	3061	184,22	215	5370	13036	7200,00
cyc10	11520	16640	0,03	7424	7424	7421	7424	0,04	199	1024	4224	7200,00	7984	3327	7984	141,21	29	14601	40406	7200,00
cyc11	28160	39424	0,01	[19016,18944]	19007	18923	19007	0,44	65	2048	8274	7200,00	-	-	-	-	-	-	-	-

Tabela 5.1: Resultados para o BCLAG e para o BCLAG1MRRP.

Considera-se, contudo, que o aspecto citado acima não é o principal responsável pelo desempenhos ruins do BCLAG1MRRP e do BCLAGDEC. Aponta-se a ineficiência dos cortes como tal. É através dos cortes que se pode, no BCLAG, elevar substancialmente os limitantes inferiores. Os cortes, porém, não conseguem, aparentemente, provocar esse efeito no BCLAG1MRRP e no BCLAGDEC.

Para o BCLAGDEC, há, ainda, outra complicação. Para vários dos problemas testados, o número de novas igualdades criadas para igualar cópias de variáveis (do tipo $x_j = y_j$) é bastante grande. Com uma quantidade significativamente maior de restrições dualizadas, mesmo as adicionais sendo de estrutura especial, a convergência do MS mostra-se mais difícil.

Os últimos dois aspectos comentados acima não estão exatamente claros até agora. É, então, relatado um experimento cujo intuito é evidenciá-los. As três versões do *branch-and-cut* são executadas somente para o primeiro subproblema da seguinte maneira: o valor ótimo é passado como limitante superior inicial. Com tal configuração, suprimem-se diferenças de desempenho decorrentes de limitantes superiores distintos conseguidos durante a execução. Desta forma, é possível isolar e revelar o impacto dos cortes para as três alternativas. Também, pode-se constatar se a convergência do MS é mais difícil para o BCLAGDEC.

Na sequência, RC-BCLAG, RC-BCLAG1MRRP e RC-BCLAGDEC denotam, respectivamente, o NDRC aplicado ao primeiro subproblema no BCLAG, no BCLAG1MRRP e no BCLAGDEC.

Na Tabela 5.3, trazem-se os resultados do experimento. É possível deduzir as informações contidas na maioria das colunas da mesma baseando-se nas tabelas apresentadas acima. As únicas novidades são os valores ótimos arredondados das seguintes relaxações lineares: do primeiro subproblema antes e do primeiro subproblema após a aplicação do NDRC, ou seja, do primeiro subproblema adicionado de cortes. Os valores relativos a antes e a após a aplicação do NDRC são rotulados, respectivamente, por $\lceil$ÓtiL$\rceil$ e $\lceil$ÓtiLCor$\rceil$. Na coluna $\lceil$ÓtiLCor$\rceil$, estão determinados os máximos limitantes inferiores atingíveis em cada versão.

Para a análise a seguir, relembre a classificação das instâncias feita na Subseção 4.2.2.

Pelos dados apresentados, pode-se constatar que os cortes são, de fato, menos eficientes para o BCLAG1MRRP e para o BCLAGDEC. Para as 34 instâncias do grupo 1, enquanto o limitante inferior máximo obtenível pelo RC-BCLAG supera o ótimo linear do primeiro subproblema antes do seu processamento 31 vezes, o mesmo ocorre em apenas 17 e 15 casos para o RC-BCLAG1MRRP e para o RC-BCLAGDEC respectivamente. Comparando os máximos atingíveis pelos algoritmos entre si, vê-se que o do RC-BCLAG1MRRP é inferior ao do RC-BCLAG em 20 testes. Nos demais, há 11 empates e apenas 2 vitórias; para as instâncias nw29 e nw36. O RC-BCLAGDEC não supera o RC-BCLAG nenhuma vez. O

Instância					BCLAG								BCLAGDEC							
					Prim subp								Prim subp							
Nom	NRes	NVar	Den	Óti	LSup	LInf	LSup	Gap	NSubp	NCor	NCorG	Tem	LSup	LInf	LSup	Gap	NSubp	NCor	NCorG	Tem
nw03	59	43749	14,10	24492	24492	24492	24492	-	1	325	513	101,08	24492	24447	24492	-	3	260	328	113,45
nw04	36	87482	20,22	16862	16868	16309	16868	3,41	475	5248	136911	7200,00	16882	14612	16876	14,07	33	304	1106	7200,00
nw06	50	6774	18,17	7810	7968	7793	7810	-	3	188	796	7,72	7968	7656	7810	-	19	128	318	25,11
nw11	39	8820	16,64	116256	116256	116256	116256	-	1	721	782	10,57	117369	61223	116256	-	805	566	5521	7000,65
nw13	51	16043	12,78	50146	50146	50146	50146	-	1	432	565	10,41	50484	31009	50154	61,74	945	600	2042	7200,00
nw17	61	118607	13,96	11115	11115	10897	11115	-	5	739	2877	961,90	11133	10332	11133	7,75	3	1098	1643	7200,00
nw18	124	10757	6,82	340160	340160	340160	340160	-	1	1618	2073	19,31	342942	319577	340160	-	681	1662	95800	3502,58
nw20	22	685	24,70	16812	16812	16812	16812	-	1	88	173	0,11	16812	16641	16812	-	3	44	49	0,68
nw21	25	577	24,89	7408	7408	7408	7408	-	1	33	53	0,09	7408	7380	7408	-	3	8	12	0,50
nw22	23	619	23,87	6984	6984	6984	6984	-	1	83	155	0,09	6984	6942	6984	-	7	23	32	0,67
nw23	19	711	24,80	12534	12534	12534	12534	-	1	132	397	0,66	12534	11770	12534	-	7	63	124	3,74
nw24	19	1366	33,20	6314	6314	6314	6314	-	1	25	38	0,28	6338	5843	6314	-	7	33	39	3,31
nw25	20	1217	30,16	5960	5960	5960	5960	-	1	42	75	0,22	5960	5852	5960	-	3	62	77	0,78
nw26	23	771	23,77	6796	6796	6796	6796	-	1	43	69	0,12	6796	6743	6796	-	3	113	127	0,92
nw27	22	1355	31,52	9933	9933	9933	9933	-	1	44	45	0,43	9933	9933	9933	-	1	9	9	0,24
nw28	18	1210	39,27	8298	8298	8298	8298	-	1	14	36	0,21	8298	8298	8298	-	1	26	56	0,21
nw29	18	2540	31,04	4274	4324	4217	4274	-	17	479	2115	10,27	4274	3193	4274	-	11	148	875	24,35
nw30	26	2653	29,63	3942	3942	3942	3942	-	1	148	195	0,90	3942	3942	3942	-	1	37	81	0,71
nw31	26	2662	28,86	8038	8038	8038	8038	-	1	47	144	0,87	8038	8004	8038	-	7	32	51	2,76
nw32	19	294	24,29	14877	14877	14877	14877	-	1	49	78	0,08	14877	14570	14877	-	9	19	83	0,82
nw33	23	3068	30,76	6678	6682	6536	6678	-	3	60	81	1,58	6678	6536	6678	-	3	87	104	2,83
nw34	20	899	28,06	10488	10488	10488	10488	-	1	33	35	0,19	10488	10488	10488	-	1	70	94	0,29
nw35	23	1709	26,70	7216	7216	7216	7216	-	1	63	89	0,48	7216	7216	7216	-	1	28	45	0,27
nw36	20	1783	36,90	7314	7322	7260	7314	-	17	367	2228	16,59	7378	6085	7314	-	11	306	887	17,64
nw37	19	770	25,82	10068	10068	10068	10068	-	1	28	30	0,13	10068	10068	10068	-	1	22	25	0,13
nw38	23	911	31,44	5558	5558	5355	5558	-	3	1080	1509	8,47	5558	5558	5558	-	1	39	97	0,18
nw39	25	677	26,55	10080	10080	10080	10080	-	1	36	56	0,14	10080	9869	10080	-	5	28	29	0,72
nw40	19	404	26,95	10809	10809	10809	10809	-	1	39	53	0,04	10809	10659	10809	-	3	14	14	0,48
nw41	17	197	22,10	11307	11307	11307	11307	-	1	10	14	0,02	11307	10973	11307	-	3	1	1	0,25
nw42	23	1079	26,32	7656	7656	7234	7656	-	3	217	1123	4,27	7656	7511	7656	-	3	88	98	1,45
nw43	18	1072	25,18	8904	8904	8904	8904	-	1	68	127	0,33	8904	8904	8904	-	1	23	65	0,22
us04	163	28016	6,52	17854	17854	17732	17854	-	3	483	504	44,71	17854	17732	17854	-	37	366	436	191,56
kl01	55	7479	13,67	1086	1087	1085	1086	-	17	1364	3178	32,89	1123	1066	1086	-	59	795	2034	207,34
kl02	71	36699	8,16	219	220	216	219	1,39	1853	182	15381	7200,00	220	216	219	1,39	4259	92	161	7200,00
aa01	823	8904	1,00	56138	57528	55367	57528	3,90	251	11136	38033	7200,00	385514	50137	61692	23,05	275	9897	47239	7200,00
aa03	825	8627	0,99	49649	50747	49616	49649	-	19	1764	2970	250,21	317506	44470	52463	17,97	489	5324	36805	7200,00
aa04	426	7195	1,70	26374	29888	25952	26801	3,27	927	6027	78575	7200,00	216238	25355	27225	7,38	425	5720	37207	7200,00
aa05	801	8308	0,99	53839	53885	53740	53839	-	831	2418	27736	3362,65	333944	50666	57825	14,13	457	4930	28295	7200,00
aa06	646	7292	1,10	27040	27293	27001	27040	-	199	2592	11134	1280,74	28270	26573	27818	4,69	1015	3564	40553	7200,00
cyc06	240	432	1,16	112	112	112	112	-	1	64	186	0,37	120	48	120	150,00	247	220	261	7200,00
cyc07	672	1120	0,45	352	352	352	352	-	1	133	390	1,20	372	117	372	217,95	7045	773	1147	7200,00
cyc08	1792	2816	0,18	1024	1036	1023	1036	1,27	1983	256	939	7200,00	1108	327	1096	235,17	4555	2123	3930	7200,00
cyc09	4608	6912	0,07	2816	2830	2814	2830	0,57	269	512	2054	7200,00	3068	1075	3068	185,40	235	5382	13413	7200,00
cyc10	11520	16640	0,03	7424	7424	7421	7424	0,04	199	1024	4224	7200,00	7984	3327	7984	139,98	17	14769	40951	7200,00
cyc11	28160	39424	0,01	[19016,18944]	19007	18923	19007	0,44	65	2048	8274	7200,00	-	-	-	-	-	-	-	-

Tabela 5.2: Resultados para o BCLAG e para o BCLAGDEC.

Instância		RC-BCLAG				RC-BCLAG1MRRP				RC-BCLAGDEC			
Nom	⌈ÓtiL⌉	LSup	LInf	NCor	⌈ÓtiLCor⌉	LSup	LInf	NCor	⌈ÓtiLCor⌉	LSup	LInf	NCor	⌈ÓtiLCor⌉
nw03	24447	24492	24492	306	24492	24492	24447	86	24447	24492	24447	158	24447
nw04	16311	16862	10645	4581	16315	16862	16190	1580	16312	16862	14698	1588	16311
nw06	7640	7810	7718	533	7792	7810	7656	51	7656	7810	7656	57	7656
nw11	116255	116256	116256	562	116256	116256	116256	215	116256	116256	60535	1399	116255
nw13	50132	50146	50146	354	50146	50146	50132	131	50132	50146	31157	380	50132
nw17	10876	11115	10912	*	10930	11115	10895	*	10906	-	-	-	-
nw18	338865	340160	339178	*	339852	340160	339058	*	339134	340160	317375	*	338865
nw20	16626	16812	16812	137	16812	16812	16641	27	16641	16812	16641	20	16641
nw21	7380	7408	7408	10	7408	7408	7380	11	7380	7408	7380	10	7380
nw22	6942	6984	6984	58	6984	6984	6942	13	6942	6984	6942	18	6942
nw23	12317	12534	12534	53	12534	12534	12321	20	12321	12534	12321	29	12321
nw24	5843	6314	6314	12	6314	6314	5843	13	5843	6314	5843	23	5843
nw25	5852	5960	5960	19	5960	5960	5852	18	5852	5960	5852	21	5852
nw26	6743	6796	6796	23	6796	6796	6743	16	6743	6796	6743	14	6743
nw27	9878	9933	9933	25	9933	9933	9933	4	9933	9933	9933	8	9933
nw28	8169	8298	8298	49	8298	8298	8298	13	8298	8298	8298	13	8298
nw29	4186	4274	4165	504	4196	4274	4246	332	4257	4274	3809	106	4190
nw30	3727	3942	3942	58	3942	3942	3942	34	3942	3942	3942	53	3942
nw31	7980	8038	8038	92	8038	8038	8038	14	8038	8038	8000	23	8000
nw32	14570	14877	14877	44	14877	14877	14570	23	14570	14877	14570	79	14570
nw33	6484	6678	6536	58	6536	6678	6536	17	6536	6678	6536	25	6536
nw34	10454	10488	10488	10	10488	10488	10488	5	10488	10488	10488	45	10488
nw35	7206	7216	7216	43	7216	7216	7216	8	7216	7216	7216	10	7216
nw36	7260	7314	7215	389	7260	7314	7268	145	7268	7314	6071	650	7260
nw37	9962	10068	10068	16	10068	10068	10068	5	10068	10068	10068	13	10068
nw38	5552	5558	5558	55	5558	5558	5552	20	5552	5558	5558	28	5558
nw39	9869	10080	10080	13	10080	10080	9869	5	9869	10080	9869	7	9869
nw40	10659	10809	10809	21	10809	10809	10659	6	10659	10809	10659	4	10659
nw41	10973	11307	11307	10	11307	11307	10973	0	10973	11307	10973	0	10973
nw42	7485	7656	7656	56	7656	7656	7511	13	7511	7656	7511	82	7511
nw43	8897	8904	8904	54	8904	8904	8897	20	8897	8904	8897	35	8897
kl01	1084	1086	1085	284	1085	1086	1084	42	1084	1086	1070	51	1084
kl02	216	219	216	39	216	219	216	0	216	219	216	262	216
us04	17732	17854	17732	62	17732	17854	17732	53	17732	17854	17732	0	17732
aa01	55536	56138	55292	1661	55596	56138	55435	880	55562	56138	54111	2330	55565
aa03	49617	49649	49616	855	49638	49649	49630	485	49638	49649	49401	941	49637
aa04	25878	26374	25947	869	25959	26374	25884	438	25890	26374	25767	775	25890
aa05	53736	53839	53740	1108	53776	53839	53741	774	53750	53839	53677	822	53747
aa06	26978	27040	26998	379	27004	27040	26998	194	27002	27040	26995	505	27002
cyc06	48	112	112	64	112	112	48	220	48	112	48	220	48
cyc07	112	352	352	133	352	352	117	774	117	352	117	772	117
cyc08	256	1024	1022	269	1024	1024	328	2139	330	1024	328	2136	330
cyc09	576	2816	2813	523	2815	2816	1077	5448	1080	2816	1074	5496	1080
cyc10	1280	7424	7421	1024	7424	7424	3327	15300	3331	7424	3314	15358	3323

Tabela 5.3: Resultados para o RC-BCLAG, para o RC-BCLAG1MRRP e para o RC-BCLAGDEC.

máximo para o RC-BCLAGDEC é pior em 22 e igual em 11 casos.

Em relação ao grupo 2, o RC-BCLAG1MRRP e o RC-BCLAGDEC se saem um pouco melhor. Agora, os limitantes inferiores alcançáveis pelos três algoritmos são maiores que o ótimo linear original em todos os 5 testes. No entanto, os valores relativos o RC-BCLAG1MRRP e o RC-BCLAGDEC ainda não são inferiores aos do RC-BCLAG em somente 1 caso. Para instância aa03, há um empate entre o RC-BCLAG e o RC-BCLAG1MRRP.

Para o terceiro grupo de testes, o cenário é o pior. Em 3 dos 4 casos, o máximo limitante obtenível pelo RC-BCLAG1MRRP e pelo RC-BCLAGDEC supera o ótimo linear ao qual são comparados. Entretanto, para todas as instâncias, os valores dos mesmos são piores que os do RC-BCLAG. Ainda, a diferença entre esses é bastante grande.

Outra constatação possível é que há, de fato, para o BCLAGDEC, mais dificuldades na convergência do MS. Para as instâncias do grupo 1, o RC-BCLAGDEC consegue atingir o máximo limitante inferior possível em praticamente igual número de casos que o RC-BCLAG e o RC-BCLAG1MRRP. Quando isso não ocorre, porém, os valores resultantes são mais distintos. Observe, por exemplo, os testes nw29, nw36 e kl01. Em média, tal diferença é de 4,63%, 0,09% e 18,4% para o RC-BCLAG, para o RC-BCLAG1MRRP e para o RC-BCLAGDEC respectivamente.

No tocante ao grupo 2, de forma coerente ao exposto na análise anterior, o RC-BCLAGDEC obtém resultados um pouco melhores. Agora, para as três alternativas, o limitante final não alcança o máximo possível em nenhum dos casos. A diferença de valores para o RC-BCLAGDEC é, contudo, de forma absoluta e em comparação às atingidas com os outros dois algoritmos, menor. Para o RC-BCLAG, para o RC-BCLAG1MRRP e para o RC-BCLAGDEC, nessa ordem, os limitantes inferiores estão a 0,14%, 0,06% e 0,74% do máximo.

Para o último grupo, o cenário é semelhante, no sentido em que os desempenhos dos três algoritmos são mais parecidos. As diferenças correspondentes são de 0,09% para o RC-BCLAG, 0,27% para o RC-BCLAG1MRRP e 0,41% para o RC-BCLAGDEC. Os limitantes produzidos pelos três algoritmos conseguem, agora, atingir o valor máximo em 2 dos 4 casos.

5.6 Conclusão

Neste capítulo, foi avaliado o impacto do uso de relaxações Lagrangianas alternativas no desempenho do *branch-and-cut* proposto para o SPP no Capítulo 4. Tais relaxações são baseadas na identificação de problemas fáceis componentes do SPP. Foram, assim, geradas duas novas versões do algoritmo descrito anteriormente. Em uma dessas, empregou-se a técnica de decomposição Lagrangiana.

As relaxações alternativas não tornaram o *branch-and-cut* mais eficiente. Ao contrário, em geral, piores limitantes foram produzidos, mais subproblemas foram criados e mais tempo foi gasto com as novas versões implementadas.

Aponta-se a ineficiência dos cortes como o principal motivo para o desempenho citado acima. É através dos mesmos que, no algoritmo testado no capítulo passado, conseguem-se elevar substancialmente os limitantes inferiores obtidos. Constata-se, também, que a heurística primal é menos eficaz para as novas versões do *branch-and-cut* e que, quando a decomposição Langrangeana é empregada, devido ao grande número de restrições dualizadas, há mais dificuldades na convergência do MS.

Capítulo 6

Cooperação entre algoritmos *relax-and-cut* para o SPP

Nos últimos anos, têm-se empregado vários algoritmos distribuídos e cooperativos para atacar problemas difíceis. E tais algoritmos têm obtido ótimos resultados. Em [17], Cung *et al.* reúnem uma numerosa lista de aplicações de sucesso, além de discutir aspectos fundamentais dessa estratégia. Em [56], é ilustrada a eficiência de uma meta-heurística GRASP paralela.

Muito do sucesso citado acima pode ser creditado à seguinte constatação. Há várias heurísticas distribuídas que, quando comparadas a correspondentes (aplicáveis ao mesmo problema) sequenciais, além do menor tempo de execução, possuem tipicamente duas significativas vantagens. Primeiramente, é comum que tais métodos mostrem-se menos sensíveis a parâmetros. Em segundo lugar, a eficiência de uma heurística sequencial é, em geral, restrita a um pequeno conjunto de classes de instâncias. Para uma respectiva paralela, esse conjunto é frequentemente maior. No entanto, é válido salientar que desenvolver um método distribuído eficiente requer muito esforço e conhecimento.

Também, em geral, é possível, para um dado algoritmo, fazer uma implementação distribuída do mesmo. Comparado ao original, o novo método tem comumente um melhor desempenho quando

1. aquele é, em grande parte, formado por passos realizáveis separadamente e,
2. no ambiente onde este é executado, o esforço computacional como um todo é dividido equilibradamente entre as unidades processadoras.

Para ser eficiente com o problema em questão, Hoitomt *et al.* [41] ressaltam que o elemento mais importante para a abordagem paralela ainda é, contudo, a qualidade do algoritmo primitivo.

No contexto de relaxação Lagrangiana, algoritmos distribuídos costumam ser especialmente proveitosos. Isto, pois é comum que, com tal técnica, a relaxação gerada possa ser decomposta em problemas menores e independentes. Um exemplo é o método desenvolvido em [41] para o *problema de escalonamento de tarefas*. Esse método é uma paralelização do algoritmo proposto em [40]. Tal algoritmo é capaz de encontrar soluções muito boas, muito próximas de serem ótimas, em pouco tempo, comparado ao tipicamente levado para conseguir escalonamentos ideais. Também, o PRL resolvido no mesmo se adequa à situação descrita acima. Em [41], um estudo realizado indica que a referida abordagem sequencial pode ser executada, em grande parte, de forma distribuída. Com a nova versão do algoritmo, os autores obtêm uma significativa redução no tempo de execução.

Uma das ideias empregadas no Capítulo 5 é identificar dois subconjuntos de restrições do SPP que representem problemas fáceis. Essa identificação permite facilmente a construção de um interessante algoritmo distribuído. Isto se dá da seguinte maneira. Lembre que tais subconjuntos somente possuem a propriedade citada se considerados isoladamente. No referido capítulo, o mecanismo usado para manter o isolamento é a decomposição Lagrangiana. Uma alternativa é resolver separadamente relaxações Lagrangianas (puras) diferentes do SPP, cada uma associada a um problema fácil, em um ambiente paralelo. Obviamente, as versões processadas nesse contexto devem cooperar entre si, com troca de informações.

Neste capítulo, é feita uma avaliação bastante rudimentar sobre a eficiência da abordagem anunciada acima. Para tanto, é implementada uma cooperação entre duas unidades processadoras presentes em um possível ambiente distribuído. Com essa implementação, não é possível discutir a fundo o desempenho do método. Pode-se, contudo, conhecer efeitos de uma colaboração entre algoritmos *relax-and-cut*.

Na seção seguinte, o referido método é descrito mais profundamente. Na Seção 6.2, apresentam-se detalhes da implementação feita aqui. As Seções 6.3 e 6.4 trazem os resultados e conclusões obtidos.

6.1 Um algoritmo distribuído para o SPP

No método distribuído mencionado na seção anterior, o SPP é processado através de um NDRC. Nesse contexto, a relaxação usada é a seguinte. Tome uma das duas versões do problema. Essa está vinculada a um subconjunto de restrições. Mantenha as mesmas no PRL e dualize todas as outras.

No ambiente distribuído, a cooperação entre as unidades processadoras A e B se dá de duas formas. A primeira dessas se baseia na observação abaixo. A mesma é descrita na sequência. Considere o problema dual da relaxação linear do SPP. Para uma relaxação

Lagrangiana do SPP, os valores ótimos das variáveis desse problema correspondentes às restrições dualizadas são também ótimos para os respectivos multiplicadores.

A unidade A pode cooperar com a B munindo a mesma com valores iniciais para os multiplicadores. Dada uma restrição dualizada em B, há dois casos possíveis. O primeiro ocorre quando essa pertence ao conjunto de igualdades mantidas em A. Em tal situação, o valor ótimo da respectiva variável dual pode ser usado. O mesmo é tipicamente obtido com a resolução do PRL em A. Isto acontece, por exemplo, se o último é solucionado através da relaxação linear. O segundo caso ocorre quando a restrição em questão também está dualizada em A. Nessa cirscunstância, o valor obtido com a resolução do PDL em A pode inicializar o relativo multiplicador em B.

Outra forma de colaboração é o fornecimento de cortes. Uma ressalva, contudo, cabe. Suponha que os dois conjuntos C_1 e C_2 de restrições destacadas do SPP englobem todas as igualdades do mesmo. Sejam A e B associadas a C_1 e a C_2 respectivamente. Um corte adicionado no NDRC em A elimina soluções relaxadas que não satisfaçam às restrições em C_2. Esse, porém, não é útil em B, já que qualquer solução relaxada em tal unidade respeita às igualdades em C_2. Isto, pois as mesmas estão mantidas no PRL. A observação também vale no sentido de B para A. No entanto, é muito provável que a suposição acima não seja verdadeira. Assim, a transferência de cortes tem importância, pois passa informações sobre as restrições não presentes em nenhum dos subconjuntos.

6.2 Implementação

A cooperação entre as unidades processadoras A e B no ambiente distribuído é implementada como abaixo. Um NDRC é executado em A. Esse é como descrito para a primeira nova versão do *branch-and-cut* avaliada no Capítulo 5. Ao fim do método, as informações obtidas são repassadas à outra unidade. Em seguida, ocorrem os mesmos processamento e transferência relativos a B. São realizadas iterações como essas até que um critério de parada seja atingido.

Uma unidade processadora colabora com a outra, então, com duas informações: valores de multiplicadores e cortes. A primeira dessas é tratada da seguinte maneira. Suponha, sem perda de generalidade, o sentido de A para B. Considere os valores de multiplicadores que determinaram o melhor limitante obtido em A. Os valores das variáveis duais correspondentes às restrições mantidas no PRL são, então, empregados na inicialização dos respectivos multiplicadores em B. A segunda informação, cortes, é utilizada de forma direta.

6.3 Resultados

Pretende-se avaliar, aqui, o desempenho de um método distribuído e cooperativo que faz uso, em cada uma das duas unidades processadoras do sistema, da detecção de uma MRRP. Não se espera que os resultados obtidos com o mesmo sejam melhores que os produzidos com o algoritmo *relax-and-cut* referência (desenvolvido em [15]). Isto, pois os aspectos prejudiciais às abordagens propostas no capítulo anterior, a saber, a pouca eficiência dos cortes e da heurística primal, ainda são, naturalmente, influentes no presente contexto. É válido, contudo, investigar se esta maneira de empregar duas MRRPs é proveitosa, ao contrário do que se mostrou a relacionada à decomposição Lagrangiana. Para realizar tal avaliação, compara-se a nova versão do algoritmo *relax-and-cut* com outras duas não distribuídas: cada uma utilizando somente um e distinto problema fácil identificado.

A execução do método distribuído, cuja implementação está descrita na seção anterior, é simulada de forma sequencial. São efetuados passos que constituem ciclos, como definidos na referida seção, os últimos ocorrendo em série. Ao final do processo, o maior tempo de otimização contabilizado dentre as unidades processadoras é considerado o total. Os tempos para troca de informações são ínfimos e, portanto, negligenciados.

Para determinar um número máximo de ciclos adequado, foi feito o seguinte experimento preliminar. Os algoritmos confrontados foram executados para um subconjunto representativo de instâncias com esse máximo sendo bastante alto: 20 ciclos. Foi constatado que, a partir do quinto desses, ou não há melhoria nos limitantes ou a mesma é marginal em relação às já conseguidas. São permitidos, então, no máximo, 4 ciclos. Ainda, para limitar o tempo das execuções a duas horas, tal número foi reduzido a 2 para os testes nw17, cyc09 e cyc10 e a instância cyc11 foi descartada.

Para realizar uma comparação justa, é preciso permitir, também, que o MS possa ser executado mais de uma vez nas alternativas não distribuídas. Uma aplicação do mesmo onde se conhecem bons limitantes e valores para os multiplicadores, obtidos anteriormente, pode culminar em melhores produtos. Assim, o conceito de ciclo é emprestado às referidas abordagens e os máximos citados acima são utilizados.

Na Tabela 6.1, mostram-se os resultados. Nas quatro primeiras colunas da mesma, apresentam-se o nome, Nom, e as dimensões das instâncias – número de restrições, NRes, número de variáveis, NVar, e densidade (em porcentagem), Den – após o pré-processamento. Nas demais, são descritas as execuções dos três algoritmos: o *relax-and-cut* que utiliza apenas a primeira MRRP identificada, RCSEQ-1, o método que emprega somente a segunda MRRP detectada, RCSEQ-2, e a abordagem distribuída, RCDIST.

Nos dois primeiros dos respectivos três grupos de colunas, revelam-se o número de ci-

Instância				RCSEQ-1							RCSEQ-2							RCDIST									
																					Problema 1				Problema 2			
Nom	NRes	NVar	Den	NCic	LSup	LInf	NCor	NCorG	VarF	Tem	NCic	LSup	LInf	NCor	NCorG	VarF	Tem	NCic	LSup	LInf	NCor	NCorG	VarF	NCor	NCorG	VarF	Tem
nw03	59	43749	14,10	1	24492	24447	121	176	99,79	71,16	1	24492	24492	113	431	99,98	58,82	1	24492	24492	121	176	99,79	99,98	431	112,26	70,00
nw04	36	87482	20,22	4	16882	16310	644	817	*	37m	4	16922	16311	571	733	*	32m	4	16868	16311	860	1083	*	*	1533	89,40	1h37m
nw06	50	6774	18,17	4	8626	7656	156	182	0,00	28,30	4	8088	7686	169	209	0,00	45,32	4	7984	7677	168	182	0,00	0,00	475	13,33	52,31
nw11	39	8820	16,64	4	116256	116255	8	273	99,89	10,23	4	116256	116255	3	215	99,92	10,55	4	116256	116255	8	273	99,89	99,92	215	135,87	10,30
nw13	51	16043	12,78	4	50146	50132	82	157	99,50	21,78	4	50146	50132	182	294	99,47	23,27	3	50146	50146	19	157	99,82	99,50	294	146,62	21,42
nw17	61	118607	13,96	2	11115	10900	1136	1641	99,70	872,41	2	11550	10900	1047	1468	0,00	1h32m	2	11115	10897	1287	1840	99,73	99,66	2102	151,35	39m
nw18	124	10757	6,82	2	342742	339049	1245	1410	70,63	51,96	2	340160	338949	977	1317	91,80	38,77	2	340160	339030	1649	1938	86,48	92,46	1877	119,61	53,36
nw20	22	685	24,70	4	16965	16641	39	58	112,90	2,33	1	16812	16812	110	234	96,64	0,28	1	16812	16812	40	49	91,70	96,64	234	117,67	0,62
nw21	25	577	24,89	4	7408	7380	4	6	133,10	1,28	1	7408	7408	0	6	100,00	0,04	1	7408	7408	4	6	97,65	100,00	6	135,45	0,34
nw22	23	619	23,87	4	6984	6942	21	24	113,75	1,36	1	6984	6984	0	160	100,00	0,07	1	6984	6984	21	24	97,18	100,00	160	116,57	0,34
nw23	19	711	24,80	4	12534	12321	9	13	91,33	1,62	4	12534	12407	1508	1647	88,79	3,12	4	12534	12407	9	13	92,39	88,79	1538	139,11	7,54
nw24	19	1366	33,20	4	6314	5843	24	31	0,00	6,91	4	6314	6175	55	125	96,33	2,97	4	6314	6175	21	26	0,00	96,33	147	143,84	8,88
nw25	20	1217	30,16	4	5960	5852	60	78	142,77	2,20	1	5960	5960	0	40	100,00	0,06	1	5960	5960	60	78	98,46	100,00	40	144,19	0,64
nw26	23	771	23,77	4	6796	6743	7	18	139,85	1,63	1	6796	6796	1	21	98,71	0,04	1	6796	6796	9	18	96,68	98,71	21	140,96	0,41
nw27	22	1355	31,52	4	9933	9878	7	7	145,90	15,09	1	9933	9933	0	25	100,00	0,13	1	9933	9933	7	7	99,24	100,00	25	146,33	0,50
nw28	18	1210	39,27	1	8298	8298	0	17	100,00	0,08	4	8298	8169	6	6	145,70	2,36	0	8298	8298	0	17	100,00	0,00	0	46,67	0,07
nw29	18	2540	31,04	4	4274	4230	1390	1678	97,05	4,59	4	4274	4189	33	48	98,03	3,63	4	4274	4228	2298	3082	97,49	98,77	70	123,65	9,08
nw30	26	2653	29,63	1	3942	3942	0	111	100,00	0,37	4	3942	3756	46	72	40,82	14,83	0	3942	3942	0	111	100,00	0,00	0	40,82	0,39
nw31	26	2662	28,86	4	8038	8000	30	33	144,82	4,32	1	8038	8038	0	63	100,00	0,34	1	8038	8038	30	33	98,79	100,00	63	146,02	1,39
nw32	19	294	24,29	4	14877	14570	18	19	108,76	1,43	1	14877	14877	120	383	98,41	0,17	1	14877	14877	18	19	91,63	98,41	383	115,54	0,35
nw33	23	3068	30,76	4	6678	6536	58	82	99,09	5,41	4	6682	6484	26	26	0,00	14,63	4	6678	6536	58	82	99,46	99,30	30	126,34	8,63
nw34	20	899	28,06	1	10488	10488	0	12	100,00	0,07	1	10488	10488	0	9	119,87	0,07	0	10488	10488	0	12	100,00	0,00	0	19,87	0,07
nw35	23	1709	26,70	1	7216	7216	0	24	100,00	0,08	1	7216	7216	0	28	121,81	0,12	0	7216	7216	0	24	100,00	0,00	0	21,81	0,08
nw36	20	1783	36,90	4	7314	7268	157	188	93,39	3,39	4	7314	7260	58	76	96,24	3,41	4	7314	7268	608	788	95,10	96,24	76	122,87	5,53
nw37	19	770	25,82	1	10068	10068	0	8	100,00	0,02	1	10068	10068	8	9	118,94	0,02	0	10068	10068	0	8	100,00	0,00	0	20,50	0,02
nw38	23	911	31,44	1	5558	5558	0	22	100,00	0,10	1	5558	5558	0	129	100,00	0,07	0	5558	5558	0	22	100,00	0,00	0	0,00	0,12
nw39	25	677	26,55	4	10080	9869	5	10	98,77	1,01	4	10080	9869	7	7	98,59	1,13	4	10080	9869	5	10	98,77	98,59	7	117,99	1,08
nw40	19	404	26,95	4	10809	10659	6	7	115,77	1,28	1	10809	10809	0	18	100,00	0,01	1	10809	10809	6	7	95,54	100,00	18	120,24	0,33
nw41	17	197	22,10	4	11307	10973	2	2	11,30	1,17	1	11307	11307	0	6	100,00	0,02	1	11307	11307	2	2	0,00	100,00	6	111,30	0,24
nw42	23	1079	26,32	4	7656	7511	19	25	117,65	2,85	1	7656	7656	0	59	100,00	0,06	1	7656	7656	19	25	97,09	100,00	59	120,56	0,76
nw43	18	1072	25,18	1	8904	8904	0	16	100,00	0,08	4	8904	8897	26	53	108,25	1,46	0	8904	8904	0	16	100,00	0,00	0	9,16	0,07
kl01	55	7479	13,67	4	1088	1084	158	284	94,12	13,67	4	1087	1084	182	259	95,87	12,00	4	1087	1084	133	306	95,94	95,90	344	121,45	15,80
kl02	71	36699	8,16	4	220	216	1	1	75,91	97,27	4	219	216	114	122	80,91	82,00	4	219	216	1	1	83,50	82,53	136	201,83	105,95
us04	163	28016	6,52	4	17854	17732	118	231	98,96	42,71	4	17854	17732	82	174	98,83	40,80	4	17854	17732	107	232	99,04	99,01	175	313,80	54,10
aa01	823	8904	1,00	4	385514	54959	8544	10959	0,00	202,48	4	385514	54600	9999	12732	0,00	275,00	4	385514	55195	11787	15629	0,00	0,00	20150	15,83	414,50
aa03	825	8627	0,99	4	317506	49348	5295	6975	0,00	142,35	4	50435	49238	5625	7602	19,60	183,20	4	50399	49565	4720	5955	27,00	23,97	8791	46,39	322,22
aa04	426	7195	1,70	4	216238	25908	6619	7567	0,00	95,40	4	216238	25913	8164	10131	0,00	133,94	4	216238	25913	8082	9560	0,00	0,00	14556	16,05	249,88
aa05	801	8308	0,99	4	54181	53493	5559	7068	37,17	203,12	4	333944	53125	6823	8942	0,00	204,41	4	53935	53590	5955	7409	60,44	58,18	9314	87,43	316,59
aa06	646	7292	1,10	4	27381	26982	3038	3638	24,78	104,38	4	27293	26973	3282	4142	35,53	130,21	4	27293	26974	4649	5922	33,40	31,40	6938	49,91	261,28
cyc06	240	432	1,16	4	116	48	220	243	0,00	3,18	4	116	48	216	218	0,00	3,17	4	116	48	220	243	0,00	0,00	218	0,00	4,27
cyc07	672	1120	0,45	4	367	117	773	1080	0,00	10,70	4	377	122	769	1133	0,00	11,09	4	367	122	773	1080	0,00	0,00	1133	0,00	15,36
cyc08	1792	2816	0,18	4	1090	329	2126	3901	0,00	52,62	4	1096	341	2106	4058	0,00	53,45	4	1090	341	2126	3900	0,00	2103,00	4062	0,00	83,66
cyc09	4608	6912	0,07	2	3061	1077	5443	12807	*	329,45	2	3075	1100	5426	13061	*	330,48	2	3061	1101	5419	12877	*	*	13096	0,00	250,67
cyc10	11520	16640	0,03	2	7984	3327	15159	38585	*	29m	2	7896	3339	15207	38878	*	34m	2	7896	3339	15201	38666	*	*	38995	0,00	53m

Tabela 6.1: Resultados para o RCSEQ-1, para o RCSEQ-2 e para o RCDIST.

clos realizados, NCic, os limitantes superior e inferior atingidos, LSup e LInf, o número de cortes ao fim do processo, NCor, o número de cortes gerados, NCorG, a porcentagem de variáveis fixadas, VarF, e o tempo de otimização, Tem (em segundos, salvo quando outra unidade está explicitamente indicada). No último grupo, as informações sobre cortes e variáveis fixadas são apresentadas para cada um dos dois problemas processados no sistema.

Ao observar a referida tabela, é possível ver que, salvo em 5 casos, o RCDIST atinge limitantes superiores e inferiores melhores ou iguais aos obtidos pelo RCSEQ-1 e pelo RCSEQ-2. É válido ressaltar que tal feito não é garantido. Isto, pois não se pode prever o impacto da troca de valores de multiplicadores nos produtos do *relax-and-cut*.

Em limitantes inferiores, o RCDIST supera o RCSEQ-1 em 25 e o RCSEQ-2 em 13 testes. Há, no total, 44 instâncias. Considerando somente os casos em que o limitante é superável, ou seja, nos quais o valor ótimo não é alcançado, os máximos números possíveis são, respectivamente, 37 e 28. É no grupo 2 de instâncias (veja a classificação feita na Subseção 4.2.2) que a melhoria de limitantes é mais frequente. É também nesse conjunto, contudo, que há um dos casos de derrota do RCDIST.

Quanto aos limitantes superiores, o valor final do RCDIST é melhor que o do RCSEQ-1 para 10 instâncias. Em comparação ao RCSEQ-2, há 9 vitórias. Novamente, restringindo-se aos casos para os quais a obtenção de um menor valor é possível, os números citados estão limitados a 16 e 15 respectivamente. Como para limitantes inferiores, é no segundo grupo de instâncias que há, em proporção, a maior quantidade de casos de sucesso do RCDIST.

Para confrontar os métodos quanto às quantidades de cortes utilizados e de variáveis fixadas, é oportuno fazer uma separação. Ao considerar esses quesitos, é mais adequado que se compare os algoritmos *relax-and-cut* aplicados aos dois problemas processados no simulado sistema distribuído ao RCSEQ-1 e ao RCSEQ-2 isoladamente. Os referidos problemas estão indexados por 1 e 2.

Por meio dos resultados apresentados, nota-se que as quantidades de cortes gerados e presentes ao final do processo para os métodos componentes do RCDIST são, em geral, maiores que as relativas ao RCSEQ-1 e ao RCSEQ-2. No entanto, vê-se, também, que os números obtidos são comumente parecidos. São poucos os testes em que se observam distinções significativas. Aqui, é possível destacar alguns casos. Para a instância aa03, a abordagem associada ao problema 1 é mais eficiente que o RCSEQ-1. Para o primeiro método, uma menor quantidade de cortes gerados proporciona um limitante inferior mais preciso. Os respectivos valores são 5955 e 6975 e 49565 e 49348. Com o teste aa06, a diferença de desempenho ocorre no sentido inverso. Os valores correspondentes são 5922 e 3638 e 26974 e 26982.

Em geral, mais variáveis são fixadas durante o processamento dos problemas no su-

posto ambiente distribuído. Para apenas 5 instâncias, tal número é menor na coluna VarF do problema 1 ou na coluna VarF do problema 2. O teste aa06 é um exemplo de superioridade das abordagens presentes no ambiente distribuído. Neste caso, a porcentagem atingida no RCSEQ-2 é menor que a resultante do método aplicado ao problema 2.

Com uma análise dos tempos de execução, pode-se considerar que o RCDIST é um pouco mais lento que o RCSEQ-1 e que o RCSEQ-2. Em vários testes, o RCDIST é mais rápido que o RCSEQ-1. Em outros, supera também o RCSEQ-2. Para todas as instâncias do grupo 2, para quase todas do grupo 3 e para a maioria geral dos casos, porém, as situações acima não ocorrem é há um aumento no tempo de computação (novamente, lembre os grupos de instâncias definidos na Subseção 4.2.2). Tal aumento, no entanto, gera, frequentemente, um valor no máximo dobrado. Nos casos contrários, tem-se comumente uma das duas seguintes circunstâncias: um limitante inferior mais preciso é obtido com o RCDIST ou os tempos comparados são muito pequenos.

É válido ressaltar que, corroborando a expectativa inicial, o desempenho do método distribuído proposto aqui não supera o do algoritmo *relax-and-cut* originário das extensões feitas nesta dissertação (tal constatação é factível mesmo sem uma comparação explícita dos mesmos). Apesar de os resultados atingidos dominarem os obtidos com as alternativas apresentadas no Capítulo 5, os aspectos prejudiciais lá citados ainda impedem melhores produtos.

Por fim, é importante observar que a cooperação feita aqui consiste essencialmente em uma partida quente para multiplicadores em um algoritmo *relax-and-cut*. Lembre que, quando tal mecanismo foi testado no Capítulo 4, este se mostrou pouco proveitoso. Aparentemente, o sucesso da técnica depende da forma de implementação.

6.4 Conclusão

Neste capítulo, foi apresentado um algoritmo *relax-and-cut* distribuído para o SPP. Em cada uma das unidades do sistema subjacente, é resolvida uma relaxação Lagrangiana onde um subconjunto de restrições do problema que represente uma MRRP não é dualizado. Ainda, as unidades cooperam entre si trocando informações sobre limitantes e cortes. Esta é uma abordagem alternativa à testada no Capítulo 5, na qual se emprega a decomposição Lagrangiana.

O método distribuído obtém melhores resultados que os outros dois algoritmos, cada um fazendo uso de somente uma submatriz de igualdades destacada. Com o método distribuído, atingem-se, em geral, melhores limitantes. Têm-se, também, maiores tempos de execução. Tal aumento no tempo, contudo, é razoável frente aos benefícios alcançados.

A abordagem proposta representa, ainda, um caso bem sucedido de uso de partida quente para multiplicadores em um algoritmo *relax-and-cut*. Um experimento realizado

com essa técnica anteriormente a revelou pouco proveitosa. A forma de implementação parece, então, ser determinante para o sucesso da mesma.

Capítulo 7

Conclusão

Nesta dissertação, estudou-se a combinação de relaxação Lagrangiana com planos de corte na resolução do SPP. Uma conhecida instância dessa combinação é classe de algoritmos *relax-and-cut*. Uma aplicação de sucesso de um método dessa classe ao SPP é relatada em [15], por Cavalcante *et al.* Tal método, apesar de muito bem sucedido, é passível de potenciais melhoramentos. Optou-se, então, por realizar o estudo proposto aqui através da sugestão, implementação e avaliação de extensões do mesmo.

Dentre as modificações apresentadas, a de maior impacto foi a transformação do algoritmo desenvolvido em [15] em um método exato. Foi construído, assim, um *branch-and-cut* baseado somente em relaxação Lagrangiana para o SPP. A abordagem exata mostrou-se mais eficiente que a heurística. Isto, no sentido em que, com a primeira, foi possível determinar, em termos práticos, ou seja, em um tempo bastante razoável, o valor ótimo para um número substancialmente maior de testes; estes sendo padrões na literatura. Quando comparado ao resolvedor comercial CPLEX, contudo, o novo algoritmo revelou-se pouco competitivo.

Na sequência, foi realizada uma investigação sobre a influência da detecção e de destaque de problemas fáceis componentes do SPP na relaxação Lagrangiana. Esperava-se, com isso, melhores produtos do algoritmo *relax-and-cut* incorporado no método exato e, consequentemente, melhor desempenho do último. Foram testadas duas novas versões do *branch-and-cut*: com uma e com duas MRRPs destacadas; na última sendo empregada a técnica de decomposição Lagrangiana. No entanto, os resultados obtidos foram piores. Apontou-se, como principais razões para o insucesso, a ineficiência dos cortes e da heurística primal utilizados quando não são todas as restrições dualizadas.

Avaliou-se, ainda, o uso da estratégia descrita acima no contexto de um sistema distribuído. Na simulação feita, a cooperação entre algoritmos *relax-and-cut* mostrou-se, de fato, proveitosa. Os resultados atingidos, porém, permaneceram aquém dos encontrados inicialmente. É preciso notar que, nesta versão cooperativa, não se tentou combater a

ineficiência dos elementos citados anteriormente. Buscou-se, somente, uma alternativa mais valiosa de emprego de relaxações Lagrangianas alternativas.

Na primeira das extensões feitas neste trabalho, pesquisou-se quais os benefícios de melhores valores iniciais, uma partida quente, para os multiplicadores de um algoritmo *relax-and-cut*. Tal procedimento revelou-se pouco vantajoso. Por outro lado, a cooperação experimentada na última das modificações efetuadas é um exemplo bem sucedido do mesmo. Aparentemente, a forma de implementação é determinante para os resultados alcançados.

Em suma, observou-se que a estratégia de combinar relaxação Lagrangiana com planos de corte pode ser usada com sucesso na resolução do SPP. Entretanto, algoritmos desta natureza ainda se mostram pouco competitivos quando confrontados a um resolvedor comercial tradicional. Ainda, as constatações negativas feitas neste estudo não são palavras finais sobre o potencial de tais algoritmos. É razoável que, com cada vez mais conhecimento sobre a estratégia e refinamento dos métodos, cada vez melhores resultados sejam obtidos.

Apêndice A

Um melhor valor inicial para um novo multiplicador de Lagrange

A.1 Objetivos

Em [15], Cavalcante et al. desenvolveram um algoritmo *relax-and-cut* para o Problema de Particionamento de Conjuntos (SPP, do inglês *Set Partitioning Problem*). Nesse algoritmo, quando uma nova restrição é adicionada ao problema, é dado valor *zero* ao seu multiplicador. Comparado aos outros, esse multiplicador terá uma menor quantidade de iterações, até a parada do algoritmo, para convergir para seu valor ótimo.

O objetivo desse experimento é computar um melhor valor inicial para o multiplicador de um nova restrição inserida e avaliar o impacto dessa modificação nos limitantes e valor final do multiplicador obtidos, bem como no tempo despendido pelo algoritmo.

A.2 Metodologia

O algoritmo proposto em [15] foi estendido, a fim de implementar as modificações citadas acima. Originalmente, o algoritmo usa o Método do Subgradiente (MS) para resolver o Problema Dual Lagrangeano, tratado a partir daqui por MS *externo*, tentando, a cada iteração, adicionar cortes ao problema, excluindo soluções subótimas e fortalecendo sua formulação. Agora, quando um corte é encontrado, um novo MS é executado, tratado a partir daqui por MS *interno*, para encontrar "rapidamente" um bom valor para o respectivo multiplicador.

Foram executadas três versões do algoritmo: (1) a versão original, (2) uma com limite de iterações sem melhoria de limitante para o MS interno sendo $\lfloor (k/2500)30 \rfloor$, onde k é a iteração atual do MS externo e (3) outra com esse limite igual ao do MS externo.

Para os três casos, o número máximo de iterações (somadas as iterações dos MS *externo* e *interno*) foi 8000 e o fator de decréscimo do parâmetro ϵ^k foi $0,75$. Para os primeiro e terceiro, tomou-se um limite 30 iterações sem melhoria de limitante.

As instâncias utilizadas nesse experimento são derivadas de situações reais, oriundas do problema de alocação de tripulação aérea. Essas compõem o conjunto de instâncias testadas em [15], onde há detalhes sobre como obtê-las.

A.3 Resultados e Discussão

O código foi escrito em `C++` e os testes executados em uma máquina com processador `Intel Core 2 Quad` e 3GB de memória RAM. O sistema operacional Linux (Kubuntu 8.04) e o compilador `g++` (versão 4.2) com opções `-O3` e `-lm` foram utilizados. Os resultados estão distribuídos nas Tabelas A.1, A.2 e A.3.

Os Algoritmos 2 e 3 apresentaram melhorias de limitantes em um pequeno número de instâncias. Além disso, na maioria dos casos, essa melhoria foi bastante pequena (Tabelas A.1, A.2 e A.3 e figuras A.1 e A.2). Em média, os Algoritmos 2 e 3 conseguiram aumentar o valor do limitante inferior (lb, do inglês *lower bound*) em apenas 0.58% e 0.65% respectivamente. Em contrapartida, conseguiram diminuir o valor do limitante superior (ub, do inglês *upper bound*) mais expressivamente. Diminuiram em 13.42% e 16.08% respectivamente (Tabela A.8).

Tabela A.1: Resultados para os Algoritmos 1, 2 e 3

Instância	**Ótimo**	**Algoritmo 1**			**Algoritmo 2**			**Algoritmo 3**		
		ub	*lb*	$t(s)$	*ub*	*lb*	$t(s)$	*ub*	*lb*	$t(s)$
aa01	56138	57528	55126	21.33	$\overline{57528}$	55610	78.14	57528	$\underline{55632}$	235.19
aa03	49649	50747	49524	16.44	$\overline{50586}$	49632	18.04	50747	$\underline{49635}$	21.42
aa04	26374	29888	25924	9.21	$\overline{29888}$	25946	44.42	29888	$\underline{25952}$	124.89
aa05	53839	55160	53677	16.71	54180	53743	18.67	$\overline{53941}$	$\underline{53767}$	34.83
aa06	27040	27293	26983	9.26	$\overline{27293}$	26995	14.78	27293	$\underline{27004}$	33.30
kl01	1086	1089	1085	11.25	1117	$\underline{1085}$	27.80	$\overline{1089}$	1085	81.88
kl02	219	220	216	42.32	$\overline{220}$	$\underline{216}$	36.58	220	216	83.77
nw02	105444	105444	105444	748.21	$\overline{105444}$	$\underline{105444}$	100.13	105444	105439	124.50
nw03	24492	$\overline{24504}$	24474	410.35	24555	24455	530.12	24558	$\underline{24492}$	189.41
nw04	16862	41858	16294	435.26	$\overline{20296}$	$\underline{16320}$	1025.50	20296	16319	19543.88
nw05	132878	132878	132878	465.45	$\overline{132878}$	$\underline{132878}$	249.92	132878	132878	133.57
nw06	7810	7810	7810	2.18	$\overline{7810}$	$\underline{7810}$	8.25	7810	7810	15.35
nw08	35894	35894	35894	0.04	$\overline{35894}$	$\underline{35894}$	0.03	35894	35894	0.04

Quanto ao tempo, em média, os Algoritmos 2 e 3 gastaram aproximadamente 3 e 10

vezes o tempo gasto pelo Algoritmo 1 (Tabela A.7).

Tabela A.2: Resultados para os Algoritmos 1, 2 e 3

Instância	**Ótimo**	**Algoritmo 1**			**Algoritmo 2**			**Algoritmo 3**		
		ub	lb	$t(s)$	ub	lb	$t(s)$	ub	lb	$t(s)$
nw10	68271	68271	68271	0.10	68271	68271	0.04	68271	68268	0.15
nw12	14118	14118	14118	0.12	14118	14118	0.04	14118	14118	0.04
nw13	50146	50146	50141	26.67	50146	50144	16.90	50146	50137	14.17
nw14	61844	72052	59428	294.76	61844	61844	3539.03	61844	61844	250.04
nw15	67743	67743	67743	0.61	67743	67340	0.39	67743	67743	0.16
nw17	11115	37938	10847	222.88	27280	10913	1732.78	27292	10918	7482.34
nw18	340160	340162	339927	52.98	340160	340160	127.35	340160	340112	300.45
nw20	16812	16812	16812	0.94	16812	16812	0.61	16812	16812	0.79
nw21	7408	7408	7408	0.02	7408	7408	0.02	7408	7408	0.03
nw23	12534	12534	12534	0.56	12534	12532	1.17	12534	12501	8.84
nw24	6314	6314	6314	0.19	6338	6295	1.62	6314	6314	4.61
nw25	5960	5960	5960	0.33	5960	5960	0.17	5960	5960	0.32
nw26	6796	6796	6796	0.05	6796	6796	0.12	6796	6796	0.14

A.4 Conclusão

Os algoritmos desenvolvidos nesse experimento não se apresentaram significativamente mais eficientes que o algoritmo original. Esses conseguiram melhorias nos limitantes obtidos, que, porém, em média, foram bastante pequenas. Além disso, no outro aspecto analisado, o aumento de tempo causado pelo novo cálculo inserido, a piora foi significativa.

Tabela A.3: Resultados para os Algoritmos 1, 2 e 3

Instância	**Ótimo**	**Algoritmo 1**			**Algoritmo 2**			**Algoritmo 3**		
		ub	lb	$t(s)$	ub	lb	$t(s)$	ub	lb	$t(s)$
nw31	8038	8038	8038	0.30	8038	8038	0.59	8038	8038	0.89
nw32	14877	14877	14877	0.08	14877	14877	0.32	14877	14877	0.16
nw33	6678	7892	6536	1.82	7840	6536	1.82	6980	6536	1.66
nw34	10488	10488	10488	0.05	10488	10488	0.08	10488	10488	0.07
nw35	7216	7216	7216	0.18	7216	7216	0.29	7216	7216	0.26
nw36	7314	7722	7231	2.95	7502	7282	12.70	7328	7281	27.09
nw37	10068	10068	10068	0.02	10068	10068	0.05	10068	10068	0.06
nw39	10080	10080	10080	0.16	10080	10080	0.13	10080	10080	0.19
nw41	11307	11307	11307	0.00	11307	11307	0.01	11307	11307	0.01
nw42	7656	7656	7647	1.58	7656	7628	1.13	7656	7656	0.28
nw43	8904	8904	8904	0.08	8904	8904	0.28	8904	8904	0.20

Tabela A.4: Comparação entre os Algoritmos 1 e 2

Algoritmo		**# Melhores limitantes**
Algoritmo 1	*ub*	3
	lb	7
Algoritmo 2	*ub*	8
	lb	12

Figura A.1: Limitantes Superiores (ub) dos Algoritmos

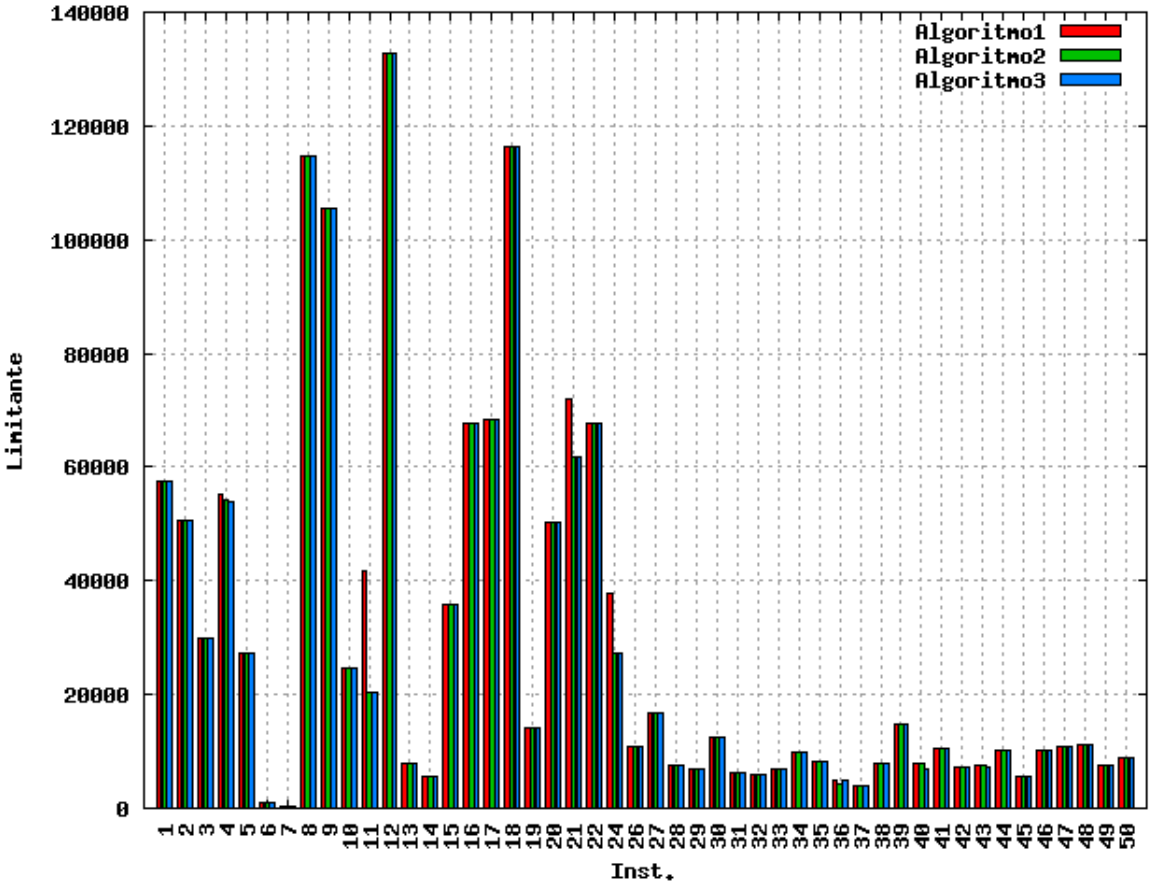


Tabela A.5: Comparação entre os Algoritmos 1 e 3

Algoritmo		**# Melhores limitantes**
Algoritmo 1	*ub*	1
	lb	5
Algoritmo 3	*ub*	7
	lb	11

Tabela A.6: Comparação entre os Algoritmos 2 e 3

Algoritmo		**# Melhores limitantes**
Algoritmo 2	*ub*	4
	lb	8
Algoritmo 3	*ub*	5
	lb	11

Figura A.2: Limitantes Inferiores (lb) dos Algoritmos

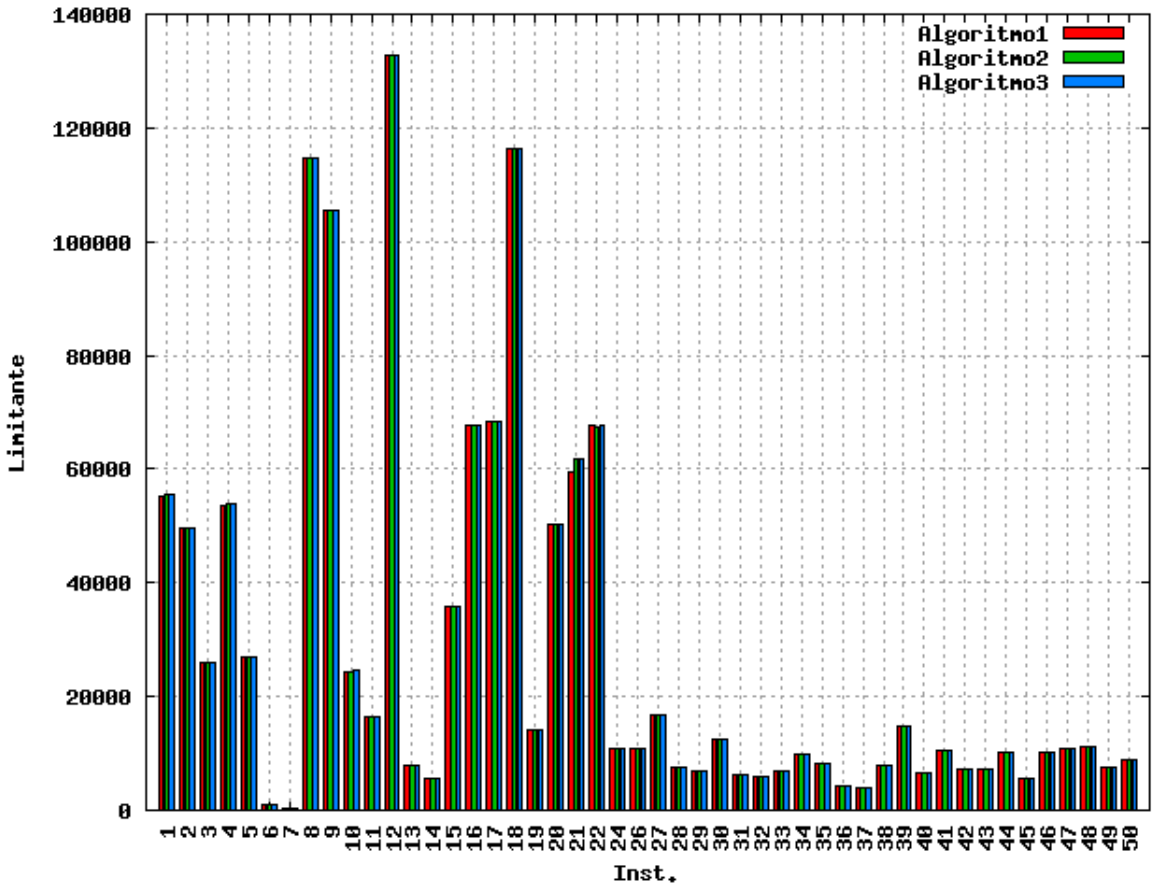


Tabela A.7: Média de tempo dos Algoritmos

Algoritmo	**Média de tempo(s)**
Algoritmo 1	57.17
Algoritmo 2	153.54
Algoritmo 3	576.41

Algoritmo		**Média(%)**
Algoritmo 2	*ub*	13.42
	lb	0.58
Algoritmo 3	*ub*	16.08
	lb	0.65

Tabela A.8: Média de melhoria de limitantes dos Algoritmos

Referências Bibliográficas

[1] R. Andrade, A. Lucena, and N. Maculan. Using Lagrangian dual information to generate degree constrained spanning trees. *Discrete Applied Mathematics*, 154(5):703–717, 2006.

[2] B. Baker and P. Maye. A heuristic for finding embedded network structure in mathematical programmes. *European Journal of Operational Research*, 67(1):52–63, 1993.

[3] E. Balas. Some valid inequalities for the Set Partitioning Problem. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 13–47. Elsevier, 1977.

[4] E. Balas and N. Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21(1):19–46, 1981.

[5] E. Balas and S. Ng. On the set covering polytope: I. All the facets with coefficients in {0, 1, 2}. *Mathematical Programming*, 43(1-3):57–69, 1989.

[6] E. Balas and S. Ng. On the set covering polytope: II. Lifting the facets with coefficients in {0, 1, 2}. *Mathematical Programming*, 45(1-3):1–20, 1989.

[7] E. Balas and M. Padberg. Set partitioning: A survey. *SIAM Review*, 18(4):710–760, 1976.

[8] J. Bartholdi. A good submatrix is hard to find. *Operations Research Letters*, 1(5):190–193, 1982.

[9] J. Beasley. Lagrangian relaxation. In *Modern heuristic techniques for combinatorial problems*, chapter 6. Wiley, 1993.

[10] A. Belloni and A. Lucena. Lagrangian heuristics for the linear ordering problem. In *Metaheuristics: Computer Decision-Making*, pages 37–63. Kluwer, 2004.

[11] R. Bixby and W. Cunningham. Converting linear programs to network problems. *Mathematics of Operations Research*, 5(3):321–357, 1980.

[12] R. Bixby and R. Fourer. Finding embedded network rows in linear programs I. Extraction heuristics. *Management Science*, 34(3):342–376, 1988.

[13] R. Borndörfer. *Aspects of Set Packing, Partitioning and Covering*. PhD thesis, School of Mathematics and Natural Sciences, Berlin Institute of Technology, 1998.

[14] F. Calheiros, A. Lucena, and C. de Souza. Optimal rectangular partitions. *Networks*, 41(1):51–67, 2003.

[15] V. Cavalcante, C. de Souza, and A. Lucena. A Relax-and-Cut algorithm for the set partitioning problem. *Computers & Operations Research*, 35(6):1963–1981, 2008.

[16] H. Crowder and M. Padberg. Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, 26(5):495–509, 1980.

[17] V. Cung, S. Martins, C. Ribeiro, and C. Roucairol. Strategies for the parallel implementation of metaheuristics. In *Essays and Surveys in Metaheuristics*, pages 263–308. Kluwer, 2002.

[18] J. da Silva. Uma heurística Lagrangeana para o problema da árvore geradora capacitada de custo mínimo. Dissertação de Mestrado, Programa de Engenharia de Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro, 2002.

[19] R. Day. On optimal extracting from a multiple file data storage system: An application of integer programming. *Operations Research*, 13(3):482–494, 1965.

[20] L. Escudero, M. Guignard, and K. Malik. A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research*, 50(1):219–237, 1994.

[21] M. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. Technical Report PA 19104, Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1990.

[22] M. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42(4):626–642, 1994.

[23] C. Furushima. Estudo de uma heurística para a extração de submatrizes de rede puras em programação linear inteira. Projeto de Iniciação Científica, Instituto de Computação, Universidade Estadual de Campinas, 2007.

[24] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., 1979.

[25] R. Garfinkel and G. Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16(8):B495–B508, 1970.

[26] R. Garfinkel and G. Nemhauser. *Integer Programming*. Wiley, 1972.

[27] B. Gavish. Augmented Lagrangean based algorithms for centralized network design. *IEEE Transactions on Communications*, 33(12):1247–1257, 1985.

[28] A. Geoffrion. Lagrangean relaxation for integer programming. In *Approaches to Integer Programming*, volume 2 of *Mathematical Programming Studies*, pages 82–114. Springer, 1974.

[29] R. Gomory. An algorithm for integer solutions to linear programs. In *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, 1963.

[30] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101(1):81–92, 1997.

[31] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[32] M. Guignard. Lagrangean relaxation. *TOP*, 11(2):151–200, 2003.

[33] M. Guignard and S. Kim. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming*, 39(2):215–228, 1987.

[34] N. Gülpinar, G. Gutin, G. Mitra, and I. Maros. Detecting embedded networks in LP using GUB structures and independent set algorithms. *Computational Optimization and Applications*, 15(3):235–247, 2000.

[35] N. Gülpinar, G. Gutin, G. Mitra, and A. Zverovitch. Extracting pure network submatrices in linear programs using signed graphs. *Discrete Applied Mathematics*, 137(3):359–372, 2004.

[36] K. Haase, G. Desaulniers, and J. Desrosiers. Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35(3):286–303, 2001.

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

[38] M. Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

[39] K. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, 1993.

[40] D. Hoitomt, P. Luh, and K. Pattipati. Job shop scheduling. In *Proceedings of First International Conference on Automation Technology*, pages 565–574, Taipei, Taiwan, 1990.

[41] D. Hoitomt, J. Perkins, and P. Luh. Distributed scheduling of job shops. In *Proceedings of 1991 IEEE International Conference on Robotics and Automation*, volume 2, pages 1067–1072, Sacramento, CA, USA, 1991.

[42] M. Hunting, U. Faigle, and W. Kern. A Lagrangian relaxation approach to the edge-weighted clique problem. *European Journal of Operational Research*, 131(1):119–131, 2001.

[43] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. *COAL Bulletin*, 21:2–7, 1992.

[44] A. Lucena. Tight bounds for the Steiner problem in graphs. In *Proceedings of NETFLOW 93*, pages 147–154, Pisa, Italy, 1993.

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

[46] R. Marsten and F. Shepardson. Exact solution of crew scheduling problems using the set partitioning model: Recent successful applications. *Networks*, 11(2):165–177, 1981.

[47] C. Martinhon, A. Lucena, and N. Maculan. Stronger k-tree relaxations for the vehicle routing problem. *European Journal of Operational Research*, 158(1):56–71, 2004.

[48] G. Nemhauser and G. Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *The Journal of the Operational Research Society*, 43(5):443–457, 1992.

[49] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1999.

[50] P. Nobili and A. Sassano. Facets and lifting procedures for the set covering polytope. *Mathematical Programming*, 45(1-3):111–137, 1989.

[51] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

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

[53] J. Pierce. Pattern sequencing and matching in stock cutting operations. *TAPPI*, 53(4):668–678, 1970.

[54] B. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3):14–29, 1969.

[55] C. Revelle, D. Marks, and J. Liebman. An analysis of private and public sector location models. *Management Science*, 16(11):692–707, 1970.

[56] C. Ribeiro and I. Rosseti. Efficient parallel cooperative implementations of GRASP heuristics. *Parallel Computing*, 33(1):21–35, 2007.

[57] J. Root. An application of symbolic logic to a selection problem. *Operations Research*, 12(4):519–526, 1964.

[58] J. Valenta. Capital equipment decisions: A model for optimal systems interfacing. Master's thesis, Sloan School of Management, Massachusetts Institute of Technology, 1969.

[59] L. Wolsey. *Integer Programming*. Wiley, 1998.