Algoritmos *relax-and-cut* para problemas de Programação Inteira 0–1

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Victor Fernandes Cavalcante e aprovada pela Banca Examinadora.

Campinas, 31 de julho de 2008.

Prof. Dr. Cid Carvalho de Souza (Orientador)

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

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Cavalcante, Victor Fernandes
C314a Algoritmos relax-and-cut para problemas de programação inteira 0-1 / Victor Fernandes Cavalcante -- Campinas, [S.P. :s.n.], 2008.

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

1. Otimização combinatória. 2. Programação inteira. 3. Algoritmos. I. Souza, Cid Carvalho de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Relax-and-cut algorithms for 0-1 integer programming problems.

Palavras-chave em inglês (Keywords): 1. Combinatorial optimization. 2. Integer programming. 3. Algorithms.

Área de concentração: Otimização Matemática

Titulação: Doutor em Ciência da Computação

Banca examinadora:

Prof. Dr. Cid Carvalho de Souza (IC-UNICAMP)
Profa. Dra. Laura Silvia Bahiense da Silva Leite (COPPE-UFRJ)
Prof. Dr. Abílio Pereira de Lucena Filho (FEA-UFRJ)
Prof. Dr. Vinícius Amaral Armentano (FEEC-UNICAMP)
Prof. Dr. Flávio Keidi Miyazawa (IC-UNICAMP)

Data da defesa: 31/07/2008

Programa de Pós-Graduação: Doutorado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 31 de julho de 2008, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Abílio Pereira de Lucena Filho
FEA – UFRJ.

Profª. Drª. Laura Silvia Bahiense da Silva Leite
COPPE – UFRJ.

Prof. Dr. Vinícius Amaral Armentano
FEEC – UNICAMP.

Prof. Dr. Flávio Keidi Miyazawa
IC – UNICAMP.

Prof. Dr. Cid Carvalho de Souza
IC – UNICAMP.

Instituto de Computação
Universidade Estadual de Campinas

Algoritmos *relax-and-cut* para problemas de Programação Inteira 0–1

Victor Fernandes Cavalcante[1]

Julho de 2008

Banca Examinadora:

- Prof. Dr. Cid Carvalho de Souza (Orientador)
- Prof. Dr. Abílio Pereira de Lucena Filho
  FEA - UFRJ
- Profa. Dra. Laura Silvia Bahiense da Silva Leite
  COPPE - UFRJ
- Prof. Dr. Vinícius Amaral Armentano
  FEEC - UNICAMP
- Prof. Dr. Flávio Keidi Miyazawa
  IC - UNICAMP
- Profa. Dra. Yoshiko Wakabayashi
  IME - USP (suplente)
- Prof. Dr. Arnaldo Vieira Moura
  IC - UNICAMP (suplente)
- Prof. Dr. Ricardo Dahab
  IC - UNICAMP (suplente)

[1]Suporte financeiro de: Bolsa da CAPES 2002–2006.

Resumo

Uma das principais motivações para o estudo de Otimização Discreta reside no elevado número de problemas do nosso cotidiano representáveis através de modelos de Otimização Inteira e Combinatória. Em particular, muitos destes problemas podem ser formulados com Programação Inteira 0–1, o que desperta especial interesse em técnicas capazes de resolver tais modelos. Dentre as inúmeras formas de solução atualmente disponíveis para problemas desta natureza, os algoritmos baseados na técnica de relaxação Lagrangiana surgem como uma alternativa que tem tido grande sucesso na prática. Além disso, avanços consideráveis ocorreram na área de Programação Inteira com o advento da Combinatória Poliédrica, intensificando o interesse pelos algoritmos de planos de corte.

Neste contexto, esta tese tem como principal objetivo verificar as potencialidades do uso combinado de Combinatória Poliédrica e relaxação Lagrangiana na resolução de dois problemas de otimização combinatória. Mais especificamente, o presente trabalho está focado no desenvolvimento dos chamados algoritmos *relax-and-cut* para o problema de particionamento de conjuntos e ao problema do separador de vértices de um grafo.

Sendo assim, são propostos algoritmos que combinam relaxação Lagrangiana e planos de cortes faciais para os dois problemas sob consideração. Em ambos os casos, os resultados obtidos com os testes computacionais realizados são comparados com os melhores resultados disponíveis na literatura.

Os principais resultados alcançados na tese mostram que: *(a)* o uso combinado de relaxação Lagrangiana e planos de corte constitui uma alternativa bastante competitiva para solucionar o problema de particionamento de conjuntos, frequentemente superando o desempenho dos melhores algoritmos disponíveis na literatura para o problema e, *(b)* no caso do problema do separador de vértices, além da combinação de técnicas Lagrangianas com o uso de planos de corte, a hibridização dos algoritmos *relax-and-cut* e *branch-and-cut* leva à resolução de instâncias da literatura mais rapidamente que o melhor algoritmo exato conhecido para o problema até então.

Abstract

One of the main motivations for the study of Discrete Optimization resides in the huge number of problems from our daily life that can be represented through Integer and Combinatorial Optimization models. In particular, many of these problems can be cast as 0–1 Integer Programs, which gives rise to special interest on how to solve such models. Among the several ways currently available to solve problems of this nature, the algorithms based on Lagrangian relaxation techniques appears as an alternative that has had great success in practice. Besides, noticeable achievements occurred with the advent of Polyhedral Combinatorics, intensifying the interest on cutting plane algorithms.

In this context, this thesis has as its main goal to verify the potentialities of the combined usage of Polyhedral Combinatorics and Lagrangian relaxation in the resolution of two combinatorial optimization problems. More specifically, the present work is focused on the development of the so-called *relax-and-cut* algorithms for the set partition problem and for the vertex separator problem on graphs.

Therefore, algorithms combining Lagrangian relaxation and cutting planes are proposed for the two problems under consideration. In both cases, the results obtained in the computational tests carried out are compared with the best ones available in the literature.

The main results achieved in the thesis show that: *(a)* the combined usage of Lagrangian relaxation and cutting planes constitutes a competitive alternative to solve the set partition problem, often outperforming the best algorithms available in the literature for the problem and, *(b)* in the case of the vertex separator problem, besides the combination of Lagrangian techniques and cutting planes, a hybridization of the *relax-and-cut* and *branch-and-cut* algorithms lead to the resolution of instances from the literature more rapidly than the best exact algorithm known for the problem so far.

Agradecimentos

À minha esposa, Cristina, pelo apoio à minha decisão de ingresso no doutorado, pelo companheirismo que estimula a minha busca por novas investidas e, sobretudo, pelo amor, que durante o doutorado ganhou novas fronteiras, valores e apelidos: Clara e Ana.

Aos meus pais e irmãos, pela torcida fraterna e respeito aos meus projetos.

Ao meu orientador, Cid, pela dedicação, amizade e competência acadêmica sempre presentes na forma com que conduziu esta pesquisa.

Aos professores Abílio Lucena, Vinícius Armentano e Arnaldo Moura, pelos valorosos comentários feitos quando do meu Exame de Qualificação de Doutorado.

A todos os professores que contribuiram de alguma forma para a minha formação acadêmica e/ou para o andamento do trabalho de pesquisa desenvolvido ao longo do doutorado.

Aos colegas da UNICAMP, pelas conversas (técnicas ou não) e pela atenção dispensada durante os momentos compartilhados ao longo destes últimos anos.

Aos funcionários do Instituto de Computação pelo ambiente agradável e pela disposição e prontidão em ajudar quando solicitados.

Aos amigos que, ao longo deste período, me incentivaram de alguma forma a persistir nos meus objetivos.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Uma das principais motivações para o estudo de otimização discreta reside no elevado número de problemas do nosso cotidiano representáveis através de modelos de otimização inteira e combinatória. Tais problemas, cuja aplicabilidade abrange os mais variados domínios do conhecimento, surgem de necessidades reais de gerenciamento eficiente de recursos escassos e, em geral, buscam o aumento de produtividade (maximização de lucros e/ou minimização de custos). Em particular, os problemas de otimização 0–1 despertam especial interesse devido à dificuldade intrínseca à combinatória subjacente a estes problemas.

Devido à grande necessidade prática de resolução de problemas desta natureza e ao impulso dado pelo crescente desenvolvimento tecnológico de recursos computacionais, inúmeras abordagens de solução têm sido propostas e experimentalmente validadas ao longo das últimas décadas. Dentre elas, algoritmos baseados na técnica de relaxação Lagrangiana têm sido aplicados com sucesso a diversos problemas de otimização discreta.

Dentro deste contexto, o presente trabalho teve por principal objetivo verificar a aplicabilidade do uso combinado de Combinatória Poliédrica e relaxação Lagrangiana na resolução de problemas de otimização combinatória 0–1. Mais especificamente, a presente pesquisa teve como cerne o desenvolvimento dos chamados algoritmos *relax-and-cut* [35] e sua aplicação a dois problemas de otimização 0–1: o problema de particionamento de conjuntos e o problema do separador de vértices de um grafo.

1.1 O problema de particionamento de conjuntos

Dentre todas as estruturas especiais de programação inteira (pura) o problema de particionamento de conjuntos, aqui denotado por SPP, do inglês *Set Partitioning Problem*, é mencionado na literatuta como um dos problemas com maior aplicabilidade. Balas e Padberg [4] o qualificam como o problema de otimização discreta com a mais vasta gama

de aplicações.

Desde meados de 1960, inúmeras applicações práticas são descritas na literatura relativa ao SPP, dentre elas: escalonamento de veículos [6, 10], escalonamento de motoristas de ônibus [17] e escalonamento de tripulações de vôo (*Crew Scheduling Problem*) [29, 38]; projeto de circuitos integrados [44] e localização de facilidades [43]. Revisões detalhadas sobre o SPP estão disponíveis em [4, 22]. Para um acompanhamento da literatura mais recente, sugerem-se as referências [10, 11, 14, 31].

O SPP é um problema $\mathcal{NP}$-*Difícil* [21] que pode ser visto como um caso especial do problema de recobrimento de conjunto ou SCP (do inglês *Set Covering Problem*). No SCP são dados um conjunto universo C e uma família $\mathcal{F}$ de subconjuntos de C com custos associados a cada um dos seus elementos. O problema consiste em encontrar subconjuntos $C_1, C_2, ..., C_k$ em $\mathcal{F}$ cuja soma dos custos é mínima e cuja união seja C. Neste caso, tem-se que $C \subseteq C' = C_1 \cup C_2 \cup \ldots \cup C_k$ e diz-se que C' cobre C.

Quando se impõe a restrição de que C' seja também um empacotamento, a solução do problema será um particionamento, ou seja, tem-se que $C_i \cap C_j = \emptyset,\ \forall C_i, C_j \subseteq C'$. O problema de otimização obtido nesta situação é o SPP. Note que, neste caso, cada elemento aparece em exatamente um subconjunto. Por este motivo, o SPP é também conhecido como problema de recobrimento com restrições de igualdade.

Alternativamente, uma maneira mais formal e compacta para enunciarmos o SPP é descrevê-lo através de um programa (linear) inteiro 0-1 da seguinte maneira:

$$z_{SPP} = \min \{cx : Ax = e_m,\ x \in \mathbb{B}^n\} \tag{1.1}$$

onde: A é uma matriz $m \times n$ de zeros e uns, cujas colunas são os vetores de incidência dos subconjuntos de C presentes em $\mathcal{F}$; c é o vetor n-dimensional contendo os custos dos subconjuntos de $\mathcal{F}$; $e_m = \{1, \ldots, 1\}$ é um vetor de $m = |C|$ entradas iguais a um e, por fim, x é um vetor de variáveis binárias cuja j-ésima componente assume o valor um se e somente se o subconjunto C_j de $\mathcal{F}$ pertencer à solução. Assim sendo, o objetivo é determinar valores para as variáveis binárias x_j que minimizem a função objetivo z_{SPP}. De acordo com esta descrição matricial, cada conjunto C_j corresponde a um subconjunto de linhas cobertas pela coluna j. Logo, dizer que cada elemento (linha) de C aparece em somente um subconjunto é equivalente a afirmar que, em uma solução do SPP, uma linha é coberta por exatamente uma coluna.

Tomando-se $N = \{1, \ldots, n\}$, o conjunto de índices dos subconjuntos, a seguinte interpretação (Balas e Padberg [4]) é dada para justificar a proveniência do nome SPP: se as linhas de A são associadas a elementos do conjunto $C = \{1, \ldots, m\}$ e cada coluna a^j de A for associada ao conjunto C_j contendo as $i \in C$ linhas cobertas pela coluna j (i.e., $a_{ij} = 1$ se $i \in C_j$), então o SPP é o problema de encontrar uma coleção de subconjuntos de custo mínimo, que seja uma partição de C, onde cada $C_j,\ j \in N$, tem custo c_j.

Em [12] e [13], o problema de partição retangular (RG-P) é formulado como um caso particular do problema de particionamento de conjuntos. Da aplicação com sucesso de um algoritmo *relax-and-cut* ao RG-P nasceu a principal motivação para investigarmos a adequabilidade desta estratégia para resolver o SPP.

1.2 O problema do separador de vértices

Um separador de vértices em um grafo não orientado é um subconjunto do conjunto de vértices do grafo cuja remoção desconecta o grafo em pelo menos duas componentes conexas. Formalmente, o problema do separador de vértices, denotado por VSP (do inglês, *Vertex Separator Problem*), pode ser enunciado como segue:

`INSTÂNCIA:` um grafo conexo não orientado $G = (V, E)$, com $|V| = n$, um inteiro $1 \leq b \leq n$ e um custo c_i associado com cada vértice $i \in V$.

`PROBLEMA:` Encontrar uma partição de V em conjuntos disjuntos A, B, C, com $|A| \geq 1$ e $|B| \geq 1$ tais que *(i)* Não existe aresta $(i, j) \in E$ tal que $i \in A$, $j \in B$; *(ii)* $\max\{|A|, |B|\} \leq b$ e *(iii)* $\sum_{j \in C} c_j$ é minimizado.

Os conjuntos A e B serão chamados aqui de bordas (do inglês, *shores*) do separador C. Um separador C que satisfaz (i) e (ii) é dito *viável*. O separador que também satisfaz (iii) é dito ótimo. Este problema é $\mathcal{NP}$-Difícil e possui inúmeras aplicações no contexto de conectividade de redes (veja [3] para discussões adicionais relacionadas a aplicações, inclusive em Álgebra Linear).

A definição do VSP, na forma como está descrito acima, foi introduzida por Balas e de Souza [3, 16]. Em [3] uma investigação poliedral sobre o VSP foi realizada, onde diversas famílias de desigualdades fortes para o politopo associado ao problema são discutidas. Em [16] os autores reportam experimentos computacionais baseados em resultados obtidos com a implementação de um algoritmo *branch-and-cut* baseado nestas desigualdades. A identificação das classes de desigualdades válidas para o VSP descritas em [3] e a qualidade dos resultados reportados em [16] foram os principais motivadores para que o VSP fosse escolhido como um dos problemas-alvo do presente trabalho de pesquisa.

Um problema similar ao VSP é descrito em [42] por Borndörfer *et al.* Este problema pode ser visto como uma generalização do VSP no sentido que o particionamento pode ser feito em mais que dois conjuntos de vértices. Todavia, contrariamente ao VSP, soluções onde A ou B ficam vazios são permitidas.

1.3 Organização do Texto

Uma parte expressiva do texto do presente trabalho corresponde a artigos publicados ou submetidos a jornais científicos e conferências internacionais. Desta forma, parte do trecho está redigida em inglês, seguindo as normas que regem a pós-graduação da Universidade Estadual de Campinas atualmente. Ainda de acordo com estas normas, o conteúdo do texto escrito em língua estrangeira reproduz fielmente os artigos publicados ou submetidos a periódicos e conferências internacionais. Apenas a formatação do texto foi alterada para ficar consistente com o estilo do restante do documento.

Assim, os capítulos 3 e 4 encontram-se estruturados no seguinte formato: `(i)` um breve texto resumindo o conteúdo do capítulo e discriminando detalhes do artigo publicado ou submetido; `(ii)` o conteúdo do artigo em si, exatamente como publicado ou submetido; `(iii)` uma seção de comentários complementares que detalham ou justificam partes do texto original do artigo que não constam no mesmo por limitações de espaço impostas pelos periódicos, mas que julgou-se serem relevantes para melhor descrição e compreensão do trabalho realizado e, por fim, `(iv)` a lista das citações referenciadas no artigo original[1]. Os comentários aos quais se refere o item `(iii)` são indicados na margem do corpo do texto original dos artigos publicados (ou submetidos) conforme ilustrado no exemplo a seguir.

" ...

The decision problem associated to the separation of `CD` inequalities is $\mathcal{NP}$-complete when the weights on the vertices are restricted to $\{0, 1, 2\}$.

... "

C1 (Pg.107)

Abaixo do ícone que aparece na margem direita encontram-se especificadas o número do comentário relativo à sentença acima e, entre parênteses, a página indicativa do início do comentário é explicitada. A numeração dos comentários é feita por capítulos, isto é, o primeiro comentário do capítulo recebe o número um, o segundo recebe o número dois e assim por diante.

O Capítulo 2 revê, de forma sucinta, alguns conceitos e definições básicas que compõem os aspectos teóricos relevantes à compreensão das técnicas apresentadas e dos modelos e algoritmos discutidos ao longo da texto.

No Capítulo 3 um algoritmo *relax-and-cut* é desenvolvido para o SPP e os diversos aspectos referentes ao *framework* Lagrangiano proposto como abordagem de solução são descritos em detalhes. Em seguida, resultados obtidos com os testes computacionais realizados são confrontados com os melhores resultados disponíveis na literatura para o

[1]Note que as citações referenciadas nos demais capítulos encontram-se listadas no final da tese.

SPP e algumas extensões do nosso trabalho são discutidas.

No Capítulo 4 variantes de algoritmos *relax-and-cut* são desenvolvidas para o VSP e as características relativas ao *framework* Lagrangiano proposto são apresentadas. Novamente são empreendidos vários testes computacionais que analisam o desempenho computacional dos algoritmos que desenvolvemos, comparando os resultados obtidos com aqueles reportados na literatura.

O Capítulo 5 contém as conclusões do trabalho e discute algumas possíveis extensões do mesmo.

O documento é composto ainda por dois apêndices que descrevem versões preliminares dos artigos detalhados, respectivamente, nos capítulos 3 e 4. Cada um destes apêndices está associado a uma publicação em conferência internacional. De forma semelhante aos capítulos escritos em inglês, faz-se referência no texto aos anais das conferências onde foram feitas as publicações.

1.4 Principais contribuições da Tese

As principais contribuições da tese estão concentradas nos Capítulos 3 e 4 deste texto.

O Capítulo 3 corresponde essencialmente ao artigo que foi publicado na revista *Computers and Operations Research*. Mostramos neste trabalho que o uso combinado de relaxações Lagrangiana e planos de corte constitui-se em uma excelente alternativa para solucionar o SPP, notadamente para instâncias onde resolvedores comerciais de programação linear inteira encontram dificuldades de computação.

No Capítulo 4 realizamos um estudo semelhante àquele que foi feito com o SPP mas, agora, com o VSP. Neste trabalho mostramos que, além da combinação de técnicas Lagrangianas com o uso de planos de corte, a hibridização de algoritmos *relax-and-cut* com o algoritmo *branch-and-cut* é extremamemte benéfica. Foi desta forma que conseguimos resolver instâncias presentes nos *benchmarks* da literatura mais rapidamente que o melhor algoritmo exato disponível para o problema até então. Uma versão preliminar deste trabalho foi apresentada em uma conferência internacional cujos anais foram publicados no *Lecture Notes in Computer Science*. Contudo, o texto em inglês constante deste documento corresponde ao artigo que está sendo submetido para publicação em um jornal internacional arbitrado.

Capítulo 2

Fundamentação teórica

2.1 Introdução

O propósito central deste capítulo é fazer uma breve descrição dos métodos e técnicas de otimização utilizados ao longo desta tese. Para tal, faz-se necessário apresentar conceitos e definições básicas que constituem a fundamentação teórica auxiliar no entendimento dos problemas, modelos e algoritmos descritos ao longo dos próximos capítulos. Os resultados e definições aqui compilados tomaram como base as referências [23, 25, 41, 45].

Problemas de Otimização Combinatória podem, com freqüência, ser descritos matematicamente e resolvidos usando-se programação linear inteira. Como resultado desta descrição obtem-se um modelo (formulação) matemático(a) que nada mais é que uma especificação do problema através de três elementos: variáveis, função objetivo e restrições. Estas últimas são usualmente citadas como o aspecto mais importante do modelo [41], pois definem o espaço de soluções viáveis do problema. No caso de Programação Linear Inteira (PLI), este modelo é usualmente denominado simplesmente de Programa Inteiro (PI).

No presente trabalho estamos particularmente interessados em problemas onde todas as variáveis envolvidas sejam binárias, ou seja, problemas de Otimização Combinatória 0-1. Entretanto, as definições apresentadas ao longo deste capítulo consideram/são válidas para o conjunto mais abrangente das variáveis inteiras.

Vale salientar que, embora tenhamos escolhido, arbitrariamente, descrever problemas de otimização em versões de maximização, as definições e resultados teóricos apresentados ao longo deste capítulo são claramente válidos também para problemas de minimização.

Definição 2.1.1 *Seja S o conjunto de pontos em $\mathbb{Z}_+^n$ associados às soluções viáveis para o problema* $\max \{cx : S \subset \mathbb{Z}_+^n\}$. *Diz-se que*

$$z = \max \{cx : Ax \leq b,\ x \in \mathbb{Z}_+^n\} \tag{2.1}$$

é uma ***formulação*** *de PLI* ***válida*** *para o problema se* $S = \{x \in \mathbb{Z}^n_+ : Ax \leq b\}$*, onde* $c \in \mathbb{R}^n$*,* $A \in \mathbb{R}^{m \times n}$*,* $b \in \mathbb{R}^m$ *e os escalares* m *e* n *são inteiros positivos.*

Definição 2.1.2 *O problema de maximização*

$$z_R = \max \{z_R(x) : x \in S_R\} \tag{2.2}$$

é uma ***relaxação*** *de (2.1) se* $S \subseteq S_R$ *e* $cx \leq z_R(x)$ *para* $x \in S$*.*

Observe que o valor de qualquer solução viável para um problema de maximização provê um limitante inferior (primal) para o valor de uma solução ótima para o problema. Para alguns problemas de PLI encontrar soluções viáveis é uma tarefa fácil e a questão principal reside em encontrar soluções de qualidade [45]. Em contrapartida, para outros problemas de PLI, encontrar uma solução viável pode ter a mesma complexidade que resolver o próprio problema. Neste caso, identificar alguma solução viável costuma ser uma tarefa árdua mesmo para algoritmos heurísticos sofisticados.

Por outro lado, relaxações são comumente empregadas na obtenção de limitantes superiores (ou duais) para problemas de maximização[1]. Uma das relaxações mais usadas é a relaxação de Programação Linear (PL), chamada de *relaxação linear*, que consiste, essencialmente, na remoção das restrições sobre a integralidade das variáveis. Assim sendo, a relaxação linear do PLI definido em (2.1) seria dada por $z_{LP} = \max \{cx : Ax \leq b, x \in \mathbb{R}^n_+\}$.

Há um número infinito de formulações distintas capazes de descrever um mesmo problema de Otimização Combinatória. Esta constatação sugere a existência de formulações mais adequadas para cada problema que se deseja resolver e pode ser facilmente visualizada com ajuda da Geometria. Note que, em geral, a avaliação da qualidade de uma formulação está relacionada ao valor do limitante dual produzido pela relaxação linear associada ao PLI. Dentro deste contexto, apresentamos a seguir algumas definições de Combinatória Poliédrica que introduzem noções básicas e apresentam elementos relacionados ao conceito de *força* de uma formulação.

Definição 2.1.3 *Um subconjunto de pontos* $\mathcal{P} \subseteq \mathbb{R}^n$ *é dito ser um* ***poliedro*** *se satisfaz um número finito de restrições lineares, ou seja, se* $\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b\}$*. Denota-se por* $\mathcal{P}(A, b)$ *o poliedro definido pela matriz* A *e pelo vetor* b*.*

Definição 2.1.4 $\mathcal{P}$ *é limitado se existe um real* α *tal que todo ponto* $x \in \mathcal{P}$ *satisfaz* $\|x\| \leq \alpha$*. Neste caso,* $\mathcal{P}$ *é chamado de* ***politopo****.*

[1]Para problemas de minimização os termos *limitante primal* e *limitante dual* são usados de forma inversa, i.e., para designar limitantes superiores e inferiores, respectivamente.

Definição 2.1.5 *Um vetor $x \in \mathbb{R}^n$ é* ***combinação linear*** *dos vetores $x_1, \ldots, x_t \in \mathbb{R}^n$ se existe $\lambda = (\lambda_1, \ldots, \lambda_t) \in \mathbb{R}^t$ tal que $x = \sum_{i=1}^{t} \lambda_i x_i$. Se, na combinação linear, for exigido que $\sum_{i=1}^{t} \lambda_i = 1$, diz-se que x é uma* ***combinação afim*** *de $x_1, \ldots, x_t$. Finalmente, se em uma combinação afim for exigido também que $\lambda_i \geq 0, i = 1, \ldots, t$, então x é dito ser* ***combinação*** *linear* ***convexa*** *de $x_1, \ldots, x_t$.*

Definição 2.1.6 *Dado um conjunto $X \subseteq \mathbb{R}^n$, a* ***envoltória convexa*** *dos elementos de X, denotada por $conv(X)$, é definida como sendo o conjunto de todos os vetores que são combinação linear convexa de um número finito de vetores de X.*

Dois resultados relacionados à definição de $conv(X)$ merecem ser destacados. São eles: `(a)` $conv(X)$ é um poliedro e `(b)` os pontos extremos de $conv(X)$ também pertencem ao conjunto X. Estes resultados nos permitem observar a equivalência entre os problemas de PLI = max $\{cx : x \in X\}$ e de PL dado por max $\{cx : x \in conv(X)\}$. Esta última formulação seria *ideal* pois, se resolvermos o problema de programação linear, uma solução ótima estaria em um ponto extremo do poliedro, ou seja, um ponto inteiro. Em teoria, esta seria uma abordagem para solução de PLIs. Todavia, na prática, este método de solução é inaplicável devido ao número usualmente elevado (tipicamente exponencial) de desigualdades necessárias para descrever $conv(X)$ e à dificuldade em caracterizá-las. Uma alternativa seria, portanto, a adoção de uma estratégia que busque obter uma aproximação da descrição de $conv(X)$. Neste sentido, uma possibilidade seria o desenvolvimento de algoritmos de planos-de-corte brevemente descritos na próxima seção.

Outra definição útil é dada a seguir.

Definição 2.1.7 *Seja S um subconjunto do $\mathbb{R}^n$. O* ***posto afim de*** *S ($\rho_A(S)$) é o número máximo de pontos afim independentes que podem ser obtidos em S. A* ***dimensão*** *de S, denotada por $\dim(S)$, é dada por $\dim(S) = \rho_A(S) - 1$.*

2.2 Algoritmos de planos-de-corte

Como auxílio ao entendimento do mecanismo implementado pelos algoritmos para geração de planos-de-corte em programação linear, considere as seguintes definições:

Definição 2.2.1 *Uma* ***desigualdade*** *$\pi x \leq \pi_0$ é dita* ***válida*** *para $\mathcal{P}$ se ela é satisfeita por todos os pontos de $\mathcal{P}$. Analogamente, uma* ***família*** *(ou classe) $\mathcal{F}$ de desigualdades é* ***válida*** *para $\mathcal{P}$ se qualquer desigualdade em $\mathcal{F}$ for válida para $\mathcal{P}$.*

Definição 2.2.2 *Dada uma desigualdade válida $\pi x \leq \pi_0$ para um poliedro $\mathcal{P}$, a* ***face*** *definida por ela neste poliedro é dada por $F(\pi, \pi_0) = \{x \in \mathcal{P} : \pi x = \pi_0\}$. Diz-se que $F(\pi, \pi_0)$ define uma* ***faceta*** *de $\mathcal{P}$ se $\dim(F(\pi, \pi_0)) = \dim(\mathcal{P}) - 1$.*

Definição 2.2.3 *Sejam $\pi x \leq \pi_0$ e $\mu x \leq \mu_0$ duas desigualdades válidas para $\mathcal{P}$. Diz-se que $\pi x \leq \pi_0$* ***domina*** *$\mu x \leq \mu_0$ se existe $u > 0$ tal que $\pi \geq u\mu$, $\pi_0 \leq u\mu_0$ e $(\pi, \pi_0) \neq (u\mu, u\mu_0)$.*

Definição 2.2.4 *Uma desigualdade válida $\pi x \leq \pi_0$ é redundante na descrição de $\mathcal{P}$ se for dominada por uma desigualdade ou se for idêntica ou dominada pela combinação linear de pelo menos duas outras desigualdades válidas para $\mathcal{P}$.*

Definição 2.2.5 *Uma desigualdade que não é dominada por nenhuma outra desigualdade válida para um poliedro $\mathcal{P}$ é dita ser* ***essencial*** *para este poliedro. Neste caso, a desigualdade define uma* ***faceta*** *de $\mathcal{P}$.*

Em síntese, a idéia básica dos algoritmos de planos-de-corte em programação linear inteira é obter, a cada passo, uma solução para a relaxação linear de um problema, $\bar{x}$, e se esta solução não for inteira, tentar eliminá-la (cortá-la) do conjunto de soluções viáveis para o problema original. A eliminação de $\bar{x}$ é feita através da adição à formulação atual do problema de pelo menos uma desigualdade válida, *plano-de-corte* (ou simplesmente *corte*), não satisfeita por $\bar{x}$. Assim procedendo, um novo poliedro é obtido e as soluções viáveis do PLI são preservadas.

Para geração de cortes válidos faz-se uso de algoritmos especializados, as chamadas *rotinas de separação*. Estas rotinas buscam identificar novas desigualdades válidas violadas pela solução fracionária encontrada pela relaxação linear. Em muitos casos esta busca por desigualdades violadas fica restrita a uma família $\mathcal{F}$ de desigualdades válidas. Tal procedimento pressupõe que um *problema de separação* seja resolvido (veja [41, 45] para uma discussão detalhada). Observe que, dada uma família $\mathcal{F}$ de desigualdades válidas, a separação para $\mathcal{F}$ pode ser um problema *fácil* (complexidade polinomial). Neste caso, dispomos de um algoritmo de separação *exato* para $\mathcal{F}$, ou seja, é conhecida uma rotina capaz de encontrar uma desigualdade válida violada por $\bar{x}$, caso ela exista. Em contrapartida, o *problema de separação* para $\mathcal{F}$ pode ser $\mathcal{NP}$*-difícil*. Nesta situação, a rotina de separação é implementada por um algoritmo heurístico, uma vez que o uso efetivo de uma classe de desigualdades $\mathcal{F}$ dentro de um algoritmo de planos-de-corte depende da eficiência da rotina de separação para $\mathcal{F}$.

A menos que $\mathcal{P} = \mathcal{N}P$, não há expectativas de conseguirmos obter uma descrição explícita para $conv(X)$ quando o problema de otimização é $\mathcal{NP}$*-difícil*. Entretanto, não estamos interessados na descrição completa da envoltória convexa de X, mas sim numa boa aproximação para a vizinhança da solução ótima do problema. Assim sendo, a incorporação sistemática de cortes válidos violados proporciona uma aproximação, pelo menos na vizinhança de $\bar{x}$, da descrição poliedral de $conv(X)$.

Como resultado da introdução de planos-de-corte e da conseqüente diminuição da região viável do PL, espera-se, a cada corte incorporado, uma melhora do limitante dual

em relação ao seu precedente. Obviamente, a eficácia do método depende da qualidade e da direção dos cortes identificados. Desta forma, fica clara a relevância da obtenção de desigualdades fortes, preferencialmente que definam facetas, pela(s) rotina(s) de separação em uso. Este fato evidencia a importância da caracterização de famílias de desigualdades que induzam facetas. Ou seja, do emprego de Combinatória Poliédrica como instrumento quase indispensável ao desenvolvimento de algoritmos de planos-de-corte.

Para uma discussão mais aprofundada sobre a teoria subjacente à tecnica de geração de planos de corte recomenda-se a leitura das referências [19], [41] e [45].

A seguir apresentamos uma técnica de decomposição bastante utilizada na solução de PLIs e que é uma das bases do presente trabalho: a relaxação Lagrangiana.

2.3 Dualidade Lagrangiana

A técnica de relaxação Lagrangiana começou a ganhar projeção no início da década de 1970 quando Held e Karp a aplicaram ao problema do Caixeiro Viajante [26, 27]. Logo a seguir, Geoffrion [23] introduziu o termo *Lagrangian Relaxation* e desenvolveu uma teoria geral sobre relaxação Lagrangiana no contexto de Programação Inteira. Desde então, o emprego dessa técnica na geração de limitantes duais (e primais) para resolver diversos problemas de otimização combinatória tem sido amplamente reportado na literatura (veja [4, 8, 11, 20, 24, 45] para uma extensa bibliografia).

Em síntese, a técnica de relaxação Lagrangiana pode ser vista como um método de decomposição que possibilita a exploração de estruturas especiais identificadas a partir de uma formulação do problema a ser resolvido [15]. Na prática, o processo implementado pelo método consiste em: (A) selecionar um subconjunto de restrições difíceis do problema, i.e., que se eliminadas, tornariam o problema resultante de mais fácil solução; (B) associar a cada uma delas uma "penalidade"(multiplicador de Lagrange) e (C) introduzí-las na função objetivo. Este mecanismo é chamado de dualização (Lagrangiana) das restrições é formalizado em detalhes na próxima seção.

Em relação ao item (A) do parágrafo anterior, observe que, apesar de não ser usual para a aplicação de relaxação Lagrangiana clássica, o tamanho deste conjunto é potencialmente muito grande – ou até mesmo exponencial no tamanho da entrada do problema. Entretanto, a dificuldade atrelada à dualização desse número potencialmente exponencial de restrições será momentaneamente desconsiderada. Mais adiante, será visto que este é um tema central da pesquisa que foi conduzida neste trabalho.

2.3.1 Relaxação Lagrangiana

A formulação de Programação Linear Inteira apresentada em (2.1) pode, sem perda de generalidade, ser reescrita da seguinte forma

$$z = \max \{cx : Ax \leq b, x \in \mathbb{X}\} \tag{2.3}$$

onde $\mathbb{X} = \{x \in \mathbb{Z}^n_+ : Dx \leq d\}$, $c \in \mathbb{R}^n$, e as matrizes (A, b) e (D, d) têm dimensões, respectivamente, $m_1 \times (n+1)$ e $m_2 \times (n+1)$, com $m = m_1 + m_2$. Note então que m e n são escalares inteiros positivos que representam, respectivamente, o número de restrições e de variáveis da formulação do problema.

Assuma que o problema acima é $\mathcal{NP}$-*difícil* e que o segundo grupo de restrições, $Dx \leq d$, apresenta uma estrutura especial de tal forma que o programa linear

$$z' = \max \{cx : \ x \in \mathbb{X}\} \tag{2.4}$$

resultante da remoção das restrições $Ax \leq b$ de (2.3) seja um problema fácil, isto é, um problema para o qual é conhecido um algoritmo *eficiente* que o resolve. Neste contexto, entende-se como sendo *eficiente* um algoritmo que resolve o problema em tempo polinomial (ou pseudo-polinomial) no tamanho da sua entrada. Assim sendo, as restrições $Ax \leq b$ são ditas *complicadoras* e o problema descrito por (2.4) é, trivialmente, uma relaxação para (2.3).

Uma vez identificadas tais restrições, $\{a_i x \leq b_i : i = 1, 2, \ldots, m_1\}$, a cada uma delas é associado um multiplicador de Lagrange $\lambda_i \in \mathbb{R}_+$ e cada termo resultante desta associação é introduzido na função objetivo. Assim procedendo, obtém-se

$$z(\lambda) = \max \{cx + \lambda(b - Ax) : \ x \in \mathbb{X}\} = \max \{(c - \lambda A)x + \lambda b : \ x \in \mathbb{X}\} \tag{2.5}$$

Este problema, denotado por PL(λ), recebe o nome de *Problema Lagrangiano Relaxado* ou *Subproblema Lagrangiano* de (2.3) relativo a $Ax \leq b$ e com parâmetro $\lambda \in \mathbb{R}^{m_1}_+$. Note que, sendo $\lambda \geq 0$, violações de $Ax \leq b$ fazem com que o termo $\lambda(b - Ax)$, relativo às penalidades, seja negativo. Assim, intuitivamente, $Ax \leq b$ será satisfeita se λ for adequadamente grande.

Considerando-se a definição 2.1.2, é fácil concluir que o PL(λ) obtido em (2.5) é uma relaxação do problema (2.3). Para tal, é bastante verificar que:

1. $S_{(2.5)} \supseteq S_{(2.3)}$, onde $S_{(2.5)}$ e $S_{(2.3)}$ denotam, respectivamente, os conjuntos de soluções viáveis de (2.5) e de (2.3), e

2. Para toda solução $x' \in S_{(2.3)}$ e $\lambda \geq 0$, o termo extra Lagrangiano $\lambda(b - Ax)$ adicionado à função objetivo em (2.5) é não negativo. Portanto, $cx' + \lambda(b - Ax') \geq cx'$.

Observe que, em particular, PL(0) é equivalente ao problema descrito por (2.4). Contudo, admitindo-se $\lambda \geq 0$, podemos afirmar que a relaxação Lagrangiana relativa às restrições $Ax \leq b$ é mais forte que aquela dada por (2.4). Ainda, dado que $z(\lambda)$ é um limitante dual (superior) válido para (2.3) tem-se que a relação $z' \geq z(\lambda) \geq z$ é verificada. Entretanto, o valor de $z(\lambda)$ pode ser tão ruim que não justifique sua utilização na prática. Na verdade o que se deseja é encontrar valores para os multiplicadores de Lagrange que produzam o melhor (menor) limitante superior possível. Assim, tem-se como objetivo resolver o chamado *Problema Dual Lagrangiano* (PDL) dado por:

$$z_D = \min_{\lambda \in \mathbb{R}^{m_1}_+} \{z(\lambda)\} = \min_{\lambda \in \mathbb{R}^{m_1}_+} \{\max \{cx + \lambda(b - Ax) : x \in \mathbb{X}\}\} \tag{2.6}$$

Quando as restrições a serem dualizadas são restrições de igualdade, os multiplicadores de Lagrange correspondentes são irrestritos, isto é, os elementos de λ podem assumir qualquer valor real. Neste caso, o Problema Dual Lagrangiano torna-se $z_D = \min_{\lambda \in \mathbb{R}^m} \{z(\lambda)\}$. Uma consideração interessante a ser levantada sob esta circunstância diz respeito à relação entre soluções viáveis para (2.3) e soluções ótimas para o PDL. Assim, seja x^* uma solução para PL(λ) onde todas as restrições dualizadas são restrições de igualdade e suponha que x^* é viável para o problema original descrito em (2.3), ou seja, $Ax^* = b$. Desta forma, temos $cx^* + \lambda(b - Ax^*) = cx^*$ e, portanto, x^* é também ótima para (2.3).

O mecanismo de dualização proposto pela técnica de relaxação Lagrangiana pressupõe a análise de várias questões. Dentre elas, duas questões merecem ser destacadas [8]:

1. que conjunto(s) de restrições dualizar? (questão estratégica)

2. como determinar valores para os multiplicadores? Ou seja, como resolver o Problema Dual Lagrangiano? (questão tática)

Estas questões estão intimamente relacionadas a dois aspectos importantes: (A) à qualidade dos limitantes duais a serem obtidos resolvendo-se o PDL e (B) à facilidade - em termos de complexidade computacional - de resolução de cada PL(λ) e do Problema Dual Lagrangiano. Ao longo das próximas seções sobre relaxação Lagrangiana, considerações teóricas e computacionais são discutidas com o intuito de ampliar a compreensão de cada um destes aspectos. Em particular, a próxima seção faz uma breve revisão de propriedades e discute alguns aspectos teóricos relevantes ao entendimento da estrutura do Problema Dual Lagrangiano.

2.3.2 O Problema Dual Lagrangiano em detalhes

Denotemos por $\overline{P}$ a relaxação linear de (2.3) definida como

$$z_{\overline{P}} = \max \{cx : Ax \leq b, x \in \overline{\mathbb{X}}\} \tag{2.7}$$

onde $\overline{\mathbb{X}} = \{x \in \mathbb{R}^n_+ : Dx \leq d\}$, b, c e d são vetores e A e B são matrizes de dimensões convenientes em conformidade com aquelas especificadas na descrição de (2.3). O resultado a seguir nos diz quão forte é o limitante gerado pelo Problema Dual Lagrangiano e serve como embasamento ao entendimento da relação estabelecida entre PDL e $\overline{P}$.

Teorema 2.3.1 $z_D = \max\{cx : Ax \leq b, x \in conv(\mathbb{X})\}$.

Prova: *dado que o conjunto* $\mathbb{X} \subseteq \mathbb{Z}^n_+$ *é finito mas potencialmente contém um número muito grande de pontos* $\{x^1, x^2, \ldots, x^T\}$, *então*

$$\begin{aligned}
z_D &= \min_{\lambda \in \mathbb{R}^{m_1}_+} \{z(\lambda)\} \\
&= \min_{\lambda \in \mathbb{R}^{m_1}_+} \{\max\ [cx + \lambda(b - Ax) :\ x \in \mathbb{X}]\} \\
&= \min_{\lambda \in \mathbb{R}^{m_1}_+} \left\{\max\ \left[cx^t + \lambda(b - Ax^t),\ t = 1, \ldots, T\right]\right\} \\
&= \left\{ \begin{array}{c} \min \eta \\ \eta \geq cx^t + \lambda(b - Ax^t), t = 1, \ldots, T \\ \lambda \in \mathbb{R}^{m_1}_+,\ \eta \in \mathbb{R}^1 \end{array} \right. \qquad (2.8)
\end{aligned}$$

Onde a nova variável η *introduzida representa um limitante superior para* $z(\lambda)$ *e o problema resultante, (2.8), é um problema de programação linear com variáveis* $(\eta, \lambda) \in \mathbb{R}^1 \times \mathbb{R}^{m_1}_+$. *Recorrendo-se à dualidade em programação linear, o problema dual de (2.8) pode ser escrito como:*

$$\begin{aligned}
z_D \quad = \quad & \max \sum_{t=1}^{T} \mu_t \left(cx^t\right) \\
& s.a. \quad \sum_{t=1}^{T} \mu_t \left(Ax^t - b\right) \leq 0 \\
& \sum_{t=1}^{T} \mu_t = 1 \\
& \mu \in \mathbb{R}^T_+
\end{aligned}$$

Logo, a prova termina fazendo-se $x = \sum_{t=1}^{T} \mu_t x^t$, *com* $\sum_{t=1}^{T} \mu_t = 1$ *e* $\mu \in \mathbb{R}^T_+$. □

Em outras palavras, este teorema diz que resolver PDL (à otimalidade) é equivalente a resolver o problema de programação linear $P' = \max\{cx : Ax \leq b, x \in conv(\mathbb{X})\}$. Este resultado teórico é também detalhado nas referências [41, 45]. Observe que $\mathbb{X} \subseteq \overline{\mathbb{X}}$, logo $conv(\mathbb{X}) \subseteq conv(\overline{\mathbb{X}}) = \overline{\mathbb{X}}$. Por conseguinte, $\overline{P}$ é claramente uma relaxação de P' e, como conseqüência do teorema acima, podemos concluir que:

Corolário 2.3.1 $z_D \leq z_{\overline{P}}$.

Em particular, se $conv(\mathbb{X}) \subsetneq \overline{\mathbb{X}}$, temos que, em teoria, $z_D < z_{\overline{P}}$. Em contrapartida, se $conv(\mathbb{X}) = \overline{\mathbb{X}}$, PDL é dito satisfazer à *Propriedade de Integralidade*. Isto é equivalente a dizer que solução de PL(λ), para qualquer valor de λ, fica inalterada quando as restrições de integralidade são relaxadas. Ou seja, neste caso, os pontos extremos de $\overline{\mathbb{X}}$ são inteiros e, em decorrência do teorema acima, temos que:

Corolário 2.3.2 *Se PDL satisfaz à Propriedade de Integralidade, então* $z_D = z_{\overline{P}}$.

Note que o problema (2.8) consiste em um programa linear com o número de restrições, T, tipicamente muito alto (cada restrição corresponde a uma solução viável em $\mathbb{X}$). Assim sendo, os resultados decorrentes do teorema 2.3.1 e o corolário acima nos dizem que encontramos uma forma de resolver um programa linear com, possivelmente, um número imenso de restrições, sem tratá-lo diretamente.

Estes resultados teóricos informam precisamente a qualidade potencial do limitante que pode ser produzido pela solução do Problema Dual Lagrangiano. Embora não determinantes, tais resultados servem como um subsídio relevante à escolha do conjunto de restrições a dualizar. Quando considerados isoladamente, eles nos levam à conclusão de que, em geral, o emprego de relaxação Lagrangiana é mais promissor quando a Propriedade de Integralidade não é satisfeita. Todavia, a dificuldade de solução dos subproblemas Lagrangianos e do PDL constituem outro aspecto igualmente importante que pode ser decisivo na escolha da relaxação Lagrangiana a ser adotada.

A demonstração do teorema 2.3.1 nos permite entender um pouco mais sobre a estrutura do Problema Dual Lagrangiano e como ele pode ser resolvido. Dentre outras, ela mostra que o PDL foi convexificado, tornando-se um programa linear. Esta característica pode ser facilmente verificada com auxílio dos resultados que se seguem.

Definição 2.3.1 *Seja* $\mathbb{C}$ *um subconjunto de* $\mathbb{R}^n$. *Diz-se que* $\mathbb{C}$ *é um* ***conjunto convexo*** *se* $\alpha x^1 + (1-\alpha)\, x^2 \in \mathbb{C}$ *para todo* x^1, $x^2 \in \mathbb{C}$ *e* $\alpha \in [0,1]$.

Definição 2.3.2 *Seja* $\mathbb{C}$ *um subconjunto convexo de* $\mathbb{R}^n$. *A* ***função*** $f : \mathbb{C} \to \mathbb{R}$ *é dita* ***convexa*** *se e somente se*

$$f\left(\alpha x^1 + (1-\alpha)\, x^2\right) \leq \alpha f\left(x^1\right) + (1-\alpha)\, f\left(x^2\right) \tag{2.9}$$

para todo x^1, $x^2 \in \mathbb{C}$ *e* $\alpha \in [0,1]$.

Com base nas definições acima podemos provar o seguinte teorema:

Teorema 2.3.2 *A **função** que descreve o problema **dual**, $z : \mathbb{R}^{m_1} \to \mathbb{R}$, definida por*

$$z(\lambda) = \max \{cx + \lambda(b - Ax) : \ x \in \mathbb{X}\} \tag{2.10}$$

*é **convexa**.*

Prova: *Sejam λ_1, $\lambda_2 \in \mathbb{R}^{m_1}$ e $\alpha \in [0, 1]$. Então*

$$\begin{aligned} z(\lambda_1) &= \max \{cx + \lambda_1(b - Ax) : \ x \in \mathbb{X}\} \ e \\ z(\lambda_2) &= \max \{cx + \lambda_2(b - Ax) : \ x \in \mathbb{X}\} \end{aligned}$$

Para $\bar{\lambda} = \alpha\lambda_1 + (1 - \alpha)\,\lambda_2$ temos que

$$\begin{aligned} z(\bar{\lambda}) &= \max \{cx + \bar{\lambda}(b - Ax) : \ x \in \mathbb{X}\} \\ &= c\bar{x} + \bar{\lambda}(b - A\bar{x}) : \ \bar{x} \in \mathbb{X} \\ &= \alpha \underbrace{[c\bar{x} + \lambda_1(b - A\bar{x})]}_{\leq z(\lambda_1)} + (1 - \alpha) \underbrace{[c\bar{x} + \lambda_2(b - A\bar{x})]}_{\leq z(\lambda_2)} \\ &\leq \alpha z(\lambda_1) + (1 - \alpha)\, z(\lambda_2) \\ z(\alpha\lambda_1 + (1 - \alpha)\,\lambda_2) &\leq \alpha z(\lambda_1) + (1 - \alpha)\, z(\lambda_2) \qquad \square \end{aligned}$$

Observe também que a estrutura da função dual é descrita como o máximo de um conjunto de funções lineares, i.e, pela expressão max $[cx^t + \lambda(b - Ax^t),\ t = 1, \ldots, T]$. Portanto, a função que descreve o PDL é claramente linear por partes e convexa. Desta forma, o Problema Dual Lagrangiano pode ser visto como o problema de minimização de uma função, z, convexa linear por partes e não diferenciável. Este fato motiva o emprego de métodos baseados em subgradiente para resolver o PDL.

2.3.3 Otimização por Subgradiente

Através da demonstração do teorema 2.3.1 vimos que PDL pode ser formulado como um problema de programação linear e que o montante de restrições resultantes da formulação corresponde ao número de elementos em $\mathbb{X}$. Como, em geral, o número de pontos viáveis em $\mathbb{X}$ é muito grande, na prática, o emprego de programação linear é quase proibitivo para resolver o problema.

Neste sentido, diversas abordagens são mencionados na literatura como alternativas para resolver o PDL: métodos de otimização por subgradiente, tais como o Método do Subgradiente [45, 25], Métodos de *Bundle*[28] e o Algoritmo do Volume [7]; métodos baseados em Ajuste de Multiplicadores [8] e os chamados Métodos Ascendentes [8]. Dentre

estas alternativas, os métodos de otimização por subgradiente se destacam pelo grande número de trabalhos reportados na literatura que os utilizam (veja [8]). Isto se deve, muito provavelmente, pelo bom desempenho que as implementações de métodos de subgradiente comumente apresentam e pela facilidade de implementação destes algoritmos.

Métodos de otimização por subgradiente são adaptações do método de gradiente [25] e têm origem em otimização não-linear e não-diferenciável. Tais métodos foram projetados para resolver problemas de minimização (maximização) de funções convexas (côncavas) e lineares por parte.

Em síntese, métodos de subgradiente são procedimentos iterativos que buscam, a cada iteração, caminhar numa direção promissora de acordo com a função de otimização considerada. A direção de cada passo tem um papel crucial no desempenho do algoritmo, sendo determinada com o auxílio de um vetor de subgradiente (definido a seguir) computado a cada iteração do método.

Definição 2.3.3 *Seja $f : \mathbb{R}^{m_1} \to \mathbb{R}$ uma função convexa. O vetor $s \in \mathbb{R}^{m_1}$ é dito* ***subgradiente*** *de f no ponto $\bar{x} \in \mathbb{R}^{m_1}$ se $f(x) \geq f(\bar{x}) + s(x - \bar{x})$, para todo $x \in \mathbb{R}^{m_1}$.*

A partir desta definição, o resultado abaixo mostra como explorar a estrutura convexa do Problema Dual Lagrangiano no cálculo do vetor subgradiente. Para sua compreensão, considere a função dual z descrita em (2.10). Além disso, denote por $\mathbb{X}^*(\bar{\lambda})$ o conjunto de soluções ótimas de $z(\bar{\lambda}) = \max\{cx + \bar{\lambda}(b - Ax) : x \in \mathbb{X}\}$, onde $\bar{\lambda} \in \mathbb{R}^{m_1}_+$. Temos, portanto, que $\mathbb{X}^*(\bar{\lambda}) = \{\bar{x} \in \mathbb{X} : z(\bar{\lambda}) = c\bar{x} + \bar{\lambda}(b - A\bar{x})\}$.

Teorema 2.3.3 *Seja a função dual $z : \mathbb{R}^{m_1} \to \mathbb{R}$. O vetor $s(\bar{x}) = b - A\bar{x}$ é um subgradiente de z no ponto $\bar{\lambda}$, onde $\bar{x} \in \mathbb{X}^*(\bar{\lambda})$.*

Prova: *Para $\bar{\lambda} \in \mathbb{R}^{m_1}_+$, $\bar{x} \in \mathbb{X}^*(\bar{\lambda})$ significa que $\bar{x}$ resolve $z(\bar{\lambda})$ e, portanto,*

$$z(\bar{\lambda}) = c\bar{x} + \bar{\lambda}(b - A\bar{x}),$$

E, para $\lambda \in \mathbb{R}^{m_1}$, temos que

$$z(\lambda) = \max\{cx + \lambda(b - Ax) : x \in \mathbb{X}\} \geq c\bar{x} + \lambda(b - A\bar{x}).$$

Portanto, subtraindo a primeira expressão desta última obtemos

$$\begin{aligned} z(\lambda) - z(\bar{\lambda}) &\geq c\bar{x} + \lambda(b - A\bar{x}) - c\bar{x} + \bar{\lambda}(b - A\bar{x}) \\ z(\lambda) &\geq z(\bar{\lambda}) + (b - A\bar{x})(\lambda - \bar{\lambda}), \ \forall \lambda \in \mathbb{R}^{m_1} \quad \square \end{aligned}$$

Em resumo, o teorema demonstrado nos mostra que podemos obter subgradientes com o auxílio de soluções ótimas dos subproblemas Lagrangianos.

Em particular, esta forma de cálculo do subgradiente é implementada a cada iteração do Método do Subgradiente (MS). Este método foi utilizado nas implementações realizadas ao longo desta tese e, por este motivo, é brevemente descrito a seguir.

2.3.4 O Método do Subgradiente

O Método do Subgradiente (MS) tem sido amplamente utilizado (veja [8]). Isto se deve, primordialmente, à simplicidade/facilidade de implementação e à boa qualidade, em geral, dos limitantes por ele produzidos.

MS é um procedimento iterativo que, a cada passo, busca resolver um Problema Lagrangiano Relaxado. Basicamente, partindo-se de um vetor inicial viável de multiplicadores, λ^0, a cada iteração k, novos valores para os elementos do vetor são determinados (λ^k), produzindo um novo Problema Lagrangiano Relaxado. Este novo PLR produzido é resolvido, gerando um novo limitante dual para o problema original em questão.

Seja $\bar{x}^k$ uma solução ótima para o PLR da iteração k cujo valor é $z(\lambda^k)$. Um vetor de subgradientes, $s_i^k, i = 1, 2, \ldots, m_1$, associados às m_1 restrições relaxadas para a solução corrente $\bar{x}^k$ é calculado da seguinte forma:

$$s_i^k = (b_i - a_i \bar{x}^k),\ i = 1, 2, \ldots, m_1 \tag{2.11}$$

A atualização dos multiplicadores de Lagrange, a cada iteração, é feita com base: `(1)` nos valores dos multiplicadores na iteração anterior, `(2)` no vetor de subgradientes e `(3)` no valor de um escalar positivo, θ^k, chamado de tamanho do passo (um passo aqui é interpretado como o deslocamento a partir do ponto/solução corrente na direção oposta à do subgradiente). O cálculo de θ^k é feito com base na fórmula (ver [8])

$$\theta^k = \frac{\alpha^k (z(\lambda^k) - \underline{z})}{\sum_{i=1}^{m_1} (s_i^k)^2} \tag{2.12}$$

onde $\alpha^k \in (0, 2]$ é um escalar cujo valor é atualizado ao longo da execução do algoritmo e $\underline{z}$ corresponde ao valor de um limitante primal (ou inferior, para problemas de maximização) conhecido para (2.3) na iteração k.

Uma vez calculado o tamanho do passo, a atualização do vetor de multiplicadores de Lagrange, λ, é dada por

$$\lambda_i^{k+1} = \max \{0; \lambda_i^k - \theta^k s_i^k\},\ i = 1, 2, \ldots, m_1 \tag{2.13}$$

A compilação dos cálculos correspondentes às expressões (2.11)–(2.13) nos permite compor a seguinte descrição de cada iteração típica do MS:

Método do Subgradiente (k-ésima iteração)
1. Resolva PL(λ^k) e obtenha uma solução ótima $\bar{x}^k$ cujo valor é $z(\lambda^k)$;
2. Calcule os valores do vetor de subgradientes, $s_i^k \leftarrow b_i - a_i\bar{x}^k,\ i = 1, 2, \ldots, m_1$;
3. Determine o valor do tamanho do passo, $\theta^k \leftarrow \frac{\alpha^k(z(\lambda^k)-\underline{z})}{\sum_{i=1}^{m_1}(s_i^k)^2}$, onde $\alpha^k \in (0, 2]$ e $\underline{z}$ é o valor disponível de uma solução viável para o problema original;
4. Atualize o vetor de multiplicadores de Lagrange referentes às restrições dualizadas, i.e., faça $\lambda_i^{k+1} = \max\{0; \lambda_i^k - \theta^k s_i^k\},\ i = 1, 2, \ldots, m_1$
5. Faça $k \leftarrow k + 1$;

Vale ressaltar que, apesar dos inúmeros relatos de bom desempenho do Método do Subgradiente, na prática, é possível nos defrontarmos com problemas indesejáveis de convergência devido à seqüência de multiplicadores gerados ao longo da execução do algoritmo. Uma forma simples de identificarmos potenciais problemas desta natureza é nos atermos à fórmula de cálculo do tamanho do passo (2.12). Observe que a medida que $z(\lambda^k)$ se aproxima de $\underline{z}$, o tamanho do passo pode ficar muito pequeno. Neste caso, uma alternativa comumente utilizada é substituir o numerador de (2.12) por $\alpha^k\left[(1+\epsilon)z(\lambda^k) - \underline{z}\right]$, onde ϵ é um valor positivo, tipicamente não maior que 0.05 [8, 15]. Também, note que a expressão de atualização do vetor λ está coerente com a noção intuitiva de que a determinação de valores adequadamente grandes dos multiplicadores de Lagrange deve, provavelmente, levar à satisfação das restrições dualizadas. Entretanto, observe que quando o número de restrições dualizadas é muito grande, o tamanho do passo expresso em (2.12) pode ficar desprezível. Conseqüentemente, problemas de convergência podem também ocorrer devido ao uso direto das fórmulas dadas pelas expressões (2.11)–(2.13). Uma possível alternativa é modificar o MS conforme o esquema proposto pelos algoritmos *relax-and-cut* discutidos a seguir.

2.4 Algoritmos *Relax-and-Cut* (R&C)

O termo *relax-and-cut* tem sido usualmente utilizado na literatura para denotar uma larga classe de algoritmos baseados em relaxação Lagrangiana, onde desigualdades[2] são dualizadas somente quando violadas pela solução corrente do Problema Lagrangiano Relaxado [9, 12, 24, 32, 34, 35, 39, 40]. Vale ressaltar que o termo *relax-and-cut* foi mencionado originalmente em um artigo de Escudero *et al.* [18]. Entretanto, neste artigo, como em [30], os algoritmos de relaxação Lagrangiana propostos são empregados de uma forma distinta daquela implementada nas citações anteriores. A distinção entre estas duas abordagens é explicitada por Lucena em [35].

[2]Ao longo deste texto os termos *restrição* e *desigualdade* são usados como sinônimos.

Devido ao esquema dinâmico de dualização proposto, algoritmos *relax-and-cut* aparecem como uma alternativa que pode viabilizar o uso da relaxação Lagrangiana para resolver problemas onde há um número muito grande (potencialmente exponencial) de restrições a serem dualizadas. Tentativas neste sentido datam do início da década de 1980 [2]. Todavia, a formalização de idéais propondo um esquema seletivo de desigualdades a serem dualizadas só surgiu com os trabalhos desenvolvidos por Lucena [32, 33] para resolver o *Problema de Steiner em Grafos*.

2.4.1 Estratégias de implementação

Essencialmente duas estratégias de implementação de algoritmos R&C são discutidas na literatura. Elas diferem, basicamente, no momento em que as desigualdades são identificadas e dualizadas. Em uma delas, a cada passo do algoritmo, várias execuções do Método do Subgradiente são realizadas. A cada execução do método, desigualdades válidas que violam a solução do PDL atual são identificadas e armazenadas. Ou seja, a identificação destas desigualdades é postergada até que o PDL atual seja considerado resolvido e a incorporação destas desigualdades à formulação atual só ocorre no início da execução subseqüente do MS. A outra abordagem corresponde ao algoritmo descrito por Lucena em [32, 33]. Nesta, uma única execução do MS é realizada e, a cada iteração do método, busca-se identificar desigualdades válidas que violam a solução do PL(λ) atual, as quais são incorporadas à formulação na iteração seguinte (detalhes em [12, 24, 32, 35, 40]).

Para ambas as abordagens, os processos se repetem até que algum critério de parada seja satisfeito. Note também que, independente da estratégia adotada, três possíveis mudanças podem ocorrer na formulação atual com a descoberta de novas restrições: `(i)` alterações na função objetivo devido à dualização das novas desigualdade encontradas `(ii)` modificação da estrutura do(s) novo(s) Subproblema(s) Lagrangiano(s), quando as desigualdades identificadas são mantidas na formulação ou `(iii)` ambas, quando apenas um subconjunto próprio das novas desigualdades é dualizado. Desta forma, independente da mudança sofrida, um novo PDL a ser resolvido é formulado. Com o objetivo de enfatizar as diferenças entre as duas estratégias, Lucena [35] denominou a primeira abordagem de *Delayed Relax-and-Cut* (DR&C). Em contrapartida, a abordagem por ele adotada em [32, 33] foi denominada *Non-Delayed Relax-and-Cut* (NDR&C). Vale observar que no Capítulo 4 propomos uma terceira forma alternativa de utilização de planos de corte combinados com algoritmos Lagrangianos.

2.4.2 Modificações no Método do Subgradiente

Nesta tese, independente da estratégia de implementação adotada, trabalhamos com uma adaptação do Método do Subgradiente descrita em [24]. Para fins de exposição das mu-

danças efetuadas no MS, considere que as restrições $\{a_i x \leq b_i : i = 1, 2, \ldots, m_1\}$ candidatas à dualização (em um esquema Lagrangiano) são aquelas explicitadas em (2.5).

Assumindo-se que m_1 pode ser muito grande, fica evidente o elevado esforço computacional requerido, a cada iteração do Método do Subgradiente, para a dualização de todas estas restrições. Afora esta questão, como já comentado, problemas de convergência podem ocorrer no MS quando se tem um número muito grande de componentes/entradas não nulas no vetor de subgradiente. Nesta situação, o uso explícito dos subgradientes no cálculo de θ provocaria mudanças apenas marginais no vetor de multiplicadores de Lagrange entre iterações consecutivas do método.

Como alternativa para contornar estas dificuldades, o R&C propõe uma estratégia dinâmica de uso das restrições candidatas à dualização nos cálculos feitos pelo MS: ao invés de todas as restrições serem explicitamente dualizadas, uma estratégia de seleção é estabelecida para escolha daquelas a serem efetivamente dualizadas a cada iteração.

A estratégia adotada é baseada na classificação das desigualdades candidatas à dualização, a cada iteração do MS, de acordo com os três grupos seguintes: o primeiro (grupo 1) é formado pelas desigualdades violadas por $\bar{x}^k$, ou seja, desigualdades tais que $s_i^k = (b_i - a_i \bar{x}^k) > 0,\ \forall i \leq m_1$; o segundo (grupo 2) contém as desigualdades às quais estão correntemente associados multiplicadores de Lagrange com valores não nulos e, finalmente, o terceiro grupo (grupo 3) é composto pelas demais desigualdades. Um ponto interessante a ressaltar nesta divisão é que pode haver interseção entre os grupos 1 e 2 de desigualdades.

A partir desta classificação é pertinente observar que multiplicadores de Lagrange associados às restrições pertencentes, na iteração corrente do MS, ao grupo 3, continuam com valores nulos ao final da iteração (uma simples análise da expressão (2.13) é suficiente para se chegar a esta conclusão). Note também que, diferentemente das restrições pertencentes aos grupos 1 e 2, restrições do grupo 3 não podem contribuir para os custos Lagrangianos na iteração corrente pois, para restrições deste grupo, o termo $\lambda_i(b_i - a_i x)$ é sempre nulo. Por estes motivos, restrições dos grupos 1 e 2 recebem a denominação de *desigualdades ativas*, enquanto restrições do grupo 3 são denominadas *desigualdades inativas*. Cabe notar também que, de uma iteração a outra do MS, uma desigualdade pode passar de ativa a inativa e vice-versa.

Estabelecida essa classificação, algoritmos *relax-and-cut* sugerem, a cada iteração do MS, proceder a dualização efetiva apenas das desigualdades ativas. Assim, somente as restrições violadas pela solução do PLR corrente ou aquelas que tenham multiplicadores de Lagrange associados não nulos são consideradas nos cálculos efetuados dentro do MS.

É importante mencionar que, no caso de relaxação Lagrangiana clássica, Beasley [8] observou que quando, simultaneamente, $\lambda_i = 0$ e $s_i^k > 0$, o multiplicador λ_i permanece com valor nulo após sua atualização. Entretanto, o valor de $(s_i^k)^2$ seria considerado na

expressão de cálculo do tamanho do passo. Assim sendo, neste caso, Beasley sugere fixar o subgradiente s_i^k em zero antes do cálculo de θ. Na prática, esta sugestão equivale à estratégia adotada nos algoritmos R&C que consiste em ignorar a dualização de restrições do grupo 3, i.e., de desigualdades inativas.

Observe que, a atualização dos multiplicadores de Lagrange com base apenas em desigualdades ativas, possibilita que apenas uma diminuta parte do total de desigualdades sejam explicitamente consideradas a cada iteração. Entretanto, vale a ressalva de que o emprego do esquema descrito acima não impede que o número de restrições ativas a cada iteração seja muito grande. Em trabalhos recentes, A. Lucena [34] e da Cunha [15] discutem esta questão e apresentam sugestões para lidar com esta situação em particular.

2.4.3 Planos de Corte e *Relax-and-Cut*

A abordagem implementada pelos algoritmos *relax-and-cut* pode ser uma alternativa adequada a problemas de otimização combinatória, independente de se dispor de uma formulação para o problema com um número muito grande de restrições candidatas a dualização. Uma situação típica desta adequação é quando novas restrições válidas para o problema (potencialmente em número exponencial) podem ser descobertas dinamicamente, de forma análoga ao que acontece no emprego de algoritmos para geração de planos de corte em programação linear.

De forma idêntica ao esquema proposto por algoritmos para geração de planos de corte, *rotinas de separação* podem ser desenvolvidas e incorporadas ao método usado para resolver a relaxação Lagrangiana (aqui, o MS). Assim, elas podem ser utilizadas na obtenção, a cada iteração, de novas restrições válidas violadas pela solução do PLR corrente. Estas novas restrições podem, então, ser dualizadas ou mantidas no novo PLR caso a permanência delas não o tornem muito mais difícil de ser resolvido.

Também como os algoritmos de planos de corte, algoritmos *relax-and-cut* podem ser utilizados para melhorar os limitantes duais de problemas de otimização combinatória e, eventualmente, conseguir provar a otimalidade de uma solução. Obviamente, a eficácia do método é bastante dependente da qualidade dos cortes encontrados e dos algoritmos (heurísticas) utilizados para obtenção de soluções viáveis.

Para ver que desigualdades encontradas por um algoritmo *relax-and-cut* podem, na prática, melhorar o limitante obtido, basta resgatarmos a interpretação do resultado demonstrado no teorema 2.3.1. De acordo com este teorema, resolver o Problema Dual Lagrangiano (à otimalidade) dado por $z_{D_{PDL}} = \min_{\lambda \in \mathbb{R}_+^{m_1}} \{\max [cx + \lambda(b - Ax) : x \in \mathbb{X}]\}$ é equivalente a resolver o problema de programação linear $z_{D_{LP}} = \max \{cx : Ax \leq b,\ x \in conv(\mathbb{X})\}$. Ou seja, em teoria, ambos apresentam o mesmo valor de solução ótima, $z_{D_{PDL}} = z_{D_{LP}} = z_D$, o qual é um limitante dual para o PLI original (2.3).

Agora, note que a adição de cortes à formulação significa aumentar o número de linhas da matriz A. Assim, um novo PL dado por $z'_{D_{LP}} = \max \{cx : A'x \leq b',\ x \in conv(\mathbb{X})\}$ é gerado, onde a $A' = \left[\frac{A}{\pi}\right]$ é a matriz de cortes inseridos em A, $b' = \left[\frac{b}{\pi_0}\right]$ e $\pi x \leq \pi_0$. Desta forma, sendo $z'_{D_{PDL}}$ o valor do PDL equivalente ao PL com a adição dos cortes $\pi x \leq \pi_0$, tem-se também que $z'_{D_{PDL}} = z'_{D_{LP}} = z'_D$. Portanto, em teoria, pode-se afirmar que $z'_D \leq z_D$. Em outras palavras, o novo limitante obtido com a adição dos cortes é pelo menos tão bom quanto o anterior. Na prática, este limitante pode não ser alcançado ou requerer muito tempo para ser obtido. Entretanto, espera-se que a inclusão dos cortes consiga melhorar o limitante, ou seja, que, embora não garantido, a solução ótima de valor z_D seja efetivamente cortada. Observe também que, junto com a melhora dos limitantes duais, os cortes adicionados podem ajudar a acelerar a convergência do algoritmo.

Vale ressaltar que mesmo com a adição de desigualdades é possível que, ao final da execução do Método do Subgradiente, o problema ainda não tenha sido resolvido. Mas, partindo-se da suposição de que as desigualdades geradas pelo *relax-and-cut* são boas, estas podem ser adicionadas à formulação original e passadas a um resolvedor baseado em PLI. O que se propõe aqui é um algoritmo híbrido que usa *relax-and-cut* como uma forma de pré-processamento para um resolvedor de PLI. Este resolvedor, portanto, usaria a relaxação Lagrangiana também para produzir, com baixo custo computacional, cortes que o auxiliariam a resolver instâncias que, de outro modo, não poderiam ser resolvidas.

De maneira diferente ao que acontece com algoritmos de planos de corte que usam relaxações lineares, a redução do custo computacional na geração de cortes em um algoritmo *relax-and-cut* é possível em alguns casos, pois os algoritmos de separação, neste contexto, precisam cortar soluções inteiras. Em [40], por exemplo, os autores mostram que a separação de uma desigualdade para o *Vehicle Routing Problem* é um problema difícil no caso geral, mas é polinomial no caso de soluções 0-1 puras.

2.5 Métodos Enumerativos

Para a maioria dos problemas de otimização discreta é inviável o emprego de procedimentos que enumeram explicitamente todas as soluções válidas para o problema em questão. Assim sendo, métodos de enumeração implícita (ou *branch-and-bound*) surgem como uma alternativa muito mais efetiva para resolução de problemas de natureza combinatória. Além destes, algoritmos *branch-and-cut*, que embutem nos algoritmos *branch-and-bound* mecanismos propostos por algoritmos de planos de corte, aparecem como uma outra estratégia disponível em alternativa ao uso de enumeração explícita. A seguir, as idéias básicas subjacentes a ambas as abordagens são brevemente descritas.

2.5.1 Algoritmos *Branch-and-Bound* (B&B)

Algoritmos B&B procuram decompor o conjunto (espaço) de soluções viáveis em subconjuntos disjuntos e sucessivamente menores (ramificação ou *branching*). Para cada subconjunto, limitantes (duais) para o valor da função objetivo relativos a cada subproblema são calculados e usados para descartar a análise de certos subconjuntos de soluções (poda ou *bounding*). Os limitantes são obtidos substituindo-se cada subproblema gerado por uma relaxação sua, de mais fácil solução, tal que o valor ótimo da relaxação produz um limitante para o subproblema. O procedimento termina quando, para cada subproblema, uma solução viável é produzida ou é constatado que nenhuma solução mais promissora do que a melhor atualmente disponível pode ser gerada. A melhor solução encontrada ao longo da execução do algoritmo é, portanto, um ótimo global para o problema.

Sendo P um problema de otimização discreta e $v(P)$ o valor de uma solução ótima para P, Balas e Toth [5] elencam os seguintes ingredientes essenciais para qualquer algoritmo B&B para P: `(i)` uma relaxação de P que, no contexto de Programação Inteira, tipicamente, recai na opção mais natural representada pela relaxação linear; `(ii)` uma regra de ramificação, ou seja, uma forma de decomposição do conjunto viável S_i para o subproblema atual P_i em subconjuntos $S_{i1}, S_{i2}, \ldots, S_{ik}$, tal que $\bigcup_{j=1}^{k} S_{ij} = S_i$; `(iii)` um procedimento para cálculo de limitantes duais, isto é, para determinar $v(R_i)$, relativo à relaxação R_i de cada subproblema P_i e `(iv)` um critério para escolha do próximo subproblema a ser considerado pelo algoritmo. Além destes, outros dois ingredientes comumente úteis são: `(v)` uma heurística para gerar soluções viáveis para P e `(vi)` procedimentos que fazem uso de implicações lógicas com base em restrições e limitantes, objetivando possibilitar a fixação de valores de variáveis e/ou prover mecanismos para descarte de subproblemas (testes de dominância).

Algoritmos *branch-and-bound* são amplamente discutidos na literatura e comumente descritos de forma detalhada em livros-texto da área de otimização discreta e/ou programação matemática. Em particular, Wolsey [45] apresenta o tópico de forma bastante didática e acessível, mesmo para leitores iniciantes na área.

2.5.2 Algoritmos *Branch-and-Cut* (B&C)

Algoritmos *branch-and-cut* são algoritmos *branch-and-bound* onde planos de cortes são identificados ao longo da árvore de enumeração gerada durante a execução do *branch-and-bound*. Estes planos de corte são desigualdades válidas violadas por $\bar{x}$, solução do problema proveniente da relaxação (linear) da formulação disponível para o problema, a qual é obtida a cada nó da árvore. Uma vez identificadas, estas desigualdades são incorporadas à formulação do problema, propiciando a obtenção de um limitante dual melhor do que o atualmente disponível.

Embora as diferenças entre os algoritmos B&B e B&C possam parecer pequenas, como ressaltado em [45], há uma distinção clara de filosofia adotada. Enquanto algoritmos B&B privilegiam uma reotimização rápida executada a cada nó da árvore de B&B, algorimos B&C normalmente dispensam muito mais esforço computacional a cada nó da árvore. Tipicamente, rotinas de separação são executadas para identificar desigualdades violadas em conjunto com pré-processamentos e/ou heurísticas primais. Além destas, a manutenção do(s) repositório(s) de desigualdades (cortes) encontrados pelo algoritmo, os chamados *pools* de desigualdades, costumam requerer um tempo de processamento não desprezível na prática.

Em acréscimo aos ingredientes já enumerados para os algoritmos B&B podemos, a exemplo de [15, 45], citar os seguintes ingredientes essenciais para algoritmos B&C: `(i)` rotinas de separação que possibilitem identificar desigualdades (cortes) violadas pela solução da relaxação executada a cada nó da árvore de enumeração e `(ii)` procedimentos para gerenciamento dos *pools* de desigualdades de acordo com políticas de manutenção de desigualdades nos mesmos. Para mais detalhes sobre algoritmos B&C, incluindo considerações de implementação, veja as referências [36] e/ou [37].

Chapter 3

A relax-and-cut algorithm for the set partitioning problem

Neste capítulo é proposto um algoritmo *relax-and-cut* para o problema de particionamento de conjuntos. O algoritmo desenvolvido é usado como abordagem de solução para o problema ou, alternativamente, como uma ferramenta de pré-processamento para identificação de desigualdades válidas para o problema de particionamento de conjuntos. O texto em inglês corresponde ao artigo *"A Relax-and-Cut algorithm for the set partitioning problem"* escrito em co-autoria com os professores Dr. Cid C. de Souza[1] e Dr. Abílio Lucena[2] tendo sido publicado no periódico *Computers & Operations Research*, 35(2008), 1963-1981. Uma versão preliminar deste trabalho foi publicada nos Anais do *XIII Congreso Latino-Iberoamericano de Investigación Operativa (CLAIO)*, Montevidéu - Uruguai, em novembro de 2006 (vide apêndice A).

Relax-and-Cut algorithms offer an alternative to strengthen Lagrangian relaxation bounds. The main idea behind these algorithms is to dynamically select and dualize inequalities (cuts) within a Lagrangian Relaxation framework. This paper proposes a Relax-and-Cut algorithm for the Set Partitioning Problem. Computational tests are reported for benchmark instances from the literature. For Set Partitioning instances with integrality gaps, a variant of the classical Lagrangian relaxation is often used in the literature. It introduces a knapsack constraint to the standard formulation of the problem. Our results indicate that the proposed Relax-and-Cut algorithm outperforms the latter approach in terms of lower bound quality. Furthermore, it turns out to be very competitive in terms of CPU time. Decisive in achieving that performance was the implementation of

[1]Instituto de Computação, UNICAMP, Brasil.
[2]Departamento de Administração, UFRJ, Brasil

dominance rules to manage inequalities in the cut pool. The Relax-and-Cut framework proposed here could also be used as a preprocessing tool for Linear Integer Programming solvers. Computational experiments demonstrated that the combined use of our framework and XPRESS improved the performance of that Linear Integer Programming solver for the test sets used in this study.

Keywords: Lagrangian relaxation, cutting planes, set partitioning, relax-and-cut algorithms.

3.1 Introduction

The term Relax-and-Cut was coined to denote a class of Lagrangian Relaxation algorithms where inequalities may only be explicitly dualized when they become violated at a Lagrangian relaxation solution (see [8, 14, 21, 30, 31, 34]). These algorithms appear to be a promising approach to strengthen Lagrangian relaxation bounds. They use a dynamic inequality dualization scheme that renders viable the application of Lagrangian Relaxation to models with an exponential number of inequalities. Indeed, attempts to attain that for the Traveling Salesman Problem [3] date from the early 1980's.

In this study, a Relax-and-Cut algorithm is proposed to the Set Partitioning Problem (SPP). The effectiveness of Lagrangian techniques in the solution of the Set Covering Problem (SCP) was confirmed in [9]. The sophisticated implementation of Lagrangian relaxation described there led the authors to win the first prize in the FASTER contest promoted by the Italian Railways in 1994. This success of Lagrangian algorithms, even when compared to modern integer programming solvers, together with the fact that SCP is closely related to SPP, justifies our interest in such algorithms. In particular, we investigate in this work the embedding of strong cutting planes in a Lagrangian framework for SPP.

The text is organized as follows. Section 3.2 presents a brief review of Lagrangian Relaxation and the Subgradient Method (SM) together with a general description of a Relax-and-Cut framework. A Relax-and-Cut algorithm for SPP is proposed in sections 3.3 and 3.4 where details are given for the Lagrangian heuristic used and for the cut pool management policy implemented. Section 3.5 reports on the computational results obtained for test sets from the literature. Finally, Section 3.6 presents some conclusions and discusses possible extensions to this study.

3.2 Lagrangian relaxation and relax-and-cut algorithms

Denote by X a subset of the set of n-dimensional 0-1 vectors $\mathbb{B}^n$ and let

$$Z = \min \{cx : Ax \leq b,\ x \in X\} \tag{3.1}$$

be a formulation for a NP-hard combinatorial optimization problem. In association with (3.1) one has $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$, where m and n are positive integral values representing, respectively, the number of constraints and the number of variables involved.

Let Z' denote the formulation obtained after removing constraints $Ax \leq b$ from (3.1). Assume that Z' can be solved in polynomial or pseudo-polynomial time in the problem size. A Lagrangian relaxation of (3.1) is obtained by bringing the term $\lambda(Ax - b)$ into the objective function of Z', where $\lambda \in \mathbb{R}^m_+$ is the corresponding vector of Lagrange multipliers. The resulting *Lagrangian Relaxation Problem* (LRP) is thus given by

$$Z(\lambda) = \min \{cx + \lambda(Ax - b) :\ x \in X\} = \min \{(c + \lambda A)x - \lambda b :\ x \in X\}. \tag{3.2}$$

It is a known fact that $Z(\lambda) \leq Z$ and, therefore, the tightest possible lower bound on Z, attainable through $LRP(\lambda)$, is given by an optimal solution to the *Lagrangian Dual Problem* (LDP)

$$Z_D = \max_{\lambda \in \mathbb{R}^m_+} \{\min \{(c + \lambda A)x - \lambda b :\ x \in X\}\}. \tag{3.3}$$

In the literature, several methods exist to attempt to compute (3.3). Among these, due to its simplicity and the acceptable results it returns, the Subgradient Method (SM) is the most widely used [5]. A brief review of that method follows since the Relax-and-Cut algorithm we suggest here is based on SM.

The subgradient method (SM). SM is an iterative algorithm where, at every iteration, a LRP of type (3.2) must be solved. Starting with a feasible vector λ^0 of Lagrangian multipliers, one generates, at iteration k, a new feasible vector λ^k of multipliers and an associated LRP.

At iteration k, let $\bar{x}^k$ be an optimal solution to (3.2), $Z(\lambda)^k$ be the cost of $\bar{x}^k$ and Z^k_{UB} be a known upper bound on (3.1). An associated subgradient vector (for the m relaxed constraints) is then computed as $g^k_i = (b_i - a_i\bar{x}^k),\ i = 1, 2, \ldots, m$. That vector is then used to update λ^k. To that order, a *step size* θ^k must be computed first. Typically, for a real valued parameter $\alpha^k \in (0, 2]$, formula

$$\theta^k = \frac{\alpha^k(Z^k_{UB} - Z(\lambda)^k)}{\sum_{i=1}^m (g^k_i)^2} \tag{3.4}$$

is used [5]. Finally, once θ^k is obtained, λ^k is updated as

$$\lambda^{k+1}_i = \max \{0; \lambda^k_i - \theta^k g^k_i\},\ i = 1, 2, \ldots, m. \tag{3.5}$$

Notice that the straightforward use of formulas (3.4-3.5) may become troublesome when a *huge* number of dualized inequalities exist. An extreme situation occurs when m, the number of dualized constraints, is an exponential function of n. An alternative is then to modify SM according to the Relax-and-Cut scheme below.

The Relax-and-Cut strategy. Two types of Relax-and-Cut algorithms essentially exist. For the first type, several passes are made where Z_D is solved. After a pass, violated valid inequalities are separated for the solution of the last LRP. These inequalities are dualized and a new pass begins unless a stopping criterion is reached. For the second type, dualization is not delayed until Z_D is solved and may occur for every $LRP(\lambda)$ solution one may come across (for instance, throughout the use of SM). These two types of algorithms are respectively denoted Delayed Relax-and-Cut (DRC) and Non Delayed Relax-and-Cut (NDRC) [32]. In a comparison carried out in [32], NDRC performed better than DRC. In this study, we propose a NDRC algorithm to the Set Partitioning Problem. Thus, from now on, when we quote Relax-and-Cut, we mean NDRC. NDRC algorithms are used in [8, 21, 30, 31, 34].

In general terms, Relax-and-Cut comes up with a dynamic dualization scheme that makes it possible to apply Lagrangian Relaxation even when the number of inequalities to dualize is exponential.

Assume that one wishes to dualize, in a traditional Lagrangian fashion, the exponentially many inequalities in (3.1). Due to the potentially huge number of inequalities involved, the use of standard SM would prove prohibitive. That applies since an overwhelming computational effort would be required to perform SM iterations. Furthermore, even if one was able to overcome that hurdle, convergence problems are bound to happen since, typically, minute values for θ should be expected at every SM iteration.

Alternatively, the Relax-and-Cut scheme proposes a dynamic strategy to selectively dualize inequalities. That strategy is based, at every SM iteration, on a classification of inequalities into three groups. The first one (group 1) contains inequalities that are violated at $\bar{x}^k$. The second group (group 2) contains inequalities with nonzero multipliers currently associated with them (in accordance with the previous section, that means $\lambda > 0$). The third group (group 3) consists of the remaining inequalities and, as mentioned in [21], contains, for nontrivial instances, the vast majority of inequalities involved. Notice that an inequality may belong, simultaneously, to group 1 and group 2.

Two main observations justify the classification above. Firstly, Lagrangian multipliers for inequalities presently in group 3 would remain at null value at the end of the current SM iteration. For that reason, these inequalities are called *inactive* and, quite clearly, do not directly contribute, at the current SM iteration, to the updating of Lagrangian costs. On the other hand, multipliers for the *active* group 1 and group 2 inequalities, may

directly contribute to such an updating.

It should be noticed that inequalities may change groups from one SM iteration to another. However, at a given iteration, the Relax-and-Cut strategy consists in only dualizing active inequalities.

Moving back to the traditional use of Lagrangian relaxation, when m is not too big, Beasley [5] reported good practical convergence of SM to (3.3) after setting to zero all subgradients associated with inequalities in group 3. Proceeding accordingly, only inequalities in groups 1 or 2 are used to compute θ. Somehow, that gives a parallel to what is done in Relax-and-Cut.

An important step in the scheme just outlined is the identification of group 1 inequalities, i.e., violated inequalities at $\bar{x}^k$. To do so, likewise polyhedral cutting-plane generation, a separation problem must be solved at every iteration of SM. Thus, for every LRP, one tries to identify at least one inequality which is violated at the corresponding LRP solution. The inequalities thus identified may be either dualized or kept provided the new resulting LRP does not become too hard to solve.

It is worth noting that Relax-and-Cut separation problems may be easier to solve than their polyhedral cutting-plane algorithm counterparts. That applies since LRP normally involves integral valued solutions (for an example, see [34]).

3.3 A Relax-and-Cut Algorithm to the Set Partitioning Problem

Balas and Padberg [4] rank Set Partitioning as the problem that has the most widespread application among all special structures in pure integer programming. In fact, since 1960, several SPP applications are described in the literature. For a partial list of applications and surveys on the problem see [4, 6, 7, 10, 12, 22, 33]. Polyhedral investigations and ways to tighten the linear representation of SPP can be found in [2, 6] and [40]. Algorithms for solving the SPP also abound. Among those, we could mention: Fischer and Kedia [15] presented a dual ascent heuristic; Hoffman and Padberg [22] developed a branch-and-cut algorithm which successfully solved airline crew scheduling problems; Chu and Beasley [10] proposed an efficient genetic algorithm, and Atamtürk, Nemhauser and Savelsbergh [1] designed a powerful heuristic that combines reduction techniques, linear programming, Lagrangian heuristic and cutting planes generation. Also, based on this last work, Linderoth, Lee and Savelsbergh [28], developed a parallel algorithm for large scale SPPs.

SPP is a NP-hard problem [16] generically formulated as

$$Z_{SPP} = \min \{cx : Ax = e_m,\ x \in \mathbb{B}^n\}, \tag{3.6}$$

where A is a $(0,1)$ $m \times n$ matrix, c is a n-dimensional vector and e_m is the m-dimensional vector with all entries equal to one.

Two other NP-hard problems [16] are closely related to SPP: the Set Packing Problem (or Stable Set Problem (SSP)) and the Set Covering Problem (SCP)[6, 9, 22]. These two problems respectively have the following Integer Programming formulations:

$$Z_{SSP} = \min \{cx : Ax \leq e_m,\ x \in \mathbb{B}^n\}, \tag{3.7}$$

$$Z_{SCP} = \min \{cx : Ax \geq e_m,\ x \in \mathbb{B}^n\}, \tag{3.8}$$

where c, x, A, e, b, B, m and n are defined as in (3.6). It could be shown that the convex hull of the set of feasible solutions to (3.6) is given by the intersection of the convex hulls of (3.7) and (3.8). Hence, it suffices to study the convex hulls of SSP and SCP to know the convex hull of SPP. Thus, not surprisingly, the known classes of valid inequalities for SPP are typically derived from SSP and SCP (see [6, 22, 37, 38]). Among these, clique inequalities are often cited as being the most practically relevant ones for solving SPPs exactly.

To study the facial structure of the SPP polytope, Edmonds [13] associated an intersection graph $G_A = (N, E)$ to the columns of the SPP input matrix A. Following [13] (see [6, 22] for details): *(1)* for $j \in N = \{1, 2, \ldots, n\}$, node j is associated to column a^j of A, and, *(2)* there is an edge $(i, j) \in E$, for $i, j \in N$ and $i \neq j$, if columns a^i and a^j of A are nonorthogonal.

Notice that every feasible solution to SPP corresponds to a subset $S \subseteq N$ of nodes, where no two nodes of S are adjacent in G_A. Thus, each feasible set S is an independent set of G_A. On the other hand, let $V \subseteq N$ be a set in G_A such that V induces a complete subgraph in G_A. Then, every independent set meets V in at most one element. As a result of that, since each node of N represents a variable in (3.7), every solution to SPP satisfies the inequality $\sum_{v \in V} x_v \leq 1$.

The inequality above defines a facet of SSP if and only if V is a maximal clique of G_A [6, 22, 36, 37, 38]. However, the number of maximal cliques grows exponentially with the graph size. Therefore, it is not viable to a priori append all of them to the formulation. Furthermore, finding the most violated of these inequalities, at a given point, turns out to be hard. Thus, a clique separation heuristic was implemented in this study. At every SM iteration, such a heuristic aims at finding, provided one exists, at least one maximal clique not satisfied by the current LRP solution $\bar{x}^k$. Our heuristic is a simple modification of the greedy algorithm in [35] where nodes are selected in decreasing order of degree magnitude. Intersection graph $G_A = (N', E)$, which is passed to the heuristic as a parameter, only contains that subset N' of columns corresponding to variables assuming value 1 in $\bar{x}^k$, i.e., $N' = \{j \in N : \bar{x}^k_j = 1\}$.

Lagrangian relaxations for SPP. Re-writing (3.6) in summation notation and dualizing all equality constraints in a Lagrangian fashion we obtain

$$\begin{aligned} \text{LRP}(\lambda) &= \min \{\sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i (1 - \sum_{j=1}^{n} a_{ij} x_j) : x_j \in \{0,1\},\ j = 1, \ldots, n\} \\ &= \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i : x_j \in \{0,1\},\ j = 1, \ldots, n\} \end{aligned} \tag{3.9}$$

where $\bar{c}_j$ is the Lagrangian cost of x_j given by $\bar{c}_j = (c_j - \sum_{i=1}^{m} \lambda_i a_{ij})$, $\lambda \in \mathbb{R}^m$ is the Lagrangian multiplier vector attached to the dualized constraint set, $a_{ij} \in \{0,1\}$ corresponds to the coefficient at position ij in $A^{m \times n}$ and c_j is the cost associated to the binary variable x_j.

It is straightforward to verify that (3.9) is solvable in linear time on n, by simple inspection. However, lower bound LRP(λ) might be further strengthened by adding to (3.9) some constraints that are redundant to the original formulation (3.1). We did that by appending constraint $\sum_{j=1}^{n} x_j \leq m$ to (3.9), thus yielding

$$\text{LRP}(\lambda) = \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i : \sum_{j=1}^{n} x_j \leq m,\ x \in \{0,1\}^n\} \tag{3.10}$$

Notice that, after sorting the Lagrangian costs involved, the LRP above could easily be solved by inspection in a $O(n \log n)$ complexity algorithm.

An attractive alternative Lagrangian relaxation exists to (3.10). It is, however, computationally more expensive than that. In any case, as a trade off, such a relaxation is likely to produce better bounds. It involves the use of knapsack constraint $\sum_{j=1}^{n} w_j x_j = m$, where w_j denotes the number of rows covered by variable x_j. The resulting LRP is given by

$$\text{LRP}(\lambda) = \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i : \sum_{j=1}^{n} w_j x_j = m, x \in \{0,1\}^n\} \tag{3.11}$$

Clearly, the knapsack constraint introduced above is valid for any feasible solution to our original SPP. It may yield a Lagrangian Dual Problem bound which is better than that given by the linear programming relaxation of (3.1) (see [5, 17, 21, 36], for details). In this case, LRP(λ) is solved via dynamic programming.

Before we describe how we incorporated cutting planes to our algorithm, we should notice that the addition of the extra knapsack constraint in (3.11) is a rudimentary example of composite relaxation as it was proposed in [18] (see also [19, 24, 25, 26, 27]). However, this relaxation is adequate for a first evaluation of the possible gains brought by the inclusion of strong cutting planes in Lagrangian algorithms for SPP. We use it

with this purpose. Successful applications of composite relaxation to SCP could be found in [23] and [29]. We left for a future work the investigation of the embedding of cutting planes in these more sophisticated types of relaxations.

Using clique inequalities within a Relax-and-Cut framework. Our clique inequality separation routine is called at every SM iteration. As a result, the original SPP formulation and all relaxations described above are modified to accommodate the inequalities identified by the separation routine. Such inequalities could be written as $\sum_{j=1}^{n} d_{kj}x_j \leq 1,\ k = 1, \ldots, |K|$, where K is the set of clique inequalities that are kept dualized at the current SM iteration and $d_{kj} \in \{0, 1\}$ are the values for the corresponding coefficients in the associated clique inequality matrix $D^{|K|\times n}$. The resulting relaxations (incorporating respectively constraints $\sum_{j=1}^{n} x_j \leq m$ and $\sum_{j=1}^{n} w_j x_j = m$) could thus be written as

$$\mathrm{LRP}_{\leq m}^{\lambda,\beta} = \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i - \sum_{k=1}^{|K|} \beta_k : \sum_{j=1}^{n} x_j \leq m,\ x \in \{0,1\}^n\} \tag{3.12}$$

$$\mathrm{LPL}_{k}^{\lambda,\beta} = \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i - \sum_{k=1}^{|K|} \beta_k : \sum_{j=1}^{n} w_j x_j = m,\ x \in \{0,1\}^n\} \tag{3.13}$$

where $\beta \in R_{+}^{|K|}$ is the vector of multipliers associated with the clique inequalities and $\bar{c}_j = (c_j - \sum_{i=1}^{m} \lambda_i a_{ij} + \sum_{k=1}^{|K|} \beta_k d_{kj})$, for $j = 1, \ldots, n$, are the corresponding Lagrangian modified costs.

Cut pool management. At each Relax-and-Cut iteration, all clique inequalities (identified by the separation routine) and all equality constraints in (3.6) are dualized. Consequently, a same clique inequality may be repeatedly identified by the separation routine, at different SM iterations. Hence, it is necessary to establish a policy to manage the pool of valid inequalities yielded by the separation routine. In order to do so, dominance relationships between inequalities (see [36] for definitions) were taken into account. Essentially, the following rules were implemented:

1. New valid inequalities are inserted into the pool if the following conditions are met:
  (1.a) they are not already contained in the pool;
    neither they are one of the inequalities in the original formulation; and
  (1.b) they are not dominated by any inequality already in the pool
    or by any inequality in the original formulation;
2. Inequalities are only deleted from the pool if they are dominated by a newly identified inequality and do not belong to the original formulation.

Although an effective pool management policy potentially improves the performance of Relax-and-Cut algorithms, a thorough discussion on this topic was not found in the literature. The only explicit mention of a mechanism to avoid repetition of equal cuts in the pool was found in [11]. The verification we made for dominance relations among cuts is broader than a simple search for repetitions in the cut pool and seems to be a novelty in the implementation of Relax-and-Cut algorithms. Obviously, the management policy suggested above introduces some additional computational effort. However, in practice, the use of conveniently chosen data structures renders the implementation of such a policy very fast. As it turns out, the time overhead required to apply the suggested rules is more than compensated for the drastic reduction in the number of inequalities to dualize. As a result, global savings in processing time are thus attained. Additionally, another positive impact results from the fact that some convergence problems, due to repetition and dominance within the pool, end up being eliminated. Consequently, some (small) dual bound improvements are also observed. Comparative tests that indicate the effectiveness of the suggested pool management policy are presented in Section 3.5.

Valid upper bounds to SPP. A Lagrangian heuristic is suggested here to generate feasible SPP solutions. At every SM iteration, the proposed heuristic attempts to convert corresponding LRP solutions into feasible SPP solutions. That is attained after making those variables set to one in the current LRP solution very attractive to be selected in a (very simple, greedy style) primal algorithm. A high level description of the heuristic is given below where the following notation is used: *(1)* S_{LRP} and S_{SPP} denote, respectively, feasible solutions to LRP and SPP; *(2)* V corresponds to the set of indices (columns) for SPP variables; *(3)* L denotes a list of rows that are candidate to be covered at an iteration of the heuristic.

1. $S_{SPP} \leftarrow \emptyset$;
   $V_k = k, k = 1, \ldots, n$, where n is the number of variables in SPP;
   $L = \{1, \ldots, n\}$;
   $k = 0$;
2. Sort V in non increasing order of the corresponding variables in S_{LRP},
   $(v_k \geq v_{k+1}, k = 1, \ldots, n-1)$;
3. While $L \neq \emptyset$ and $k < n$
   Select x_{v_k} such that: V sorting is obeyed, i.e., x_{v_k} is only considered after
   $x_{v_{k-1}}$ and x_{v_k} does not cover a row already covered by S_{SPP};
   $S_{SPP} = S_{SPP} \cup \{x_{v_k}\}$;
   Delete from L the rows covered by x_{v_k};
4. If $L = \emptyset$, return the feasible solution composed of the variables (columns) in
   S_{SPP} else, return `fail`.

Contrary to what prevails for SCP and SSP, it is hard [16] to determine whether or not an instance of SPP is feasible. Therefore, there is no guarantee that the heuristic suggested above will succeed in generating a feasible SPP solution (even when one does indeed exists). Therefore the procedure that follows was proposed to (roughly) estimate an upper bound on the optimal SPP solution value.

1. For $j = 1, \ldots, n$, compute LC_j as the number of rows covered by column j;

2. Sort LC in nondecreasing order and determine the smallest k such that $\sum_{j=1}^{k} LC_j \geq m$. If $\sum_{j=1}^{k} LC_j > m$, do $k = k - 1$.

3. Return the upper bound given by $\sum_{j \in C} c_j$, where C is the set of indices for the k variables with the highest costs in the problem.

Variable fixing tests. Provided a SPP variable x_k is proven to be suboptimal, it can be eliminated from the problem (corresponding column eliminated from matrix A). Likewise, if x_k is proven to be part of an optimal solution, the corresponding column and all rows covered by it may be eliminated from A.

The variable fixing tests suggested here use Lagrangian relaxation SPP lower bounds, SPP upper bounds and Lagrangian modified costs in attempts to price variables in and out of an optimal solution. These tests are similar to those described for SCP by Beasley in [5]. The only difference between the two being that, for our procedure, a few mechanisms had to be incorporated to accommodate the additional constraint found in our LRPs.

3.4 The Relax-and-Cut Framework Execution Flow

The implementation of our Relax-and-Cut framework is structured in modules. These modules, which are briefly described below, are executed sequentially as shown in Figure 3.1:

Lagrangian preprocessing (LPREP). This is a fast preprocessing routine aimed at reducing instance size. It performs a few iterations of a standard SM algorithm (over relaxation (3.10)) and, within these iterations, implements variable fixing tests. Moreover, the best upper bound thus obtained is saved and is later used as an input to the module that follows.

In Figure 3.1, the LPREP module is identified by a dashed line box to emphasize the fact that its use, within the proposed framework, is optional.

Relax-and-Cut (RC). The module corresponds to the Relax-and-Cut algorithm described in section 3.3 and is the core of our framework. Within it, variable fixing tests are disabled. Furthermore, the next two modules are only executed if optimality is not proven during this phase.

Lifting (LIFT). The module attempts, at the end of the Relax-and-Cut run, a 0-1 lifting for these cuts left at the cut pool. The procedure works as follows. Given a clique inequality, consider its intersection graph support. One then attempts to enlarge that clique by adding additional nodes to it until a maximal clique is obtained. After every successful lifting, the pool is updated. Notice that the (stronger) cuts thus generated might dominate several different clique inequalities currently found in the pool. Provided this is the case, dominated inequalities are discarded from the pool. As a result, the number of pool inequalities, at the end of the module, may be smaller than that in the beginning of it.

C1 (Pg.52)

LR with cuts (LRC). This module is equivalent to implementing SM over a standard (non Relax-and-Cut) Lagrangian relaxation of a strengthened SPP formulation. Such a formulation includes all the original SPP constraints plus those clique inequalities left in the cut pool at the end of the previous module. In practice, that implies applying SM (over relaxation (3.12)) knowing, in advance, which clique inequalities will be left in the pool. The module takes as an input the best upper bound and the best Lagrangian multipliers (associated with the original SPP formulation constraints) obtained at the Relax-and-Cut run. Besides that, the best lower bound found so far is also passed as an

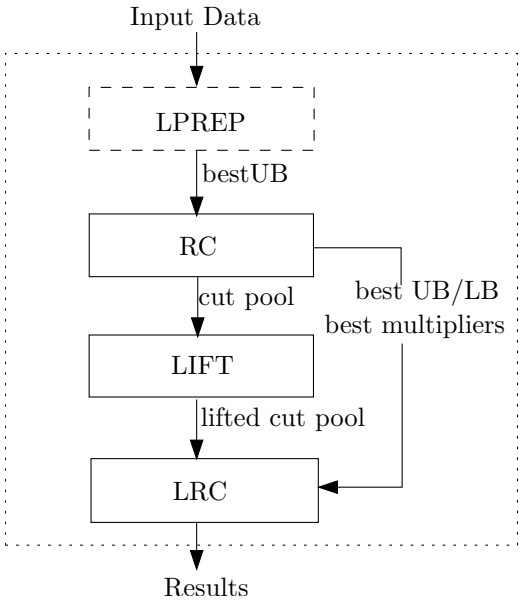

Figure 3.1: *Relax and Cut* Framework Execution Flow.

input to the module. This is done with the aim of attaining *better* step size values for SM iterations (see equation 3.4).

Essentially, two observations motivated the inclusion of this module in our proposed framework. The first one is that after every successful lifting operation (in the previous module) a new clique inequality is obtained which dominates at least one of the inequalities in the pool. As a result, one may possibly obtain a dual bound stronger than that computed at the Relax-and-Cut module. The second observation is that, for the proposed Relax-and-Cut scheme, it was experimentally noticed that inequalities identified at early SM iterations are more likely to contribute to the computation of *better* bounds than those identified later on. Thus, applying *traditional* Lagrangian relaxation to the reinforced SPP formulation described above is capable of attaining a Lagrangian relaxation bound at least as good as that obtained in the previous module.

In terms of the nomenclature discussed in Section 3.2, our suggested framework represents a combination of Non Delayed Relax-and-Cut (module **RC**) and a somewhat Delayed Relax-and-Cut approach (this module).

Finally, a common observation to all modules in our framework is that, at every SM iteration (irrespective of the module where the method is being used) the Lagrangian heuristic in Section 3.3 is applied in attempts to generate stronger SPP upper bounds.

3.5 Computational Results

This section discusses a few implementation details of our Relax-and-Cut algorithm for SPP and reports on our computational experiments. The main objective was to determine whether or not the use of cutting planes, within our proposed Relax-and-Cut framework for SPP, pays off. Tests were carried out on a Pentium IV machine having 1GB of RAM and running under Linux OS (distribution Debian 4.0.1-4). Our codes were written in `C++`, using resources of the Standard Template Library [41]. We used the free compiler `g++` (`gcc` version 4.0.2) with options `-O3` and `-lm` selected.

The test sets. Two test sets were used in our experiments. The first, denoted **I1**, consists of real-world SPP instances from OR-Library, available from

`http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.

They originate from the airline crew scheduling problem [22]. Among the available instances, we considered only those instances having an integrality gap. An exception to that is instance `us01`, containing 1053137 columns, which was not attempted to be solved here due to CPU memory limitations. The second test set, denoted **I2**, contains only those SCP instances cited in [20] as being hard to solve. To each SCP instance in that set we appended an identity matrix to the original set of columns (in order to guarantee existence of feasible SPP solutions). The original SCP instances used in set **I2** may be obtained from the same site where set **I1** was obtained. These SCP instances are derived from an old question posed by the mathematician Paul Erdös: *what is the minimal number of edges of a hypercube that can be chosen so that every cycle of 4 edges contains at least one chosen edge?* An interesting property of these instances is that their optimal fractional (SCP or SPP) solutions have a cost of $n/4$, where n is the number of variables in the corresponding original SCP instance (for more details see [20]).

Preprocessing rules. Within the context of the airline industry, SPP instances may have special structures and, usually, it is possible to reduce their input size by applying simple logical rules. Since instances in test set **I1** originate from airline industry applications, a small subset of the simple preprocessing rules proposed in [6, 10, 22] was used in an attempt to reduce their input size. Following the notation in [10], one has

$I = 1, \ldots, m$: set of rows;

$J = 1, \ldots, n$: set of columns;

$\alpha_i = \{j : a_{ij} = 1, j \in J\}$: set of columns that cover row i, and

$\beta_j = \{i : a_{ij} = 1, i \in I\}$: set of rows covered by column j.

The preprocessing rules we use are:

1. **Rule 1:** For all $(j,k) \in J, j \neq k$, such that $\beta_j = \beta_k$, if $c_j \leq c_k$, delete (the duplicated) column k.

2. **Rule 2:** If $|\alpha_i| = 1, i \in I$, then a column $j \in \alpha_i$ must be in every optimal solution ($x_j = 1$). Hence, delete j and all rows $k \in \beta_j$, as well as all columns in α_k, such that $k \in \beta_j$.

3. **Rule 3:** For all $(i,k) \in I, i \neq k$, such that $\alpha_i \subseteq \alpha_k$, delete all columns $j \in (\alpha_k - \alpha_i)$ and row k. This reduction is due the fact that any column covering row i also covers row k.

General parameters. The following settings were used for the basic parameters of the subgradient algorithm: *(1)* in equation (3.4), α is initially set to 2 and is multiplied by 0.75 after every 100 consecutive SM iterations without an overall improvement on the lower bound; *(2)* the primal heuristic is called at every SM iteration. In the first iteration, if the heuristic fails, it is called again with set V being sorted in nondecreasing order; *(3)* the algorithm stops when the limit of 8000 SM iterations is reached or when, in equation (3.4), $\alpha^k \leq 0.00001$; whatever occurs first. When the Lagrangian Preprocessing module is not used, at most 4000 SM iterations are performed during each of the **RC** and **LRC** phases. When the Lagrangian Preprocessing is used, since the idea is to use it as a fast attempt to price variables in and out of optimal solutions, the following settings are applied: *(a)* α is multiplied by 0.75 after every 20 consecutive iterations without an overall improvement on the lower bound and *(b)* at most 800 SM iterations are executed during preprocessing. In this case, the remaining maximum of 7200 SM iterations are equally divided among the two additional modules that use subgradient optimization.

Relax-and-Cut specific parameters. Setting up large intersection graphs at each call of the separation routine proved far too expensive. In order to save on CPU time, at each SM iteration, the maximum number of nodes in the intersection subgraphs used (for separation purposes) was conveniently bounded. The corresponding default limit was set to 200 nodes. For our computational tests, the only exception was the largest instance in group **I2** which involves a very high number of rows. For this instance, a limit of 500 nodes was applied. Selection of intersection subgraph nodes was based on the Lagrangian costs associated with each corresponding SPP variable. Accordingly, only variables taking a value one in the solution of the Lagrangian relaxation problem in hand were used. Furthermore, only these subgraphs associated with nodes corresponding

to variables having the most negative Lagrangian costs were explored in the separation routine (when searching for violated inequalities).

Lagrangian relaxation results. We investigate now the gain, if any, of embedding cutting planes in a Lagrangian framework. Table 3.1 shows the results obtained by Lagrangian relaxations (3.10), (3.11) and the Relax-and-Cut algorithm implied by (3.12). For each configuration used, SPP upper and lower bounds and the time (in CPU seconds) required to attain them are reported. Since all variable costs are integral, lower bounds are rounded up. For the Relax-and-Cut framework, when the Lagrangian Preprocessing module is enabled, the percentage of priced out variables is also given. The first three columns in the table respectively give the number of rows, the number of columns and the corresponding density for each preprocessed instance. Preprocessing rules were repeatedly applied to SPP instances until no further reduction is attained. Suitable data structures were implemented so that the resulting preprocessing times were negligible (typically less than one second). Optimal integer solution values and optimal LP relaxation values are known for all instances in set **I1**. For the three smaller instances in **I2**, corresponding optimal solution values were obtained through solver XPRESS Optimizer 16.01.05 (`http://www.dashoptimization.com/`). Experiments conducted with the three largest instances indicated that XPRESS was not able to solve any of them within the imposed time limit of ten CPU hours. For these instances, table entries give the upper and (rounded up) lower bounds available at the moment the time limit was reached.

Results in Table 3.1 indicate that Lagrangian relaxation (3.11), i.e. the one involving the additional knapsack constraint, was able to close gaps between SPP upper and lower bounds for only five of the instances tested. However, for a few instances, the relaxation proved worse than (3.10), i.e. the *pure* Lagrangian relaxation. As it turns out, in that case, the upper bounds obtained through the use of the Lagrangian relaxation scheme associated with (3.11) proved very poor. Furthermore, for large scale instances, memory requirements (of the Dynamic Programming algorithm used) did not allow the use of the associated knapsack constraint. Therefore, this relaxation was not applied to the two largest instances in set **I2**.

Another point to stress is that the LRP solutions produced by (3.11) typically have few variables with value one. As a result, the use of cutting planes combined with the knapsack constraint suggested in (3.13) was discarded. The reason for that was the fact that the intersection graphs associated with corresponding LRP solutions would typically imply small cliques (thus leading to weak inequalities). Such a fact was confirmed in our preliminary tests with relaxation (3.13) (over a small subset of instances).

Turning now to Relax-and-Cut table columns, it is possible to see that 27 out of the 45 instances tested were solved to proven optimality, i.e., 60% of the total. In relation

Instance				Optimal	LP	XP	LR (3.10)			LR (3.11)			R&C (3.12)			
label	m	n	$d(\%)$			$t(s)$	ub	lb	$t(s)$	ub	lb	$t(s)$	ub	lb	$t(s)$	$fix(\%)$
nw03	59	38964	14.12	24492	24447	3	24558	24447	386.2	34917	24447	165.0	24492	**24492**	65.4	99.76
nw04	36	46190	19.94	16862	16310.67	19	20286	16311	571.4	57172	16313	178.51	17856	16328	258.17	78.34
nw06	50	5977	28.27	7810	7640	<1	8464	7640	66.0	50670	7640	24.3	7810	**7810**	18.9	68.76
nw11	39	6488	16.68	116256	116254.5	<1	116259	116255	42.1	116265	116255	22.8	116256	**116256**	8.5	99.77
nw13	51	10905	12.57	50146	50132	<1	50154	50132	119.0	52908	50132	47.4	50146	50140	18.1	97.98
nw17	61	78186	13.96	11115	10875.7	22	37938	10876	1158.4	91848	10875	423.1	11295	10914	243.5	92.61
nw18	124	8460	6.83	340160	338864.3	<1	394104	338863	93.0	548236	338862	82.1	340160	**340160**	142.4	0.00
nw20	22	566	24.99	16812	16626	<1	17478	16626	4.4	18267	16626	0.8	16812	**16812**	0.9	81.98
nw21	25	426	24.32	7408	7380	<1	7408	7380	1.9	7442	7380	0.4	7408	**7408**	0.4	96.24
nw22	23	531	24.22	6984	6942	<1	7102	6942	2.8	7158	6942	0.5	6984	**6984**	0.6	93.60
nw23	18	473	24.86	12534	12317	<1	13034	12317	2.6	13904	12317	0.4	12534	**12534**	1.3	70.82
nw24	19	926	33.22	6314	5843	<1	6462	5843	4.7	6462	5922	0.8	6314	**6314**	1.1	94.17
nw25	20	844	30.15	5960	5852	<1	6580	5852	4.3	7178	5852	0.7	5960	**5960**	1.1	79.62
nw26	23	542	23.11	6796	6743	<1	6804	6743	2.5	8180	6743	0.5	6796	**6796**	0.6	81.00
nw27	22	926	30.76	9933	9877.5	<1	10344	9878	4.7	9933	**9933**	0.3	9933	**9933**	1.0	96.33
nw28	18	825	38.43	8298	8169	<1	8298	8169	4.2	8298	**8298**	0.1	8298	**8298**	0.9	99.03
nw29	18	2034	30.99	4274	4185.9	<1	4814	4186	19.3	4856	4190	2.6	4430	4257	7.8	76.80
nw30	26	1884	29.81	3942	3726.8	<1	4294	3727	17.8	6072	3754	5.1	3942	**3942**	2.3	96.76
nw31	26	1823	29.21	8038	7980	<1	8038	7980	11.1	8822	8009	2.4	8038	**8038**	2.2	98.85
nw32	18	251	25.81	14877	14570	<1	14877	14570	1.4	16329	14583	0.4	14877	**14877**	0.2	90.83
nw33	23	2415	30.75	6678	6484	<1	7840	6484	13.8	7968	6484	3.5	7840	6536	6.9	82.70
nw34	20	750	28.16	10488	10453.5	<1	10713	10454	3.6	10488	**10488**	0.3	10488	**10488**	0.8	98.53
nw35	23	1403	27.02	7216	7206	<1	7456	7206	8.5	7216	**7216**	0.4	7216	**7216**	1.6	91.87
nw36	20	1408	36.14	7314	7260	<1	7502	7260	11.1	7550	7265	1.5	7328	7281	20.1	0.00
nw37	19	639	25.89	10068	9961.5	<1	10068	9962	3.1	10068	**10068**	0.1	10068	**10068**	0.7	99.06
nw38	23	911	31.44	5558	5553	<1	5558	5552	4.7	8402	5552	0.1	5558	**5558**	1.0	96.38
nw39	25	567	26.27	10080	9868.5	<1	10545	9869	2.9	10758	10056	0.6	10080	**10080**	0.6	91.89
nw40	19	336	26.87	10809	10658.3	<1	10809	10659	2.4	12180	10666	0.4	10809	**10809**	0.3	96.43
nw41	17	177	22.33	11307	10972.5	<1	11307	10973	0.7	11457	11084	0.3	11307	**11307**	0.2	90.96
nw42	23	895	26.05	7656	7485	<1	7656	7485	4.5	10826	7485	0.9	7656	**7656**	1.0	97.21
nw43	17	982	26.43	8904	8897	<1	8904	8897	5.2	11158	8897	0.7	8904	**8904**	1.1	98.98
us04	105	12169	7.65	17854	17731.7	<1	17865	17732	176.2	17865	17732	100.6	17907	17834	210.0	0.00
aa01	624	7767	1.04	56138	55535.4	35	57528	55319	111.5	385614	52330	365.0	57528	55393	147.8	0.00
aa03	569	7208	1.11	49649	49616.4	3	50747	49614	100.9	322702	47068	315.0	50747	49627	121.5	0.49
aa04	343	6215	1.80	26374	25877.6	38	29888	25873	86.8	216238	25870	145.3	29888	25956	114.9	0.00
aa05	555	6585	1.07	53839	53735.9	5	54014	53685	90.4	351061	50611	283.7	54014	53703	113.9	0.02
aa06	516	6276	1.18	27040	26977.2	3	27293	26977	86.3	157755	26974	201.0	27293	26998	91.5	13.08
kl01	47	5970	13.45	1086	1084	1	1091	1084	68.5	1139	1084	31.9	1086	1085	11.9	93.85
kl02	69	16734	8.34	219	215.3	2	221	216	217.3	230	216	113.0	220	216	275.3	3.24
cyc06	240	432	1.16	112	48	1	112	48	3.0	164	48	3.3	120	112	25.2	-
cyc07	672	1120	0.45	352	112	6	352	112	8.8	472	112	23.7	352	**352**	4.0	-
cyc08	1792	2816	0.18	1024	256	2177	1036	256	28.2	1318	256	152.8	1048	1024	86.8	-
cyc09	4608	6912	0.07	[2879, 2367]	576	36000	2844	576	76.9	3404	576	960.0	2844	2816	234.2	-
cyc10	11520	16640	0.03	[7656, 6543]	1280	36000	7424	1280	313.9	-	-	-	7424	**7424**	435.1	-
cyc11	28160	39424	0.01	[19619, 11529]	2816	36000	19007	2816	1304.0	-	-	-	19016	18944	2316.4	-

Table 3.1: Computational results for SPP instances.

with this, recall that the Lagrangian relaxation scheme associated with (3.11) managed to solve (to proven optimality) only about 11% of these instances. Additionally, for all instances tested, lower bounds obtained by the Relax-and-Cut algorithm proved at least as good as those returned by the two other Lagrangian algorithms. The Relax-and-Cut edge over these algorithms becomes more impressive for the `cyc` instances in the last six rows of Table 3.1 (possibly due to the *huge* integrality gaps associated with these instances). Furthermore, for 43 instances (i.e. 96% of the total) Relax-and-Cut lower bounds are strictly better than the bounds produced by the LP relaxation of the standard SPP formulation. Despite the fact that such a dual bound improvement was predictable in theory, it is well-known that sometimes it may not be attained in practice (due to SM convergence problems).

Finally, notice that the XPRESS Optimizer (under default settings) was not able to solve instance `cyc10` after 10 CPU hours. However, Relax-and-Cut solved that instance to proven optimality in about 7 CPU minutes. Indeed, for test set **I2**, clique inequalities proved very effective in strengthening the lower bounds. The hardness involved in providing optimality certificates for these instances came from the primal side of our algorithm (since our heuristic failed to return better, ideally optimal, upper bounds).

Though our goal was to compare the performance of Lagrangian algorithms, a few words are in place for XPRESS results. Especially for the **I1** set, the solver is extraordinarily fast. XPRESS also uses clique inequalities as cuts, thus, it can be viewed as a thorough implementation of a linear programming cutting plane algorithm (LP-CPA) for SPP. In 50% of the cases, the Relax-and-Cut algorithm had a comparable performance but, in the remaining instances, it was largely beaten. However, rather than being discouraging, these results highlight the potential of Relax-and-Cut algorithms for SPP. This applies since ours is the very first Relax-and-Cut algorithm for that problem, in opposition to the many years already spent in developing and refining LP-CPA for SPP. Besides, when we focus on the **I2** set, the situation is reversed and our algorithm performs remarkably well while XPRESS stalls. A promising cooperative framework involving the two techniques is discussed in the next paragraphs.

Combined results. From the results in Table 3.1 one may conclude that XPRESS is very suitable for solving instances in test set **I1**. However, that commercial IP package fails to solve (to proven optimality) several instances in test set **I2**. Differently from the instances in set **I1**, the latter instances have large integrality gaps, unitary costs and very low matrix densities.

An alternative to tackle the instances where XPRESS behaved poorly, is to devise a hybrid approach that combines Lagrangian relaxation with Integer Linear Programming (ILP), in the style suggested in [8]. In such an approach, optimization is split into two

steps: a preprocessing phase, based on our Relax-and-Cut framework, and the subsequent use of an ILP solver. Accordingly, our Relax-and-Cut framework would thus be used to generate cuts that allow the ILP solver to operate over a tighter SPP formulation. In this way, after running our Relax-and-Cut algorithm, lifted (active) cuts in the pool are passed as an input to XPRESS (alongside the best known integral SPP solution). The execution flow corresponding to such an approach is described in Figure 3.2.

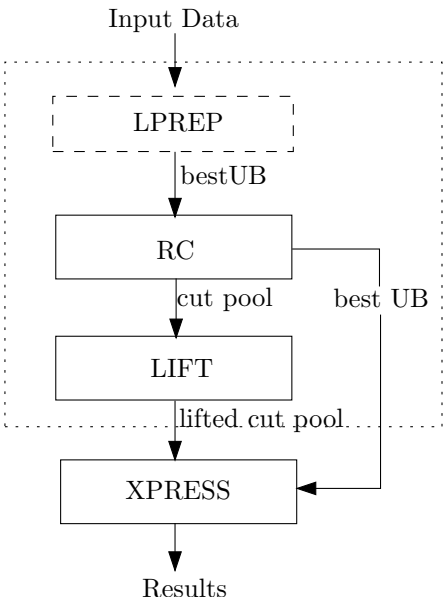

Figure 3.2: Execution flow of the *Relax-and-Cut* Framework applied as a preprocessing tool to provide XPRESS Optimizer with clique inequalities.

Table 3.2 gives the results obtained by our proposed hybrid algorithm. Only the four hardest instances in set **I2** were used in these experiments. The table gives, for every instance tested: the best upper bound value passed to XPRESS, the lower bound produced by the Relax-and-Cut algorithm[3], and the CPU time spent up to the end of the lifting procedure (and that of the associated procedure that prepares Relax-and-Cut input data to XPRESS). Heading `XPC` refers to the bounds produced by XPRESS and the CPU time required to obtain them. Finally, the last table column indicates the total CPU time spent by the proposed hybrid algorithm.

Comparing the results obtained in the experiments above and those returned by XPRESS, operating as a stand alone procedure (see Table 3.1), the immediate conclusion is that our hybrid approach outperforms the latter. A point to highlight is the *enormous*

[3]Here, no SM execution is performed after lifting. Hence, these lower bounds could be quite different from, typically weaker than, those reported in Table 3.1.

Instance	R&C (3.12)				XPC			Time (s)
	ub	*lb*	$t1(s)$	*pool*	*ub*	*lb*	$t2(s)$	$t1+t2$
cyc08	1048	744	80.0	256	1024	**1024**	54	134
cyc09	2844	2139	483.3	512	2816	**2816**	765	1248.3
cyc10	7424	3792	949.7	1024	7424	**7424**	28464	29413.7
cyc11	19016	6007	2006.2	2048	-	18944	36000	38006.2

Table 3.2: Results for SPP instances using Relax-and-Cut as preprocessing.

reduction (of about 94%) in the CPU time required to solve instance `cyc08` to proven optimality. Likewise, it should be stressed that our Lagrangian preprocessing made it possible to XPRESS to solve instances `cyc09` and `cyc10` within the pre-specified CPU time limit (recall that previously that was not possible). Specifically for instance `cyc09`, which previously could not be solved to proven optimality in 10 CPU hours, an optimal solution is now obtained in less than 20 minutes of CPU. Furthermore, although the hybrid algorithm was not capable of solving (within the imposed time limit) instance `cyc11` to proven optimality, the final dual bound reached was much better than that displayed in Table 3.1. Indeed, for that particular instance, XPRESS had previously to be stopped even before it was able to compute the very first LP relaxation solution. Contrary to that, we are now able to say (using the feasible integral solution provided by Relax-and-Cut) that the gap remaining to be closed for `cyc11` is of approximately 0.38%.

SPP upper bound results. The main focus of this study was to strengthen Lagrangian relaxation SPP lower bounds. However, the good performance of our proposed Lagrangian heuristic must be stressed. After examining Table 3.1, one could see that the heuristic was capable of generating optimal solutions for 29 out of the 45 instances tested (thus attaining a 64% success rate).

In order to further evaluate the quality of our heuristic, we compared the solutions it returns with those given by two SPP heuristics from the literature. The first of these is an LP-based heuristic (LP-H) developed by Hoffman and Padberg [22]. The other is a genetic algorithm (GA) proposed by Chu and Beasley [10]. For both of these heuristics, results are only presented for test set **I1**. Hence, our comparisons are restricted to that set.

After examining the column associated with our Lagrangian algorithm in Table 3.1 (see *ub* under heading R&C (3.12)), one could see that the Lagrangian heuristic was able to find feasible solutions in all cases. In that respect, it could be said that our heuristic is much more effective than their two counterparts (which failed to find feasible solutions

# wins		# ties
LR-H	LP-H	
30 (77%)	5 (13%)	4 (10%)

Table 3.3: Lagrangian heuristic (LR-H) versus LP-Based heuristic (LP-H) proposed by Padberg and Hoffman [22].

# wins		# ties
LR-H	GA Average	
10 (26%)	7 (18%)	22 (56%)

# wins		# ties
LR-H	GA Median	
7 (18%)	**7 (18%)**	25 (64%)

# wins		# ties
LR-H	GA Best	
5 (13%)	**8 (21%)**	26 (66%)

Table 3.4: Lagrangian heuristic (LR-H) compared with the genetic algorithm (GA) proposed by Chu and Beasley [10].

for several test set instances).

Table 3.3 compares the quality of the upper bounds generated by the Lagrangian heuristic (LR-H) with those given by the LP-based heuristic (LP-H). As one could see, the Lagrangian heuristic was capable of returning better quality solutions than the LP-based heuristic for the majority of the instances tested (30 out of 39) and is only beaten in 5 cases. For the remaining 4 instances, the quality of the upper bound was the same for both heuristics.

The GA algorithm of Chu and Beasley is randomized. As a result, we computed, for each test set, the average and median values for the ten executions reported in [10]. Next, we compared our upper bounds against the average value, the median value and the best solution values found by the genetic algorithm. Each of these comparisons is presented separately in Table 3.4. From the results displayed for averages and medians, we can conclude that only in 18% of the cases the results of GA outperformed those produced by LR-H. On the other hand, on 82% of the cases our heuristic produced results at least as good as the ones returned by GA. As shown in the last columns, only when we restrict ourselves to the best solutions generated by GA, the latter can be said to slightly outperform LR-H.

Thus, besides its ability to compute very good dual bounds, our Relax-and-Cut algorithm also proved very effective in generating good quality SPP upper bounds.

A few words on the sophisticated heuristic described in [1] are in place. This algorithm

	Instances							
	nw13	nw17	nw33	kl01	us04	aa06	cyc06	cyc08
Time reduction (%)	52.2	30.2	42.7	68.7	39.9	65.1	99.9	60.6
Pool reduction (%)	97.5	89.6	96.6	99.0	97.4	99.5	99.9	99.7
LB improvement (%)	5.1	0.4	0.0	0.0	0.1	0.2	0.0	0.2

Table 3.5: Dominance check benefits over the selected SPP instances.

solves to optimality all nw instances in set **I1**. Decisive for such performance was the aggressive strategy for reducing the problem sizes. For practical instances from the airline industry, this often spares the method to resort to cutting planes. We cannot predict if these problem reduction technique perform well in instances with characteristics close to those in the set **I2**.

Dominance checking results. As mentioned earlier, a noteworthy contribution of this paper (to the implementation of Relax-and-Cut algorithms) refers to the usage of dominance rules in the management of the cut pool. In cutting plane algorithms for ILP, the same cut is never generated twice and no special treatment is needed to avoid repetitions in the cut pool. For the case of Lagrangian relaxation, this does not necessarily happen. For instance, LRPs in different SM iterations may have the same solution which, when processed by deterministic separation routines, may produce the same violated cuts again. Besides avoiding multiple repetitions of cuts in the pool, we also discard the insertion of dominated inequalities in the data structures. To our knowledge, none of the earlier implementations of Relax-and-Cut algorithms included any special treatment for dominance. However, our experiments indicate that processing cuts to avoid dominance and repetition dramatically affects the performance of the method.

Table 3.5 highlights the benefits attained by a Relax-and-Cut algorithm that includes dominance checks. These tests were performed on a subset of instances arbitrarily chosen among those not solved to proven optimality by the Relax-and-Cut algorithm. The table indicates, for each instance, the reductions attained with our algorithm after the inclusion of a cut dominance checking procedure, both in terms of execution time and of cut pool size. It also shows the corresponding lower bound improvement. As an example, for instance nw13, running time was reduced by 52.2% and cut pool storage space was reduced by 97.5%. Furthermore, a lower bound improvement of 5.1% was also reached.

Notice that, for the eight instances tested, dominance checking led to a drastic decrease on the number of cuts in the pool. Likewise, substantial savings in running time were also attained (despite the additional book keeping cost involved). Besides that, the resulting lower bounds were always at least as large as their previous corresponding values. Finally,

Instance	% improvement
nw04	0.15
nw13	0.07
nw17	0.31
nw18	0.27
nw23	0.29
nw29	0.57
nw33	0.00
nw36	0.10
us04	0.03
aa01	0.16
aa03	0.03
aa04	0.18
aa05	0.02
aa06	0.01
kl01	0.00
kl02	0.00

Table 3.6: LB improvements due to lifting: **I1** dataset.

it is worth mentioning that, for some instances, convergence problems occurred when dominance checking was not used. That was probably caused by the (much larger) number of inequalities being dualized.

Lifting procedure results. Computational experiments were also conducted to assess the benefits attained through the use of the lifting procedure. Through these experiments, our intent was to measure the eventual reductions in pool size reached with the use of the procedure. Notice that no lifting is applied while an instance is being processed at the **RC** module. Therefore our computational experiments only apply to the subgroup of 21 instances where the lifting procedure was eventually used.

For all such instances we compared the dual bounds computed prior to the execution of module `LIFT` and after that of module `LRC` (i.e., after lifting took place) in Figure 3.1. Results for sets **I1** and **I2** are summarized in Tables 3.6 and 3.7, respectively. One can see that lifting led to dual bounds that are at least as good as those attained when the procedure is not applied. Though the gains obtained for the first test set were typically smaller than 1%, substantial larger gains were attained for test set **I2**, especially for instances `cyc10` and `cyc11`.

Instance	% improvement
cyc06	0.0
cyc08	0.1
cyc09	8.5
cyc10	93.2
cyc11	103.0

Table 3.7: LB improvements due to lifting: **I2** dataset.

reduction ranges	$[0-1)$	$[1-25)$	$[25-50)$	$[50-75)$	$[75-100]$
% of instances	10	5	47	19	19

Table 3.8: Pool reduction due to lifting procedure.

Significant reductions in the pool size were also observed while experimenting with the lifting procedure. These reductions can be appreciated by examining Table 3.8. As one may observe, for 85% of the instances tested, pool size reductions (brought about by the use of the lifting procedure) ranged from 25% to 100%. For 38% of the instances tested, pool sizes were reduced to less than 50% of their original sizes. These substantial reductions made it possible for us to tackle the larger instances in set **I2** through our proposed hybrid algorithm.

Odd cycle inequalities. An odd cycle inequality can be stated as $x(C) \leq k$, where k is a positive integer and C is the set of nodes of a cycle with $|C| = 2k+1$. It is well-known that odd cycle inequalities are valid for SPP. They were used by Hoffman and Padberg [22] but have been neglected by many authors. In their work on the stable set problem [35], Nemhauser and Sigismondi reported that these cuts are effective only for low density graphs. In his PhD thesis [6], Borndörfer says that the results of cutting plane algorithms (with linear programming solvers) for classes of valid inequalities other than cliques were disappointing, reporting that their only effect was to increase running times. Also Atamtürk, Nemhauser and Savelsbergh [1] did not apply other cutting planes than the clique inequalities. Finally, it is worth noting that the `XPRESS` solver includes separation for clique inequalities but not for odd cycles.

Despite those many indications from the literature that the odd cycle inequalities were not very effective to solve SPP, we carry out tests with these cuts. We decided to report them here for two reasons. First, we believe this will contribute to the debate on whether or not these cuts are useful for SPP computations. Secondly, the separation of odd cycle

inequalities for binary vectors, like the solution of Lagrangian subproblems, differs from that of fractional points and led to an elegant polynomial solution.

The commonly used modeling to find odd cycles inequalities violated by a fractional optimal solution $\bar{x}$ on a graph $G = (V, E)$ goes as follows. First it creates a bipartite graph $G' = (V \cup V', E')$. For every edge $(u, v) \in E$, it creates edges (u, v') and $(v, u') \in E'$ and assigns to them identical weights $w_{(u,v')} = w_{(v,u')} = 1 - \bar{x_u} - \bar{x_v}$. Given that, the identification of a violated odd cycle inequality reduces to find a minimum weight path from $v_0 \in V$ to $v'_0 \in V'$ in G' (see [35] for details).

But, Lagrangian subproblems have integer solutions and the modeling just as above leads to a different situation. Similarly, let $\bar{x}$ be an optimal solution to a Lagrangian subproblem and $G = (V, E)$ be a graph where to each edge $(u, v) \in E$ is assigned a weight $w_{(u,v)} = 1 - \bar{x_u} - \bar{x_v}$. Clearly, $w_{(u,v)} \in \{-1, 0, 1\}$. Now, given an odd cycle C, if the corresponding cycle inequality is violated by $\bar{x}$, then $\bar{x}(C) \geq k + 1$. In this case, if we compute the sum of the edge weights along the cycle, we obtain:

$$\sum_{(u,v)\in C} w(u, v) = 2k + 1 - 2\bar{x}(C) \leq 2k + 1 - 2(k + 1) = -1.$$

Therefore, the search for a violated odd cycle inequality can be cast as the problem of finding a negative cycle in an undirected graph which, in turn, can be reduced in polynomial time to the problem of finding a shortest T-join in an auxiliary graph [39]. Hence, apart from the effort needed to do such reduction, the detection algorithm has complexity $O(n^3)$ (for details, see chapter 29 in [39]). We conclude that the separation of odd cycle inequalities can be done in polynomial time.

However, we developed a fast heuristic separation routine by restricting our search to odd cycles belonging to the intersection subgraphs induced by variables currently at value one. Clearly, in this case, every cycle has negative weight. Thus, our problem can be solved quickly by a modified version of the standard breadth-first-search on graphs, which stops whenever the first odd cycle is found. Once such a cycle has been identified, a lifting procedure is performed similar to that described in [35].

Unfortunately, the results we obtained with the cycle inequalities were very disappointing. Minute improvements on the dual bounds were observed in few instances and at the expense of an increasing in running time. In particular, for the difficult instances in the `cyc` series no gain at all was observed.

C2 (Pg.54)

3.6 Conclusions and future works

In this paper an approach based on a Relax-and-Cut algorithm was proposed and implemented for the Set Partitioning problem. The basic idea of the algorithm is to embed a

cutting-plane procedure into a Lagrangian relaxation framework. At each iteration of the Subgradient Method, valid inequalities that are violated at the solution of an associated Lagrangian relaxation problem are identified (separated) and dualized. In our Relax-and-Cut implementation, separation was restricted to the class of clique inequalities. Besides Relax-and-Cut algorithm development, a scheme involving preprocessing, variable fixing and a simple lifting procedure was also devised.

Computational results were obtained for instances from the literature that are known to have integrality gaps. For comparison purposes, two additional Lagrangian algorithms were also coded. One where all original SPP constraints are dualized (3.10), an other where a knapsack constraint was incorporated to the model and kept in the Lagrangian subproblem (3.11). Since, for the latter case, the subproblem is solved exactly via dynamic programming, in theory, the lower bound returned by that Lagrangian relaxation is no smaller than the LP relaxation bound of (3.6). This also applies for our Relax-and-Cut algorithm, because the additional clique cuts may cutoff the fractional LP solution.

Computational results demonstrated that our Relax-and-Cut framework is competitive, in terms of CPU time, with the other two Lagrangian algorithms considered here. More importantly, it always generated equivalent, if not better, lower bounds than the others. As a result, optimality was proven for a substantial number of test instances. Furthermore, our approach succeed in solving, in approximately 7 CPU minutes, a SPP instance which could not be solved by the commercial ILP solver XPRESS in 10 CPU hours.

These results were attained through an implementation that carefully dealt with dominance relations among the generated cuts. Tests that were carried out have shown that the treatment of dominance that we proposed here was crucial for the good performance of the algorithm.

Moreover we also proposed a hybridization of the method (in combination with ILP solvers). The hybrid algorithm, which uses our Relax-and-Cut framework for preprocessing, proved to be very effective for hard SPP instances from the literature. Its use led to *huge* speed ups over XPRESS.

As for primal bounds, our simple Lagrangian heuristic turned out to be competitive with good algorithms recently suggested in the Set Partitioning literature. Though we focused on improving dual bounds, our heuristic was capable of generating high-quality solutions, with optimal values being attained for 64% of the instances tested.

Further developments and implementation issues should be considered to possibly improve the performance of our Relax-and-Cut algorithm. According to the authors, the algorithm presented in [1] relies on an aggressive preprocessing procedure that is repeated several times inside a loop, removing dominated rows and duplicated columns. This is done in such a way that in many cases not a single cut had to be generated to

prove optimality. This approach has proved to be very effective for real world instances where logical implications help to diminish the instance sizes. We could think of adding a similar preprocessing mechanism to our code. However, it should be understood that, the instances solved completely through reduction techniques, like some of those tested in [1], are not the ones that are best suited to the application of cutting planes.

Other natural issues to consider would be to separate different known classes of valid inequalities for SPP and to devise a cleverer primal heuristic. Despite its surprisingly good performance, our primal heuristic is rather simple and improvements for it should possibly be conceived, maybe embedding some of the ideas of the the Lagrangian heuristic from [1].

Acknowledgments. Cid de Souza was supported by grants 307773/2004-3 from CNPq and 03/09925-5 from FAPESP. Abilio Lucena was supported by grants 300149/94-8 from CNPq and E26/71.906/00 from FAPERJ. These authors are also supported by CNPq, Project PROSUL-Proc. no. 490333/04-4.

3.7 Comentários

C1: Experimentos com grafos de interseção

As tabelas 3.9 e 3.10 mostram os resultados obtidos por duas versões do *relax-and-cut* que diferem apenas na rotina de separação de desigualdades clique. A primeira versão (veja coluna `R&C(1)`) mostra os resultados do nosso algoritmo em sua versão final, i.e., onde busca-se identificar desigualdades clique a partir de grafos induzidos apenas por variáveis com valor um na solução do subproblema Lagrangiano. Note que, posteriormente, as desigualdades violadas encontradas desta forma serão candidatas ao processo de *lifting*, o qual é feito com base no grafo de interseção completo, ou seja, construído considerando todas as variáveis, e não epenas aquelas com valor um na solução do subproblema Lagrangiano. Em contrapartida, as colunas sob `R&C(2)` mostram os resultados obtidos quando as desigualdades clique são identificadas diretamente sobre o grafo de interseção completo.

Observe que, em `R&C(2)`, o grafo de interseção é construído apenas uma vez, no início da execução do algoritmo. Hoffman e Padberg [22] apresentaram formas de evitar trabalhar com o grafo inteiro durante a execução da rotina de separação. Basicamente, eles começam com um subgrafo induzido por alguma solução fracionária de uma relaxação linear do SPP, de forma similar ao que é feito aqui para se obter os resultados em `R&C(1)` os quais, em seguida, são submetidos ao processo de *lifting*.

Os resultados revelam que, em geral, nossa abordagem final, i.e., `R&C(1)`, é bem mais rápida que `R&C(2)`, especialmente para as instâncias maiores/mais difíceis. Note que,

Tabela 3.9: Desempenho do R&C para estratégias distintas de separação de desigualdades clique.

Instance	Optimal	LP	R&C (1)			R&C (2)		
			ub	*lb*	$t(s)$	*ub*	*lb*	$t(s)$
nw03	24492	2447	24492	24492	65.35	24492	24492	65.13
nw04	16862	16310.67	17856	16328	268.17	18796	16297	2885.65
nw06	7810	7640	7810	7810	18.89	7968	7748	147.87
nw11	116256	116254.5	116256	116256	8.54	116256	116256	8.53
nw13	50146	50132	50146	50140	18.13	50146	50119	27.99
nw17	11115	10875.7	11295	10914	243.49	11115	10871	984.72
nw18	340160	338864.3	340160	340160	142.35	340160	340149	2422.72
nw20	16812	16626	16812	16812	0.91	16812	16812	1.27
nw21	7408	7380	7408	7408	0.42	7408	7408	0.42
nw22	6984	6942	6984	6984	0.56	6984	6984	0.56
nw23	12534	12317	12534	12534	1.27	12534	12534	1.28
nw24	6314	5843	6314	6314	1.11	6314	6314	1.11
nw25	5960	5852	5960	5960	1.06	5960	5960	1.44
nw26	6796	6743	6796	6796	0.62	6796	6796	0.88
nw27	9933	9877.5	9933	9933	1.00	9933	9933	0.99
nw28	8298	8169	8298	8298	0.86	8298	8298	0.87
nw29	4274	4185.9	4430	4257	7.79	4424	4211	21.36
nw30	3942	3726.8	3942	3942	2.34	3942	3942	2.38
nw31	8038	7980	8038	8038	2.17	8038	8038	2.18
nw32	14877	14570	14877	14877	0.24	14877	14877	0.23
nw33	6678	6484	7840	6536	6.86	7320	6536	12.56
nw34	10488	10453.5	10488	10488	0.81	10488	10488	0.81
nw35	7216	7206	7216	7216	1.58	7216	7216	1.68
nw36	7314	7260	7328	7281	20.05	7328	7256	65.48
nw37	10068	9961.5	10068	10068	0.67	10068	10068	0.67
nw38	5558	5553	5558	5558	0.99	5558	5558	0.98
nw39	10080	9868.5	10080	10080	0.59	10080	10080	0.64
nw40	10809	10658.3	10809	10809	0.32	10809	10809	0.32
nw41	11307	10972.5	11307	11307	0.16	11307	11307	0.16
nw42	7656	7485	7656	7656	0.97	7656	7656	0.97
nw43	8904	8897	8904	8904	1.06	8904	8904	1.06

em vários casos, os limitantes inferiores gerados por `R&C(2)` não são melhores mas sim, usualmente, piores do que aqueles obidos por `R&C(1)`. A vantagem de se limitar o grafo de interseção torna-se mais evidente analisando-se os resultados para as instâncias com maior número de linhas; por exemplo, aquelas rotuladas como `aa` e `cyc`. Como mencionado na descrição dos parâmetros específicos na seção 3.5, o número de nós de qualquer sub(grafo) de interseção em `R&C(1)` foi limitado a $\min\{200, m\}$ (ou $\min\{500, m\}$ para a instância `cyc11`). Finalmente, vale ressaltar que a ausência de entrada para a instância `cyc11` na tabela 3.10 sob `R&C(2)` se deve ao enorme tempo gasto para manuseio do grafo de interseção (inteiro) durante a rotina de separação.

Como conclusão, estes experimentos confirmam que, conforme proposto pelo fluxo de

execução do nosso *framework*, trabalhar com grafos de interseção menores e em seguida efetuar o *lifting* das desigualdades sobre o grafo inteiro é a estratégia mais promissora dentre as duas analisadas.

É importante ressaltar também que outras estratégias foram consideradas em nossos experimentos. Por exemplo, buscou-se identificar cliques maximais a partir de grafos de interseção compostos por todos os nós cujas variáveis correspondentes foram parte de alguma solução ótima do subproblema Lagrangiano em alguma iteração anterior. Infelizmente, dado que o subgrafo resultante geralmente crescia muito rapidamente a cada iteração, esta estratégia não obteve bom desempenho e foi abandonada.

Tabela 3.10: Desempenho do R&C para estratégias distintas de separação de desigualdades clique (cont.)

Instance	Optimal	LP	R&C (1)			R&C (2)		
			ub	lb	$t(s)$	ub	lb	$t(s)$
us04	17854	17731.7	17907	17834	210.00	17865	17830	2068.74
aa01	56138	55535.4	57528	55393	147.76	57528	55009	4084.25
aa03	49649	49616.4	50747	49627	121.48	50747	49583	3883.67
aa04	26374	25877.6	29888	25956	114.85	29888	25913	2165.92
aa05	53839	53735.9	54014	53703	113.91	55160	53600	3558.65
aa06	27040	26977.2	27293	26998	91.49	27293	26986	2606.48
kl01	1086	1084	1086	1085	11.89	1088	1085	26.78
kl02	219	215.3	220	216	275.33	221	215	2510.54
cyc06	112	48	120	112	25.18	120	112	104.79
cyc07	352	112	352	352	4.02	352	352	41.49
cyc08	1024	256	1048	1024	86.77	1048	1024	6156.42
cyc09	2816	576	2844	2816	234.18	2844	2816	44340.24
cyc10	7424	1280	7424	7424	435.05	7424	7424	35713.79
cyc11	–	2816	19016	18944	2316.4	19016	18944	–

C2: Experimentos com desigualdades de ciclo ímpar

Em nossa implementação, a regra responsável pela gerência do *pool* de desigualdades do tipo *lifted odd cycle* (LOH) é muito simples: um novo corte só é inserido se não for idêntico a nenhum outro presente no *pool*. É também verificado se o novo corte é dominado por uma desigualdade clique presente no *pool* de desigualdades clique.

Experimentos computacionais foram realizados com o objetivo de avaliar vários aspectos de duas principais configurações dos nossos algoritmos R&C: `(a) R&C(C)`, refere-se à implementação do R&C separando/dualizando apenas desigualdades clique; `(b) R&C(C+LOH)`, correspondente ao algoritmo R&C onde ambas, cliques e desigualdades de ciclo ímpar, são identificadas e dualizadas. Nestes experimentos, uma instância foi submetida a `(b)` somente quando `R&C(C)` não conseguiu resolvê-la à otimalidade. As exceções

foram `cyc07` e `cyc09`. Nós decidimos manter estas instâncias nos testes com `R&C(C+LOH)` porque o algoritmo *branch-and-bound* do XPRESS (versão 16.01.05, configuração padrão) encontrou dificuldades em resolvê-las.

As sete primeiras colunas da tabela 3.11 mostram, para cada instância selecionada, os limitantes alcançados por ambas as configurações, bem como os tempos computacionais necessários para atingir os valores listados. As quatro colunas seguintes apresentam os limitantes da relaxação linear obtidos quando excutando o resolvedor de PL do XPRESS sobre: `(i)` a formulação original; `(ii)` a formulação original acrescida das desigualdades LOH identificadas durante a execução do R&C separando apenas LOHs; `(iii)` a formulação original acrescida das desigualdades clique identificadas durante a execução de R&C separando somente desigualdades clique; `(iv)` a formulação original acrescida de ambas, desigualdades clique e LOH, identificadas quando executando o R&C para separação de ambas as classes de desigualdades. Estes limitantes nos permitiram avaliar o *gap* de integralidade para cada instância/configuração (veja tabelas 3.9 e 3.10 para valores ótimos de limitantes superiores). Finalmente, a última coluna mostra a percentagem de ganho/perda, em termos de *gap*, resultante da introdução da separação das desigualdades de ciclo ímpar.

Tabela 3.11: Resultados computationais para instâncias do SPP.

Instance	R&C (C)			R&C (C+LOH)			LP	XPRESS(LP+R&C (Cuts))			
	ub	*lb*	*t*(*s*)	*ub*	*lb*	*t*(*s*)		*LOH*	*C*	*C* + *LOH*	*Gain*(%)
nw04	17856	16328	268.2	17856	16326	314.4	16310.7	16310.7	16339.4	16335.1	-0.03
nw13	50146	50140	18.1	50146	50142	22.0	50132.0	50132.0	50146.0	50146.0	0.00
nw17	11295	10914	243.5	11115	10921	252.3	10875.7	10881.4	10954.0	10977.9	0.22
nw29	4430	4257	7.8	4430	4257	9.5	4185.9	4185.3	4257.0	4257.0	0.00
nw33	7840	6536	6.9	6678	6636	7.5	6484.0	6607.3	6536.0	6636.0	1.53
nw36	7328	7281	20.1	7328	7286	23.0	7260.0	7260.0	7282.5	7286.0	0.05
us04	17907	17834	210.0	17854	17826	208.3	17731.7	17745.1	17846.3	17846.3	0.00
aa01	57528	55393	147.8	57528	55269	157.3	55535.4	55542.9	55632.4	55631.3	0.00
aa03	50747	49627	121.5	50747	49604	126.4	49616.4	49629.5	49637.3	49649.0	0.02
aa04	29888	25956	114.9	29888	25953	119.8	25877.6	25885.4	25974.3	25975.0	0.00
aa05	54014	53703	113.9	55160	53684	122.1	53735.9	53735.9	53727.4	53739.1	0.02
aa06	27293	26998	91.5	27293	26999	95.1	26977.2	26996.3	27006.2	27011.5	0.02
kl01	1086	1085	11.9	1088	1085	14.0	1084.0	1084.1	1084.1	1085.0	0.08
kl02	220	216	275.3	220	216	276.5	215.3	215.0	215.3	215.3	0.00
$\sum$			1641.4			1748.2					
cyc06	120	112	25.2	120	112	23.4	48.0	48.0	112.0	112.0	0.00
cyc07	352	352	4.0	352	352	35.0	112.0	112.0	352.0	352.0	0.00
cyc08	1048	1024	86.8	1048	1018	92.3	256.0	256.0	1024.0	1021.0	-0.29
cyc09	2844	2816	234.2	2844	2816	238.1	576.0	576.0	2816.0	2816.0	0.00
cyc10	7424	7424	435.1	7424	7410	649.0	1280.0	1280.0	7424.0	7424.0	0.00
cyc11	19016	18944	2316.4	19016	18927	2414.5	2816.0	2816.0	18944.0	18944.0	0.00
$\sum$			3101.7			3452.3					
$\sum$			4743.1			5200.5					

A análise destes resultados foi dividida por grupos de instâncias. Primeiro, considere os resultados sobre o subconjunto de instâncias provenientes da OR-Library (denotado por **I1**). Inspecionando-se as colunas relativas a `R&C(C)` e a `R&C(C+LOH)`, podemos concluir que não há dominância em termos de qualidade dos limitantes (superiores ou inferiores) e que o tempo total gasto na separação de LOH é um pouco maior (aproximadamente 6.5%). Contudo, os resultados obtidos com o resolvedor de PL do XPRESS revelam que os cortes originários de `R&C(C+LOH)` melhoraram os limitantes duais em aproximadamente 50% dos casos. Apenas em dois deles o uso de desigualdades clique sozinhas produziu melhor limitante.

Realizando-se a mesma análise para o conjunto de instâncias denotado por **I2** (advindas originalmente de instâncias do SCP), os resultados foram favoráveis ao uso apenas de desigualdades clique. As colunas relativas às execuções de algoritmos *relax-and-cut* mostram que `R&C(C)` tem um desempenho um pouco melhor que `R&C(C+LOH)`, em termos de limitantes duais e, para 5 dos 6 casos, `R&C(C)` é mais rápido que `R&C(C+LOH)`. O tempo total para identificação de desigualdades de ciclo ímpar também é aproximadamente 11% maior que o tempo gasto quando somente desigualdades clique são identificadas. Para estas instâncias, os resultados com o resolvedor de PL do XPRESS mostram que os cortes vindos de `R&C(C+LOH)` nunca melhoram os limitantes duais obtidos pela formulação com cortes gerados por `R&C(C)`, sendo pior para `cyc08`.

Finalmente, a última linha da tabela 3.11 mostra que a separação de desigualdades de ciclo ímpar causou uma aumento de aproximadamente 457 segundos do tempo total de execução do *relax-and-cut*.

Tabelas 3.12 e 3.13 mostram resultados da abordagem híbrida obtida com formulações fortalecidas com a introdução, respectivamente, de cortes (ativos) gerados por `R&C(C)` e `R&C(C+LOH)`. Estes testes foram realizados para as instâncias consideradas mais difíceis no subconjunto **I2**. As tabelas mostram, para cada instância: o valor do melhor limitante superior passado para o XPRESS; o *gap* entre os valores da relaxação linear e do limitante dual produzido pelo algoritmo R&C, e o tempo requerido até o final do procedimento de *lifting*. Na tabela, `XPC` refere-se aos limitantes produzidos pelo XPRESS e aos tempos gastos pata obtê-los. A última coluna mostra o tempo total requerido pela abordagem híbrida.

Nós decidimos separar os resultados para `cyc11` dos demais porque o tempo limite de 10 horas foi alcançado ainda enquanto o primeiro LP estava sendo resolvido para esta instância! Estes experimentos revelam que o uso das desigualdade de ciclo ímpar trouxe apenas uma contribuição marginal na redução do tempo total de processamento para as duas menores instâncias `cyc08` e `cyc09`. Para `cyc10`, separar apenas desigualdades clique pareceu muito mais apropriado e, para `cyc11`, adicionar desigualdades LOH revelou deteriorar o desempenho do resolvedor de PL.

Compilando todos estes experimentos concluímos que as desigualdades de ciclo ímpar não contribuiram de forma significativa para melhorar os limitantes duais. Assim, acreditamos que os ganhos marginais obtidos no cálculo para as instâncias de **I1** não justificam a inclusão de desigualdades de ciclo ímpar como parte da configuração final do nosso *framework* de R&C.

Tabela 3.12: Resultados para instância do SPP usando R&C como pré-processamento: apenas desigualdades clique são identificadas.

Instance	R&C (C)				XPC			Time (s)
	ub	*LP gap*	$t1(s)$	*pool*	*ub*	*lb*	$t2(s)$	$t1+t2$
cyc08	1048	2.34	63.43	256	1024	1024	67	130.43
cyc09	2844	0.99	163.54	512	2816	2816	838	1001.54
cyc10	7424	0.00	423.29	1024	7424	7424	24277	24700.29
$\sum$								25832.26
cyc11	19016	0.38	1530.94	2048	–	18944	36000	37530.94

Tabela 3.13: Resultados para instâncias do SPP usando R&C como pré-processamento: desigualdades clique e de ciclo ímpar são identificadas.

Instance	R&C (C+LOH)				XPC			Time (s)
	ub	*LP gap*	$t1(s)$	*pool*	*ub*	*lb*	$t2(s)$	$t1+t2$
cyc08	1048	2.64	68.15	1244	1024	1024	60	128.15
cyc09	2844	0.99	158.63	1829	2816	2816	842	1000.63
cyc10	7424	0.00	429.41	2513	7424	7424	27455	27884.41
$\sum$								29013.19
cyc11	19016	0.38	1617.57	3330	–	18944	36000	37617.57

Bibliography

[1] A. Atamtürk, G.L. Nemhauser, and M.W.P. Savelsbergh. A combined Lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems. *Journal of Heuristics*, 1:247–259, 1996.

[2] E. Balas. Some valid inequalities for the set partitioning. *Annals of Discrete Mathematics*, 1:13–47, 1977.

[3] E. Balas and N. Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21:19–46, 1981.

[4] E. Balas and M.W. Padberg. *Combinatorial Optimization*, chapter Set Partitioning - A Survey. John Wiley and Sons, Chichester, NY, Brisbane, Toronto, 1979.

[5] J.E. Beasley. *Modern Heuristic Techniques*, chapter 6. Blackwell Scientific Press, Oxford, 1993.

[6] R. Borndörfer. *Aspects of Set Packing, Partitioning and Covering.* PhD thesis, Faculty of Mathematics at Tech.University of Berlin, 1998.

[7] C.Y Byun. *Lower Bounds for Large Scale Set Partitioning Problems.* PhD thesis, Faculty of Mathematics at Tech. University of Berlin, Berlin, January 2001.

[8] F. Calheiros, A. Lucena, and C. C. de Souza. Optimal rectangular partitions. *Networks*, 41:51–67, 2003.

[9] A. Caprara, M. Fischetti, and Toth P. A heuristic method for the set covering problem. *Operations Research*, 47, 1999.

[10] P.C. Chu and J.E. Beasley. Constraint handling in genetic algorithms: The set partitioning problem. *Journal of Heuristics*, 11:323–357, 1998.

[11] J.B.C da Silva. Uma heurística Lagrangeana para o problema da Árvore geradora capacitada de custo mínimo. Master's thesis, COPPE, Federal University of Rio de Janeiro, Brazil (in Portuguese), 2002.

[12] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. *Working Paper*, 1993.

[13] J. Edmonds. Covers and packings in a family of sets. In *Bulletin of the Am. Math. Soc. 68*, pages 29–32, 1962.

[14] L. Escudero, M. Guignard, and K. Malik. A Lagrangean relax and cut approach for the sequential ordering with precedence constraints. *Annals of Operations Research*, 50:219–237, 1994.

[15] M.L. Fisher and P. Kedia. Optimal solution of set covering/partitionning problems using dual heuristics. *Management Science*, 36(6), June 1990.

[16] M.R. Garey and D.S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W.H. Freeman and Co., New York, 1979.

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

[18] F. Glover. Surrogate constraint duality in mathematical programming. *Operations Research*, 23:434–451, 1975.

[19] H.J. Greenberg and W.P. Pierskalla. Surrogate mathematical programming. *Operations Research*, 18:924–939, 1970.

[20] T. Grossman and A. Wool. Computacional experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101:81–92, 1997.

[21] M. Guignard. Lagrangean relaxation. In M.A. Lòpez-Cerdá and I. García-Jurado, editors, *Top*, volume 11, pages 151–200. Springer Berlin/ Heidelberg, 2003.

[22] K.L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6), June 1993.

[23] C.G. John and G.A. Kochenberger. Using surrogate constraint in a Lagrangian relaxation approach to set covering problems. *Journal of the Operational Research Society*, 29:681–685, 1988.

[24] M.H. Karwan and R.L. Rardin. Some relationships between Lagrangian and surrogate duality in integer programming. *Mathematical Programming*, 17:320–334, 1979.

[25] M.H. Karwan and R.L. Rardin. Searchability of the composite and multiple surrogate dual functions. *Operations Research*, 28:1251–1257, 1980.

[26] M.H. Karwan and R.L. Rardin. Surrogate duality in a branch and bound procedure. *Naval Research Logistics Quarterly*, 28:93–101, 1981.

[27] M.H. Karwan and R.L. Rardin. Surrogate dual multiplier search procedures in integer programming. *Operations Research*, 32:52–69, 1984.

[28] J.T. Linderoth, E.K. Lee, and M.W.P. Savelsbergh. A parallel, linear programming based heuristic for large scale set partitioning problems. *INFORMS J. On Computing*, 13:1–19, 2001.

[29] L.A.N. Lorena and F.B. Lopez. A surrogate heuristic for set covering problems. *European Journal of Operational Research*, 79:138–150, 1994.

[30] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. In *COAL Bulletin*, volume 21. Mathematical Programming Society, 1992.

[31] A. Lucena. Relax and cut algorithms. In *IV ALIO-EURO Meeting*, Pucon, 2002.

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

[33] R.E. Marsten and F. Shepardson. Exact solution of crew scheduling problems using the set partitioning mode: Recent successful applications partitioning. *Networks*, 11:165–177, 1981.

[34] C. Martinhon, A. Lucena, and N. Maculan. Stronger k-tree relaxations for the vehicle routing problem. *European Journal on Operational Research*, 158:56–71, 2004.

[35] G.L. Nemhauser and G. Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *J.Opl. Res. Soc.*, 43(5):443–457, 1992.

[36] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.

[37] M. Padberg. *Essays in Integer Programming*. PhD thesis, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1971.

[38] M. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(199-215), 1973.

[39] A. Schrijver. *Combinatorial Optimization*. Springer, Berlin Heidelberg, 2003.

[40] H.D. Sherali and Y. Lee. Tighter representations for set partitioning problems. *Discrete Applied Mathematics*, 1996.

[41] B. Stroustrup. *The C++ Programming Language*, chapter The Standard Library, pages 427–688. Massachusetts, Addison-Wesley, 1997, 1997.

Chapter 4

A relax-and-cut algorithm for the vertex separator problem

O presente capítulo descreve um *framework* baseado em variantes de um algoritmo *relax-and-cut* para o problema do separador de vértices introduzido por Balas e de Souza em [3]. O texto em inglês corresponde à íntegra do artigo *Exact Algorithms for the Vertex Separator Problem in Graphs* desenvolvido em co-autoria com o professor Dr. Cid C. de Souza[1] e submetido para possível publicação a um periódico internacional arbitrado. Os resultados reportados neste capítulo foram apresentados na conferência 7^{th} *Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, Gargnano - Itália, em maio de 2008. Versões preliminares deste trabalho foram apresentadas nas conferências *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE)*, Zhejiang University, Hangzhou - China, em abril de 2007 (vide apêndice B)[2] e 19^{th} *International Symposium on Mathematical Programming*, UFRJ, Rio de Janeiro - Brasil, em julho/agosto de 2006.

In this paper we propose a Lagrangian relaxation framework to solve the vertex separator problem (VSP) based on relax-and-cut algorithms. The approach developed suggests the combination of cutting planes and Lagrangian algorithms together with the hybridization of Lagrangian relaxation and Integer Programming. The several relax-and-cut versions implemented have embedded cutting plane procedures derived from valid inequalities to the VSP. Computational results obtained with benchmarks from literature turn our hybrid framework, to our knowledge, the best exact algorithm available for the

[1] Instituto de Computação, UNICAMP.

[2] Anais: *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, volume 4614/2007, páginas 471-482, ISSN 0302-9743 (versão impressa), ISSN 1611-3349 (versão digital) .

VSP to date. Besides, our experiments showed that our Lagrangian heuristic is a very effective algorithm for the vertex separator problem, often producing optimal solutions extremely fast.
Keywords: Lagrangian relaxation, cutting planes, Integer Programming, relax-and-cut algorithms, Vertex Separator Problem.

4.1 Introduction

A *vertex separator* in an undirected graph is a subset of the vertices, whose removal disconnects the graph in at least two nonempty connected components. Recently, Balas and de Souza [3, 11] studied the vertex separator problem (VSP) which can formally be stated as follows.
`INSTANCE:` a connected undirected graph $G = (V, E)$, with $|V| = n$, an integer $1 \leq b \leq n$ and a cost c_i associated with each vertex $i \in V$.
`PROBLEM:` find a partition of V into disjoint sets A, B, C, with A and B nonempty, such that *(i)* E contains no edge (i, j) with $i \in A$, $j \in B$, *(ii)* $\max\{|A|, |B|\} \leq b$, *(iii)* $\sum_{j \in C} c_j$ is minimized.

The sets A and B are called the *shores* of the separator C. A separator C that satisfies (i) but violates (ii) is termed *infeasible*; one that satisfies (i) and (ii) is *feasible*; and a separator that satisfies (i), (ii), (iii) is optimal. Unless otherwise specified, the term separator is used here to denote a feasible one. The VSP is $\mathcal{NP}$-hard and has widespread applicability in network connectivity. Further discussion on applications appears in [3].

In that paper Balas and de Souza also conducted the first polyhedral investigation on the VSP. They introduced several classes of strong valid inequalities for the polytope associated to the problem. In a companion paper to that study, the same authors reported extensive computational experiments with a branch-and-cut (B&C) algorithm based on those inequalities. A similar problem is discussed in [20] by Borndörfer *et al.* This problem can be considered a generalization of the VSP in the sense that the partitioning can be done in more than two sets of vertices. However, contrarily to the VSP, solutions where one of the shores remains empty are allowed.

Based on the Integer Programming (IP) model and on the strong valid inequalities introduced by Balas and de Souza, we propose an algorithm that combines Lagrangian relaxation with cutting plane techniques to solve the VSP. Our method belongs to a class of Lagrangian relaxation algorithms where constraints of certain families of inequalities may only be explicitly dualized when they become violated at some Lagrangian relaxation solution (see [6, 15, 17, 19]). These so-called Relax-and-Cut (R&C) algorithms appear as a promising alternative approach to strengthen Lagrangian relaxation bounds. Relax-and-Cut algorithms are presented in several recent works in literature [6, 15, 16, 17,

18, 19]. These algorithms use a dynamic inequality dualization scheme that renders viable the application of Lagrangian Relaxation to models with an exponential number of inequalities. Indeed, a similar approach for the traveling salesman problem [2] date from the early 80's.

Furthermore, we describe a framework that proposes a hybridization between our R&C algorithm and a modified version of the B&C algorithm presented in [11]. Basically, this hybridization consists in using our R&C as a preprocessing to the B&C algorithm and we denote it by Relax-and-Cut-and-Branch-and-Cut, or simply HYBRID. Such hybrid approaches were already tried on other problems [8, 9, 10], however this work presents the first attempt to use it in the exact solution of VSP instances. The results we obtain in this way are very promising, outperforming those achieved by the B&C algorithm when used alone.

The paper is organized as follows. Section 4.2 presents the IP formulation for the VSP given in [3, 11] and used here. Section 4.3 briefly reviews the Lagrangian relaxation technique and the subgradient method (SM) and gives a general description of R&C algorithms. A general description of the R&C elements that are part of the algorithmic framework we developed for the VSP is done in section 4.4. It includes details of the Lagrangian relaxations taken into account, descriptions of the separation routines implemented and of the Lagrangian heuristic we devised. Section 4.5 discusses how we integrated Lagrangian relaxation with other Integer Linear Programming techniques to design an exact algorithm to solve the VSP. Our test environment setup is detailed in section 4.6. of our algorithm and reports on the computational results obtained for test instances gathered from the literature. Finally, in section 4.8, we draw some conclusions and discuss possible extensions of this study.

4.2 An IP formulation for the VSP

We describe here the mixed IP formulation presented in [3, 11] on which our Lagrangian relaxation is based. For every vertex $i \in V$, two binary variables are defined: $u_{i1} = 1$ if and only if $i \in A$ and $u_{i2} = 1$ if and only if $i \in B$. For $S \subseteq V$ and $k \in \{1, 2\}$, let $u_k(S)$

denote $\sum(u_{ik} : i \in S)$, and $u(S) = u_1(S) + u_2(S)$. An IP model for the VSP is given by

$$\max \sum_{i \in V} c_i(u_{i1} + u_{i2})$$

$$u_{i1} + u_{i2} \leq 1, \qquad \forall\ i \in V \tag{4.1}$$

$$u_{i1} + u_{j2} \leq 1,\ u_{j1} + u_{i2} \leq 1, \qquad \forall\ (i,j) \in E \tag{4.2}$$

$$u_1(V) \geq 1, \tag{4.3}$$

$$u_2(V) \leq b, \tag{4.4}$$

$$u_1(V) - u_2(V) \leq 0, \tag{4.5}$$

$$u_{i2} \geq 0,\ u_{i1} \in \{0,1\}, \qquad \forall\ i \in V. \tag{4.6}$$

Inequality (4.1) forces every vertex to belong to at most one shore. Inequalities (4.2) prohibits the extremities of an edge to be on distinct shores. Inequalities (4.3) to (4.5) limit the size of the shores and, at the same time, reduce the symmetry of the model by forcing the size of shore A to be bounded by that of shore B. As observed in [11], if the u_{i1} variables are integer for all $i \in V$, the integrality of the u_2 variables can be dropped from the formulation. Though this observation is not taken into account by our Lagrangian relaxation, it is relevant for IP solvers.

4.3 Relax-and-Cut (R&C) algorithms

For completeness, we briefly review the basics on Lagrangian relaxation and relax-and-cut algorithms that are relevant to us. Denote by X a subset of $\mathbb{B}^n = \{0,1\}^n$ and let

$$Z = \max \{cx : Ax \leq b,\ x \in X\} \tag{4.7}$$

be a formulation for a $\mathcal{NP}$-hard combinatorial optimization problem. In association with (4.7) one has $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$, where m and n are positive integral values representing, respectively, the number of constraints and the number of variables involved. Assuming that m is an exponential function of n, let Z' denote the formulation obtained after removing constraints $Ax \leq b$ from (4.7). Also, assume that Z' can be solved in polynomial or pseudo-polynomial time in the problem size.

A Lagrangian relaxation of (4.7) is obtained by bringing the term $\lambda(b - Ax)$ into the objective function of Z', where $\lambda \in \mathbb{R}^m_+$ is the corresponding vector of Lagrange multipliers. The resulting *Lagrangian relaxation Problem* (LRP) is

$$Z(\lambda) = \max \{cx + \lambda(b - Ax) :\ x \in X\} = \max \{(c - \lambda A)x + \lambda b :\ x \in X\}. \tag{4.8}$$

It is a known fact that $Z(\lambda) \geq Z$ and, therefore, the tightest possible upper bound on Z, attainable through $LRP(\lambda)$, is given by an optimal solution to the *Lagrangian dual problem* (LDP) $Z_D = \min_{\lambda \in \mathbb{R}^m_+} \{\max \{(c - \lambda A)x + \lambda b : x \in X\}\}$. In the literature, several methods exist to compute the LDP. Among these, due to its simplicity and the acceptable results it returns, the subgradient method (SM) is the most widely used [5]. A brief review of that method follows since the R&C algorithm we suggest here for the VSP is deeply based on SM.

SM is an iterative algorithm which solves a succession of LRPs like the one in (4.8). It starts with a feasible vector λ^0 of Lagrangian multipliers and, at iteration k, generates a new feasible vector λ^k of multipliers and an associated LRP.

At iteration k, let $\bar{x}^k$ be an optimal solution to (4.8) with cost $Z(\lambda^k)$ and let z^k_{LB} be a known lower bound on (4.7). An associated subgradient vector (for the m relaxed constraints) is then computed as $g^k_i = (b_i - a_i\bar{x}^k),\ i = 1, 2, \ldots, m$. That vector is then used to update λ^k. To that order, a *step size* θ^k must be computed first. Typically, for a real valued parameter $\pi^k \in (0, 2]$, formula

$$\theta^k = \frac{\pi^k(Z(\lambda^k) - z^k_{LB})}{\sum_{i=1}^m (g^k_i)^2} \tag{4.9}$$

is used [5]. Finally, once θ^k is obtained, λ^k is updated as

$$\lambda^{k+1}_i = \max\{0; \lambda^k_i - \theta^k g^k_i\},\ i = 1, 2, \ldots, m. \tag{4.10}$$

Notice that the straightforward use of formulas (4.9-4.10) may become troublesome when a *huge* number of dualized inequalities exist. An alternative may be to modify SM according to the R&C scheme discussed below.

In the literature two strategies to implement R&C algorithms are discussed. They differ, basically, on the moment at which the new inequalities are identified and dualized. In a Delayed Relax-and-Cut (DR&C), several executions of SM are made and the cuts found during one such execution are added only at the beginning of the next one. In a Non Delayed Relax-and-Cut (NDR&C), typically a single SM run is done and cuts are dualized along the iterations as they are found (see [6, 15, 17, 18, 19] for details). In a comparison carried out in [18], NDR&C performed better than DR&C. However, in our work, we decide to implement both strategies in order to compare them in the context of the VSP. Also, we propose a third strategy which combines ideas borrowed from the previous ones. We denote it by *Postponed (non-delayed) Relax-and-Cut* (PR&C). As for NDR&C, in PR&C the cuts are separated at each SM iteration. However, these cuts are not immediately dualized. Instead, they are stored in a buffer. Then, similarly to what happens in DR&C, various rounds of SM iterations are executed. However, in the beginning of each round, the buffer is emptied and all its cuts are dualized for the next SM round.

Clearly, if there are exponentially many inequalities in (4.7), the use of traditional Lagrangian relaxation becomes impracticable. Alternatively the R&C scheme proposes a dynamic strategy to dualize inequalities. In this process, one should be able to identify inequalities that are violated at $\bar{x}^k$. To do so, likewise polyhedral cutting-plane generation, a separation problem must be solved at every iteration of SM. Thus, one tries to find at least one inequality violated by the current LRP solution. The inequalities thus identified are candidates to be dualized. It is worth noting that separation problems arising in R&C algorithms may be easier than their polyhedral cutting-plane algorithm counterparts. That applies since LRP normally has integral valued solutions (cf. [19]).

4.4 Relax-and-cut algorithms for the VSP

Different Lagrangian relaxations can be devised from the formulation given in section 4.2. During this work we evaluated some of them, always considering the trade-off between two aspects: `(a)` the strength (sharpness) of the resulting Lagrangian dual bounds and `(b)` the difficulty of solving the Lagrangian primal and dual problems, which influence on the amount of computation required to obtain the bounds. With this in mind, we considered three relaxations, all of which can be easily seen to satisfy the integrality property. Then, in all three cases, the best dual bound attainable is equal to the value of the VSP linear programming relaxation. Therefore, what prevailed in our choice of the Lagrangian relaxation to be used was the computational effort involved in solving the LRP and LDP problems.

We decided to start with a simple relaxation where the constraint sets (4.1) and (4.2) are dualized by means of the vector multipliers $\lambda \in \mathbb{R}_+^{|V|}$, $\beta^1 \in \mathbb{R}_+^{|E|}$ and $\beta^2 \in \mathbb{R}_+^{|E|}$, respectively. Also, observe that symmetry is not of primary concern for the Lagrangian relaxation. Thus, we consider an alternative IP formulation where the inequalities (4.3) and (4.4) are replaced, respectively, by $1 \leq u_l(V) \leq b$, with $l = 1, 2$, and inequality (4.5) is dropped. Accordingly, the resulting LRP is given by

$$\begin{aligned} \mathrm{LRP}(\lambda, \beta^1, \beta^2) = \max \{ & \sum_{i \in V} (\bar{c}_{i1} u_{i1} + \bar{c}_{i2} u_{i2} + \lambda_i) + \sum_{\substack{(i,j) \in E \\ i<j}} (\beta^1_{i,j} + \beta^2_{i,j}) : u_{kl} \in \{0,1\}, \\ & \forall\, k \in V \text{ and } l = 1, 2, \text{ satisfying } 1 \leq u_l(V) \leq b \text{ and (4.6)} \} \end{aligned} \tag{4.11}$$

where $\bar{c}_{k1} = c_k - \lambda_k - \sum_{\substack{(k,j) \in E \\ k<j}} \beta^1_{k,j} - \sum_{\substack{(i,k) \in E \\ i<k}} \beta^2_{i,k}$ and $\bar{c}_{k2} = c_k - \lambda_k - \sum_{\substack{(i,k) \in E \\ i<k}} \beta^1_{i,k} - \sum_{\substack{(k,j) \in E \\ k<j}} \beta^2_{k,j}$ are the Lagrangian costs of, respectively, u_{k1} and u_{k2}. Notice that (4.11) can be solved in $O(|V| \log |V|)$ time by sorting the variables according to their Lagrangian costs and after performing a few simple calculations.

The second relaxation we experimented is very similar to the first one, differing only by the fact that inequality (4.1) is not dualized anymore. The resulting LRP is thus

$$\begin{aligned}\mathrm{LRP}(\beta^1,\beta^2) = \max\{ &\sum_{i\in V}(\bar{c}_{i1}u_{i1} + \bar{c}_{i2}u_{i2}) + \sum_{\substack{(i,j)\in E\\ i<j}}(\beta^1_{i,j} + \beta^2_{i,j}) : u_{kl} \in \{0,1\},\\ &\forall\, k \in V \text{ and } l = 1,2, \text{ satisfying (4.1)}, 1 \le u_l(V) \le b \text{ and (4.6)}\}\end{aligned} \tag{4.12}$$

where $\bar{c}_{k1} = c_k - \sum_{\substack{(k,j)\in E\\ k<j}}\beta^1_{k,j} - \sum_{\substack{(i,k)\in E\\ i<k}}\beta^2_{i,k}$ and $\bar{c}_{k2} = c_k - \sum_{\substack{(i,k)\in E\\ i<k}}\beta^1_{i,k} - \sum_{\substack{(k,j)\in E\\ k<j}}\beta^2_{k,j}$ are the Lagrangian costs of, respectively, u_{k1} and u_{k2}. $\mathrm{LRP}(\beta^1,\beta^2)$ can be solved by dynamic programming in $O(|V|^3)$.

C1 (Pg.107)

The third relaxation comes from the observation that a matrix formed by the coefficients of the set of constraints described in (4.1) and (4.2) is totally unimodular (TU). Thus, when all but these constraints are dualized, the resulting LRP is a well-solved problem that can be computed in polynomial-time using a specialized network flow algorithm or an interior point method for linear programming. Now, given the vectors of Lagrangian multipliers $\theta \in \mathbb{R}^1_+$, $\eta \in \mathbb{R}^1_+$ and $\gamma \in \mathbb{R}^1_+$, the resulting LRP is

$$\begin{aligned}\mathrm{LRP}(\theta,\eta,\gamma) = \max\{ &\sum_{i\in V}(\bar{c}_{i1}u_{i1} + \bar{c}_{i2}u_{i2}) - \theta + \eta b :\\ &u_{kl},\ \forall\, k \in V \text{ and } l = 1,2, \text{ satisfy (4.1) and (4.2)}\}\end{aligned} \tag{4.13}$$

where $\bar{c}_{k1} = c_k + \theta - \gamma$ and $\bar{c}_{k2} = c_k - \eta + \gamma$ are the Lagrangian costs of u_{k1} and u_{k2}, respectively.

Among the three relaxations discussed above, the first one provided the best trade-off between the strength of dual bounds and the computation time required to solve the Lagrangian subproblem. For this reason, it was the one adopted in the final configuration of our relax-and-cut algorithm.

4.4.1 Classes of valid inequalities and separation problems

The relax-and-cut algorithms developed here are based on two families of valid inequalities introduced by Balas and de Souza in their polyhedral study of the VSP [3]. Inequalities in both families have dominators as part of their support graphs. The first is related to minimal connected dominators and the inequalities belonging to it are called *CD inequalities*. The second family is associated to minimal but not necessarily connected dominators and have their strength increased through a tricky lifting procedure. The latter inequalities are termed *LD inequalities*.

The CD and LD inequalities are described below. In the discussion that follows, P is defined as the convex hull of the integer solutions of the IP model given in section 4.2, i.e.,

$P := \text{conv}\{u \in \{0,1\}^{2|V|} : u \text{ satisfies (4.1)–(4.6)}\}$. The point $\bar{u} = (\bar{u}_1, \bar{u}_2)$, to which we apply our separation routines, refers to an optimal solution of the LRP currently under consideration. Also, given $G = (V, E)$, for any $S \subset V$, Adj(S) refers to the set of all vertices in $V \backslash S$ which are adjacent to at least one vertex in S (when $S = \{i\}$ we write $Adj(i)$ to denote $Adj(\{i\})$). Similarly, for a certain $k \in V \backslash S$, we denote $Adj_s(k) := \{i \in S : (i, k) \in E\}$.

CD inequalities

Balas and de Souza [3] call a valid inequality for VSP *symmetric* if, for all $j \in V$, the coefficients of the variables u_{j1} and u_{j2} in the inequality are the same. Besides, they show that vertex separators are intimately related to vertex dominators. A vertex dominator is a subset of vertices of the graph such that all the remaining vertices are adjacent to at least one of them. The dominator is said to be connected if the subgraph induced by its vertices is connected. Balas and de Souza then stated the following property: every separator and every connected dominator have at least one vertex in common. From this observation, they derived a class of symmetric inequalities associated with connected dominators, the so-called CD inequalities. If $S \subset V$ is a connected dominator, the CD inequality for S is given by

$$u(S) \leq |S| - 1. \tag{4.14}$$

Inequality (4.14) is clearly valid for the VSP polytope P. It is non dominated only if S is *minimal* with respect to vertex removal. Notice that minimality here applies to both the dominance and the connectivity properties. Though, necessary and sufficient conditions for CD inequalities to define facets are not known in general, they are shown in [11] to be very effective in computations.

A valuable characteristic of our R&C algorithms is the fast separation routine that looks for violated CD inequalities at $\bar{u}$. A high level description of our procedure is given in Figure 4.1. The routine starts by constructing the subgraph $G_{\bar{u}} = (W, F)$ of the input graph $G = (V, E)$ which is induced by the vertices $i \in V$ with $\bar{u}_{i1} + \bar{u}_{i2} \geq 1$. It is easy to see that, if W is a dominator and $G_{\bar{u}}$ is connected then the CD inequality associated to W is violated by $\bar{u}$. Unfortunately, the converse is not true in general. It holds when constraints (4.1) are satisfied, in which case, as cited before, LRP can be solved by dynamic programming.

C1 (Pg.107)

Thus, our separation routine can be viewed as a heuristic. Step 5 of the algorithm tries to strengthen the inequality since the minimality of the dominator is a necessary condition for a CD inequality to be facet defining. It checks if the removal of a limited number of vertices preserves the connectedness of the graph induced by W and the dominance property. The separation routine implemented has $O(|V|(|V| + |E|))$ time complexity.

```
CD-Separation(G)
1. Construct G_ū = (W, F);
2. Determine n_CC, the number of connected components of G_ū;
3. if n_CC = 1 then                          /* G_ū is connected */
4.   if V ⊆ (W ∪ Adj(W)) then                /* W is a dominator of V */
5.     Turn W into a minimal CD;
6.     return the CD inequality u(W) ≤ |W| − 1;
7.   else return FAIL;          /* no new cut is returned for dualization */
```

Figure 4.1: Routine separation of CD inequalities.

In our R&C algorithm the separation procedure is called at every SM iteration. Since we implemented two greedy ways to obtain minimal CD inequalities, at most two cuts are produced at each SM iteration. Every new cut separated is stored into a pool and dualized in a Lagrangian fashion. The relaxation in (4.11) is then modified to incorporate this constraint. As a result, the term $\sum_{k=1}^{|pool|} \mu_k(|S_k| - 1 - u(S_k))$ is added to the cost function of (4.11), where $\mu \in \mathbb{R}_+^{|pool|}$ is the vector of multipliers of the CD inequalities that are currently dualized.

Conditional (CD) Cuts

According to de Souza and Balas, in [11], for unit costs, one can adapt the separation routine to search for more stringent CD inequalities. These inequalities are valid for all vectors $u \in P$ satisfying $u(V) \geq z_{LB} + 1$, but chop off several feasible solutions with smaller costs. Their usage preserves optimality and is conditioned to the existence of a lower bound z_{LB}. We call them *conditional cuts*, in an analogy to what is done for the set covering problem in [4]. For the VSP, these cuts are obtained computing $\alpha = max\{z_{LB} - b + 1, 1\}$ and searching minimal dominators that cover at least $k = |V| - \alpha + 1$ vertices (k-dominators). Thus, given a lower bound z_{LB} for the optimum, the separation routine can be changed to identify minimal connected dominators that *cover* at least k vertices. Obviously, the interesting situation occurs when $z_{LB} > b$, meaning that not all $|V|$ vertices need to be covered. Conditional cuts are used both in the B&C algorithm in [11] and in the R&C algorithm presented here. In our implementation, conditional CD cuts are considered already along the execution of the R&C algorithm. When a conditional CD cut is identified, it replaces any CD inequality it dominates.

LD inequalities

Let $S \subset V$ be a dominator of V. For $i \in S$, $P(i) = \{k \in V \backslash S : Adj_s(k) = \{i\}\}$ is the set of pendent vertices of i. Also, if S is minimal and $P(i) = \emptyset$, for some $i \in S$, the presence

of i in S is needed only to dominate i itself. We call such a vertex a *self-dominator*. Now, take $S \subset V$ a minimal dominator of G, not necessarily connected. Then, the inequality

$$u_1(S) \leq |S| - 1. \tag{4.15}$$

is trivially valid[3] for the VSP polytope P and is facet defining only under some special conditions, according to the following proposition:

Proposition 4.4.1 (Balas and de Souza[3]) *The inequality (4.15), where S is a minimal dominator of G, defines a facet of P if and only if the following conditions are satisfied: (a) $V \backslash S = \bigcup_{i \in S} P(i)$; (b) S contains no self-dominator, and (c) S is an independent set.*

Balas and de Souza[3] propose two forms of lifting the inequality (4.15) when some of the conditions in proposition 4.4.1 are not satisfied. In the R&C algorithm designed here, we apply the first lifting devised by them, which alters the coefficients of the variables associated to the assignment of vertices of S to the shore B. It applies when the dominator S is not an independent set. Since the resulting inequalities are associated with minimal dominators and a certain lifting procedure, they are called LD (Lifting Dominator) inequalities.

Thus, let S be a minimal dominator that is not an independent set. Further, let $S_1, S_2, \ldots, S_k$ be the vertex sets of the components of $G[S]$ (the graph induced by S in G) such that $|S_l| > 1, l = 1, \ldots, k$. According to [3], for each component $G[S_l]$, one must build an ordered set of vertices $I_l = \{v_1, v_2, \ldots, v_q\}$ having the following properties: `(c1)` I_l is an independent set of $G[S_l]$; `(c2)` for all $i \in \{2, \ldots, q\}$, v_i is at (edge) distance two from the vertex set $\{v_1, v_2, \ldots, v_{i-1}\}$ and `(c3)` I_l is maximal. Such a set always exists and is usually not unique. Balas and de Souza designed an algorithm to find such a set which computes a spanning tree T_l of $G[S_l]$ as follows.

Initially all the vertices in S_l are unmarked. The algorithm starts by arbitrarily choosing $v \in S_l$ as the root of $T_l = (V_{T_l}, E_{T_l})$ and mark v. Also, all the vertices $w \in Adj(v)$ in $G[S_l]$ and all the edges joining them to v in $G[S_l]$ are put into T_l. Then, for each $w \in S_l \backslash T_l$, $w \in Adj(V_{T_l})$ in $G[S_l]$, the following steps are repeated until all the vertices of S_l have been included in T_l: (i) w is marked and put into T_l by joining it through an edge from $G[S_l]$ to some (arbitrarily chosen) unmarked vertex of T_l; (ii) using edges from $G[S_l]$, add to T_l all the vertices in $(S_l \backslash T_l) \cap Adj(w)$ (the adjacency here is defined over $G[S_l]$).

It is not hard to see that the vertices marked in T_l form an ordered set satisfying the conditions defined earlier for I_l. Moreover, because of the freedom one has to choose the unmarked vertex of T_l to which a newly marked vertex is joined by an edge to T_l,

[3] We assume that $|S| \leq b$, for otherwise (4.15) would be implied by (4.4), hence redundant.

the tree is not unique. Figure 4.2 shows an example of component $G[S_l]$, along with two distinct ordered sets satisfying conditions (c1), (c2) and (c3). The spanning trees corresponding to each of the ordered sets are also depicted. Besides, the marked vertices and their degrees are highlighted.

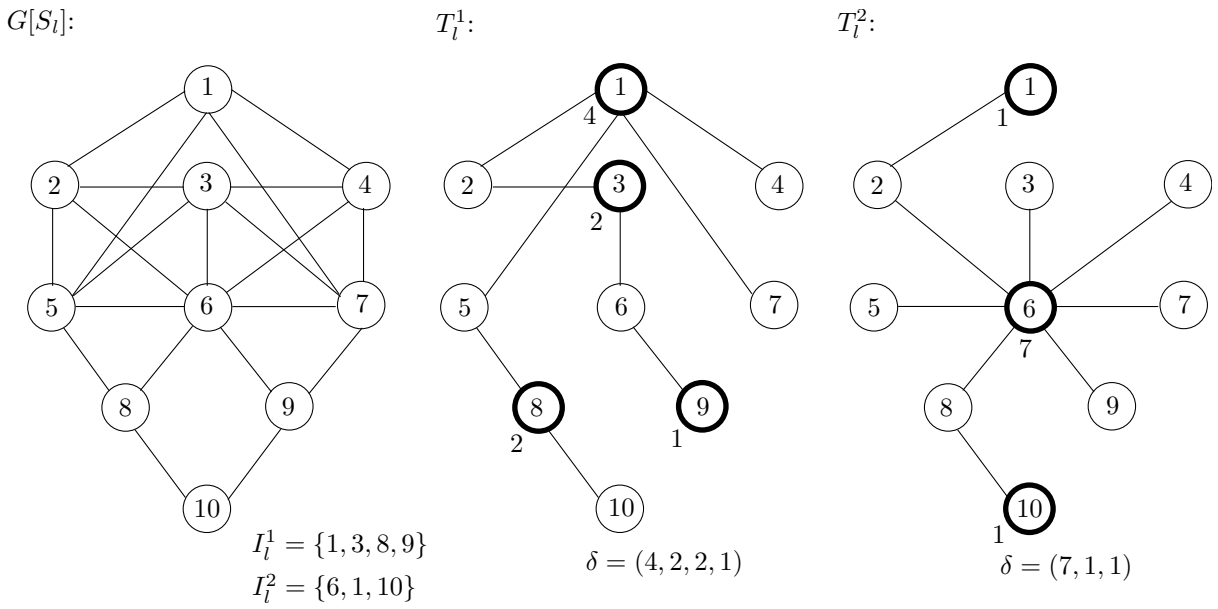

Figure 4.2: Two ordered sets satisfying conditions (c1), (c2) and (c3) and the corresponding spanning trees associated with each one of them.

Suppose that the algorithm above executed for each component $G[S_l]$ resulting in a spanning tree T_l and an ordered set I_l. Balas and de Souza [3] prove that the inequality

$$u_1(S) + \sum_{j \in S} \delta_j u_{j2} \leq |S| - 1 \tag{4.16}$$

is valid and facet defining for the polytope P, where each δ_j is equal to the degree of v_j in T_l if this vertex is marked, otherwise it is null.

To identify LD inequalities we implemented a heuristic separation routine which uses this first lifting procedure from Balas and de Souza. The procedure is detailed in Figure 4.3. Like in the CD separation, the routine starts by constructing the subgraph $G_{\bar{u}} = (W, F)$ of G which is induced by the vertices $i \in V$ with $\bar{u}_{i1} + \bar{u}_{i2} \geq 1$. Then, to save computation time, in step 3 we restrict the separation routine to the cases where $G_{\bar{u}}$ has at most two connected components. Though restrictive, this procedure allows us to generate LD cuts both for connected and non connected dominating sets, contrarily to the CD inequality case. Thus, the lifting of variables in the smaller set is produced with the aid of at most two spanning trees (steps 6–8). In our final implementation, the selection

```
LD-Separation(G)
1. Construct G_ū = (W, F);
2. Determine n_CC, the number of connected components of G_ū;
3. if n_CC ≤ 2 then                /* G_ū is connected or has at most two components */
4.   if V ⊆ (W ∪ Adj(W)) then      /* W is a dominator of V */
5.     Turn W into a minimal dominator with n_CC components of G[W];
6.     for l = 1, .., n_CC do
7.       Construct T_l and identify an independent set I_l = {v_1, v_2, ..., v_q} ⊂ V_{T_l};
8.       Determine δ_{v_j} = δ(v_j) for all v_j ∈ I_l. Set δ_{v_j} = 0 for all v_j ∈ V_{T_l}\I_l;
9.     return the LD inequality u_1(W) + Σ_{v_j∈W} δ_{v_j} u_{v_j 2} ≤ |W| − 1;
10. return FAIL;                   /* no new cut is returned for dualization */
```

Figure 4.3: Routine separation of LD inequalities.

of vertices to mark and to connect each newly marked vertex is done in increasing order of vertex labels[4] Finally, a LD inequality associated to the dominator W that cuts off $\bar{u}$ is constructed. The computational complexity of the separation routine for LD inequalities is the same as that of the CD inequalities, i.e., $O(|V|(|V| + |E|))$. In practice, however, the separation routine is quite fast because minimal dominators are obtained from already small W dominators.

Similarly to what is reported by Balas and de Souza in [11], CD inequalities showed, experimentally, to be much more effective than LD cuts. Moreover, we noticed that the LD inequalities over connected dominator often produced better dual bound than those ones over not connected dominators. Thus, in our final experiments, we decided to search LD inequalities only when our CD separation is turned on and just considering connected dominators. Furthermore, in practice, these choices resulted in time savings during LD separation, mainly because W became an input parameter. These changes resume our LD separation routine in the steps 6–9 of the original algorithm with n_{CC} set to 1.

In our R&C algorithm the LD separation routine is called at every SM iteration. It produces at most two cuts per iteration and the lifting procedure is fired even when the (4.15) yielded is not violated by the solution of the current Lagrangian subproblem. Every new cut separated is stored in a pool and dualized in a Lagrangian fashion. The relaxation in (4.11) is then modified to incorporate this constraint. As a result, the expression $\sum_{k=1}^{|pool|} \varphi_k(|S_k| - 1 - u_2(S_k) - \sum_{j \in S_k} \delta_j u_{j1})$ is annexed to the cost function of (4.11), where $\varphi \in \mathbb{R}_+^{|pool|}$ is the vector of multipliers of the LD inequalities currently dualized.

[4]With the aim of generating LD constraints with smaller supports we experimented the selection of vertices with highest degree in $G[S_l \backslash V_{T_l^{(i)}}]$, at each choice point (i) during T_l construction. However, since no gain was observed and some additional computation was required, this strategy was abandoned.

Notice that, due to the inequality dualization scheme within relax-and-cut algorithms, the same cut may be repeatedly identified by the separation routines. Managing the cut pools of CD and LD inequalities is quite simple and is restricted to redundancy checks, i.e., a new inequality is inserted only if it is not identical to another inequality already in the pool or in the original formulation. The use of suitable data structures[5] may render the implementation of redundancy verification very fast. It can be implemented to have $O(s_1 s_2)$ time complexity, where s_1 is the current size of the biggest list resulting from collisions in the hash tables designed to handle our pools and s_2 is the size of the current candidate constraint to be inserted in the pool. In practice, with mid-high density graphs, s_1 and s_2 are both much smaller than $|V|$. However, computational effort with redundancy verification can be noticed with sparse graphs because, typically in this case, our separation routines produce (CD and LD) cuts with a reasonable number of not null coefficients.

4.4.2 A Lagrangian heuristic

The generation of good primal bounds is important for the computation of the step size (4.9) in the SM and to assess the duality gap along the iterations of the algorithm. In order to compute lower bounds for the VSP, we devise a simple greedy heuristic whose steps are summarized in Figure 4.4. Initially, the set L containing the vertices that are candidates to be part of the shores is built. This excludes the universal vertices, i.e., those which are adjacent to all other vertices. Clearly, universal vertices must be in any separator. The heuristic chooses arbitrarily two nonadjacent vertices of L and assigns them to different shores so that, in the end, they will not be empty. It proceeds by assigning vertices to shores, prioritizing the assignments corresponding to the variables with higher weighted Lagrangian costs. The choice of the weighting method is controlled by the parameters $\bar{\rho}(k) \in \{0, 1\}, k = 1, 2$. It is implemented by multiplying or dividing the Lagrangian cost of the variable associated to a vertex v by the degree of v, $\delta(v)$, as seen in step 5, and can be distinct for variables associated to the same vertex but to different shores. Notice that such a weighting permit us to guide the choice of vertices (to put into any of the two shores) based on both, costs and vertice degrees. Introducing information about vertice degrees lead us to consider connectivity/neighborhood questions/concerns when selecting vertices along the execution of our Lagrangian heuristic. Notoriously in unitary costs, multiplying (dividing) lead us to give priority to put vertices of highest (lowest) degrees into the shores. Preliminary tests with a subset of instances showed that only multiplying the costs produce slightly better solutions than the other combinations. Hence, in our

[5]Experimental results have shown that the appropriate tunning of hash tables may cause the redundancy check to be quite efficient.

Lagrangian heuristic $(G = (V, E), c, \overline{c}, \bar{\rho}, \Gamma)$
1. $L \leftarrow V \backslash \{$universal vertices in $G\}$;
2. $v_0 \leftarrow \{$any vertex in L that maximizes $\bar{c}(u_{i1})\}$;
3. Initialize shore A: $A \leftarrow \{v_0\}$, $L \leftarrow L \backslash \{v_0\}$ and $L' \leftarrow L \backslash \mathrm{Adj}(v_0)$;
4. Initialize shore B: $B \leftarrow \{v_1 \in L' : \delta(v_1) \geq \delta(v), \forall v \in L'\}$ and $L \leftarrow L \backslash \{v_1\}$;
5. `for` $k = 1, 2$ `do`:
   `for` all $i \in L$, compute $w_{u_{ik}} \leftarrow \bar{c}(u_{ik}) * [\bar{\rho}(k) * \delta(i) + (1 - \bar{\rho}(k))/\delta(i)]$;
   Let S_k be the list of variables u_{ik} sorted nonincreasingly by $w_{u_{ik}}$;
   `for all` $j \in Adj(v_{2-k})$ `do` $S_k \leftarrow S_k \backslash \{u_{jk}\}$;
6. `while` $|A| < b$ `or` $|B| < b$ `do`
   $f_1 \leftarrow \{$vertex corresponding to the first variable in $S_1\}$;
   $f_2 \leftarrow \{$vertex corresponding to the first variable in $S_2\}$;
   `if` $\bar{c}(u_{f_1,1}) > \bar{c}(u_{f_2,2})$ `then`
     $A \leftarrow A \cup \{f_1\}$; $S_1 \leftarrow S_1 \backslash \{u_{f_1,1}\}$;
     `for` all $j \in \mathrm{Adj}(f_1)$ `do` $S_2 \leftarrow S_2 \backslash \{u_{j,2}\}$;
   `else`
     $B \leftarrow B \cup \{f_2\}$; $S_2 \leftarrow S_2 \backslash \{u_{f_2,2}\}$;
     `for` all $j \in \mathrm{Adj}(f_2)$ `do` $S_1 \leftarrow S_1 \backslash \{u_{j,1}\}$;
   `if` $|A| = b$, $\bar{c}(u_{f_1,1}) \leftarrow -\infty$; /* avoids new vertices in A */
   `if` $|B| = b$, $\bar{c}(u_{f_2,2}) \leftarrow -\infty$; /* avoids new vertices in B */
7. Compute the separator: $C \leftarrow V \backslash \{A \cup B\}$
8. `if` $\gamma(\sum_{j \in C} c_j) < \Gamma$, `call Local Search`$(G, A, B, C, c)$;
9. `return` (A, B, C)

Figure 4.4: Lagrangian heuristic.

default setting, we fixed $\bar{\rho}(1) = \bar{\rho}(2) = 1$. All the assignments of vertices to shores are made so as to maintain the viability and to respect the maximum size of the shores. As a final step, a local search subroutine is ran in an attempt to improve on the solution produced by the heuristic. Aiming to try a further improvement over the feasible solution generated, the Lagrangian heuristic may call a local search heuristic. Let z be the cost of this solution and $\gamma(z)$, a function that returns the number of solutions to the VSP having cost z found along the R&C execution. The local search is executed only when $\gamma(z) < \Gamma$, where Γ is a parameter that specify a limit to the number of trials of improvements over solutions with same value. It was implemented as a manner to limit additional overhead with primal improvement trials when running the relax-and-cut algorithm.

The local search is described in Figure 4.5 starts by enlarging the separator with as many vertices of the shores belonging to its adjacency as possible (steps 1 to 5). Then vertices are transferred from the separator back to the shores in step 6 in an arbitrary order. However, the choice of the destination shore is made so as to increase the chances

of future moves from the separator to the shores. This is evaluated via the simple computations in steps 6.i to 6.m. The overall complexity of the Lagrangian heuristic, including the local search procedure, is $O(|V| \log |V| \times |E|)$.

4.5 Integrating R&C and B&C

An alternative to be more effective to solve VSP problems to optimality is to devise a hybrid approach that combines Lagrangian relaxation with Integer Linear Programming (IP), in the style suggested in [6]. We named this hybridization as Relax-and-Cut-and-Branch-and-Cut (or simply HYBRID). In such combination, optimization is split into three steps: *(i)* the LR phase, based on our relax-and-cut framework, whose output are pools of valid inequalities and a primal bound; *(ii)* a remodelling phase, where the IP formulation is tightened according to the information gathered during the first phase and, subsequently, *(iii)* the LP phase where a branch-and-cut code is executed over the new IP model. Among the cuts used in this last phase, we include those cuts separated throughout the execution of the R&C algorithm in the initial phase. We call them the *Lagrangian cuts*.

The two first phases are generically termed as the *preprocessing phase* of our hybrid algorithm. The execution flow of the algorithm is depicted in Figure 4.6.

We now describe the three phases of the hybrid algorithm in more detail.

The LR phase

The LR phase is comprised of an R&C module. It corresponds to one of the implementations of the relax-and-cut algorithm described in section 4.4 and is the core of our framework. During its execution, valid CD and/or LD inequalities to the VSP are identified and inserted into the corresponding pool.

After completing the execution of the relax-and-cut algorithm in this module, the final gap is verified. If the problem is not solved during the R&C run, some information are passed as the input of the next phases. This includes not only the cut pools, but also the best primal solution and its cost, i.e., the best lower bound found so far.

The remodelling phase

In addition to CD and LD inequalities identified in the previous phase, some constraints may be added or adapted to strengthen the original formulation presented in section 4.2.

The first constraint considered comes from the observation that universal vertices must belong to any separator. Thus, given the VSP input graph $G = (V, E)$ and $U = \{i \in V : |Adj(i)| = |V| - 1\}$, the constraint $\sum_{i \in U}(u_{i1} + u_{i2}) = 0$ is trivially valid for the

Local Search (G, A, B, C, c)

/ initializations */*

1. Let A_C be the vertices in A that have neighbors in C;
2. Let B_C be the vertices in B that have neighbors in C;
3. `if` $A = A_C$ `then` $A_C \leftarrow A_C \backslash \{$arbitrarily chosen vertex of A$\}$;
4. `if` $B = B_C$ `then` $B_C \leftarrow B_C \backslash \{$arbitrarily chosen vertex of B$\}$;
5. $A' \leftarrow A \backslash A_C$; $B' \leftarrow B \backslash B_C$; $C' \leftarrow C \cup A_C \cup B_C$;

/ main loop */*

6. `for` every vertex $v \in C'$ `do`:

6.a `if` $|A'| = b$ and $|B'| = b$ `then break`;

6.b `if` $|\text{Adj}(v) \cap A'| \neq \emptyset$ and $|\text{Adj}(v) \cap B'| \neq \emptyset$, `then continue`;

6.c $C' \leftarrow C' \backslash \{v\}$;

6.d `if` $\text{Adj}(v) \subset C'$ `then`

6.e `if` $|A'| = b$ `then` $B' \leftarrow B' \cup \{v\}$;

6.f `else`

6.g `if` $|B'| = b$ `then` $A' \leftarrow A' \cup \{v\}$;

6.h `else`

6.i $n_A \leftarrow 0$; $n_B \leftarrow 0$;

6.j `for` all $w \in \text{Adj}(v)$ `do`

6.k $n_A \leftarrow n_A + |\text{Adj}(w) \cap A|$; $n_B \leftarrow n_B + |\text{Adj}(w) \cap B|$;

6.l `if` $n_A > n_B$ `then` $A' \leftarrow A' \cup \{v\}$;

6.m `else` $B' \leftarrow B' \cup \{v\}$;

6.n `else`

6.o `if` $\text{Adj}(v) \subset A' \cup C'$ and $|A'| < b$ `then` $A' \leftarrow A' \cup \{v\}$;

6.p `else` /* $\text{Adj}(v) \subset B' \cup C'$ */

6.q `if` $|B'| < b$ `then` $B' \leftarrow B' \cup \{v\}$;

7. `if` $\sum_{i \in C} c_i > \sum_{i \in C'} c_i$ `then` $A \leftarrow A'$, $B \leftarrow B'$, $C \leftarrow C'$.

Figure 4.5: Primal heuristic: the local search procedure

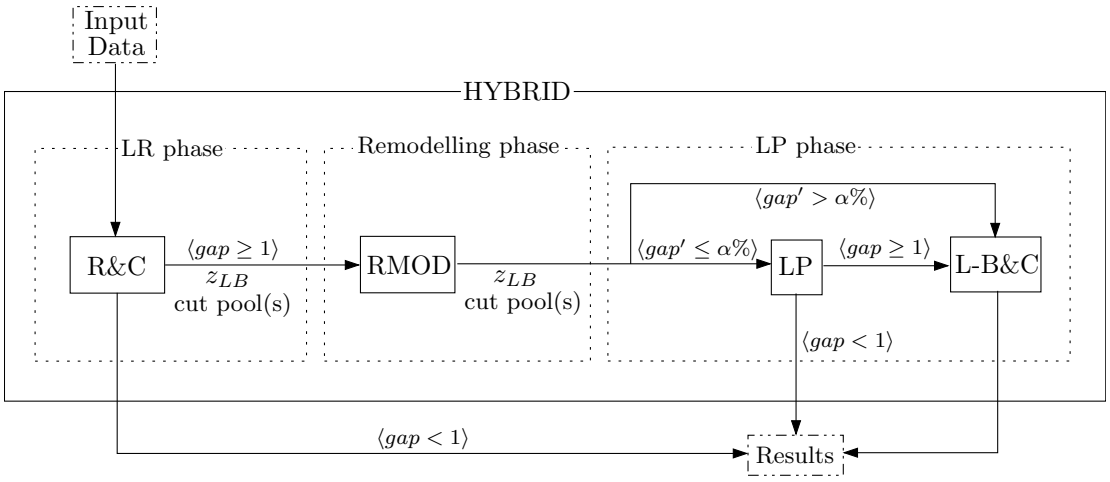

Figure 4.6: Flow Diagram of HYBRID to the VSP.

problem. This constraint was not used during the LR phase because it resulted in some degradation of the R&C performance in terms of dual and primal bounds yielded. Also, this constraint is not taken into account in [3]. However, when dealing with high density graphs, the occurrence of universal vertices is very frequent. In practice, the benefits with the addition of this constraint to the IP model justified its inclusion as part of our remodelling phase.

Now, we concentrate on how to use the lower bound yielded by the R&C module to tighten our IP model. To this end, we focus on unit cost instances, i.e., those for which $c_i = 1, \forall i \in V$. We do so because these instances are very frequent in practical applications.

Assume that z_{LB} is the cost of the best known solution computed in this case. Since $u_2(V) \geq u_1(V)$, we can deduce that $u_2(V) \geq \left\lfloor \frac{z_{LB}}{2} \right\rfloor + 1$ must be satisfied by any solution with cost higher than z_{LB}. For any such solution, it is also straightforward to conclude that if $z_{LB} - b > 1$, constraint (4.3) can be replaced by the stronger inequality $u_1(V) \geq z_{LB} - b$.

Although the previous modifications rely on rather simple arguments, in this phase we incorporate them to the model. As a matter of fact, with the exception of the last change, preliminary experiments we carried out with these modifications in the IP model revealed an improvement in the performance of our modified branch-and-cut algorithm.

The LP phase.

The LP phase has as its input the cut pools, the best solution and the best primal bound from the LR phase and the new IP model from the remodelling phase. It has two modules that we discuss below.

Separation Strategy(sep, CDP, LDP)
1. `forall` active node
   / *call table lookup routines to obtain Lagrangian cuts* */*
1.a `if` there is Lagrangian `CD` cut violated `then`
1.b runLagrangianSeparation(`CD,CDP`); */* `L-CD` *is always run* */*
1.c `if sep` $\in \{\langle$`L-CD,CD,L-LD,LD`$\rangle, \langle$`L-CD,CD,L-LD`$\rangle, \langle$`L-CD,L-LD`$\rangle\}$ `then`
1.d `if` there is Lagrangian `LD` cut violated `then`
1.e runLagrangianSeparation(`LD,LDP`); */* *add* `L-LD` *cuts* */*
   / *test and call de Souza and Balas' routines to generate cuts* */*
1.f `if sep` $\notin \{\langle$`L-CD,L-LD`$\rangle\}$ and there is no Lagrangian cut violated `then`
1.g runCDSeparation(); */* *call* `CD` *separation* */*
1.h `if sep` $= \langle$`L-CD,CD,L-LD,LD`$\rangle$ `then`
1.i runLDSeparation(); */* *call* `LD` *separation* */*

Figure 4.7: The separation strategy executed by L-B&C module.

Linear Programming Solver (LP). This module solves the LP corresponding to the relaxation of the IP model coming from the remodelling phase, appended with the cuts present in the cut pool. This model is computed only if the (relative) Lagrangian gap resulting from the R&C module, given by $gap' = 100 \times (z_{UB} - z_{LB})/z_{LB}$ is lower than a threshold value α. The purpose here is to use linear programming to avoid running the B&C module unnecessarily. It is well-known that, in practice, computing dual bounds within R&C algorithms commonly produce meager values than the linear relaxation optimum value. Thus, this module is a possible workaround to bypass some numerical difficulty in closing the integrality gap.

Branch-and-Cut with Lagrangian cuts (L-B&C). This module runs only if the R&C (LR phase) and/or the LP solver fail to prove optimality (i.e., $gap \geq 1$). Recall that the preprocessing phase yields as outputs the sets of (conditional) CD cuts and/or LD inequalities which are candidates to be added to the formulation given as input of L-B&C. Moreover, L-B&C is also given the values of z_{LB} (best incumbent), which may help to prune the enumeration earlier.

Figure 4.7 shows the separation strategy adopted at each node of the enumeration tree during the execution of the L-B&C module. This strategy is fixed according to the contents of the orderd sequence `sep`. The elements of `sep` are taken in the set {`CD, LD, L-CD, L-LD`}. The meanings of these strings are: `CD` and `LD` correspond to the separation routines for CD and LD inequalities as implemented in [11], while `L-CD` and `L-LD` are the

runLagrangianSeparation (C, $Pool$)
1. nbCutsViolated $\leftarrow$ 0;
2. `while` nbCutsViolated $<$ `MaxCutsPerNode do`
3. Search a cut c of class C violated by the current LP solution in $Pool$;
4. Add c to the IP model;
5. Delete c from $Pool$;
6. Increment nbCutsViolated;

Figure 4.8: Separation of Lagrangian cuts.

separation routines for Lagrangian cuts implemented by a table lookup scheme. According to this notation, `sep=<L-CD,CD>` means that the separation of a fractional solution is first made by the table lookup procedure for Lagrangian CD cuts and then by de Souza and Balas' routine for CD cuts. Lagrangian CD and LD cuts are stored in pools CDP and LDP, respectively.

The calls to `runCDSeparation` and `runLDSeparation` in lines `1.g` e `1.i` of the algorithm above correspond to calls to the separation routines from [3]. As for the Lagrangian cuts, the separation is described in Figure 4.8.

Interesting to remark here is the simplicity of separation routines inside L-B&C algorithm (briefly described just below). They consist basically of algorithms that scan linearly the cut pools, trying to identify (Lagrangian) inequalities that may cutoff the current optimal LP solution available.

It was experimentally observed that, L-B&C performance was very sensitive to the way CD inequalities were added during the branch-and-cut execution. Thus, several experiments were performed in order to setup an interesting value to `MaxCutsPerNode`. The most promising settings took into account the density of the graphs underlying the instances (see Table 4.1 for details)

4.6 Test Environment Setup

This section describes the setup of the enviroment under which our tests were carried out. The algorithms were coded in `C` and `C++`, using resources of the Standard Template Library and prepared to be executed under Linux OS. We used the free compiler `g++` (`gcc` version 4.0.3) with options `-O3` and `-lm` selected. Tests were ran on a Pentium IV machine having 1GB of RAM and XPRESS Optimizer 17.01.02 was used as the IP solver.

4.6.1 Data sets

Our main experiments were made on a subset of instances taken from [11] which can be downloaded from `www.ic.unicamp.br/~cid/Problem-instances/VSP.html`. Additionally, hard instances from the `MIPLIB`[20] subset were used to perform further tests. Initially, from the more than 140 instances used in [11], we select the ones that required more than a minute of CPU time to be solved by the branch-and-bound (B&B) algorithm of XPRESS in its default configuration. At this point, it is worth noting that XPRESS default configuration implements cut separation routines that would permit us to classify its default algorithm as a branch-and-cut algorithm, rather than as branch-and-bound. However, to distinguish it more easily from the several algorithms we compare throughout our experimentations, we will refer to XPRESS default algorithm as being a branch-and-bound (B&B) one.

We end up with 62 instances for our tests, all of which, with cost vector equals to the sum vector. The parameter b delimiting the maximum size of a shore is always set to $\lceil 2n/3 \rceil$ but, for the MIPLIB instances, that value is computed as $\lfloor 1.05 \times n/2 \rfloor$.

The majority of our reports relies on comparing results for instances that were solved by at least one of the algorithms used in our experiments. Thus, among the 62 instances initially selected, only 51 were broadly used in performance comparisons, since 11 instances were not solved by any of the implemented algorithms within the time limit imposed.

A common characteristic of the bulk of these 51 instances is the mid-high ($> 20\%$) density of the graphs underlying them. In accordance with what was already mentioned in [11] and [7] the use of cutting-plane algorithms (specially based on `CD` inequalities) is likely to be more effective for mid-high density graphs. Nevertheless, the few VSP instances rising from low density graphs were not discarded in our experiments. Proceeding like that we could be able to infer about further experiments over important data sets of hard instances (like the biggest instances from `MIPLIB` set).

As for de Souza and Balas in [11], our results are reported by classes of instances: `DIMACS` graphs, `MatrixMarket` graphs – divided in three categories, `MM-I, MM-II` and `MM-HD`, according to common characteristics used in their construction – and coefficient matrices from `MIPLIB` instances. Moreover, within each class, the instances are listed in increasing order of graph density.

4.6.2 Parameter settings

General parameters. The following settings were used for the basic parameters of the subgradient algorithm: `(a)` the Lagrangian heuristic is called at every SM iteration; `(b)` the local search heuristic is called just after Lagrangian heuristic execution. However,

along the SM execution, the number of improvement trials for solutions with same cost was limited to at most $\Gamma = 5$. Cost repetition is easily identified in our case since there are only $O(|V|)$ possible values for the cost function; `(c)` the algorithm stops when the limit of 2000 SM iterations is reached or when $\pi^k \leq 10^{-5}$ in equation (4.9), what occurs first.

Algorithm dependent parameters. When the SM is called inside the NDR&C algorithm, π, in equation (4.9), is initially set to 2 and multiplied by 0.5 each 90 consecutive SM iterations without improvement on the upper bound. Also, the call to the routine in charge of trying to generate conditional cuts occurs whenever a minimum amount of new `CD` cuts are added into the pool. In our final tests this number corresponds to 10% of the maximum number of SM iterations. If any `CD` inequality is generated, the minimum number of calls performed is one.

When running the PR&C algorithm, however, the Lagrangian dual problem is solved typically several times using SM. We call each complete execution of SM a *pass* . The total number of passes is an input parameter for the postponed relax-and-cut algorithms, denoted by Δ. In our final experiments we adopted $\Delta = 15$. Now, let δ be the number of the current pass. In equation (4.9), $\pi^0_{(\delta=1)}$ is initially set to 2 and, for the other passes, $\pi^0_{(\delta>1)}$ is computed by the recurrence relation: $\pi^0_{(\delta)} = \pi^0_{(\delta-1)} \times f(\delta)$, where $f(\delta) = 1 - (\delta - 1)/\Delta^2$. Observe that π^0 decreases monotonically and smoothly as δ increases. In our experiments, the small decreases in the initial values of π in equation (4.9) proved to be beneficial for the computation of tighter dual bounds.

Moreover, along each pass, the π value is update at each 20 consecutive iterations without improvement on the upper bound. Here, similarly to NDR&C strategy, the routine in charge of trying to generate conditional cuts is called. In this case, it is done every time a Lagrangian subproblem is solved. Nevertheless, the dualization of inequalities identified along the execution of a pass is done only when the SM terminates.

During L-B&C execution, the amount of `CD` *Lagrangian* cuts added at each node is mainly determined by the input graph density. Additionally, it was experimentally observed that adding many cuts at the first node often speedup the search. This last behavior was observed mainly when testing instances associated to mid-high density graphs. So, we adopted a strategy that tries to match this observation with graph density information. As a result, we compute the range size (rs) like that: $rs = (hdi - ldi)/nr$, where `(i)` hdi and ldi are input data and refers to, respectively, the higher and lower density of a graph among the instances in the subset tested and `(ii)` nr is the number of ranges used in the final tests. Figure 4.1 shows our final configuration where, for instance, when dealing with graphs having density in $(35.6\%, 64.3\%]$, in the first node we put up to 50% of the cuts in the pool. After, for the other nodes, at most 10 `CD` cuts violated are added. In

the case of LD inequalities, the amount of *Lagrangian* cuts added at each node followed the tunning used by de Souza and Balas in [11], i.e., 10 cuts per node. Moreover, as in [11], the experiments performed with any version of hybrid algorithm developed obeyed a CPU time limit of 30 minutes.

Table 4.1: Ranges and number of cuts adopted as final settings when adding Lagrangian `CD` cuts within L-B&C algorithm. Notice: $< rs, ldi, hdi > = < 28.7, 7, 93 >$.

Density range	Maximum number of cuts	
	first node	other nodes
$\leq 35.6\%$	10	2
$(35.6\%, 64.3\%]$	$0.5 \times$ pool size	10
$> 64.3\%$	$0.75 \times$ pool size	10

Parameter tunings To determine the settings discussed above, tunings of parameter values were carefully performed with a representative subset of instances containing at least one representative of each class. The main characteristics of these tests were: `(i)` the set of values of Γ tested with the instances selected was $\{1, 2, 3, 5, 10\}$; `(ii)` experiments with the number of consecutive iterations without improvement on the upper bound considered the values in $\{30, 50, 75, 90, 100, 120, 150, 200\}$ and $\{10, 15, 20, 25, 30, 50\}$ for, respectively, NDR&C and PR&C algorithms; and `(iii)` alternatives considered for the number of passes of the PR&C algorithms (Δ) were 10, 15, 20 and 25. Moreover, to compute the recurrence relation $\pi^0_{(\delta)} = \pi^0_{(\delta-1)} \times f(\delta)$, we also considered to do tests with constant value functions. Thus, we did some preliminary tests with functions defined as $f(\delta) = c$, $c \in \{0.90, 0.95, 1.00\}$, were performed.

4.7 Computational results

In this section we report the computational tests carried out with the several configurations of relax-and-cut algorithms and hybridizations implemented for the VSP.

4.7.1 Relax-and-cut results

The main results of the computational experiments done with the relax-and-cut algorithms developed are documented in tables 4.2, 4.3 and 4.4. Double horizontal lines in these tables split instances from `DIMACS`, `MM-I, MM-II` and `MM-HD` and `MIPLIB`. Also, these tables are divided in three groups of columns. The first group, relative to (columns 1–4), describes the instance characteristics: name (`label`), number of nodes (`n`), graph densities

(d) and the optimum value (Opt) or the best known solution value (when it appears underlined). The second and third groups of columns report the results, respectively, concerning the non-delayed relax-and-cut (NDR&C) and postponed (non-delayed) relax-and-cut (PR&C) algorithms developed. Both groups have the following format of columns: ub, the upper bound obtained; the value of the best solution found (lb) and the total time, t(s), required to run each algorithm. Additionally, although not detailed here, some preliminary tests were performed with a delayed relax-and-cut algorithm. However, as suggested by a previous comparison carried out by Lucena ([18]) the results we obtained confirmed that, NDR&C strategies performed better than DR&C.

C2 (Pg.115)

Before taking a look toward the quality of dual bounds produced (against the linear relaxation values) let's discuss this aspect. First, it should be noticed that the IP model from section 4.2 is rather weak. In the linear relaxation, setting all variables to $1/2$ satisfies all the constraints for any sufficiently large value of b (which is the case for all instances in our dataset). This gives the worst dual bound one could come up with: n ! Thus, poor dual bounds are expected unless strong cuts are added to the formulation. Results reported in [11] show that CD inequalities fulfill this requirement. However, a drawback to use such inequalities comes from the fact that the corresponding separation problem is $\mathcal{NP}$-hard in general. The authors had then to resort to a heuristic procedure to perform the task. Their heuristic is of quadratic-time complexity and, in practice, more expensive than our (almost) linear-time complexity heuristic used to separate integral points (see section 4.4.2).

Based on results reported in tables 4.2, 4.3 and 4.4 we can conclude that:

1. in terms of *optimality*, only four instances have been solved to proven optimality when separating CD inequalities and, in this aspect, PR&C seems to have a better performance than NDR&C algorithm.

2. concerning *dual bounds* we can highlight that: (i) bounds produced by NDR&C(LD) and PR&C(LD) are very poor, with values typically near to the linear programming bounds; (ii) in most of the cases, the algorithms that embed CD inequalities separation produced much stronger dual bounds than LP relaxation bound.

3. taking into account the *primal bounds* obtained by our heuristic (lb column), we conclude that it attained a considerable success rate of generating optimal value[6] solutions: always floating in the range of $[65\%, 71\%]$, depending on the relax-and-cut version. Alternatively, if considering the success rate of generating solutions as good as the best known (i.e., considering all the 62 instances tested), the range remains quite similar and even tighter: $[66\%, 71\%]$.

[6]Entries for column Opt reveal that optimum values are known for 51 out of the 62 instances tested.

4. comparing NDR&C and PR&C algorithms using the results obtained with these tests we are not able to infer that there is a dominant approach.

Additionally, inspecting the columns corresponding to total times required by the various configurations we see that, in general, the running times seem to be quite acceptable (although, usually, identifying LD inequalities seems to imply increase of CPU time).

A first analysis of these results might lead us to conclude that the algorithms implemented are not suitable when we are interested to prove the optimality of VSP solutions (this happened only in 6.5% of the cases). Thus, in this since, our algorithms could be seem as good heuristics to the VSP with some (experimental) quality guarantee. However, rather than being disappointing or dishearten, these first results pushed us toward the conception of a very effective hybrid approach that combines Lagrangian relaxation and integer linear programming techniques (similarly to the framework presented in [8]).

4.7.2 Combined results: HYBRID algorithms

Results in tables 4.3 and 4.4 reveal the good performances of our relax-and-cut algorithms when separating `CD` inequalities: they often produce good dual and primal bounds rapidly. However, they fail to solve (to proven optimality) the vast majority of the instances. Moreover, inspecting the behavior of the B&C algorithm developed in [11], which we had access to, we noticed that a couple of `CD` inequalities needed to be separated and added to the model before good dual bounds are computed. Thus, it would be very helpful if we could quickly generate a set of initial `CD` cuts.

Thus, an alternative to tackle not solved instances is to devise the hybridization proposed in section 4.5. Accordingly, any of our relax-and-cut algorithms might be used to generate cuts that would allow the ILP solver to operate over a tighter VSP formulation. In this way, after running a relax-and-cut algorithm, cuts in the pool(s) are passed as input to the modified B&C (L-B&C module). The execution flow corresponding to such an approach is presented in Figure 4.6.

From all the 62 instances selected as benchmark to test the relax-and-cut algorithms developed, only the 58 not solved by any relax-and-cut version were kept. Moreover, within the time limit of 30 minutes, 11 instances remained not solved. We are indeed left with only 47 instances that were solved by at least one of the approaches tested. Besides, since no further test extending that time limit was carried out, only results over this subset of instances are discussed.

Computational experiments were performed taking into account a considerable variety of HYBRID configurations. Their results revealed a floating performance of the algorithms, depending on the density of the (input) graph underlying the instances used. This observation confirms some expectations rising from previous discussions made over

Table 4.2: Results for VSP instances: NDR&C and PR&C separating only Lifted Dominating inequalities.

Instance				NDR&C (LD)			PR&C (LD)		
label	n	d	Opt	ub	lb	t(s)	ub	lb	t(s)
dim.DSJC125.1	125	0.09	90	124	90	14.32	125	90	14.37
dim.games120	120	0.09	102	120	97	6.07	120	99	5.02
dim.myciel7	191	0.13	156	194	153	6.34	193	154	6.33
dim.myciel6	95	0.17	76	94	75	1.90	93	75	2.57
dim.queen12_12	144	0.25	97	141	97	20.53	139	97	22.87
dim.queen11_11	121	0.27	81	116	81	15.29	115	81	17.36
dim.queen10_10	100	0.30	67	95	67	11.72	95	67	15.17
dim.queen8_12	96	0.30	65	92	65	10.82	91	65	15.03
dim.queen9_9	81	0.33	55	76	55	8.99	76	55	11.72
dim.queen8_8	64	0.36	43	59	43	5.99	59	43	7.29
dim.miles1000	128	0.40	110	127	109	7.10	126	109	9.42
dim.queen7_7	49	0.40	31	44	31	2.73	44	31	6.53
dim.DSJC125.5	125	0.50	74	116	74	13.21	114	74	13.8
dim.DSJC125.9	125	0.90	22	125	22	4.74	125	22	4.91
mat.can96	96	0.20	72	93	72	7.00	93	72	9.52
mat.can73	73	0.25	53	71	53	5.47	69	53	6.36
mat.rw136	136	0.07	121	136	120	1.73	135	119	1.79
mat.gre__115	115	0.09	95	115	92	7.52	115	91	10.83
mat.L125.gre__185	125	0.15	104	123	104	17.45	124	104	15.17
mat.can__144	144	0.16	126	141	126	16.10	142	126	23.76
mat.L125.can__161	125	0.16	97	123	95	18.02	123	95	17.88
mat.lund_a	147	0.26	118	142	116	14.68	142	116	18.33
mat.L125.bcsstk05	125	0.35	101	121	101	12.10	119	101	12.64
mat.L125.dwt__193	125	0.38	95	115	95	7.33	115	95	7.69
mat.L125.fs_183_1	125	0.44	98	135	95	2.41	135	96	2.53
mat.bcsstk04	132	0.68	84	128	84	6.53	128	84	6.57
mat.arc130	130	0.93	88	136	88	12.96	129	88	13.41
mat.L100.steam2	100	0.36	76	93	76	9.45	92	76	11.03
mat.L120.fidap025	120	0.39	102	115	102	5.59	114	102	7.03
mat.L120.cavity01	120	0.42	99	123	99	3.49	124	98	4.04
mat.L120.fidap021	120	0.43	98	122	98	4.37	123	98	5.63
mat.L120.rbs480a	120	0.46	88	114	88	6.55	109	88	6.98
mat.L120.wm2	120	0.47	98	127	92	1.93	127	92	2.29
mat.L100.rbs480a	100	0.52	73	100	73	2.59	97	73	2.8
mat.L80.wm2	80	0.58	61	84	59	1.71	84	61	1.73
mat.L100.wm3	100	0.59	77	103	77	2.42	102	77	2.38
mat.L120.e05r0000	120	0.59	90	124	90	3.74	122	90	3.43
mat.L100.wm1	100	0.60	74	103	71	2.68	102	71	2.55
mat.L120.fidap022	120	0.60	84	119	84	3.87	117	84	4.02
mat.L100.fidapm02	100	0.62	69	99	69	4.02	98	69	3.94
mat.L120.fidap001	120	0.63	82	116	82	4.87	114	82	4.94
mat.L100.e05r0000	100	0.64	70	108	70	1.77	107	70	2.23
mat.L80.fidapm02	80	0.65	53	77	53	2.40	77	53	2.34
mat.L120.fidapm02	120	0.65	86	119	86	5.43	119	86	5.53
mat.L100.fidap001	100	0.68	64	101	64	2.71	101	64	2.62
mat.L100.fidap022	100	0.68	62	102	62	2.77	99	62	2.92
mat.L80.fidap001	80	0.72	54	83	54	1.54	83	54	1.75
mat.L80.fidap022	80	0.76	41	82	41	1.57	82	41	1.81
mat.L100.fidap027	100	0.81	69	102	69	3.16	101	69	3.37
mat.L100.fidap002	100	0.82	66	108	66	2.35	107	66	2.73
mat.L120.fidap002	120	0.82	68	128	68	3.08	127	68	3.79
mat.L120.fidap027	120	0.85	83	123	83	4.48	123	83	4.51
miplib.noswot.p	182	0.09	167	190	146	2.04	189	138	5.76
miplib.khb05250.p	100	0.27	75	100	75	0.96	100	75	1.25
miplib.stein27_r.p	118	0.32	62	121	62	17.18	116	62	22.4
miplib.10teams.p	210	0.34	120	220	120	12.17	219	120	23.76
miplib.mod010.p	146	0.38	90	152	85	4.54	151	85	6.78
miplib.l152lav.p	97	0.40	61	99	58	2.39	98	59	2.64
miplib.lp4l.p	85	0.46	50	85	48	1.89	85	47	2.07
miplib.air03.p	124	0.61	75	126	72	4.63	127	73	4.75
miplib.misc03.p	96	0.63	52	96	52	3.75	91	52	5.7
miplib.misc07.p	212	0.80	116	222	114	10.80	215	115	7.06

Table 4.3: Results for VSP instances: NDR&C and PR&C separating only Connected Dominating inequalities.

Instance				NDR&C (CD)			PR&C (CD)		
label	n	d	Opt	ub	lb	t(s)	ub	lb	t(s)
dim.DSJC125.1	125	0.09	90	122	89	4.12	122	88	4.37
dim.games120	120	0.09	102	121	99	2.14	120	99	1.67
dim.myciel7	191	0.13	156	193	155	3.71	188	155	3.86
dim.myciel6	95	0.17	76	90	75	1.43	89	75	1.18
dim.queen12_12	144	0.25	97	131	97	6.69	131	97	7.57
dim.queen11_11	121	0.27	81	109	81	5.56	108	81	5.97
dim.queen10_10	100	0.30	67	88	67	4.04	88	67	4.47
dim.queen8_12	96	0.30	65	85	65	4.02	85	65	3.69
dim.queen9_9	81	0.33	55	69	55	2.73	70	55	3.21
dim.queen8_8	64	0.36	43	53	43	1.67	53	43	2.10
dim.miles1000	128	0.40	110	119	109	4.06	119	109	3.98
dim.queen7_7	49	0.40	31	40	31	0.90	40	31	1.25
dim.DSJC125.5	125	0.50	74	101	74	5.11	101	74	6.11
dim.DSJC125.9	125	0.90	22	63	22	5.73	62	22	6.20
mat.can96	96	0.20	72	89	72	1.78	87	72	2.33
mat.can73	73	0.25	53	65	53	1.60	64	53	1.69
mat.rw136	136	0.07	121	136	120	2.49	133	119	1.62
mat.gre__115	115	0.09	95	114	91	2.98	113	93	3.45
mat.L125.gre__185	125	0.15	104	120	104	4.64	119	104	4.19
mat.can__144	144	0.16	126	136	126	5.60	138	126	5.75
mat.L125.can__161	125	0.16	97	119	95	4.07	118	97	4.10
mat.lund_a	147	0.26	118	130	116	4.83	129	116	5.17
mat.L125.bcsstk05	125	0.35	101	108	101	3.48	104	101	3.68
mat.L125.dwt__193	125	0.38	95	105	95	3.49	102	95	3.68
mat.L125.fs_183_1	125	0.44	98	135	95	2.32	135	97	2.49
mat.bcsstk04	132	0.68	84	94	84	4.63	91	84	4.83
mat.arc130	130	0.93	88	102	88	7.51	100	88	7.80
mat.L100.steam2	100	0.36	76	82	76	2.85	82	76	2.84
mat.L120.fidap025	120	0.39	102	110	102	2.57	111	102	2.76
mat.L120.cavity01	120	0.42	99	120	99	3.36	119	99	2.60
mat.L120.fidap021	120	0.43	98	115	98	2.84	114	98	2.77
mat.L120.rbs480a	120	0.46	88	95	88	3.40	96	88	3.60
mat.L120.wm2	120	0.47	98	127	92	1.73	125	92	2.24
mat.L100.rbs480a	100	0.52	73	82	73	2.26	82	73	2.46
mat.L80.wm2	80	0.58	61	84	60	1.37	82	60	1.42
mat.L100.wm3	100	0.59	77	100	77	2.43	99	74	1.87
mat.L120.e05r0000	120	0.59	90	108	90	2.39	107	90	2.71
mat.L100.wm1	100	0.60	74	102	71	2.23	90	73	2.30
mat.L120.fidap022	120	0.60	84	91	84	3.87	90	84	4.10
mat.L100.fidapm02	100	0.62	69	70	69	2.39	70	69	2.28
mat.L120.fidap001	120	0.63	82	88	82	4.08	87	82	4.40
mat.L100.e05r0000	100	0.64	70	84	70	1.92	85	70	1.99
mat.L80.fidapm02	80	0.65	53	54	53	1.54	**53**	**53**	0.89
mat.L120.fidapm02	120	0.65	86	94	86	3.44	92	86	3.50
mat.L100.fidap001	100	0.68	64	71	64	2.76	69	64	2.87
mat.L100.fidap022	100	0.68	62	71	62	2.83	71	62	2.91
mat.L80.fidap001	80	0.72	54	62	54	1.40	62	54	1.52
mat.L80.fidap022	80	0.76	41	53	41	1.65	52	41	1.85
mat.L100.fidap027	100	0.81	69	70	69	2.48	**69**	**69**	1.60
mat.L100.fidap002	100	0.82	66	86	66	1.91	85	66	2.21
mat.L120.fidap002	120	0.82	68	91	68	3.09	89	68	3.15
mat.L120.fidap027	120	0.85	83	84	83	3.51	**83**	**83**	3.02
miplib.noswot.p	182	0.09	167	187	139	3.24	186	146	2.77
miplib.khb05250.p	100	0.27	75	99	75	1.10	95	75	1.20
miplib.stein27_r.p	118	0.32	62	116	62	3.78	106	62	4.20
miplib.10teams.p	210	0.34	120	203	120	10.55	180	120	11.21
miplib.mod010.p	146	0.38	90	145	88	3.99	126	86	5.26
miplib.l152lav.p	97	0.40	61	97	60	1.73	79	60	2.34
miplib.lp4l.p	85	0.46	50	80	47	1.87	63	48	1.95
miplib.air03.p	124	0.61	75	124	73	3.45	107	73	4.64
miplib.misc03.p	96	0.63	52	83	52	3.54	72	52	2.81
miplib.misc07.p	212	0.80	116	218	113	12.64	212	114	11.80

Table 4.4: Results for VSP instances: NDR&C and PR&C separating Lifted Dominating and Connected Dominating inequalities.

Instance				NDR&C (CD, LD)			PR&C (CD, LD)		
label	n	d	Opt	ub	lb	t(s)	ub	lb	t(s)
dim.DSJC125.1	125	0.09	90	122	88	13.23	124	89	11.28
dim.games120	120	0.09	102	120	99	2.79	120	99	4.64
dim.myciel7	191	0.13	156	192	153	5.19	188	155	5.84
dim.myciel6	95	0.17	76	92	73	2.05	89	75	2.59
dim.queen12_12	144	0.25	97	133	97	21.71	132	97	25.83
dim.queen11_11	121	0.27	81	110	81	17.76	109	81	20.23
dim.queen10_10	100	0.30	67	89	67	12.28	89	67	16.74
dim.queen8_12	96	0.30	65	86	65	13.26	86	65	18.30
dim.queen9_9	81	0.33	55	70	55	8.84	71	55	13.57
dim.queen8_8	64	0.36	43	54	43	7.28	54	43	9.53
dim.miles1000	128	0.40	110	120	110	8.23	120	109	9.04
dim.queen7_7	49	0.40	31	40	31	3.17	40	31	6.36
dim.DSJC125.5	125	0.50	74	101	74	13.49	102	74	17.00
dim.DSJC125.9	125	0.90	22	63	22	6.63	62	22	6.61
mat.can96	96	0.20	72	89	72	6.52	89	72	9.09
mat.can73	73	0.25	53	66	53	6.91	66	53	7.76
mat.rw136	136	0.07	121	136	120	5.17	135	119	18.66
mat.gre__115	115	0.09	95	114	90	9.01	114	91	9.82
mat.L125.gre__185	125	0.15	104	120	104	20.01	122	104	22.37
mat.can__144	144	0.16	126	138	126	24.88	140	126	24.48
mat.L125.can__161	125	0.16	97	119	95	14.98	120	97	15.38
mat.lund_a	147	0.26	118	136	116	17.08	130	116	16.97
mat.L125.bcsstk05	125	0.35	101	116	101	8.35	107	101	10.30
mat.L125.dwt__193	125	0.38	95	107	95	8.60	106	95	6.91
mat.L125.fs_183_1	125	0.44	98	135	95	2.59	134	98	2.70
mat.bcsstk04	132	0.68	84	91	84	4.61	90	84	5.36
mat.arc130	130	0.93	88	103	88	12.65	100	88	13.21
mat.L100.steam2	100	0.36	76	83	76	9.43	83	76	10.76
mat.L120.fidap025	120	0.39	102	108	102	4.49	110	102	6.08
mat.L120.cavity01	120	0.42	99	121	99	4.77	122	98	4.88
mat.L120.fidap021	120	0.43	98	114	98	4.59	116	98	4.76
mat.L120.rbs480a	120	0.46	88	97	88	6.44	96	88	5.72
mat.L120.wm2	120	0.47	98	127	92	1.89	125	92	2.37
mat.L100.rbs480a	100	0.52	73	81	73	2.74	82	73	2.70
mat.L80.wm2	80	0.58	61	84	59	1.66	80	61	1.96
mat.L100.wm3	100	0.59	77	103	71	1.96	99	76	2.64
mat.L120.e05r0000	120	0.59	90	108	90	2.90	108	90	3.20
mat.L100.wm1	100	0.60	74	102	71	2.39	95	73	3.42
mat.L120.fidap022	120	0.60	84	92	84	4.70	91	84	4.20
mat.L100.fidapm02	100	0.62	69	70	**69**	2.79	**69**	**69**	2.70
mat.L120.fidap001	120	0.63	82	87	82	5.60	87	82	5.48
mat.L100.e05r0000	100	0.64	70	84	70	2.10	85	70	2.19
mat.L80.fidapm02	80	0.65	53	54	**53**	1.73	54	**53**	1.74
mat.L120.fidapm02	120	0.65	86	93	86	4.94	93	86	4.37
mat.L100.fidap001	100	0.68	64	73	64	3.06	70	64	3.20
mat.L100.fidap022	100	0.68	62	71	62	3.12	71	62	3.34
mat.L80.fidap001	80	0.72	54	62	54	1.47	62	54	1.75
mat.L80.fidap022	80	0.76	41	53	41	1.95	51	41	2.00
mat.L100.fidap027	100	0.81	69	**69**	**69**	1.94	**69**	**69**	2.12
mat.L100.fidap002	100	0.82	66	86	66	2.05	85	66	2.45
mat.L120.fidap002	120	0.82	68	88	68	3.41	89	68	3.37
mat.L120.fidap027	120	0.85	83	84	**83**	3.83	**83**	**83**	2.33
miplib.noswot.p	182	0.09	167	189	140	5.27	188	146	4.52
miplib.khb05250.p	100	0.27	75	99	75	1.20	95	75	1.31
miplib.stein27_r.p	118	0.32	62	118	62	18.84	110	62	23.56
miplib.10teams.p	210	0.34	120	205	120	25.06	188	120	25.99
miplib.mod010.p	146	0.38	90	149	85	6.74	131	85	19.19
miplib.l152lav.p	97	0.40	61	95	58	2.60	83	60	6.47
miplib.lp4l.p	85	0.46	50	81	48	1.82	70	49	4.32
miplib.air03.p	124	0.61	75	123	74	5.83	109	73	7.90
miplib.misc03.p	96	0.63	52	82	52	9.85	78	52	11.03
miplib.misc07.p	212	0.80	116	214	115	20.19	212	114	13.92

the computational results reported in [11]. There, instances are classified in high ($\geq 35\%$) or low density instances, according to the density of the graphs. We proceeded similarly in our preliminary tests. However, these tests lead us to regroup the instances according to a diferrent classification of ranges: mid-high ($> 20\%$) and low density instances, corresponding to a total of, respectively, 38 and 9 instances in each range. Accordingly, the results reported in this section are organized in light of these groupings. Furthermore, for each group, the comparison measure used as a basis for the majority of our conclusions was the total amount of time (and/or nodes) required to solve all the instances in the group.

Combined results for mid-high density graphs

We compile and report in this section the main results obtained with mid-high density input graphs. Because they were gotten over the majority of the instances in our final test set, they compose the most representative results we published to the VSP.

Table 4.5 compares the main results obtained with the most promising combinations yielded for mid-high density instances. A total of three postponed and non-delayed relax-and-cut configurations have their results reported[7]. Also, the results returned by two variations of [11] - to our knowledge, the best algorithm available in literature to solve the VSP - and by XPRESS Optimizer under default settings are shown.

C2 (Pg.115)

The number of nodes and the time required for each algorithm are reported. When the time is ≥ 1800 seconds, it means that the instance was not solved by the corresponding algorithm. The main headings that identify the relax-and-cut configurations whose results are presented in Table 4.5 have the following meaning: `(i) Only CD Cuts`: corresponds to the usage of our NDR&C(CD) (or PR&C(CD)) algorithm followed by L-B&C, with sep $=\langle$`L-CD,CD`$\rangle$; `(ii) CD Cuts and L-LD Cuts`: combines the usage of NDR&C(CD,LD) (or PR&C(CD,LD)) as relax-and-cut algorithm with L-B&C, with sep $=\langle$`L-CD,CD,L-LD`$\rangle$; `(iii) CD and LD Cuts`: refers to NDR&C(CD,LD) (or PR&C(CD,LD)) preceding L-B&C, with sep $=\langle$`L-CD,CD,L-LD,LD`$\rangle$. Also, columns `B&C(CD)` and `B&C(CD,LD)` correspond to the algorithm described in [11] separating, respectively, only CD and both, CD and LD inequalities. Finally, XPRESS Optimizer results are reported on the last two columns.

At the bottom of each column, for each algorithm, three summations are shown. The first of them corresponds to the total time (or total number of nodes yield by the search trees) only for those algorithms that solved to optimality the whole set of instances listed in the table. To understand the other summations, let S' be the subset of instances in Table 4.5 that are solved by any HYBRID version and any B&C configuration within 30

[7] These configurations give priority to the use of CD inequalities, as suggest the results for high-density instances reported in [11]

CPU minutes, i.e., all those in Table 4.5 but `miplib.misc07.p`. Likewise, let $S'' \subset S'$ be the subset of instances in S' also solved by XPRESS algorithm (under default settings) within the same imposed time limit. (i.e., $S' \backslash S'' = \{$`dim.DSJC125.9`, `mat.lund_a`, `mat.bcsstk04`, `mat.L120.fidap001` and `miplib.air03.p`$\}$) The penultimate (last) line contains the total number of nodes and time needed by each approach to solve all the instances in S' (S'') subset. If not explicitly stated otherwise, our comparisons of results are restricted to instances in S'' only when XPRESS Optimizer results are also under consideration.

Entries appearing with no value (symbol "–") correspond to the instances that were solved after the LP module execution, i.e., before branching. These entries permit us to conclude that, beyond the four instances already solved during Lagrangian phase, up to six more instances were solved with no need of branching (see Figure 4.6).

Comparing the results obtained by the nine approaches listed in Table 4.5, that are summarized in the last three lines of the table, we can conclude that: `(1)` all the six combinations of relax-and-cut algorithms with linear programming proposed outperform the B&C algorithm detailed in [11]; `(2)` over the subset S'' of instances, the HYBRID algorithms developed are also better, in terms of time, than the other three approaches. However, in number of nodes of the search tree, the B&C(CD) reached a slightly better performance than our best HYBRID configurations; `(3)` algorithms based principally on `CD` inequalities seem to be the most promising approaches currently available to tackle mid-high density VSP instances. This observation reinforces the conclusions reported in [11] about the use and the strength of `CD` inequalities to solve high ($\geq 35\%$) density instances. Nevertheless, the outcomes resulting from the experiments carried out in the present work showed that, for instances `mat.can73`, `mat.lund_a` and `miplib.khb05250.p`, our hybrid algorithms and the branch-and-cut described in [11] outperform `XPRESS`. Hence, it admit to enlarge the frontier of the conclusions from de Souza and Balas to mid density (over 20%) graphs.

Besides, we would like to remark that five out of our six configurations were able to solve the instance `miplib.misc07.p` whereas no other approach tested solved it within 30 minutes. Thus, confronting only results obtained by those five algorithms over all the instances in Table 4.5 we are able to rank the several HYBRID versions implemented. For instance, proceeding like that, we conclude that our three best results come from using only CD cuts and from using NDR&C(CD,LD) as relax-and-cut algorithm combined with L-B&C, with sep =⟨`L-CD,CD,L-LD`⟩.

In addition to that conclusion about our best combined versions, to compare our framework with the algorithms described in [11] and with XPRESS Optimizer, we need to discard results obtained with instance `miplib.misc07.p`. Doing that and taking into account the results in Table 4.5, we are driven to adopt PHYBRID(CD) as being the best algorithm we

Table 4.5: Results for VSP instances using HYBRID configurations, B&C algorithms from [11] and XP Optimizer.

Mid-high density instance		Only CD Cuts				CD Cuts and L-LD Cuts				CD and LD Cuts				B&C [ref.[11]]				B&B (XP)	
		NDHYBRID		PHYBRID		NDHYBRID		PHYBRID		NDHYBRID		PHYBRID		B&C(CD)		B&C(CD,LD)			
label	$d(>20\%)$	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)
dim.queen8_8	0.36	1807	75.02	1707	90.22	1863	85.01	2283	136.40	2541	125.79	2283	136.03	4315	70.51	3143	58.27	23131	126.29
dim.miles1000	0.40	17	12.97	13	11.62	7	10.82	9	14.05	9	11.52	9	14.65	11	18.37	35	26.22	287	83.96
dim.queen7_7	0.40	391	14.11	313	16.73	403	16.56	243	20.48	253	17.13	243	20.89	431	10.77	265	9.93	27833	78.53
dim.DSJC125.9	0.90	4275	596.45	4143	537.01	4229	532.45	6405	722.50	6375	743.80	6417	775.66	33833	1107.29	28475	1291.54	51261	1800.00
mat.can73	0.25	5343	46.15	5505	50.57	4975	64.53	5157	59.63	5367	57.37	5195	59.35	5123	71.50	5615	46.44	33195	147.62
mat.lund_a	0.26	3145	401.39	2715	384.14	2033	454.57	3565	543.09	3345	479.10	3005	511.01	2231	462.03	2709	332.09	27506	1800.00
mat.L125.bcsstk05	0.35	709	85.65	-	4.53	-	7.25	-	8.02	-	7.25	-	8.02	1573	326.20	1625	196.01	1635	211.38
mat.L125.dwt_193	0.38	21	23.11	27	25.07	21	34.49	39	40.78	41	45.06	39	40.66	131	97.31	721	134.80	17767	1301.43
mat.L125.fs_183_1	0.44	29	28.11	21	26.16	29	32.82	746	69.37	29	43.27	27	41.02	25	35.45	29	31.74	1515	182.78
mat.bcsstk04	0.68	13	22.90	-	4.35	13	22.02	-	5.68	31	31.14	-	5.68	133	124.60	247	132.27	16572	1800.00
mat.arc130	0.93	83	160.19	83	163.80	103	186.66	73	231.78	75	246.58	73	235.17	101	370.67	101	329.59	957	926.12
mat.L100.steam2	0.36	45	21.02	41	20.67	45	25.73	79	38.06	77	34.79	69	35.92	149	40.83	241	37.98	11577	229.98
mat.L120.fidap025	0.39	-	2.50	-	2.59	-	3.58	-	4.75	-	3.58	-	4.75	13	12.00	49	17.67	889	107.73
mat.L120.cavity01	0.42	13	9.13	15	10.50	11	9.62	35	18.35	35	14.87	15	11.83	13	16.88	41	21.52	813	91.69
mat.L120.fidap021	0.43	5	5.63	7	5.88	3	7.14	3	6.98	3	7.34	3	7.00	35	24.74	67	36.63	1031	150.22
mat.L120.rbs480a	0.46	125	64.01	141	67.75	123	71.81	75	59.20	75	60.34	75	61.14	367	218.67	3007	249.27	15619	1308.79
mat.L120.wm2	0.47	33	28.04	35	36.68	75	19.47	73	26.55	75	20.25	33	38.57	33	47.82	33	45.71	351	88.71
mat.L100.rbs480a	0.52	59	11.90	67	15.60	61	18.18	45	15.44	49	17.81	45	16.76	63	21.33	91	21.34	2951	189.73
mat.L80.wm2	0.58	9	3.28	11	4.31	13	4.82	51	6.07	13	5.90	13	6.53	13	4.90	15	5.65	379	67.22
mat.L100.wm3	0.59	11	7.63	13	10.48	17	12.00	61	16.24	19	18.64	13	10.66	17	13.26	15	13.34	379	65.70
mat.L120.e05r0000	0.59	5	7.05	3	7.63	7	7.70	7	7.74	5	8.51	7	7.82	9	11.49	43	25.31	2703	543.05
mat.L100.wm1	0.60	19	10.74	13	9.18	17	9.59	71	18.40	27	14.35	17	12.00	25	15.67	35	24.36	877	94.07
mat.L120.fidap022	0.60	77	22.53	17	13.75	81	27.72	43	20.60	41	20.28	43	21.22	53	38.17	81	53.04	13319	1522.86
mat.L120.fidap001	0.63	-	4.94	-	5.07	-	6.71	-	6.66	-	6.71	-	6.66	31	32.57	189	84.70	33120	1800.00
mat.L100.e05r0000	0.64	15	8.39	17	8.17	15	8.75	19	6.46	33	8.50	19	6.59	19	11.19	39	12.49	3559	284.25
mat.L120.fidapm02	0.65	-	2.87	-	2.91	-	4.28	-	3.65	-	4.28	-	3.65	17	24.52	57	55.66	4457	552.37
mat.L100.fidap001	0.68	35	7.10	35	7.54	33	9.00	29	9.87	35	11.46	29	9.82	49	15.96	73	23.21	34321	950.38
mat.L100.fidap022	0.68	109	22.66	105	22.36	99	23.75	54	16.17	63	18.00	54	17.19	171	52.20	93	27.31	57415	1594.48
mat.L80.fidap001	0.72	-	1.55	-	1.55	-	1.61	-	1.75	-	1.61	-	1.75	1	1.76	33	5.20	3523	101.25
mat.L80.fidap022	0.76	197	14.25	159	11.90	135	11.92	57	7.18	55	7.89	57	7.68	173	15.28	45	5.51	19279	308.05
mat.L100.fidap002	0.82	5	3.22	3	3.52	5	3.48	5	4.00	3	3.10	5	4.05	7	4.75	29	10.19	2111	240.58
mat.L120.fidap002	0.82	5	6.88	1	5.73	5	6.60	3	5.56	3	6.72	3	5.58	73	41.59	93	48.59	10415	1284.46
miplib.khb05250.p	0.27	119	3.94	121	5.26	119	4.28	121	5.16	119	4.48	121	5.43	91	3.21	111	4.84	3641	66.37
miplib.l152lav.p	0.40	213	46.27	185	54.66	245	64.35	611	72.76	749	103.57	129	60.48	283	70.12	853	101.98	22885	567.68
miplib.lp4l.p	0.46	275	34.27	271	33.10	321	37.02	2083	93.82	1295	69.37	265	50.19	551	50.10	5965	151.72	27523	409.73
miplib.air03.p	0.61	115	111.94	117	119.98	117	111.50	135	139.86	119	146.10	111	163.87	135	167.35	135	180.01	14215	1800.00
miplib.misc03.p	0.63	2993	111.95	2717	121.67	3293	138.31	2589	168.23	2785	178.78	2225	173.42	4819	138.07	3947	155.22	53417	1794.42
miplib.misc07.p	0.80	173	1280.36	177	1519.38	169	1162.75	1102	1809.86	117	1402.40	147	1704.75	125	1800.00	78	1800.00	2243	1800.00
$\sum$ L-B&Cs		20480	3320.10	18789	3442.02	18685	3258.85	-	-	23861	4007.64	20789	4303.45	-	-	-	-	-	-
$\sum$ B&Cs		20315	2039.74	18621	1922.64	18516	2096.10	24769	2631.29	23744	2605.24	20642	2598.70	47706	3286.19	58345	4007.35	-	-
$\sum$ B&Cs and XP		15912	1303.51	14361	1256.23	14157	1423.42	18229	1756.59	17219	1677.49	14114	1646.83	13574	1854.38	29299	2318.83	427260	17471.88

have in terms of time performance. So, its assumption is taken in all comparisons carried out from now on through this section when considering total time comparisons. Accordingly, Figure 4.9 highlights the results obtained with PHYBRID(CD) and B&C(CD), the best branch-and-cut version from [11]. Essentially, the performance of these algorithms is measured as a percentage of the time required by XPRESS Optimizer B&B to solve the instances in S''. First, notice that, in general, both algorithms are much faster than XPRESS B&B default version. Actually, only for instance `mat.L125.bcsstk05` XPRESS Optimizer surpassed B&C(CD). Also, on average, PHYBRID(CD) is about two times faster than B&C(CD): the first requires, on average, about 10.6% of the computation time used by XPRESS to solve the instances while the last needs aproximately 19.3% of the same amount of time.

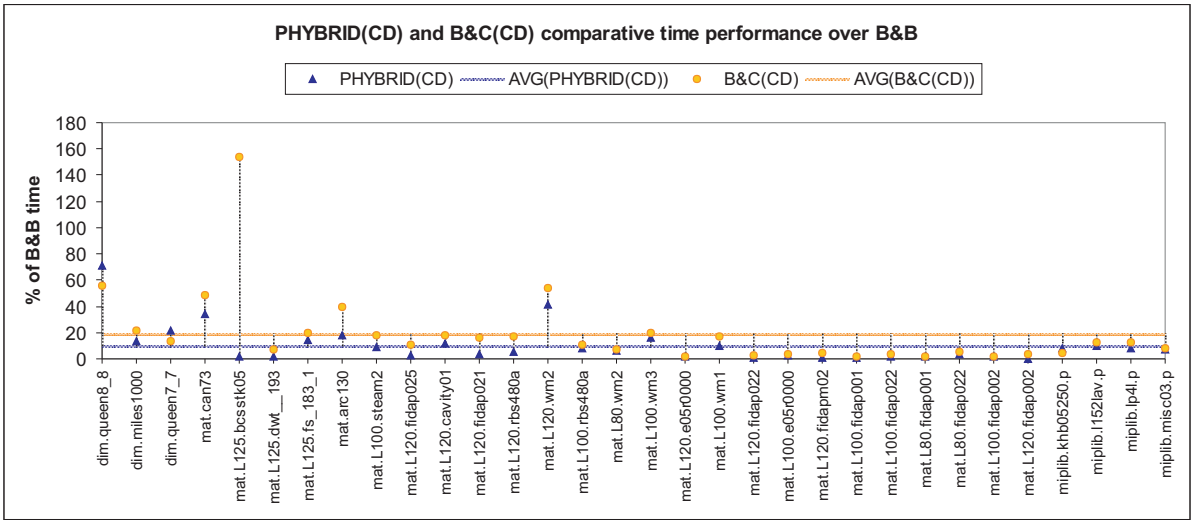

Figure 4.9: Time performance: comparing results of PHYBRID and B&C(CD) against XPRESS Optimizer default B&B algorithm.

C2 (Pg.115)

Figures 4.10 and 4.11 exhibit a graphical comparison of the six combined approaches reported in Table 4.5 against the two best B&C versions implemented in [11]. Thus, regarding computation time, Figure 4.10 shows the big savings obtained by our algorithms when compared with the best branch-and-cut algorithms currently available to solve the VSP. For instance, the effect of replacing B&C(CD,LD) by PHYBRID(CD) would be a CPU time reduction of 52%. Also, on average, considering the results for the six variations of our hybrid approach, time savings over B&C(CD,LD) and B&C(CD) are, respectively, of about 42% and 30%.

It deserves to highlight that, in our graphics, a simplified notation was adopted to identify our R&C algorithms. Basically, the term/parameter `L-CD` (`L-LD`) is used to denote that along L-B&C execution only Lagrangian CD (LD) cuts are identified. Hence,

for instance, PR&C(CD,L-LD) means that PR&C(CD,LD) was used as preprocessing with sep $=\langle$`L-CD,CD,L-LD`$\rangle$ during L-B&C.

Together with time performance, a criterion commonly used to compare enumeration algorithms is the total number of nodes. This number corresponds to the amount of nodes associated to all enumeration trees generated when solving the set of benchmarks. With regard to this aspect, Figure 4.11 suggests that our algorithms are stronger than the branch-and-cut algorithms described in [11]. However, this enormous seeming advantage over B&C(CD), the best B&C configuration in [11], comes from the difficulty found by the latter algorithm to solve instance `dim.DSJC15.9` (see Table 4.5). A fairest analysis should, perhaps, not take that instance into account, what would lead us to a (statistically) more acceptable conclusion. Proceeding like that, we could state that, in number of nodes, B&C(CD) is as good as our best algorithms.

In fact, going back to Table 4.5, we can verify that instance `dim.DSJC125.9` has a big contribution in both measures, total number of nodes and total time. Nevertheless, that instance does not belong to S'' and our time advantages are also notorious over S''. For instance, comparing PHYBRID(CD) with B&C(CD) one can notice that: considering S' we had a time reduction of 41.5% and over S'' this reduction drops to 32.3%. Moreover, PHYBRID(CD) beats B&C(CD) in 92% of the cases, namely, 35 out of the 38 mid-high density instances tested! This can be easily seem with the help of figures 4.13 or 4.14.

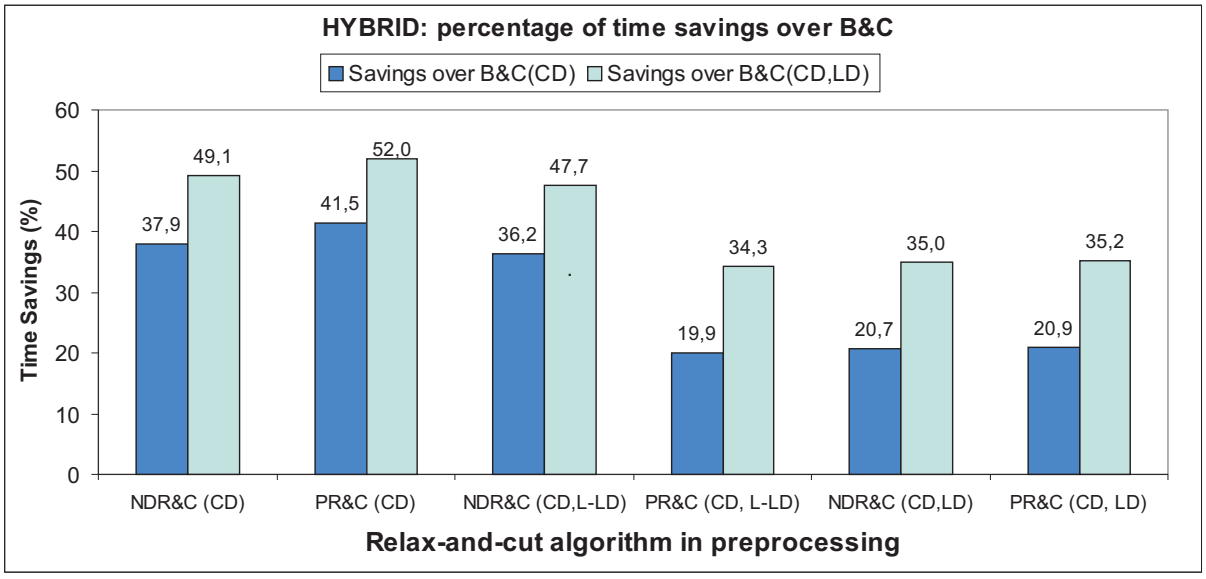

Figure 4.10: Time performance: comparing results of HYBRID versions against results with execution of B&C algorithms described in [11].

Although usual in literature, analysis of computational results restricted to total time analysis may mislead the conclusions. So, at this point, a few words may be necessary

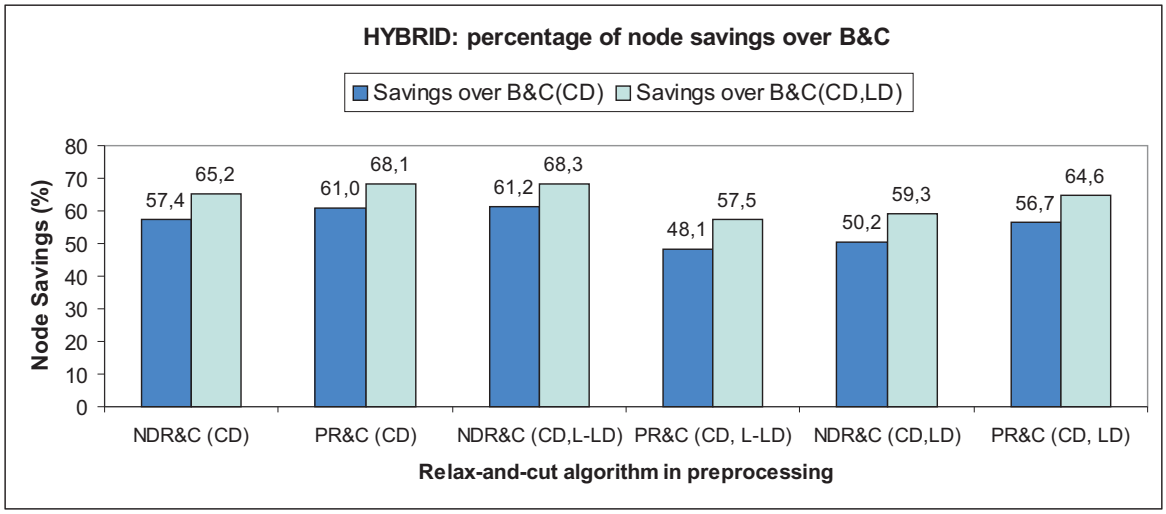

Figure 4.11: Sizes of the search trees: comparing results of HYBRID versions against results with execution of B&C algorithms described in [11].

to discuss the distortion that the inclusion of instance `dim.DSJC125.9` might cause when interpreting the results obtained. In regard to that point, we constructed the graphic in Figure 4.12 that shows an alternative way of composing results on time performance. There, for each HYBRID algorithm, a percentage of B&C(CD) CPU time spent to solve each instance in S' is computed and an arithmetic mean is taken. Basically, the average expression is given by $AVG(a) = 100 \times \frac{1}{|S'|} \times \sum_{i \in S'} (\frac{t(i,\texttt{HYBRID}(a))}{t(\texttt{i,B\&C(CD)})}), a \in \{1, ..., 6\}$, where a are labels corresponding to the HYBRID algorithms tested and $t(i, A)$ refers to the CPU time algorithm labeled as A takes to solve instance i. Hence, those values refer to the percentage of B&C(CD) execution time one should expect to be spent, on average, by each one of our algorithms. The premise adopted is that, under discrepancies of running times, replacing total time by percentage may lead to a more suitable analysis.

Comparing graphics in Figure 4.10 and Figure 4.12 we see, however, that quite similar values were generated, presenting deviation inferior to 4%. For example, the time reduction with NDHYBRID(CD) in Figure 4.10 is of 37.9% while in Figure 4.12 it is of 41.3%. Notice that, rather than unexpected, the similarities between both results reinforce what is shown in Figure 4.10. Taking into account these computational results concerning time performance we can conclude that NDHYBRID(CD) and PHYBRID(CD) are the best algorithms we developed. To better illustrate that in details, graphics in Figures 4.13 and 4.14 carry out a comparison between both approaches. Similarly to Figure 4.12, it was adopted percentages over B&C(CD) running times.

Based on results shown in Figure 4.10 and Figure 4.12 it seems straightforward to conclude that PHYBRID(CD) and NDHYBRID(CD) are the two most promising algo-

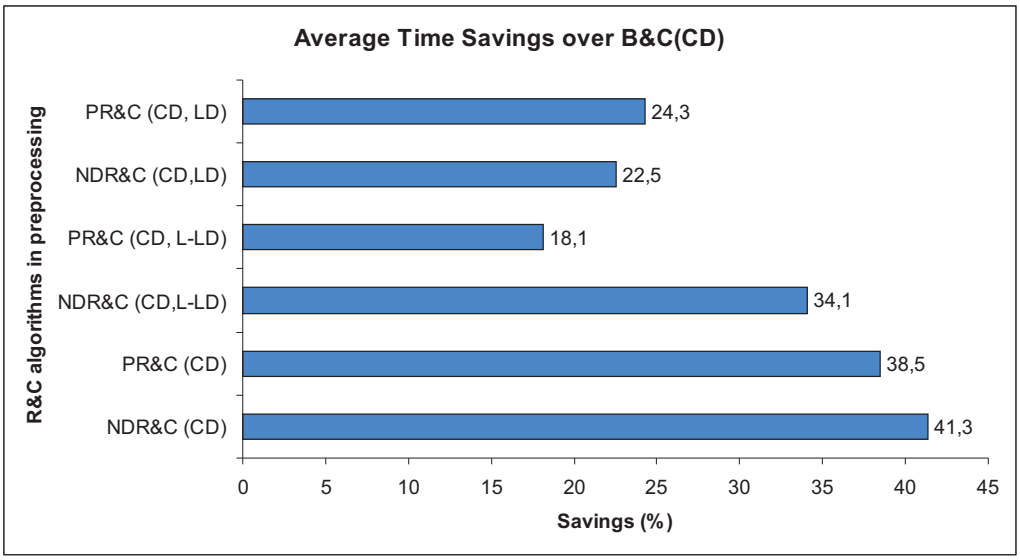

Figure 4.12: Average time savings of each HYBRID algorithm developed computed over percentage of B&C(CD) CPU time required to solve each instance in S'.

rithms to solve instances in S'. However, these results do not render possible to assert that one algorithm excels the other one. Thus, aiming to distinguish when using one algorithm may be more suitable than applying the other, two further investigations were performed. Both adopt percentages over B&C(CD) running times in order to compare the algorithms.

The first examination is documented in Figure 4.13. It tried to identify whether, for a certain class of instances, an algorithm surpasses the other one. However, only a weak evidence of superiority was observed regarding the results with instances from MIPLIB repository. For that group of instances, in general, NDHYBRID(CD) was faster than the corresponding postponed version. Nevertheless, just a few instances form this group. Moreover, only a relatively small difference of performance between both approaches was noticed. Hence, we are not able to deduce any matching between class of instances and more adequacy of the algorithms to tackle each class.

Figure 4.14 shows results of our second investigation. It contains, essentially, the same data shown in Figure 4.13, but the results are presented in increasing order of graph density. As a result, we can remark that for instances having input graph density below to 60%, in general, NDHYBRID(CD) outperforms the postponed version. On the other hand, PHYBRID(CD) shows to be more suitable to solve higher density instances. So, this experimentation provide us with the (weak) suggestion of using graph densities to decide when to apply one of these two algorithms.

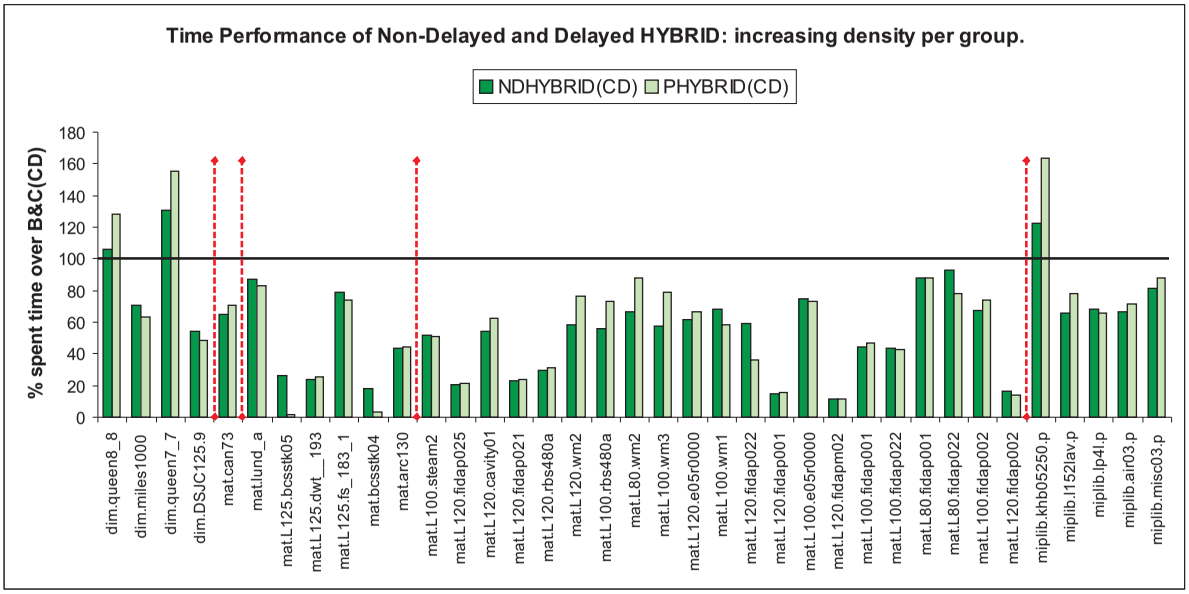

Figure 4.13: Comparing time performances of postponed and non-delayed versions: grouping by instance classes (dashed lines split the instances according to the five classes).

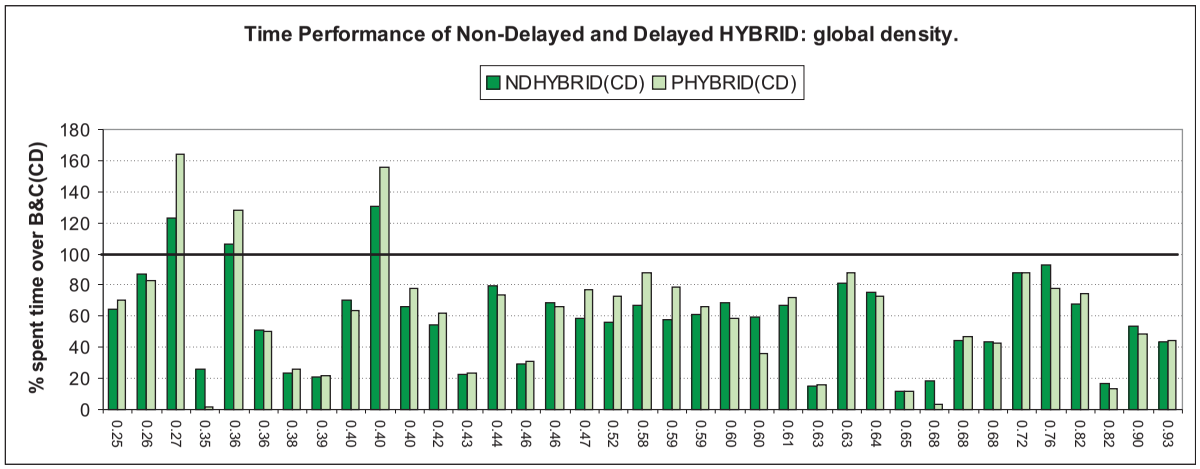

Figure 4.14: Comparing time performances of postponed and non-delayed versions: grouping by input graph densities.

Applying the Wilcoxon signed-rank (WSR) test

Wilcoxon signed rank test is a well known nonparametric statistical test [5, 14] that have been used in the optimization literature [1, 12, 14, 21] to compare any two heuristics. The final outcome of the test is always given in terms of the null hypothesis: we either reject the null hypothesis or fail to reject it. When we reject the null hypothesis we have only shown that it is highly unlikely to be true - we have not proven it in the mathematical sense. Rejecting the null hypothesis then, suggests that an alternative hypothesis may be true. Alternative hypothesis may be one-tailed or two-tailed. A one-tailed hypothesis claims that a parameter is either larger or smaller than the value given by the null hypothesis. A two-tailed hypothesis claims that a parameter is simply not equal to the value given by the null hypothesis - the direction does not matter.

Here, we apply WSR test to try to infer, statistically, any superiority between PHYBRID(CD) and NDHYBRID(CD) algorithms when solving mid-high density VSP instances (the null hypothesis claims that there is no dominance of one algorithm over the other one). Hence, we adopted a directional (one-tailed) hypothesis when applying the test and we carried out the test over results corresponding to both algorithms used to build Figure 4.12.

Let W and W' be, respectively, the sum of the signed ranks and the critical value computed in WSR test. As the number of instances used in the test, n, increases, the distribution of W tends toward the normal distribution. Furthermore, for $n \geq 10$, the critical value W' can be approximated by $W' = Z(\alpha)\sqrt{n(n+1)(2n+1)/6}$, where $Z(\alpha)$ corresponds to the standard normal fractile such that a proportion α of the area is to the left of $Z(\alpha)$. In fact, α is the term used to express the level of significance we will accept. As an example, for 90% confidence, $\alpha = 0.10$, and testing a hypothesis at the $\alpha = 0.10$ level or establishing a 90% confidence interval are essentially the same thing. In both cases the critical values and the region of rejection are the same.

In our case, the differences between matched pairs of results computed in WSR were obtained subtracting values obtained by NDHYBRID(CD) from values corresponding to PHYBRID(CD) results. Thus, if $W > W'$, the null hypothesis should be rejected at the α significance level [14]. As a consequence, for that confidence assumed when computing W', it would mean that NDHYBRID(CD) has better performance than PHYBRID(CD). Actually, the sum of the signed ranks computed with our results was 181 and, if we set $\alpha = 0.10$ ($Z(\alpha) = 1.282$), we obtain $W' = 169.96$. This tell us that if the null hypothesis is true, then in only 10% of the cases W is expected to exceed 169.96. Hence, we refute the null hypothesis at the significance level of 90% and infer that NDHYBRID(CD) outperformes PHYBRID(CD).

Applying Wilcoxon signed-rank test clearly increases our confidence in the comparison carried out between both algorithms and graphically shown in Figure 4.12. However, it

is noteworthy that we may not be able to reject the null hypothesis if we try to come out with a stronger evidence of superiority of NDHYBRID(CD). A manner to conclude that is using a smaller α value. For instance, if we set $\alpha = 0.05$ ($Z(\alpha) = 1.645$), we get $W' = 218.08$. It implies, according WSR test, that we failed to reject the null hypothesis for 95% confidence, i.e., no dominance is verified at that significance level.

C3 (Pg.124)

Combined results for sparse graphs

Computational results reported in [11] discourage, in the case of sparse graphs, the use of cutting planes corresponding to CD or LD inequalities. There, in general, the increase in computing time per search tree node resulted in an increase of total computing time. In other words, using XPRESS Optimizer with default settings was, normally, more advantageous for instances presenting low-density input graphs. Despite of these reports, we decided to investigate if that conclusion goes on being true when using a more recent XPRESS solver version. Thus, some variations of our hybrid approach were tested, as well as the two best branch-and-cut configurations from [11]. Tables 4.6 and 4.7 and Figure 4.15 document the main results obtained with this computational experiments.

Table 4.6 shows, essentially, four configurations of our hybrid algorithms: two postponed and two non-delayed relax-and-cut were used as preprocessing. The number of nodes and the total CPU time required are reported for each algorithm. When the time is $\geq$ 1800 seconds, it means that the instance was not solved by the corresponding algorithm. The main headings that identify the relax-and-cut configurations whose results are presented in Table 4.6 have the following meaning: `(i) Only CD Cuts`: corresponds to the usage of our NDR&C(CD) (or PR&C(CD)) followed by L-B&C, with sep $=\langle$`L-CD,CD`$\rangle$; `(ii) L-CD Cuts and L-LD Cuts`: combines the usage of NDR&C(CD,LD) (or PR&C(CD,LD)) preceding L-B&C, with sep $=\langle$`L-CD,L-LD`$\rangle$. `(iii) CD Cuts and L-LD Cuts`: regards to the usage of NDR&C(CD,LD) (or PR&C(CD,LD)) as relax-and-cut algorithm with L-B&C, having sep $=\langle$`L-CD,CD,L-LD`$\rangle$; `(iv) CD and LD Cuts`: refers to NDR&C(CD,LD) (or PR&C(CD,LD)) preceding L-B&C, with sep $=\langle$`L-CD,CD,L-LD,LD`$\rangle$.

Complementary, in Table 4.7, columns `B&C(CD)` and `B&C(CD,LD)` correspond to the algorithm described in [11] separating, respectively, only CD and both, CD and LD inequalities. Also, the results returned by XPRESS Optimizer under default settings are shown. In both tables, at the bottom of each column, two summations are shown for each algorithm. To understand them, let $\dot{S}$ be the subset of instances with sparse graphs that are solved by any HYBRID and any B&C configuration within 30 CPU minutes. In addition, let $\ddot{S} \subset \dot{S}$ be the subset of instances in $\dot{S}$ also solved by XPRESS algorithm (under default settings) within the same imposed time limit. The penultimate (last) line contains the total number of nodes and time needed by each approach to solve all the

instances in $\dot{S}$ ($\ddot{S}$) subset.

Examining the values corresponding to the variants of our combined approach (Table 4.6, subset $\dot{S}$) we can deduce that our best performances were attained by NDHYBRID(L-CD,L-LD) and PHYBRID(L-CD,L-LD). This observation lead us to infer that, in this case, cuts discovered during the Lagrangian phase were helpful.

Confronting B&C algorithms via results in Table 4.7 we can conclude that separating both, `CD` and `LD` cuts, is better than identifying only CD inequalities. So, as opposed to what has been seen to mid-high density instances, B&C(CD,LD) outperforms B&C(CD).

Inspecting the behavior of XPRESS Optimizer in Table 4.7, we can see that it was the only algorithm to solve instances `mat.can96` and `mat.rw136`. Also, together with NDHYBRID(CD,L-CD), they were the only ones to solve instance `mat.can__144`. On the other hand, XPRESS was the single approach to fail when trying to solve `dim.myciel7`. Now, restricting ourselves to the other five instances (i.e., $\ddot{S}$ set), results suggest that our three best hybrid algorithms, NDHYBRID(L-CD,L-LD), PHYBRID(L-CD,L-LD) and PHYBRID(CD,L-LD), outperform both, B&C from [11] and XPRESS Optimizer, when the criterion is processing time. These results can be more clearly visualized in Figure 4.15 that compile the results of all approaches, confronting them to XPRESS B&B results. Observe that XPRESS is by far the worst algorithm with respect to the number of nodes explored by the search tree. Similarly, in terms of processing time, it is worse than all approaches, except NDHYBRID(CD,L-LD) version.

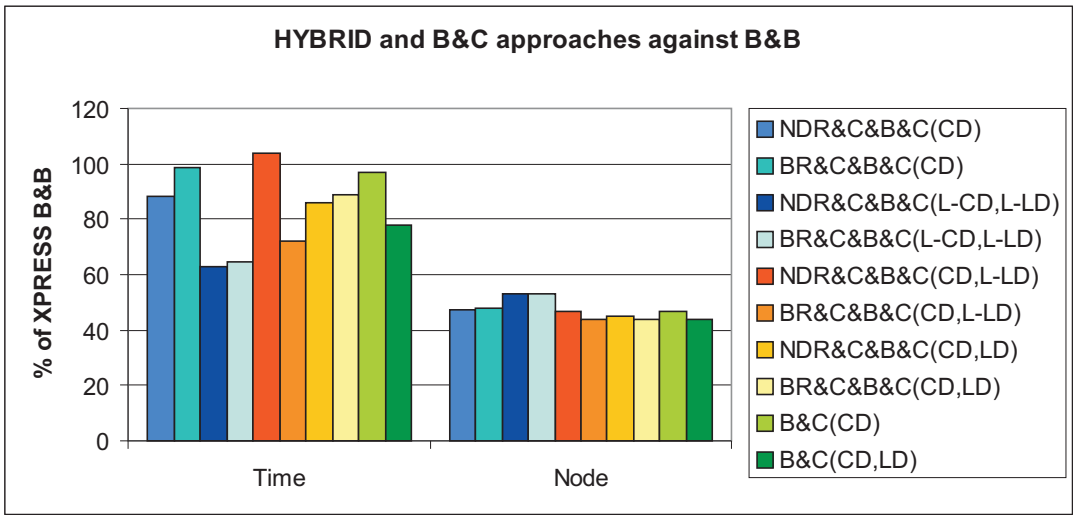

Figure 4.15: Sparse graphs results for instances in $\ddot{S}$.

Table 4.6: Computational results for VSP low density ($\leq 20\%$) instances: NDR&C and PR&C

Low density instance		Only CD Cuts				L-CD Cuts and L-LD Cuts				CD Cuts and L-LD Cuts				CD and LD Cuts			
		NDHYBRID		PHYBRID		NDHYBRID		PHYBRID		NDHYBRID		PHYBRID		NDHYBRID		PHYBRID	
label	d	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)	nodes	t(s)
dim.games120	0.09	84625	789.82	86767	978.87	96985	740.16	97349	754.08	82677	1066.70	85615	869.15	87667	1071.31	85787	1026.52
dim.myciel7	0.13	1823	304.02	2493	383.73	2499	287.90	2799	270.30	2251	374.36	1991	269.26	2779	442.20	2491	429.75
dim.myciel6	0.17	323	9.32	333	10.86	415	10.30	395	10.03	373	13.44	385	11.03	409	14.47	319	13.15
mat.can96	0.20	191769	1801.32	162842	1801.58	192310	1803.07	157351	1803.77	131426	1803.07	134256	1803.77	159271	1803.07	129012	1803.77
mat.rw136	0.07	72063	1802.28	80603	1801.70	345446	1803.05	155538	1812.90	367218	1803.05	88741	1812.90	334492	1803.05	71316	1812.90
mat.gre_115	0.09	7185	92.91	6471	92.90	6393	93.32	6371	89.77	6623	129.86	6047	111.01	6939	127.35	5993	113.05
mat.L125.gre_185	0.15	1169	94.53	1089	92.68	1655	83.39	1639	77.56	1139	105.65	1255	106.63	1191	105.05	1255	111.04
mat.can_144	0.16	38265	1500.97	39849	1807.43	29193	1818.34	24692	1818.96	26048	1818.34	21465	1818.96	24977	1818.34	21666	1818.96
miplib.noswot.p	0.09	27697	1265.17	27759	1351.65	29227	687.80	29219	715.29	27833	1342.92	17853	740.22	18879	882.22	17783	1012.83
$\sum$ B&Cs		122822	2555.77	124912	2910.69	137174	1902.87	137772	1917.03	120896	3032.93	113146	2107.30	117864	2642.60	113628	2706.34
$\sum$ B&Cs and XP		120999	2251.75	122419	2526.96	134675	1614.97	134973	1646.73	118645	2658.57	111155	1838.04	115085	2200.40	111137	2276.59

Table 4.7: Computational results for VSP low density ($\leq 20\%$) instances: B&C and B&B.

Low density instance		B&C				B&B (XP)	
		B&C(CD)		B&C(CD,LD)			
label	d	nodes	t(s)	nodes	t(s)	nodes	t(s)
dim.games120	0.09	82963	886.29	85051	980.73	161485	1077.10
dim.myciel7	0.13	3009	562.94	2033	328.17	28881	1800.00
dim.myciel6	0.17	377	14.60	423	11.69	5243	62.32
mat.can96	0.20	107849	1800.00	167804	1800.00	177163	1569.57
mat.rw136	0.07	62049	1800.00	82714	1800.00	10083	81.55
mat.gre_115	0.09	6517	103.87	6539	83.22	37177	295.66
mat.L125.gre_185	0.15	1205	131.53	1603	71.00	15795	273.68
mat.can_144	0.16	19742	1800.00	27700	1800.00	12683	339.10
miplib.noswot.p	0.09	27891	1343.88	17763	845.51	34719	849.71
$\sum$ B&Cs		121962	3043.11	113412	2320.32	–	–
$\sum$ B&Cs and XP		118953	2480.17	111379	1992.15	254419	2558.47

4.7.3 Primal bound results

Analogously to the study performed with the Set Partitioning Problem [8], our main focus with our relax-and-cut algorithms was to strengthen the dual bounds. Nevertheless, on the primal side, excellent results were achieved. As seen in columns `lb` of tables 4.2, 4.3 and 4.4 and shortly highlighted in section 4.7.1, in about 70% of the cases our simple Lagrangian heuristic found an optimal solution, with slight variations, depending on the relax-and-cut version. To illustrate the quality of our primal heuristic, consider the results obtained by PR&C(CD,LD) algorithm. In this case, the average error of the heuristic was lower than 1.4% and only for 6 instances this error was higher than 5%. However, the maximum error was 19.3% for `miplib.noswot.p`, the only instance for which the error exceeded 8.5%.

For a better appreciation of the performance of the Lagrangian heuristic (`LR-H`), we compare their times with the time needed by B&C(CD,LD) primal heuristic (`LP-H`) to find its best solution. This comparison can be visualized by inspecting the histogram in Figure 4.16 where, to be able to compare processing times, we restricted ourselves to the cases for which both, `LR-H` and `LP-H`, reached an proved optimum. This histogram reveals that `LR-H` finds optimal solutions much quicker than the LP based heuristic from de Souza and Balas. Besides, it shows that in 80% of the cases, the optimum was found in at most 0.01 seconds and, for all instances, `LR-H` reached the optimum in at most one second. On the other hand, in 80% (40%) of the cases, `LP-H` needed at least one (five) second(s) to found an optimum.

C2 (Pg.115)

4.7.4 Results on hard MIPLIB instances

A similar problem that can be considered a generalization of the VSP is discussed in [20] by Borndörfer *et al.* There is described a branch-and-cut algorithm that use cutting planes different from [11] and computational results are reported on a vast number of instances. Several instances from MIPLIB are used and a subset of them demonstrated, experimentally, to be very hard to solve (or remained unsolved) within the time limit imposed to the algorithm execution. Three among these instances have already their results reported in tables 4.5, 4.6 and/or 4.7: `miplib.nosvot.p` and `miplib.misc03.p` were solved by all the approaches tested whereas `miplib.misc07.p` was solved to proven optimality only by the hybrid framework we proposed.

Computational experiments were carried out aiming to assess the performance of our algorithms when tackling also the other instances[8] from MIPLIB not solved in [20]. Likewise, B&C algorithms from de Souza and Balas [11] and XPRESS Optimizer were tested.

[8]Since we are interested in identifying `CD` inequalities, the instance was discarded when the graph underlying it is not connected.

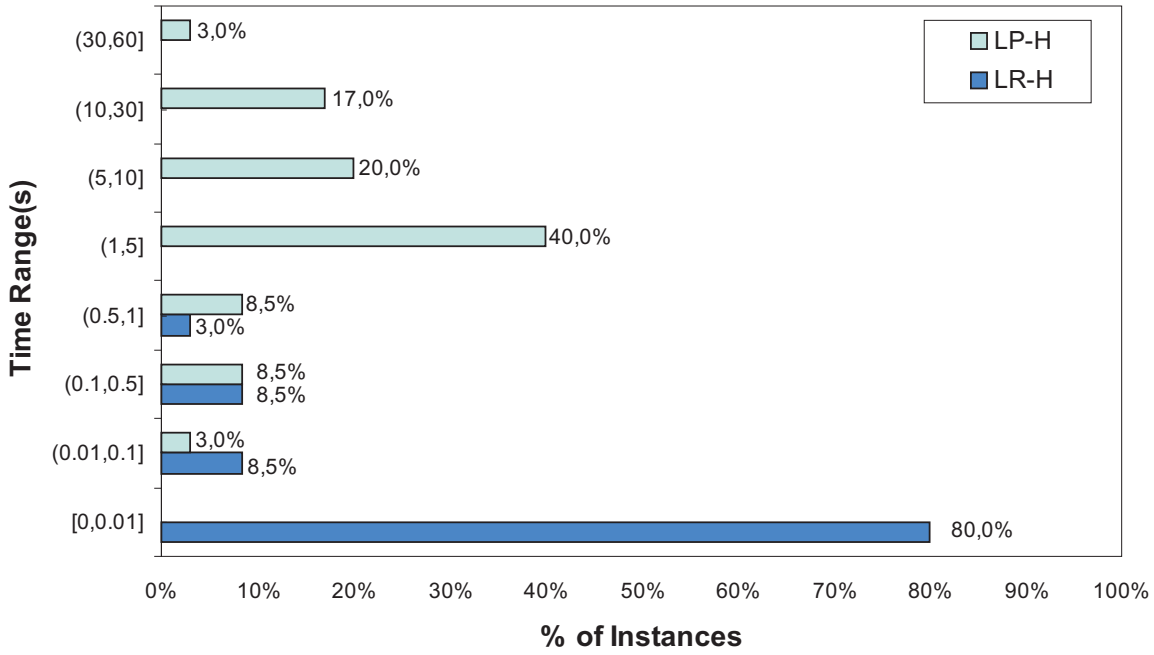

Figure 4.16: Time to optimum for Lagrangian (`LR-H`) and LP-based (`LP-H`) heuristics.

The results obtained are detailed in tables 4.8 and 4.9. They have the same format: columns `R&C` shows computation times spent by the relax-and-cut algorithm used as preprocessing and under `gap` columns we list the final gaps concerning each algorithm. Remark that, with the exception of computation times, these results can also be confronted with those reported in [20][9].

The selection of HYBRID implementations to test these hard MIPLIB instances was based on the results reported in the previous sections. Accordingly, to test mid-high and low density instances we choose, respectively, NDHYBRID(CD) and PHYBRID(L-CD,L-LD) versions. The choice concerning mid-high density instances was quite simple and performed according to results reported in Table 4.5 and graphics in figures 4.12 and 4.13. However, due to the very similar performance between NDHYBRID(L-CD,L-LD) and PHYBRID(L-CD,L-LD) reported in Table 4.6, we proceeded our choice like that: let $t_{i,NDH}$ ($t_{i,PH}$) the the time reported to NDHYBRID(L-CD,L-LD) (PHYBRID(L-CD,L-LD)) to solve instance $i = 1, \ldots, 6$ (only the instances solved by both approaches were took into account). First, we computed the ratio ($r_i = \frac{t_{i,NDH}}{t_{i,PH}}$) for each pair of corre-

[9]Notice: the numbers reported to VSP represent the joint size of both shores in each instance; they are the complements, with respect to the number of vertices, of the separator size in each instance (which is the number reported in [20]).

sponding instance. Further, the value corresponding to the average ratio was computed for those instances, i.e., $AVG(r) = \frac{1}{6} \times \sum_{i=1}^{6} r_i$. Since $AVG(r) = 1.02$, we concluded that NDHYBRID(L-CD,L-LD), on average, was 2% slower than PHYBRID(L-CD,L-LD) to solve those six instances. Hence, we decide to choose PHYBRID(L-CD,L-LD).

Results in Table 4.8 confirm the difficulty found by the various algorithms we tested to solve those instances within the time limit of 30 minutes. However, regarding final gaps (tabled in percentage), they reveal that our hybrid algorithm outperforms the other ones for 5 out of the 7 instances. Also, it was able to solve `p0282` in less than 30% of the time needed by the second best algorithm.

Final gaps reported in Table 4.9 show that, for 12 out of the 14 instances, our combined approach is at least as good as the other algorithms. Notice that our algorithm and the B&C described in [11] were capable to solve the same subset of instances. However, our algorithm did it faster than the latter. On the other hand, the number of nodes yielded by our search trees were usually higher than the amount of nodes generated when applying de Souza and Balas' algorithm.

Table 4.8: Computational results for MIPLIB open problems: mid-high density (> 20%) VSP instances.

Instance			NDHYBRID(CD)				B&C(CD)			B&B			B&C[20]
label	$d(>20\%)$	n	R&C	total t(s)	nodes	gap(%)	t(s)	nodes	gap(%)	t(s)	nodes	gap(%)	gap(%)
fast0507	20.82	484	24.57	1824.57	134	57.20	1800.00	194	68.49	1800.00	1188	86.39	59.14
stein27_r	32.20	118	4.06	1804.06	22930	34.21	1800.00	20577	36.42	1800.00	89909	32.64	38.71
air05	34.37	408	39.03	1839.03	173	75.76	1800.00	233	83.43	1800.00	317	88.19	77.63
10teams	34.45	210	10.36	1810.36	1937	57.29	1800.00	3450	35.43	1800.00	13827	48.10	39.17
mod010	37.97	146	5.04	1805.04	12336	3.58	1800.00	10146	13.69	1800.00	39346	27.82	13.79
misc05	40.09	266	13.61	1813.61	246	55.57	1800.00	129	70.54	1800.00	5467	60.13	58.67
p0282	40.89	161	4.43	31.58	23	0.00	114.99	73	0.00	1112.33	2399	0.00	10.40

Table 4.9: Computational results for MIPLIB open problems: low density ($\leq$ 20%) VSP instances.

Instance			PHYBRID(L-CD,L-LD)				B&C(CD,LD)			B&B			B&C[20]
label	d	n	R&C	total t(s)	nodes	gap	t(s)	nodes	gap	t(s)	nodes	gap	gap
set1al	0.78	492	11.98	162.30	13567	0.00	471.28	11439	0.00	1695.38	39175	0.00	1.25
set1cl	0.78	492	11.98	165.72	13567	0.00	464.22	11439	0.00	1731.34	39175	0.00	1.25
set1ch	0.81	477	10.80	153.31	13512	0.00	430.26	11373	0.00	1800.00	30008	0.36	1.08
fixnet3_r	1.10	478	12.91	20.73	77	0.00	21.24	71	0.00	406.00	5399	0.00	0.22
misc06	1.21	696	18.07	1818.07	12426	5.52	1800.00	7363	5.60	1800.00	13862	15.63	10.54
qnet1_o	3.59	369	7.64	60.03	677	0.00	215.86	497	0.00	1800.00	16989	2.38	4.69
qnet1	3.60	407	10.57	212.52	1085	0.00	532.12	1115	0.00	1800.00	19023	8.32	7.26
danoint	4.49	664	52.14	1852.14	130	27.11	1800.00	1513	29.04	1800.00	5114	70.70	38.66
gams	5.13	291	8.42	373.46	5235	0.00	419.52	4869	0.00	1800.00	22587	1.52	2.22
adrud	5.50	795	43.40	1254.74	193	0.00	1325.10	205	0.00	1800.00	1110	2.76	2.88
p0548	7.82	257	5.57	1805.57	125611	18.28	1800.00	24086	10.69	1800.00	41048	13.62	11.54
air04	16.67	782	71.29	1871.29	17	73.54	1800.00	220	87.29	1800.00	18	89.67	82.44
air06	16.82	570	51.00	1851.00	67	70.42	1800.00	184	85.51	1800.00	710	83.49	162.50
stein45_r	19.59	331	30.83	1830.83	88	79.95	1800.00	723	84.61	1800.00	7253	73.29	80.79

Notice that twelve MIPLIB instances remained unsolved after our experiments with our combined approach. For these problems, we kept our focus on the primal side. Thus, we decided to run again one of our relax-and-cut algorithms to attempt to obtain better

C2 (Pg.115)

primal bounds. In order to do it, we rerun PR&C(CD,LD) with different settings of parameters. Thus, we adopted: $\Delta = 25$ and π was update at each 50 consecutive iterations without improvement on the upper bound and the maximum number of iterations was limited to 5000. Besides, at each SM iteration, the primal heuristic was called at most four times, stopping at the first call with success in obtaining an improvement on the best primal bound known so far. At each call, a distinct choice of the parameter $\bar{\rho}$ that controls the weighting method is performed. After some preliminary tests, the final sequence of calls adopted was: $\bar{\rho} = \langle 1, 0 \rangle, \bar{\rho} = \langle 0, 1 \rangle, \bar{\rho} = \langle 1, 1 \rangle$ and $\bar{\rho} = \langle 0, 0 \rangle$. The results obtained are shown in figures 4.17, 4.18 and 4.19.

Figure 4.17 promotes a qualitative comparison between results produced by our heuristic against de Souza and Balas primal results reported in [11][10]. The graphic shows, for each instance, how much better/worse is the best solution found by each approach when compared to the bounds reported in [20][11]. Thus, a point above x axis (zero value line) represent a better solution than the best we know from literature. Observe that, for this subset of instances, the solutions corresponding to our Lagrangian heuristic are, in general, much better (almost dominate) that ones representing the heuristic embedded in de Souza and Balas algorithm. A summary of these results can be easily visualized in Figure 4.18 and help us to conclude that our Lagrangian heuristic is the best heuristic reported in literature to tackle that subset of hard instances. As an example, observe that in 83% of the cases our heuristic produced results as good as the best VSP solution already reported.

Now, regarding the execution time to reach the best solution, we build up the graphic shown in Figure 4.19. Observe that for all instances but two, `air04` and `air05`, our Lagrangian heuristic needed less time than de Souza and Balas heuristic to yield their best solutions.

C2 (Pg.115)

4.8 Conclusions and future works

This paper describes a Lagrangian relaxation framework based on relax-and-cut algorithms proposed to solve the Vertex Separator problem. The approach developed suggests not only the combination of cutting planes and Lagrangian algorithms, but also the hybridization of Lagrangian relaxation and Integer Programming. The several relax-and-cut versions implemented have embedded cutting-plane procedures derived from valid inequalities to the VSP presented on [3].

Computational results were obtained for benchmarks from the literature and compared

[10] Since, for all instances used in this test, the running time of `PR&C(CD,LD)` remained below 90 seconds, we imposed this time limit to the execution of de Souza and Balas algorithm.

[11] Bound values are computed and reported in accordance with those in tables 4.8 and 4.9.

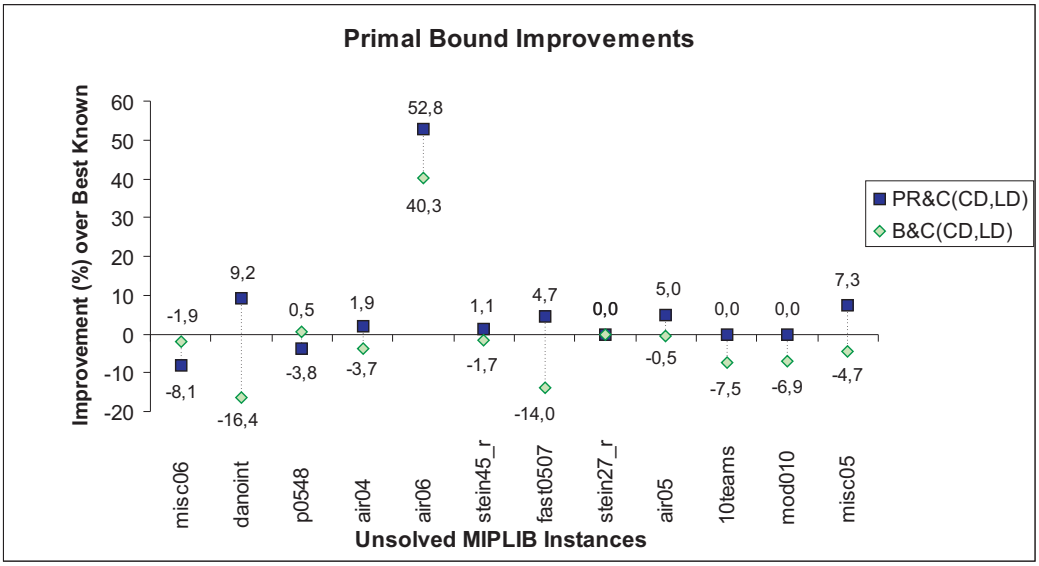

Figure 4.17: Primal results to MIPLIB not solved instances: shows how better/worse are the best solution obtained by the our relax-and-cut algorithm against the best values yielded by B&C in [11] within a time limit of 90 seconds.

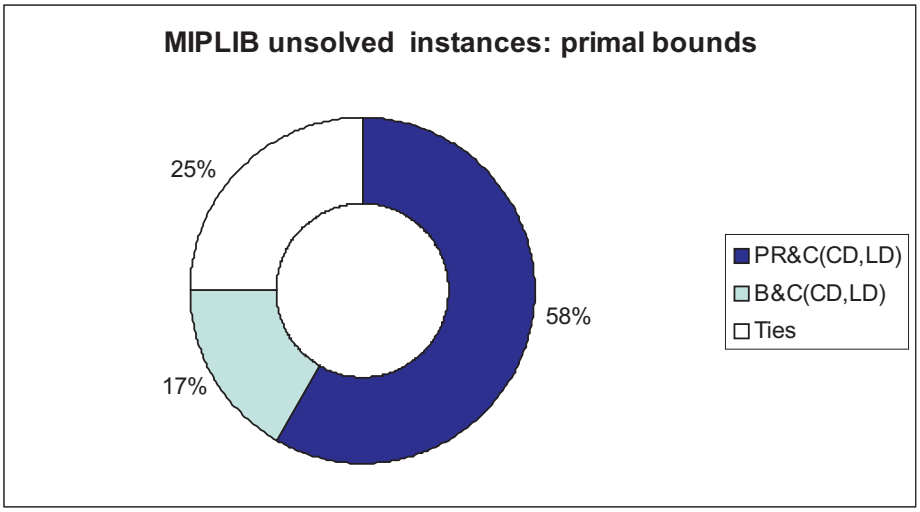

Figure 4.18: Primal results to MIPLIB not solved instances: shows percentage of cases where only PR&C(CD,LD) or B&C produced the best primal bounds or the three algorithms (including `B&C` from [20]) tied with the best primal bound assuming the same value.

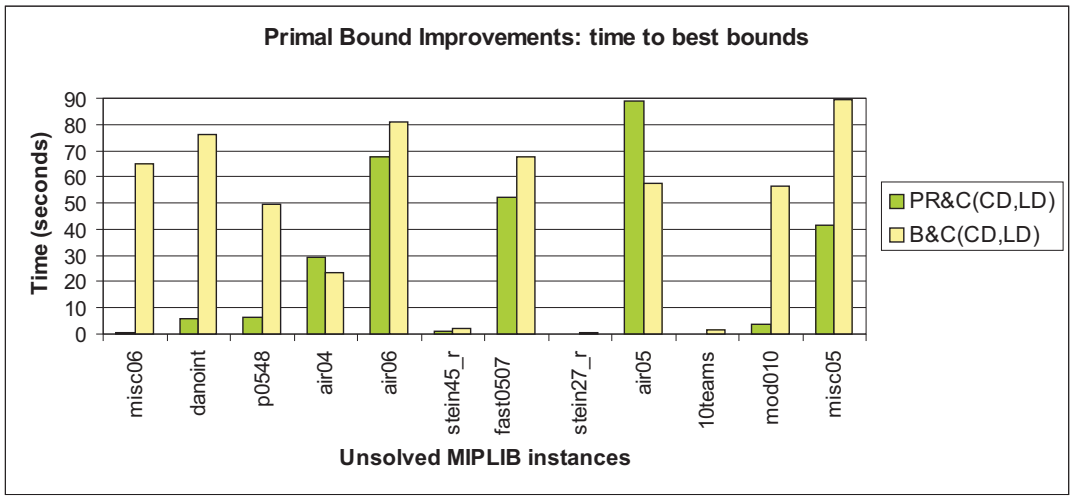

Figure 4.19: Primal results to MIPLIB not solved instances: confronts the computation times required by PR&C(CD,LD) and by B&C(dSB) to produce their best bounds within the time limit imposed.

with the best known results to the problem. These experiments show that the best variants of the (hybrid) exact method we developed outperform the pure B&C algorithm introduced by de Souza and Balas in [11]. It is remarkable that for mid-high density instances, our algorithm beats B&C(CD) in 92% of the cases, namely, in 35 out of the 38 instances. To our knowledge, these results turn our hybrid framework the best exact algorithm available for the VSP to date.

Besides, we show that the Lagrangian phase is a very effective heuristic for the VSP, often producing optimal solutions extremely fast. Moreover, for the `MIPLIB` instances whose optimal remains unknown, our Lagrangian heuristic often produced stronger primal bounds than those reported in the literature.

Further developments and implementation issues should be considered to possibly improve the performance of our current framework. Future works could include the study of different relaxations, better primal heuristics and identification of new valid inequalities for the VSP polytope discussed in [3] that could be introduced as cutting planes in new *relax-and-cut* algorithms.

4.9 Comentários

C1: Separação polinomial das desigualdades `CD`

Os resultados aqui apresentados tomam como base a formulação para o VSP descrita na seção 4.2 e, seja LRP o problema de relaxação Lagrangiana resultante da relaxação do

conjunto de restrições (4.2).

Proposição 4.9.1 *Seja $G = (V, E)$ o grafo de entrada para uma instância qualquer do VSP e $\bar{u}$ uma solução ótima para LRP. Seja $S \subseteq V$, o conjunto de vértices $i \in V$ tais que $\bar{u}_{i1} + \bar{u}_{i2} = 1$ e $G[S]$ o subgrafo induzido por S em G. Então, existe $W \subset V$ tal que a desigualdade* CD *definida é violada por $\bar{u}$ se e somente se S é um dominador e $G[S]$ é conexo.*

Prova:

($\Rightarrow$) *Se existe alguma desigualdade* CD *violada por $\bar{u}$, existe um dominador conexo W tal que $\bar{u}_{i1} + \bar{u}_{i2} = 1$, para todo $i \in W$. Portanto $u(W) = |W|$ e, por construção de S, qualquer vértice contribuindo para $u(W)$ está em S. Logo, $W \subseteq S$ e S é um dominador também.*

Suponhamos que S é desconexo. Então, dado que $W \subseteq S$ e W é conexo, existe $v \in S \backslash W$ sem vértices adjacentes em W. Isso é uma contradição pois W é um dominador.

($\Leftarrow$) *Faça $W := S$* $\square$

Fato 4.9.1 *O algoritmo de Programação Dinâmica descrito a seguir resolve LRP em tempo polinomial quando $u_{i1} + u_{i2} = 1, \forall\ i \in V$ é satisfeita.*

Sejam c_1 e c_2, os vetores que armazenam, para cada conjunto A e B, os vértices ordenados não-crescentemente pelos custos c_{i1} e c_{i2}, $i \in V$, respectivamente (note: estes valores correspondem aos custos Lagrangianos relativos às variáveis associadas a cada vértice do grafo). No pseudo-código da Figura 4.21 ForaDaFaixa(k, p, q) retorna verdadeiro se $p < 0$ ou $q < 0$ ou $(p + q) > k$. Caso contrário, a função retorna falso. Além destas, são consideradas definições dadas na descrição da programação dinâmica (Figura 4.20)

Observe que as inicializações do algoritmo de programação dinâmica proposto (Figura 4.21) têm complexidade $O(|V| \log |V|)$, devido às ordenações para cálculo de $z[k, p, 0]$ e $z[k, 0, q]$. Além disso, como $b \in O(|V|)$, a matriz de programação dinâmica tem $O(|V|^3)$ células que requerem cada uma tempo $O(1)$ para serem preenchidas. Portanto a complexidade do algoritmo é $O(|V|^3)$.

Corolário 4.9.1 *A separação de desigualdades do tipo* CD *sobre $G[S]$ é feita em tempo polinomial quando são mantidas no LRP as restrições $u_{i1} + u_{i2} \leq 1, \forall\ i \in V$.*

Note que se $u_{i1} + u_{i2} \leq 1, \forall\ i \in V$ não é satisfeita, $\bar{u}$, solução ótima para o subproblema Lagrangiano LRP, permite colocar o mesmo vértice i em conjuntos distintos. Como consequência, o contra-exemplo construído da Figura 4.22 mostra que quando não é garantido que $\bar{u}_{i1} + \bar{u}_{i2} = 1$, para todo $i \in W$, a proposição 4.9.1 não é verdadeira. Na figura, os vértices cinzas e pretos formam um dominador conexo e os os pares (u_{i1}, u_{i2})

Descrição PD (LRP') /* $LRP' \equiv LRP$, onde (4.1) é satisfeita */

1. Definições:

z: matriz $(n+1) \times (b+1) \times (b+1)$, $n = |V|$, com as soluções dos subproblemas.

$z[k,p,q]$: armazena o máximo da função objetivo considerados que os vértices de $k+1$ até n estão no separador C e tendo exatamente p vértices no conjunto A e q vértices no conjunto B.

c_{k1}, c_{k2}: custos de adiconar o vértice k aos conjuntos A e B, respectivamente.

2. Inicializações: supor que, para $j = 1, 2$, $c_{\varphi(i)j} \geq c_{\varphi(i+1)j}$, para todo i em V, ou seja, $\{\varphi(1), \varphi(2), \ldots, \varphi(n)\}$ é a seqüência dos vértices ordenados não-crescentemente pelos custos $c_{\cdot j}$.

$z[0,0,0] = z[k,0,0] \equiv 0$, $\forall k$.

$$z[k,p,0] = \begin{cases} 0, & \text{se } p > k \\ \sum_{i=1}^{\min\{p,k\}} c_{\varphi(i)1}, & \text{caso contrário.} \end{cases}$$

$$z[k,0,q] = \begin{cases} 0, & \text{se } q > k \\ \sum_{i=1}^{\min\{q,k\}} c_{\varphi(i)2}, & \text{caso contrário.} \end{cases}$$

$z[k,p,q] \equiv -\infty$, quando $p < 0$ ou $q < 0$ ou $p + q > k$.

3. Recorrência (relaciona as soluções dos subproblemas):

$z[k,p,q] \equiv \max\{z[k-1,p,q], c_{k1} + z[k-1,p-1,q], c_{k2} + z[k-1,p,q-1]\}, \forall k,p,q.$

4. Valor da solução ótima:

$z^* = \max\{z[|V|,p,q]\}$, $1 \leq p \leq b$ e $1 \leq q \leq b$.

Figura 4.20: Descrição da Programação Dinâmica.

Algoritmo PD (z, c_1, c_2, b)

1. Inicializações:
   $z[k,p,q] \leftarrow -\infty$, $\forall k, \forall p, \forall q$;
   $z[k,0,0] \leftarrow 0$, $\forall k$;
   `para` $k = 1, ..., |V|$
      Ordene os k primeiros vértices de c_1 e de c_2;
      `para` $r = 1, ..., k$ `faça`
         $z[k,r,0] \leftarrow 0$; $z[k,0,r] \leftarrow 0$;
         `para` $i = 1, ..., r$ `faça`
            $z[k,r,0] \leftarrow z[k,r,0] + c_{i1}$; /* calcula $z[k,p,0]$ */
            $z[k,0,r] \leftarrow z[k,0,r] + c_{i2}$; /* calcula $z[k,0,q]$ */
2. Cálculo da recorrência:
   `para` $k = 1, ..., |V|$ `faça`
      `para` $p = 1, ..., \min\{k, b\}$ `faça`
         `para` $q = 1, ..., k - p$ `faça`
            $kFora \leftarrow -\infty$; $kA \leftarrow -\infty$; $kB \leftarrow -\infty$;
            $todosFora =$ `VERDADEIRO`;
            `se` $\neg$ ForaDaFaixa$(k-1, p, q)$ `então`
               $kFora \leftarrow z[k-1,p,q]$; $todosFora =$ `FALSO`;
            `se` $\neg$ ForaDaFaixa$(k-1, p-1, q)$ `então`
               $kA \leftarrow z[k-1,p-1,q] + c_{k1}$; $todosFora =$ `FALSO`;
            `se` $\neg$ ForaDaFaixa$(k-1, p, q-1)$ `então`
               $kB \leftarrow z[k-1,p,q-1] + c_{k2}$; $todosFora =$ `FALSO`;
            `se` $\neg\, todosFora$ `então`
               $kB \leftarrow z[k,p,q] \leftarrow \max\{kFora, kA, kB\}$;
3. Obtenção do valor de uma solução ótima:
   $z^* \leftarrow -\infty$; $|A| \leftarrow 0$; $|B| \leftarrow 0$;
   `para` $p = 1, ..., b$ `faça`
      `para` $q = 1, ..., \min\{b, |V| - p\}$ `faça`
         $z^* \leftarrow z[|V|, p, q]$;
         $|A| \leftarrow p$; $|B| \leftarrow q$;
4. Recuperação de uma solução ótima:
   Percorra a matriz z a partir de $z[|V|, |A|, |B|]$ até $z[1, p, q]$ e identifique as escolhas ótimas (atribuições de vértices a A, B ou C) feitas para cada valor de k.
   `retorne` (A, B, C).

Figura 4.21: Algoritmo de Programação Dinâmica.

indicam os valores para todos os vértices i, quando diferetes de (0,0). Note que os vértices em cinza junto com os vértices pretos formam um dominador conexo cuja desigualdade é violada e $G[S]$ é desconexo.

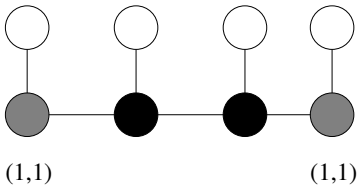

Figura 4.22: Contra-exemplo para a proposição 4.9.1 quando $u_{i1} + u_{i2} = 1$ não é satisfeita.

Corolário 4.9.2 *Se S é definido como o conjunto de vértices tais que $\bar{u}_{i1}+\bar{u}_{i2} \geq 1, i \in V$, e $G[S]$ o subgrafo induzido por S em G. Então, se S é um dominator e $G[S]$ é conexo, existe uma desigualdade* `CD` *sobre S violada por $\bar{u}$.*

Este resultado mostra que a rotina de separação descrita em 4.4.1, página 70 é, de fato, uma heurística e não um algoritmo exato para o problema de separação de desigualdades `CD`. Para descobrir a complexidade do problema de separação, vamos enunciá-lo escrevendo cada desigualdade `CD` associada a um dominador conexo W de G da seguinte forma:

$$u(W) \leq |W| - 1 \;\equiv\; u(W) < |W| \;\equiv\; \sum_{i \in W}(u_{i1} + u_{i2}) < |W|.$$

Conseqüentemente, se $\bar{u}$ viola a desigualdade `CD` de W, tem-se que $\sum_{i \in W}(u_{i1}+u_{i2}) \geq |W|$.

Agora, seja o problema do dominador conexo de peso mínimo de um grafo G enunciado a seguir:

INSTÂNCIA: Um grafo não orientado $G = (V, E)$ e peso $w_i \in \mathbb{Z}$ para todo vértice $i \in V$. PROBLEMA *mWCD (Minimum Weighted Connected Dominator)*: Encontre um dominador conexo W cujo peso, dado por $\sum_{i \in W} w_i$, seja mínimo.

Fazendo-se $w_i = \bar{u}_{i1} + \bar{u}_{i2}$ para todo $i \in V$, vê-se que o problema de separação de uma desigualdade `CD` pode ser resolvido através do problema de otimização acima. Se o valor ótimo for maior ou igual a $|W|$, encontrou-se uma desigualdade `CD` violada. Do contrário, todas as desigualdades desta família estarão satisfeitas por $\bar{u}$.

Doravante, concentremos nossa atenção na versão de decisão do *mWCD* onde um valor inteiro adicional k é passado na entrada e o problema é decidir se existe um dominador conexo de peso menor ou igual a k em G. Além disso, no que se segue, vamos supor que os pesos w_i estão restritos ao conjunto $\{0, 1, 2\}$, que é o caso que ocorre ao tratarmos o problema de separação das desigualdades `CD`.

Considera-se agora o seguinte problema de decisão:

INSTÂNCIA: Um grafo não orientado $G = (V, E)$, $|V| = n$ e um inteiro positivo k.
PROBLEMA *MLST (Maximum Leaf Spanning Tree)*: Existe uma árvore geradora T para G na qual k' ou mais vértices têm grau um, ou seja, são folhas ?

Fato 4.9.2 *MLST é $\mathcal{NP}$-Completo (veja [ND2] em [13]).*

Proposição 4.9.2 *mWCD é $\mathcal{NP}$-Completo.*

Prova: *Inicialmente é fácil ver que mWCD está em $\mathcal{NP}$. Agora, mostra-se que MLST α_p mWCD.*

Para tanto, basta tomar $k = n - k'$ no mWCD e $w_i = 1$ para todo $i \in V$. Com isto, no mWCD, estamos procurando um dominador conexo contendo no máximo $n - k'$ vértices. Note que, removendo-se as folhas de uma árvore geradora T qualquer, o conjunto de vértices restantes é um dominador conexo de G. Assim, se T é uma solução para MLST, pelo menos k' vértices são folhas em T e o restante deles forma um dominador conexo de tamanho no máximo $n - k' = k$ vértices. Por outro lado, se W é uma solução para mWCD, existe uma árvore geradora T' em $G[W]$. Como W é um dominador, cada um dos $n - |W|$ vértices de $V \setminus W$ está ligado a W por pelo menos uma aresta. Acrescentando-se uma destas arestas para cada um destes $n - |W|$ vértices à árvore T', obtém-se uma árvore geradora para G com pelo menos $n - |W| \geq n - k = k'$ folhas. Isso completa a prova. □

Note que a prova acima mostra que, em sua versão mais geral, o $mWCD$ é $\mathcal{NP}$-*completo*, sugerindo que o problema de separação também o seja. No entanto, vimos anteriormente que o problema de separação é resolvido em tempo polinomial no caso em que o problema $mWCD$ equivalente possui todos os vértices com pesos iguais a zero ou um e o valor de k é $|W|$. Porém a prova acima ainda não nos permite concluir que o $mWCD$ permanece difícil quando temos os pesos no conjunto $\{0, 1, 2\}$ e k vale $|W|$. Este é o caso do problema de separação referente à solução $\bar{u}$ da relaxação Lagrangiana quando as desigualdades $\bar{u}_{i1} + \bar{u}_{i2} \leq 1$, $i \in V$, são todas dualizadas.

Denotemos por mCD o caso especial do problema $mWCD$ não ponderado (i.e., onde $w_i = 1$ para todo $i \in V$) . Com base na demonstração anterior, relativa à proposição 4.9.2, mCD é $\mathcal{NP}$-*Completo*. Além disso, observe que se um grafo G tem um dominador conexo de tamanho $p \leq k$, então G tem um dominador conexo de tamanho l onde $p \leq l \leq n$. Logo, o problema de determinar a existência de um dominador conexo de tamanho exatamente k (descrito a seguir), o qual denominaremos $mCD^{=}$, também é $\mathcal{NP}$-*Completo*.

INSTÂNCIA: Um grafo não orientado $G = (V, E)$ e um inteiro positivo k.
PROBLEMA *$mCD^{=}$ (Minimum Connected Dominator)*: Existe um dominador conexo $W \subseteq V$ em G de tamanho k?

Consideremos agora a separação das desigualdades `CD` sobre um grafo G onde os pesos nos vértices estão restritos ao conjunto $\{0, 1, 2\}$ e k tem valor $|W|$. Este problema pode ser resolvido através do problema de otimização associado ao seguinte problema de decisão:

INSTÂNCIA: Um grafo não orientado $G = (V, E)$, $|V| = n$, peso $w_i \in \{0, 1, 2\}$ para todo vértice $i \in V$ e um inteiro positivo $t \leq n$.
PROBLEMA *SEP*: Existe um dominador conexo W em G tal que $\sum_{i \in W} w_i = t \geq |W|$?

Fato 4.9.3 *$mCD^{=}$ é $\mathcal{NP}$-Completo.*

Proposição 4.9.3 *SEP é $\mathcal{NP}$-Completo.*

Prova: *Não é difícil ver que SEP está em $\mathcal{NP}$. Assim, nossa prova concentra-se em mostrar que $mCD^{=}$ α_p SEP. Em outras palavras, mostramos que a resposta para $mCD^{=}$ é SIM se e somente se a resposta para SEP também é SIM.*

Para tal, a Figura 4.23 apresenta uma tranformação (polinomial) de uma instância qualquer de $mCD^{=}$, $I(mCD^{=})$, em uma instância de SEP, $I(SEP)$. Na figura são detalhados os conjuntos de vértices, seus respectivos pesos e as conexões (arestas) entre os pares de vértices, bem como a expressão que define o parâmetro de entrada t.
(I): *$mCD^{=}$ SIM $\Rightarrow$ SEP SIM.*

Seja $W \subseteq V$ um dominador conexo de G tal que $|W| = k$. S.p.g., podemos supor que $W = \{v_1, v_2, \ldots, v_k\}$ (vértices em preto na Figura 4.23). Seja $W' \subset V'$ dado por $W' = W \cup A \cup \{b_1, c_1\} \cup \{c_{k+1}, \ldots, c_n\} \cup \{d\}$ e denotemos $p(w) = \sum_{i \in W} w_i$. Assim, com base na transformação de $I(mCD^{=})$ em $I(SEP)$ podemos concluir que:

Fato 4.9.4 *Como $G[W]$ é conexo, $G'[W']$ também é conexo.*

Fato 4.9.5 *W' é um dominador, pois: (i) todos os nós de V são cobertos por W; (ii) d está em W'; (iii) todo vértice $i \in A \cup \{c_{k+1}, \ldots, c_n\}$ está em W'; (iv) b_1 e c_1 estão em W'; (v) $\{b_2, \ldots, b_k\}$ estão cobertos por W enquanto $\{b_{k+1}, \ldots, b_n\}$ estão cobertos por $\{c_{k+1}, \ldots, c_n\}$; (vi) $\{c_2, \ldots, c_k\}$ estão cobertos por d e (vii) $\{e\}$ está coberto por a_{n-k+2}.*

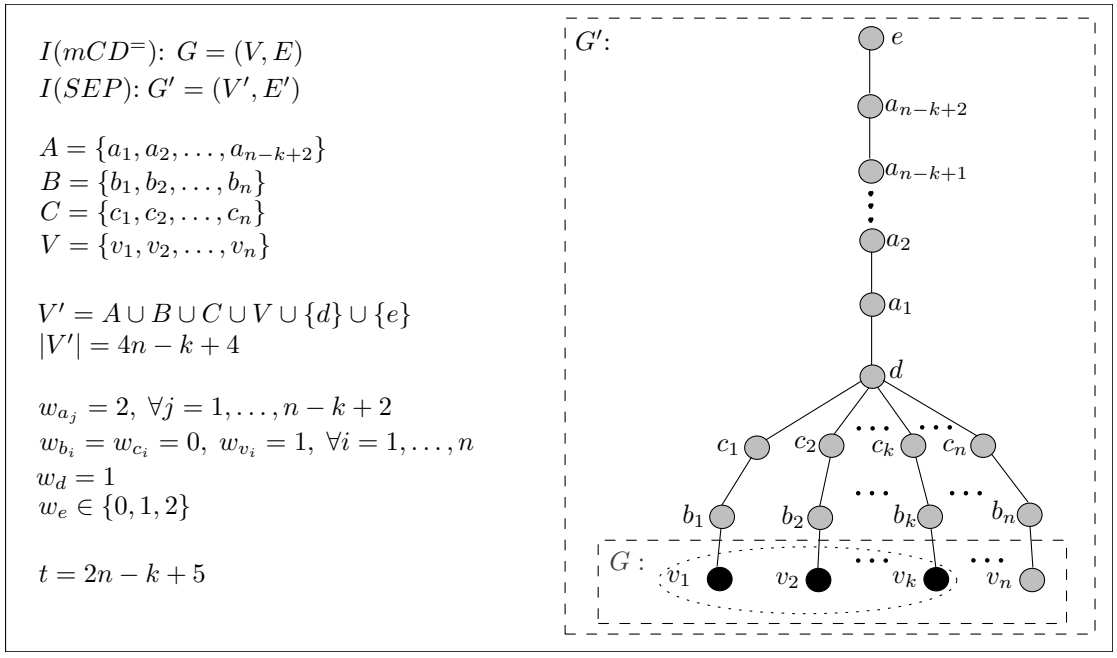

Figura 4.23: Tranformação de uma instância de $mCD^{=}$ em uma instância de SEP.

Fato 4.9.6 *$t \geq |W|$ é satisfeita. De fato:*

$$
\begin{aligned}
p(w') &= \overbrace{k}^{W} + \overbrace{2(n-k+2)}^{A} + \overbrace{0}^{\{b_1\}} + \overbrace{0}^{\{c_1\}} + \overbrace{0}^{\{c_{k+1},\ldots,c_n\}} + \overbrace{1}^{\{d\}} = 2n-k+5 \\
|W'| &= \underbrace{k}_{W} + \underbrace{(n-k+2)}_{A} + \underbrace{1}_{\{b_1\}} + \underbrace{1}_{\{c_1\}} + \underbrace{n-k}_{\{c_{k+1},\ldots,c_n\}} + \underbrace{1}_{\{d\}} = 2n-k+5 \\
t &= p(w') = \sum_{i \in W'} w_i = |W'| = 2n-k+5
\end{aligned}
$$

Logo, se $mCD^{=}$ tem resposta sim SEP também terá. □

(II): *SEP SIM $\Rightarrow$ $mCD^{=}$ SIM.*

Vamos supor que W' é um dominador conexo de G' que satisfaz $p(w') = t = 2n-k+5 \geq |W'|$. Note que $\{d, a_1, \ldots, a_{n-k+2}\}$ estão em qualquer dominador conexo de G'. Queremos mostrar que $W \subseteq V$ é um dominador conexo de tamanho k. Para tal, suponha também que $W = W' \cap V = \{V_1, V_2, \ldots, V_p\}$ onde cada V_i é uma componente conexa de G.

Observe que $p(w') = |W| + 2|A| + |d| = 2n - k + 5 = k + 2(n-k+2) + 1$. Logo, W tem que ter exatamente k vértices e $|W'|$ poderia conter até mais $n-k+2$ vértices com peso zero sem que $p(w') \geq |W'|$ fosse violada. S.p.g., suponha que $V \backslash W = \{v_{k+1}, \ldots, v_n\}$. Os vértices $\{b_{k+1}, \ldots, b_n\}$ não podem estar em W' pois, caso contrário, não teríamos W' conexo e satisfazendo a $p(w') \geq |W'|$ simultaneamente. Assim, os $n-k$ vértices de $\{c_{k+1}, \ldots, c_n\}$ estão em W'. Agora, cada componente de G induzida por vértices em W

requer um par de vértices $\{b_i, c_i\}$ *para se conectar ao resto de* W'*. Ou seja, precisamos de pelo menos* $2p$ *vértices (1 para cada componente conexa de* G*) a mais em* W' *além dos* $2n - k + 3$ *já identificados para garantir a conexide de* W'*. Mas, como* W' *só pode ter até* $2n - k + 5$ *vértices, sobram 2 vértices, no máximo, para colocar em* W'*. Portanto,* W *só pode ter uma componente conexa (*$p = 1$*), a qual deve cobrir os demais vértices de* V*. Logo,* W *é um dominador conexo de* G *com* k *vértices.* □

C2: Resultados computacionais em detalhes

Esta seção detalha alguns dos principais resultados provenientes de nossos experimentos utilizando o *framework* híbrido proposto para o VSP. Por este motivo, diversos pontos da tese que compilam resultados obtidos fazem referência à presente seção. O objetivo desta seção de comentários é, portanto, explicitar detalhes referentes aos resultados computacionais discutidos que possam ser relevantes para a compreensão do texto.

Para melhor entendimento das tabelas que se seguem, o glossário abaixo descreve o conteúdo das colunas cujos rótulos possam não ser considerado auto-explicativos:

- `d`: densidade do grafo associado a cada instância do VSP;
- `best LB`: tempo necessário para produzir a solução de melhor valor encontrada durante a execução do algoritmo *relax-and-cut* correspondente;
- `R&C`: tempo de execução do algoritmo *relax-and-cut*;
- `I/O`: tempo gasto com a escrita do(s) *pool(s)* de desigualdades e do modelo de programação matemática (arquivo .lp);
- `total`: soma dos tempos anteriores, i.e., R&C + I/O;
- `pool size`: número de restrições nos *pools* de desigualdades ao final da execução do algoritmo *relax-and-cut*;
- `LP`: (A) `t(s)`): tempo de execução do LP, incluindo estimativa[12] de tempo gasto pelo `XPRESS` para geração e escrita do modelo com todos os cortes identificados durante a execução do algoritmo *relax-and-cut*. Nota: os tempos reportados pelo `XPRESS` Optimizer em seu módulo iterativo são números inteiros. Assim sendo, quando o valor reportado foi 0 (zero), assumimos que 0.5 segundos foram gastos e (B) `gap`: *gap* de integralidade (diferença inteira entre melhores limitantes duais e primais) ao final da execução do LP.

[12]Esta estimativa foi feita tomando-se como base o tempo que nossa implementação leva para executar a mesma tarefa.

- `LP + R&C`: soma dos tempos reportados em LP (t(s)) + R&C;
- `L-B&C`: tempo total e número total de nós gerados durante execução do L-B&C descrito na seção 4.5;
- `HYBRID`: tempo total de execução do framework híbrido: soma das fases Lagrangianas e de programação linear;

Além destas, para cada tabela e para cada conjunto de instâncias, é apresentada a somatória dos tempos para: `(i)` instâncias resolvidas apenas pelos algoritmos B&C de [11] e nosso algoritmo híbrido e `(ii)` instâncias resolvidas também pelo algoritmo *default* do `XPRESS` Optimizer. As instâncias estão agrupadas por densidades superiores ou não a 35% (instâncias consideradas densas). Este valor foi utilizado porque os testes realizados estão baseados em conclusões de de Souza e Balas sobre experimentos reportados em [11].

Tabela 4.10: Resultados computacionais onde apenas desigualdades CD são separadas durante a execução do algoritmos NDR&C e L-B&C.

Instance		R&C (CD)					LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool						
label	d	best LB	R&C	I/O	total	size	t(s)	gap	t(s)	t(s)	nodes	t(s)
High Density	$> 35\%$											
dim.queen8_8	0.36	0.00	1.03	0.02	1.07	1501	–	–	–	73.95	1807	75.02
dim.miles1000	0.40	0.01	2.71	0.01	2.73	1239	0.51	3	3.24	9.73	17	12.97
dim.queen7_7	0.40	0.00	0.60	0.01	0.62	953	–	–	–	13.49	391	14.11
dim.DSJC125.9	0.90	0.00	3.85	0.02	3.89	988	–	–	–	592.56	4275	596.45
mat.L125.bcsstk05	0.35	0.01	2.54	0.01	2.56	859	1.01	1	3.57	82.08	709	85.65
mat.L125.dwt_193	0.38	0.20	2.62	0.01	2.64	1108	3.01	1	5.65	17.46	21	23.11
mat.L125.fs_183_1	0.44	0.01	1.52	0.01	1.54	18	–	–	–	26.57	29	28.11
mat.bcsstk04	0.68	0.00	3.23	0.02	3.27	749	2.02	2	5.29	17.61	13	22.90
mat.arc130	0.93	0.02	5.28	0.02	5.32	1018	4.02	9	9.34	150.85	83	160.19
mat.L100.steam2	0.36	0.00	1.90	0.01	1.92	1499	2.01	2	3.93	17.09	45	21.02
mat.L120.fidap025	0.39	0.00	1.97	0.01	1.99	530	0.51	0	2.50	–	–	–
mat.L120.cavity01	0.42	0.70	2.53	0.01	2.55	482	–	–	–	6.58	13	9.13
mat.L120.fidap021	0.43	0.00	2.18	0.01	2.20	587	0.51	1	2.71	2.92	5	5.63
mat.L120.rbs480a	0.46	0.02	2.58	0.01	2.60	1010	2.01	3	4.61	59.40	125	64.01
mat.L120.wm2	0.47	0.00	1.11	0.01	1.13	0	–	–	–	26.91	33	28.04
mat.L100.rbs480a	0.52	0.00	1.76	0.00	1.76	414	0.50	2	2.26	9.64	59	11.90
mat.L80.wm2	0.58	0.37	0.83	0.00	0.83	58	–	–	–	2.45	9	3.28
mat.L100.wm3	0.59	0.91	1.72	0.01	1.74	59	–	–	–	5.89	11	7.63
mat.L120.e05r0000	0.59	0.00	1.64	0.01	1.66	352	–	–	–	5.39	5	7.05
mat.L100.wm1	0.60	0.01	1.49	0.01	1.51	203	–	–	–	9.23	19	10.74
mat.L120.fidap022	0.60	0.00	2.75	0.02	2.79	845	1.02	1	3.81	18.72	77	22.53
mat.L100.fidapm02	0.62	0.00	2.13	0.01	2.15	708	0.51	0	2.66	–	–	–
mat.L120.fidap001	0.63	0.00	2.88	0.02	2.92	840	2.02	0	4.94	–	–	–
mat.L100.e05r0000	0.64	0.00	1.41	0.01	1.43	259	–	–	–	6.96	15	8.39
mat.L80.fidapm02	0.65	0.00	1.38	0.01	1.40	781	0.51	0	1.91	–	–	–
mat.L120.fidapm02	0.65	0.00	2.34	0.01	2.36	639	0.51	0	2.87	–	–	–
mat.L100.fidap001	0.68	0.00	1.97	0.01	1.99	358	0.51	1	2.50	4.60	35	7.10
mat.L100.fidap022	0.68	0.00	1.97	0.01	1.99	488	0.51	3	2.50	20.16	109	22.66
mat.L80.fidap001	0.72	0.00	1.02	0.01	1.04	216	0.51	0	1.55	–	–	–
mat.L80.fidap022	0.76	0.00	1.25	0.00	1.25	169	–	–	–	13.00	197	14.25
mat.L100.fidap027	0.81	0.00	2.30	0.01	2.32	474	0.51	0	2.83	–	–	–
mat.L100.fidap002	0.82	0.00	1.28	0.01	1.30	122	–	–	–	1.92	5	3.22
mat.L120.fidap002	0.82	0.00	2.25	0.01	2.27	236	–	–	–	4.61	5	6.88
mat.L120.fidap027	0.85	0.00	3.33	0.02	3.37	469	1.02	0	4.39	–	–	–
miplib.l152lav.p	0.40	0.07	1.14	0.01	1.16	262	–	–	–	45.11	213	46.27
miplib.lp4l.p	0.46	0.56	1.21	0.01	1.23	667	–	–	–	33.04	275	34.27
miplib.air03.p	0.61	0.98	2.09	0.01	2.11	491	–	–	–	109.83	115	111.94
miplib.misc03.p	0.63	0.01	2.28	0.02	2.32	1095	–	–	–	109.63	2993	111.95
miplib.misc07.p	0.80	0.02	8.01	0.05	8.11	334	–	–	–	1272.25	173	1280.36
$\sum$ B&Cs											11708	1576.40
$\sum$ B&Cs and XP											7294	845.11
Mid-Low Density	$\leq 35\%$											
dim.games120	0.09	0.16	1.54	0.02	1.58	734	–	–	–	788.24	84625	789.82
dim.myciel7	0.13	0.08	2.78	0.01	2.80	323	–	–	–	301.22	1823	304.02
dim.myciel6	0.17	0.24	1.07	0.00	1.07	294	–	–	–	8.25	323	9.32
mat.can96	0.20	0.00	1.30	0.01	1.32	534	–	–	–	1800.00	191769	1801.32
mat.can73	0.25	0.08	1.04	0.00	1.04	1094	–	–	–	45.11	5343	46.15
mat.rw136	0.07	0.02	1.78	0.00	1.78	562	0.50	16	2.28	1800.00	72063	1802.28
mat.gre_115	0.09	0.12	1.92	0.03	1.98	1101	–	–	–	90.93	7185	92.91
mat.L125.gre_185	0.15	0.00	2.71	0.04	2.79	2031	2.02	9	4.81	89.72	1169	94.53
mat.can_144	0.16	0.00	3.28	0.04	3.36	2208	4.02	5	7.38	1493.59	38265	1500.97
mat.lund_a	0.26	0.01	3.29	0.02	3.33	1208	5.02	10	8.35	393.04	3145	401.39
miplib.noswot.p	0.09	0.03	2.34	0.02	2.38	781	–	–	–	1262.79	27697	1265.17
miplib.khb05250.p	0.27	0.00	0.74	0.00	0.74	25	–	–	–	3.20	119	3.94
$\sum$ B&Cs											131429	3007.25
$\sum$ B&Cs and XP											126461	2301.84

Tabela 4.11: Resultados computacionais onde desigualdades CD e LD são separadas durante a execução do algoritmos NDR&C e L-B&C.

Instance		R&C (CD,LD)						LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool size							
label	d	best LB	R&C	I/O	total	CD	LD	t(s)	gap	t(s)	t(s)	nodes	t(s)
High Density	$> 35\%$												
dim.queen8_8	0.36	0.00	2.96	0.04	3.04	1548	1562	–	–	–	122.75	2541	125.79
dim.miles1000	0.40	0.49	4.32	0.02	4.36	1047	1062	1.01	2	5.37	6.15	9	11.52
dim.queen7_7	0.40	0.00	1.61	0.02	1.65	940	1108	–	–	–	15.48	253	17.13
dim.DSJC125.9	0.90	0.00	4.46	0.03	4.52	1105	92	–	–	–	739.28	6375	743.80
mat.L125.bcsstk05	0.35	0.06	4.19	0.02	4.23	1288	1046	3.02	0	7.25	–	–	–
mat.L125.dwt__193	0.38	0.49	4.53	0.03	4.59	1402	1288	4.02	2	8.61	36.45	41	45.06
mat.L125.fs_183_1	0.44	0.01	1.96	0.02	2.00	26	32	–	–	–	41.27	29	43.27
mat.bcsstk04	0.68	0.00	3.57	0.02	3.61	726	14	1.02	2	4.63	26.51	31	31.14
mat.arc130	0.93	0.04	7.40	0.04	7.48	866	1544	6.03	9	13.51	233.07	75	246.58
mat.L100.steam2	0.36	0.00	4.12	0.03	4.18	1682	1612	3.02	3	7.20	27.59	77	34.79
mat.L120.fidap025	0.39	0.00	3.05	0.01	3.07	472	600	0.51	0	3.58	–	–	–
mat.L120.cavity01	0.42	0.14	3.34	0.02	3.38	490	584	–	–	–	11.49	35	14.87
mat.L120.fidap021	0.43	0.00	3.19	0.01	3.21	568	588	1.01	1	4.22	3.12	3	7.34
mat.L120.rbs480a	0.46	0.02	3.77	0.02	3.81	1028	762	2.02	3	5.83	54.51	75	60.34
mat.L120.wm2	0.47	0.00	1.28	0.01	1.30	0	0	–	–	–	18.95	75	20.25
mat.L100.rbs480a	0.52	0.00	1.87	0.01	1.89	436	140	0.51	2	2.40	15.41	49	17.81
mat.L80.wm2	0.58	0.15	1.06	0.01	1.08	32	58	–	–	–	4.82	13	5.90
mat.L100.wm3	0.59	0.07	1.36	0.01	1.38	15	28	–	–	–	17.26	19	18.64
mat.L120.e05r0000	0.59	0.00	2.14	0.02	2.18	355	58	–	–	–	6.33	5	8.51
mat.L100.wm1	0.60	0.01	1.49	0.01	1.51	205	156	–	–	–	12.84	27	14.35
mat.L120.fidap022	0.60	0.00	3.22	0.01	3.24	924	120	1.01	1	4.25	16.03	41	20.28
mat.L100.fidapm02	0.62	0.00	2.63	0.01	2.65	867	262	0.51	0	3.16	–	–	–
mat.L120.fidap001	0.63	0.00	3.65	0.02	3.69	1114	256	3.02	0	6.71	–	–	–
mat.L100.e05r0000	0.64	0.00	1.58	0.01	1.60	259	0	–	–	–	6.90	33	8.50
mat.L80.fidapm02	0.65	0.00	1.65	0.01	1.67	805	182	0.51	0	2.18	–	–	–
mat.L120.fidapm02	0.65	0.01	3.25	0.01	3.27	736	286	1.01	0	4.28	–	–	–
mat.L100.fidap001	0.68	0.00	2.15	0.01	2.17	373	18	0.51	2	2.68	8.78	35	11.46
mat.L100.fidap022	0.68	0.00	2.19	0.02	2.23	469	44	1.02	3	3.25	14.75	63	18.00
mat.L80.fidap001	0.72	0.00	1.08	0.01	1.10	202	14	0.51	0	1.61	–	–	–
mat.L80.fidap022	0.76	0.00	1.37	0.01	1.39	280	14	–	–	–	6.50	55	7.89
mat.L100.fidap027	0.81	0.00	1.94	–	–	483	86	–	–	–	–	–	–
mat.L100.fidap002	0.82	0.00	1.42	0.01	1.44	122	0	–	–	–	1.66	3	3.10
mat.L120.fidap002	0.82	0.00	2.56	0.02	2.60	251	6	–	–	–	4.12	3	6.72
mat.L120.fidap027	0.85	0.00	3.77	0.02	3.81	492	78	1.02	0	4.83	–	–	–
miplib.l152lav.p	0.40	0.89	1.55	0.01	1.57	317	342	–	–	–	102.00	749	103.57
miplib.lp4l.p	0.46	0.07	1.09	0.01	1.11	327	186	–	–	–	68.26	1295	69.37
miplib.air03.p	0.61	0.20	3.55	0.02	3.59	517	468	–	–	–	142.51	119	146.10
miplib.misc03.p	0.63	0.13	4.72	0.04	4.80	1374	1396	–	–	–	173.98	2785	178.78
miplib.misc07.p	0.80	0.00	14.03	0.05	14.13	464	462	–	–	–	1388.27	117	1402.40
$\sum$ B&Cs												14913	2040.86
$\sum$ B&Cs and XP												8383	1119.82
Mid-Low Density	$\leq 35\%$												
dim.games120	0.09	0.01	1.88	0.02	1.92	261	240	–	–	–	1069.39	87667	1071.31
dim.myciel7	0.13	0.02	3.32	0.01	3.34	375	404	–	–	–	438.86	2779	442.20
dim.myciel6	0.17	0.06	1.29	0.00	1.29	316	306	0.50	15	1.79	12.68	409	14.47
mat.can96	0.20	0.00	2.99	0.04	3.07	1068	976	–	–	–	1800.00	159271	1803.07
mat.can73	0.25	0.12	2.83	0.03	2.89	1373	1210	–	–	–	54.48	5367	57.37
mat.rw136	0.07	0.02	2.55	0.00	2.55	210	280	0.50	16	3.05	1800.00	334492	1803.05
mat.gre__115	0.09	0.03	4.52	0.07	4.66	983	852	–	–	–	122.69	6939	127.35
mat.L125.gre__185	0.15	0.00	7.63	0.07	7.77	1896	1728	6.04	9	13.81	91.24	1191	105.05
mat.can__144	0.16	0.00	9.08	0.10	9.28	2331	2210	9.06	5	18.34	1800.00	24977	1818.34
mat.lund_a	0.26	0.10	7.42	0.05	7.52	1837	1716	7.03	10	14.55	464.55	3345	479.10
miplib.noswot.p	0.09	0.04	3.19	0.02	3.23	406	444	–	–	–	878.99	18879	882.22
miplib.khb05250.p	0.27	0.00	0.84	0.00	0.84	25	0	–	–	–	3.64	119	4.48
$\sum$ B&Cs												126695	3183.55
$\sum$ B&Cs and XP												120571	2262.25

Tabela 4.12: Resultados computacionais onde desigualdades CD e LD são separadas durante a execução do algoritmo NDR&C. O algoritmo L-B&C é alimentado com todos os cortes identificados durantes o NDR&C e, durante sua execução, apenas desigualdades CD são separadas.

Instance		R&C (CD, L-LD)						LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool size							
label	d	best LB	R&C	I/O	total	CD	LD	t(s)	gap	t(s)	t(s)	nodes	t(s)
High Density	$> 35\%$												
dim.queen8_8	0.36	0.00	2.96	0.04	3.04	1548	1562	–	–	–	81.97	1863	85.01
dim.miles1000	0.40	0.49	4.32	0.02	4.36	1047	1062	1.01	2	5.37	5.45	7	10.82
dim.queen7_7	0.40	0.00	1.61	0.02	1.65	940	1108	–	–	–	14.91	403	16.56
dim.DSJC125.9	0.90	0.00	4.46	0.03	4.52	1105	92	–	–	–	527.93	4229	532.45
mat.L125.bcsstk05	0.35	0.06	4.19	0.02	4.23	1288	1046	3.02	0	7.25	–	–	–
mat.L125.dwt_193	0.38	0.49	4.53	0.03	4.59	1402	1288	4.02	2	8.61	25.88	21	34.49
mat.L125.fs_183_1	0.44	0.01	1.96	0.02	2.00	26	32	–	–	–	30.82	29	32.82
mat.bcsstk04	0.68	0.00	3.57	0.02	3.61	726	14	1.02	2	4.63	17.39	13	22.02
mat.arc130	0.93	0.04	7.40	0.04	7.48	866	1544	6.03	9	13.51	173.15	103	186.66
mat.L100.steam2	0.36	0.00	4.12	0.03	4.18	1682	1612	3.02	3	7.20	18.53	45	25.73
mat.L120.fidap025	0.39	0.00	3.05	0.01	3.07	472	600	0.51	0	3.58	–	–	–
mat.L120.cavity01	0.42	0.14	3.34	0.02	3.38	490	584	–	–	–	6.24	11	9.62
mat.L120.fidap021	0.43	0.00	3.19	0.01	3.21	568	588	1.01	1	4.22	2.92	3	7.14
mat.L120.rbs480a	0.46	0.02	3.77	0.02	3.81	1028	762	2.02	3	5.83	65.98	123	71.81
mat.L120.wm2	0.47	0.00	1.28	0.01	1.30	0	0	–	–	–	18.17	75	19.47
mat.L100.rbs480a	0.52	0.00	1.87	0.01	1.89	436	140	0.51	2	2.40	15.78	61	18.18
mat.L80.wm2	0.58	0.15	1.06	0.01	1.08	32	58	–	–	–	3.74	13	4.82
mat.L100.wm3	0.59	0.07	1.36	0.01	1.38	15	28	–	–	–	10.62	17	12.00
mat.L120.e05r0000	0.59	0.00	2.14	0.02	2.18	355	58	–	–	–	5.52	7	7.70
mat.L100.wm1	0.60	0.01	1.49	0.01	1.51	205	156	–	–	–	8.08	17	9.59
mat.L120.fidap022	0.60	0.00	3.22	0.01	3.24	924	120	1.01	1	4.25	23.47	81	27.72
mat.L100.fidapm02	0.62	0.00	2.63	0.01	2.65	867	262	0.51	0	3.16	–	–	–
mat.L120.fidap001	0.63	0.00	3.65	0.02	3.69	1114	256	3.02	0	6.71	–	–	–
mat.L100.e05r0000	0.64	0.00	1.58	0.01	1.60	259	0	–	–	–	7.15	15	8.75
mat.L80.fidapm02	0.65	0.00	1.65	0.01	1.67	805	182	0.51	0	2.18	–	–	–
mat.L120.fidapm02	0.65	0.01	3.25	0.01	3.27	736	286	1.01	0	4.28	–	–	–
mat.L100.fidap001	0.68	0.00	2.15	0.01	2.17	373	18	0.51	2	2.68	6.32	33	9.00
mat.L100.fidap022	0.68	0.00	2.19	0.02	2.23	469	44	1.02	3	3.25	20.50	99	23.75
mat.L80.fidap001	0.72	0.00	1.08	0.01	1.10	202	14	0.51	0	1.61	–	–	–
mat.L80.fidap022	0.76	0.00	1.37	0.01	1.39	280	14	–	–	–	10.53	135	11.92
mat.L100.fidap027	0.81	0.00	1.94	–	–	483	86	–	–	–	–	–	–
mat.L100.fidap002	0.82	0.00	1.42	0.01	1.44	122	0	–	–	–	2.04	5	3.48
mat.L120.fidap002	0.82	0.00	2.56	0.02	2.60	251	6	–	–	–	4.00	5	6.60
mat.L120.fidap027	0.85	0.00	3.77	0.02	3.81	492	78	1.02	0	4.83	–	–	–
miplib.l152lav.p	0.40	0.89	1.55	0.01	1.57	317	342	–	–	–	62.78	245	64.35
miplib.lp4l.p	0.46	0.07	1.09	0.01	1.11	327	186	–	–	–	35.91	321	37.02
miplib.air03.p	0.61	0.20	3.55	0.02	3.59	517	468	–	–	–	107.91	117	111.50
miplib.misc03.p	0.63	0.13	4.72	0.04	4.80	1374	1396	–	–	–	133.51	3293	138.31
miplib.misc07.p	0.80	0.00	14.03	0.05	14.13	464	462	–	–	–	1148.62	169	1162.75
$\sum$ B&Cs												11389	1549.29
$\sum$ B&Cs and XP												7025	883.32
Mid-Low Density	$\leq 35\%$												
dim.games120	0.09	0.01	1.88	0.02	1.92	261	240	–	–	–	1064.78	82677	1066.70
dim.myciel7	0.13	0.02	3.32	0.01	3.34	375	404	–	–	–	371.02	2251	374.36
dim.myciel6	0.17	0.06	1.29	0.00	1.29	316	306	0.50	15	1.79	11.65	373	13.44
mat.can96	0.20	0.00	2.99	0.04	3.07	1068	976	–	–	–	1800.00	131426	1803.07
mat.can73	0.25	0.12	2.83	0.03	2.89	1373	1210	–	–	–	61.64	4975	64.53
mat.rw136	0.07	0.02	2.55	0.00	2.55	210	280	0.50	16	3.05	1800.00	367218	1803.05
mat.gre_115	0.09	0.03	4.52	0.07	4.66	983	852	–	–	–	125.20	6623	129.86
mat.L125.gre_185	0.15	0.00	7.63	0.07	7.77	1896	1728	6.04	9	13.81	91.84	1139	105.65
mat.can_144	0.16	0.00	9.08	0.10	9.28	2331	2210	9.06	5	18.34	1800.00	26048	1818.34
mat.lund_a	0.26	0.10	7.42	0.05	7.52	1837	1716	7.03	10	14.55	440.02	2033	454.57
miplib.noswot.p	0.09	0.04	3.19	0.02	3.23	406	444	–	–	–	1339.69	27833	1342.92
miplib.khb05250.p	0.27	0.00	0.84	0.00	0.84	25	0	–	–	–	3.44	119	4.28
$\sum$ B&Cs												128023	3556.31
$\sum$ B&Cs and XP												123739	2726.88

Tabela 4.13: Resultados computacionais onde apenas desigualdades CD são separadas durante a execução dos algoritmos PR&C e L-B&C.

Instance		PR&C (CD)					LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool						
label	d	best LB	R&C	I/O	total	size	t(s)	gap	t(s)	t(s)	nodes	t(s)
High Density	$> 35\%$											
dim.queen8_8	0.36	0.00	1.03	0.03	1.09	1828	–	–	–	89.13	1707	90.22
dim.miles1000	0.40	0.02	2.71	0.01	2.73	936	0.51	4	3.24	8.38	13	11.62
dim.queen7_7	0.40	0.00	0.6	0.01	0.62	1339	–	–	–	16.11	313	16.73
dim.DSJC125.9	0.90	0.00	3.85	0.02	3.89	1258	–	–	–	533.12	4143	537.01
mat.L125.bcsstk05	0.35	0.01	2.50	0.01	2.52	1211	2.01	0	4.53	–	–	–
mat.L125.dwt_193	0.38	1.25	2.73	0.01	2.75	1317	3.01	1	5.76	19.31	27	25.07
mat.L125.fs_183_1	0.44	0.73	1.68	0.01	1.70	30	–	–	–	24.46	21	26.16
mat.bcsstk04	0.68	0.00	3.29	0.02	3.33	966	1.02	0	4.35	–	–	–
mat.arc130	0.93	0.03	5.42	0.02	5.46	1170	6.02	8	11.48	152.32	83	163.80
mat.L100.steam2	0.36	0.00	1.90	0.01	1.92	1532	2.01	3	3.93	16.74	41	20.67
mat.L120.fidap025	0.39	0.00	2.06	0.01	2.08	789	0.51	0	2.59	–	–	–
mat.L120.cavity01	0.42	0.18	2.02	0.01	2.04	402	–	–	–	8.46	15	10.50
mat.L120.fidap021	0.43	0.00	2.10	0.01	2.12	777	0.51	1	2.63	3.25	7	5.88
mat.L120.rbs480a	0.46	0.01	2.67	0.02	2.71	1139	2.02	3	4.73	63.02	141	67.75
mat.L120.wm2	0.47	0.00	1.53	0.01	1.55	2	–	–	–	35.13	35	36.68
mat.L100.rbs480a	0.52	0.00	1.88	0.01	1.90	469	0.51	2	2.41	13.19	67	15.60
mat.L80.wm2	0.58	0.64	0.93	0.01	0.95	22	–	–	–	3.36	11	4.31
mat.L100.wm3	0.59	0.57	1.31	0.01	1.33	16	–	–	–	9.15	13	10.48
mat.L120.e05r0000	0.59	0.00	2.00	0.02	2.04	427	0.52	2	2.56	5.07	3	7.63
mat.L100.wm1	0.60	1.19	1.68	0.01	1.70	433	–	–	–	7.48	13	9.18
mat.L120.fidap022	0.60	0.00	2.87	0.02	2.91	912	1.02	1	3.93	9.82	17	13.75
mat.L100.fidapm02	0.62	0.00	2.14	0.01	2.16	808	0.51	0	2.67	–	–	–
mat.L120.fidap001	0.63	0.00	3.04	0.01	3.06	959	2.01	0	5.07	–	–	–
mat.L100.e05r0000	0.64	0.00	1.48	0.01	1.50	309	–	–	–	6.67	17	8.17
mat.L80.fidapm02	0.65	0.00	0.89	–	–	737	–	–	–	–	–	–
mat.L120.fidapm02	0.65	0.00	2.38	0.01	2.40	580	0.51	0	2.91	–	–	–
mat.L100.fidap001	0.68	0.00	2.02	0.01	2.04	529	0.51	2	2.55	4.99	35	7.54
mat.L100.fidap022	0.68	0.00	2.02	0.01	2.04	492	0.51	3	2.55	19.81	105	22.36
mat.L80.fidap001	0.72	0.00	1.02	0.01	1.04	229	0.51	0	1.55	–	–	–
mat.L80.fidap022	0.76	0.00	1.31	0.01	1.33	243	–	–	–	10.57	159	11.90
mat.L100.fidap027	0.81	0.00	1.60	–	–	474	–	–	–	–	–	–
mat.L100.fidap002	0.82	0.00	1.62	0.02	1.66	158	–	–	–	1.86	3	3.52
mat.L120.fidap002	0.82	0.00	2.30	0.02	2.34	263	–	–	–	3.39	1	5.73
mat.L120.fidap027	0.85	0.00	3.02	–	–	561	–	–	–	–	–	–
miplib.l152lav.p	0.40	0.13	1.64	0.01	1.66	1250	–	–	–	53.00	185	54.66
miplib.lp4l.p	0.46	0.48	1.32	0.00	1.32	670	–	–	–	31.78	271	33.10
miplib.air03.p	0.61	0.28	3.08	0.02	3.12	972	–	–	–	116.86	117	119.98
miplib.misc03.p	0.63	0.02	2.01	0.02	2.05	1131	–	–	–	119.62	2717	121.67
miplib.misc07.p	0.80	2.81	8.22	0.02	8.26	299	–	–	–	1511.12	177	1519.38
$\sum$ B&Cs											10280	1461.67
$\sum$ B&Cs and XP											6006	804.68
Mid-Low Density	$\leq 35\%$											
dim.games120	0.09	0.34	1.54	0.01	1.56	467	–	–	–	977.31	86767	978.87
dim.myciel7	0.13	1.69	2.78	0.01	2.80	518	–	–	–	380.93	2493	383.73
dim.myciel6	0.17	0.02	1.07	0.00	1.07	310	0.50	14	1.57	9.29	333	10.86
mat.can96	0.20	0.00	1.54	0.02	1.58	1121	–	–	–	1800.00	162842	1801.58
mat.can73	0.25	0.14	1.09	0.02	1.13	1208	–	–	–	49.44	5505	50.57
mat.rw136	0.07	0.00	1.20	0.00	1.20	264	0.50	11	1.70	1800.00	80603	1801.70
mat.gre_115	0.09	0.86	2.21	0.04	2.29	1256	–	–	–	90.61	6471	92.90
mat.L125.gre_185	0.15	0.00	2.55	0.03	2.61	1631	2.02	9	4.63	88.05	1089	92.68
mat.can_144	0.16	0.00	3.30	0.05	3.40	2176	4.03	5	7.43	1800.00	39849	1807.43
mat.lund_a	0.26	0.01	3.44	0.02	3.48	1557	4.02	10	7.50	376.64	2715	384.14
miplib.noswot.p	0.09	0.65	2.01	0.01	2.03	548	–	–	–	1349.62	27759	1351.65
miplib.khb05250.p	0.27	0.00	0.88	0.01	0.90	28	–	–	–	4.36	121	5.26
$\sum$ B&Cs											133253	3350.66
$\sum$ B&Cs and XP											128045	2582.79

Tabela 4.14: Resultados computacionais onde desigualdades CD e LD são separadas durante a execução dos algoritmos PR&C e L-B&C.

Instance		PR&C (CD, LD)						LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool size							
label	d	best LB	R&C	I/O	total	CD	LD	t(s)	gap	t(s)	t(s)	nodes	t(s)
High Density	$> 35\%$												
dim.queen8_8	0.36	0.00	3.59	0.06	3.71	1891	1894	–	–	–	132.32	2283	136.03
dim.miles1000	0.40	0.00	4.73	0.03	4.79	1148	1170	2.02	3	6.81	7.84	9	14.65
dim.queen7_7	0.40	0.00	2.29	0.03	2.35	1389	1664	–	–	–	18.54	243	20.89
dim.DSJC125.9	0.90	0.00	4.44	0.03	4.50	1242	72	–	–	–	771.16	6417	775.66
mat.L125.bcsstk05	0.35	0.01	4.94	0.03	5.00	1357	1336	3.02	0	8.02	–	–	–
mat.L125.dwt_193	0.38	1.15	4.21	0.03	4.27	1198	1082	4.02	2	8.29	32.37	39	40.66
mat.L125.fs_183_1	0.44	0.18	1.86	0.01	1.88	36	32	–	–	–	39.14	27	41.02
mat.bcsstk04	0.68	0.00	3.65	0.01	3.67	1050	62	2.01	0	5.68	–	–	–
mat.arc130	0.93	0.07	8.67	0.02	8.71	1298	2312	7.02	8	15.73	219.44	73	235.17
mat.L100.steam2	0.36	0.00	4.53	0.04	4.61	1812	1778	3.02	3	7.63	28.29	69	35.92
mat.L120.fidap025	0.39	0.00	3.67	0.03	3.73	954	1046	1.02	0	4.75	–	–	–
mat.L120.cavity01	0.42	0.32	3.32	0.01	3.34	541	618	–	–	–	8.49	15	11.83
mat.L120.fidap021	0.43	0.00	3.14	0.02	3.18	693	682	1.02	1	4.2	2.80	3	7.00
mat.L120.rbs480a	0.46	0.03	3.72	0.03	3.78	1081	694	2.02	3	5.8	55.34	75	61.14
mat.L120.wm2	0.47	0.00	1.72	0.00	1.72	2	4	–	–	–	36.85	33	38.57
mat.L100.rbs480a	0.52	0.00	2.03	0.01	2.05	491	140	0.51	2	2.56	14.20	45	16.76
mat.L80.wm2	0.58	0.43	1.43	0.01	1.45	61	92	–	–	–	5.08	13	6.53
mat.L100.wm3	0.59	0.55	1.90	0.01	1.92	72	102	–	–	–	8.74	13	10.66
mat.L120.e05r0000	0.59	0.00	2.40	0.02	2.44	409	52	–	–	–	5.38	7	7.82
mat.L100.wm1	0.60	1.94	2.32	0.01	2.34	453	396	–	–	–	9.66	17	12.00
mat.L120.fidap022	0.60	0.00	3.16	0.01	3.18	894	88	1.01	1	4.19	17.03	43	21.22
mat.L100.fidapm02	0.62	0.00	2.56	0.01	2.58	765	218	0.51	0	3.09	–	–	–
mat.L120.fidap001	0.63	0.00	3.60	0.02	3.64	1065	192	3.02	0	6.66	–	–	–
mat.L100.e05r0000	0.64	0.00	1.66	0.01	1.68	309	0	–	–	–	4.91	19	6.59
mat.L80.fidapm02	0.65	0.00	1.66	0.01	1.68	882	120	0.51	0	2.19	–	–	–
mat.L120.fidapm02	0.65	0.01	3.12	0.01	3.14	629	276	0.51	0	3.65	–	–	–
mat.L100.fidap001	0.68	0.00	2.25	0.01	2.27	494	24	0.51	2	2.78	7.04	29	9.82
mat.L100.fidap022	0.68	0.00	2.29	0.01	2.31	629	66	0.51	3	2.82	14.37	54	17.19
mat.L80.fidap001	0.72	0.00	1.22	0.01	1.24	233	16	0.51	0	1.75	–	–	–
mat.L80.fidap022	0.76	0.00	1.44	0.01	1.46	225	8	–	–	–	6.22	57	7.68
mat.L100.fidap027	0.81	0.00	2.12	–	–	460	60	–	–	–	–	–	–
mat.L100.fidap002	0.82	0.00	1.84	0.01	1.86	165	4	–	–	–	2.19	5	4.05
mat.L120.fidap002	0.82	0.00	2.50	0.02	2.54	258	2	–	–	–	3.04	3	5.58
mat.L120.fidap027	0.85	0.00	2.33	–	–	427	50	–	–	–	–	–	–
miplib.l152lav.p	0.40	1.58	2.92	0.03	2.98	1284	1260	–	–	–	57.50	129	60.48
miplib.lp4l.p	0.46	0.40	2.38	0.03	2.44	741	598	–	–	–	47.75	265	50.19
miplib.air03.p	0.61	1.71	4.40	0.03	4.46	864	936	–	–	–	159.41	111	163.87
miplib.misc03.p	0.63	0.01	5.11	0.07	5.25	1438	1658	–	–	–	168.17	2225	173.42
miplib.misc07.p	0.80	3.57	9.76	0.05	9.86	286	316	–	–	–	1694.89	147	1704.75
$\sum$ B&Cs												12321	1992.40
$\sum$ B&Cs and XP												5765	1052.87
Mid-Low Density	$\leq 35\%$												
dim.games120	0.09	0.00	2.66	0.02	2.70	603	484	–	–	–	1023.82	85787	1026.52
dim.myciel7	0.13	0.04	3.98	0.02	4.02	556	588	–	–	–	425.73	2491	429.75
dim.myciel6	0.17	0.29	1.62	0.01	1.64	523	554	0.51	13	2.15	11.00	319	13.15
mat.can96	0.20	0.00	3.67	0.05	3.77	1389	1266	–	–	–	1800.00	129012	1803.77
mat.can73	0.25	0.20	2.90	0.04	2.98	1520	1340	–	–	–	56.37	5195	59.35
mat.rw136	0.07	0.04	6.90	0.00	6.90	844	1064	6.00	11	12.9	1800.00	71316	1812.90
mat.gre_115	0.09	0.04	4.66	0.14	4.94	1000	926	–	–	–	108.11	5993	113.05
mat.L125.gre_185	0.15	0.01	7.56	0.07	7.70	2000	1888	7.04	9	14.74	96.30	1255	111.04
mat.can_144	0.16	0.00	8.70	0.10	8.90	2396	2168	10.06	5	18.96	1800.00	21666	1818.96
mat.lund_a	0.26	0.01	7.19	0.04	7.27	1831	1708	8.02	10	15.29	495.72	3005	511.01
miplib.noswot.p	0.09	0.34	2.92	0.01	2.94	296	274	–	–	–	1009.89	17783	1012.83
miplib.khb05250.p	0.27	0.00	0.99	0.00	0.99	28	0	–	–	–	4.44	121	5.43
$\sum$ B&Cs												121949	3282.13
$\sum$ B&Cs and XP												116453	2341.37

Tabela 4.15: Resultados computacionais onde desigualdades CD e LD são separadas durante a execução do algoritmo PR&C. O algoritmo L-B&C é alimentado com todos os cortes identificados por PR&C e, durante sua execução, apenas desigualdades CD são separadas.

Instance		PR&C (CD, L-LD)						LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool size							
label	d	best LB	R&C	I/O	total	CD	LD	t(s)	gap	t(s)	t(s)	nodes	t(s)
High Density	$> 35\%$												
dim.queen8_8	0.36	0.00	3.59	0.06	3.71	1891	1894	–	–	–	132.69	2283	136.40
dim.miles1000	0.40	0.00	4.73	0.03	4.79	1148	1170	2.02	3	6.81	7.24	9	14.05
dim.queen7_7	0.40	0.00	2.29	0.03	2.35	1389	1664	–	–	–	18.13	243	20.48
dim.DSJC125.9	0.90	0.00	4.44	0.03	4.50	1242	72	–	–	–	718.00	6405	722.50
mat.L125.bcsstk05	0.35	0.01	4.94	0.03	5.00	1357	1336	3.02	0	8.02	–	–	–
mat.L125.dwt_193	0.38	1.15	4.21	0.03	4.27	1198	1082	4.02	2	8.29	32.49	39	40.78
mat.L125.fs_183_1	0.44	0.18	1.86	0.01	1.88	36	32	–	–	–	67.49	746	69.37
mat.bcsstk04	0.68	0.00	3.65	0.01	3.67	1050	62	2.01	0	5.68	–	–	–
mat.arc130	0.93	0.07	8.67	0.02	8.71	1298	2312	7.02	8	15.73	216.05	73	231.78
mat.L100.steam2	0.36	0.00	4.53	0.04	4.61	1812	1778	3.02	3	7.63	30.43	79	38.06
mat.L120.fidap025	0.39	0.00	3.67	0.03	3.73	954	1046	1.02	0	4.75	–	–	–
mat.L120.cavity01	0.42	0.32	3.32	0.01	3.34	541	618	–	–	–	15.01	35	18.35
mat.L120.fidap021	0.43	0.00	3.14	0.02	3.18	693	682	1.02	1	4.2	2.78	3	6.98
mat.L120.rbs480a	0.46	0.03	3.72	0.03	3.78	1081	694	2.02	3	5.8	53.40	75	59.20
mat.L120.wm2	0.47	0.00	1.72	0.00	1.72	2	4	–	–	–	24.83	73	26.55
mat.L100.rbs480a	0.52	0.00	2.03	0.01	2.05	491	140	0.51	2	2.56	12.88	45	15.44
mat.L80.wm2	0.58	0.43	1.43	0.01	1.45	61	92	–	–	–	4.62	51	6.07
mat.L100.wm3	0.59	0.55	1.90	0.01	1.92	72	102	–	–	–	14.32	61	16.24
mat.L120.e05r0000	0.59	0.00	2.40	0.02	2.44	409	52	–	–	–	5.30	7	7.74
mat.L100.wm1	0.60	1.94	2.32	0.01	2.34	453	396	–	–	–	16.06	71	18.40
mat.L120.fidap022	0.60	0.00	3.16	0.01	3.18	894	88	1.01	1	4.19	16.41	43	20.60
mat.L100.fidapm02	0.62	0.00	2.56	0.01	2.58	765	218	0.51	0	3.09	–	–	–
mat.L120.fidap001	0.63	0.00	3.60	0.02	3.64	1065	192	3.02	0	6.66	–	–	–
mat.L100.e05r0000	0.64	0.00	1.66	0.01	1.68	309	0	–	–	–	4.78	19	6.46
mat.L80.fidapm02	0.65	0.00	1.66	0.01	1.68	882	120	0.51	0	2.19	–	–	–
mat.L120.fidapm02	0.65	0.01	3.12	0.01	3.14	629	276	0.51	0	3.65	–	–	–
mat.L100.fidap001	0.68	0.00	2.25	0.01	2.27	494	24	0.51	2	2.78	7.09	29	9.87
mat.L100.fidap022	0.68	0.00	2.29	0.01	2.31	629	66	0.51	3	2.82	13.35	54	16.17
mat.L80.fidap001	0.72	0.00	1.22	0.01	1.24	233	16	0.51	0	1.75	–	–	–
mat.L80.fidap022	0.76	0.00	1.44	0.01	1.46	225	8	–	–	–	5.72	57	7.18
mat.L100.fidap027	0.81	0.00	2.12	–	–	460	60	–	–	2.–	–	–	–
mat.L100.fidap002	0.82	0.00	1.84	0.01	1.86	165	4	–	–	–	2.14	5	4.00
mat.L120.fidap002	0.82	0.00	2.50	0.02	2.54	258	2	–	–	–	3.02	3	5.56
mat.L120.fidap027	0.85	0.00	2.33	–	–	427	50	–	–	–	–	–	–
miplib.l152lav.p	0.40	1.58	2.92	0.03	2.98	1284	1260	–	–	–	69.78	611	72.76
miplib.lp4l.p	0.46	0.40	2.38	0.03	2.44	741	598	–	–	–	91.38	2083	93.82
miplib.air03.p	0.61	1.71	4.40	0.03	4.46	864	936	–	–	–	135.40	135	139.86
miplib.misc03.p	0.63	0.01	5.11	0.07	5.25	1438	1658	–	–	–	162.98	2589	168.23
miplib.misc07.p	0.80	3.57	9.76	0.05	9.86	286	316	–	–	–	1800.00	1102	1809.86
$\sum$ B&Cs												15926	1982.86
$\sum$ B&Cs and XP												9358	1120.50
Mid-Low Density	$\leq 35\%$												
dim.games120	0.09	0.00	2.66	0.02	2.70	603	484	–	–	–	866.45	85615	869.15
dim.myciel7	0.13	0.04	3.98	0.02	4.02	556	588	–	–	–	265.24	1991	269.26
dim.myciel6	0.17	0.29	1.62	0.01	1.64	523	554	0.51	13	2.15	8.88	385	11.03
mat.can96	0.20	0.00	3.67	0.05	3.77	1389	1266	–	–	–	1800.00	134256	1803.77
mat.can73	0.25	0.20	2.90	0.04	2.98	1520	1340	–	–	–	56.65	5157	59.63
mat.rw136	0.07	0.04	6.90	0.00	6.90	844	1064	6.00	11	12.9	1800.00	88741	1812.90
mat.gre_115	0.09	0.04	4.66	0.14	4.94	1000	926	–	–	–	106.07	6047	111.01
mat.L125.gre_185	0.15	0.01	7.56	0.07	7.70	2000	1888	7.04	9	14.74	91.89	1255	106.63
mat.can_144	0.16	0.00	8.70	0.10	8.90	2396	2168	10.06	5	18.96	1800.00	21465	1818.96
mat.lund_a	0.26	0.01	7.19	0.04	7.27	1831	1708	8.02	10	15.29	527.80	3565	543.09
miplib.noswot.p	0.09	0.34	2.92	0.01	2.94	296	274	–	–	–	737.28	17853	740.22
miplib.khb05250.p	0.27	0.00	0.99	0.00	0.99	28	0	–	–	–	4.17	121	5.16
$\sum$ B&Cs												121989	2714.68
$\sum$ B&Cs and XP												116433	1902.33

Tabela 4.16: Resultados computacionais para instâncias de média e baixa densidades onde desigualdades CD e LD são separadas apenas durante a execução do algoritmo NDR&C.

Instance		R&C (L-CD, L-LD)						LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool size							
label	d	best LB	R&C	I/O	total	CD	LD	t(s)	gap	t(s)	t(s)	nodes	t(s)
Mid-Low Density	$\leq 35\%$												
dim.games120	0.09	0.00	2.66	0.02	2.70	603	484	–	–	–	738.24	96985	740.94
dim.myciel7	0.13	0.04	3.98	0.02	4.02	556	588	–	–	–	284.56	2499	288.58
dim.myciel6	0.17	0.29	1.62	0.01	1.64	523	554	0.5	13	1.79	8.51	415	10.30
mat.can96	0.20	0.00	3.67	0.05	3.77	1389	1266	–	–	–	1800.00	192310	1803.77
mat.can73	0.25	0.20	2.90	0.04	2.98	1520	1340	–	–	–	55.32	7089	58.30
mat.rw136	0.07	0.04	6.90	0.00	6.90	844	1064	0.50	11	3.05	1800.00	345446	1803.05
mat.gre_115	0.09	0.04	4.66	0.14	4.94	1000	926	–	–	–	88.66	6393	93.60
mat.L125.gre_185	0.15	0.01	7.56	0.07	7.70	2000	1888	6.04	9	13.81	68.58	1655	82.39
mat.can_144	0.16	0.00	8.70	0.10	8.90	2396	2168	9.06	5	18.34	1800.00	29193	1818.34
mat.lund_a	0.26	0.01	7.19	0.04	7.27	1831	1708	7.03	10	14.55	587.87	5815	602.42
miplib.noswot.p	0.09	0.34	2.92	0.01	2.94	296	274	–	–	–	684.57	29227	687.51
miplib.khb05250.p	0.27	0.00	0.99	0.00	0.99	28	0	–	–	–	3.49	119	4.48
$\sum$ B&Cs												150197	2566.83
$\sum$ B&Cs and XP												141883	1676.51

Tabela 4.17: Resultados computacionais para instâncias de média e baixa densidades onde desigualdades CD e LD são separadas apenas durante a execução do algoritmo PR&C.

Instance		PR&C (L-CD, L-LD)						LP		LP+R&C	L-B&C		HYBRID
		t(s)				pool size							
label	d	best LB	R&C	I/O	total	CD	LD	t(s)	gap	t(s)	t(s)	nodes	t(s)
Mid-Low Density	$\leq 35\%$												
dim.games120	0.09	0.00	2.66	0.02	2.70	603	484	–	–	–	751.38	97349	754.08
dim.myciel7	0.13	0.04	3.98	0.02	4.02	556	588	–	–	–	266.28	2799	270.30
dim.myciel6	0.17	0.29	1.62	0.01	1.64	523	554	0.51	13	2.15	7.88	395	10.03
mat.can96	0.20	0.00	3.67	0.05	3.77	1389	1266	–	–	–	1800.00	157351	1803.77
mat.can73	0.25	0.20	2.90	0.04	2.98	1520	1340	–	–	–	57.39	7207	60.37
mat.rw136	0.07	0.04	6.90	0.00	6.90	844	1064	6.00	11	12.9	1800.00	155538	1812.90
mat.gre_115	0.09	0.04	4.66	0.14	4.94	1000	926	–	–	–	84.83	6371	89.77
mat.L125.gre_185	0.15	0.01	7.56	0.07	7.70	2000	1888	7.04	9	14.74	62.82	1639	77.56
mat.can_144	0.16	0.00	8.70	0.10	8.90	2396	2168	10.06	5	18.96	1800.00	24692	1818.96
mat.lund_a	0.26	0.01	7.19	0.04	7.27	1831	1708	8.02	10	15.29	634.52	6465	649.81
miplib.noswot.p	0.09	0.34	2.92	0.01	2.94	296	274	–	–	–	712.35	29219	715.29
miplib.khb05250.p	0.27	0.00	0.99	0.00	0.99	28	0	–	–	–	3.96	121	4.95
$\sum$ B&Cs												151565	2631.66
$\sum$ B&Cs and XP												142301	1711.55

Tabela 4.18: Número de cortes nos *pools* de desigualdades ao final da execução dos algoritmos de R&C para testes com instâncias difíceis da MIPLIB.

Instance		Pool size	
label	d	CD cuts	LD cuts
set1al	0.78	0	0
set1cl	0.78	0	0
set1ch	0.81	1	2
fixnet3_r	1.10	72	50
misc06	1.21	0	0
qnet1_o	3.59	0	0
qnet1	3.60	1	2
danoint	4.49	2000	2030
gams	5.13	117	140
adrud	5.50	597	476
p0548	7.82	2	0
air04	16.67	1616	1402
air06	16.82	1945	1688
stein45_r	19.59	3743	3632
fast0507	20.82	1519	-
stein27_r	32.20	1679	-
air05	34.37	1493	-
10teams	34.45	2789	-
mod010	37.97	1576	-
misc05	40.09	486	-
p0282	40.89	73	-

C3: Teste de Wilcoxon em detalhes

O teste estatístico de Wilcoxon tem despertado interesse recente como uma técnica alternativa para comparação de desempenhos de pares de algoritmos heurísticos. Uma das motivações para tal é a possibilidade de incorporar análises probabilísticas, realizadas de uma maneira científica, a avaliações puramente empíricas de algoritmos heurísticos.

O emprego do teste de Wilcoxon, contudo, não é restrito a avaliações de métodos heurísticos. No presente trabalho o teste é usado para confrontrar resultados obtidos por dois métodos exatos. Neste caso, é usada como métrica a relação entre tempos de processamento requeridos por cada algoritmo para provar a otimalidade das soluções quando comparados àqueles do melhor algoritmo reportado na literatura para o VSP, descrito em [11].

O teste de Wilcoxon pressupõe, entre outras (veja [14] para detalhes a respeito de premissas assumidas pela técnica), que as instâncias utilizadas formam uma amostra representativa para o problema em questão. No caso presente, estamos restringindo o emprego do teste para inferirmos conclusões acerca de instâncias do VSP cujos grafos de entrada apresentam média e alta densidades (i.e., maiores que 20%). Além disso, como todos os custos envolvidos nas instâncias utilizadas são unitários, nossas conclusões também são restritas por esta característica comum.

Desta forma, sejam P, ND e BC, respectivamente, rótulos atribuídos aos algoritmos PHYBRID(CD), NDHYBRID(CD) e BC(CD) descritos na seção 4.7.2. Assim, denotemos por t_a^i o tempo que o algoritmo $a \in A$ leva para resolver a instância i à otimalidade e por $r(\frac{a}{a'})^i = t_a^i/t_{a'}^i$ as razões entre os tempos de processamento dos algoritmos a e a' quando resolvendo a instância $i = 1, 2, \ldots, n$. A segunda e terceira colunas da Tabela 4.19 exibem estes valores[13] para $A = \{P, ND, BC\}$. Eles formam as séries de resultados obtidos pelos algoritmos analisados e são a base para os cálculos referentes ao teste estatístico de Wilcoxon, cujos passos são descritos de forma resumida a seguir:

1. Para cada instância i, calcule $|d(\frac{a_1,a_2}{a})^i|$, onde $d(\frac{a_1,a_2}{a})^i = r(\frac{a_1}{a'})^i - r(\frac{a_2}{a'})^i$ é a diferença entre as razões correspondentes aos pares de algoritmos (a_1, a') (a_2, a');

2. Classifique (*rank*) os resultados a partir dos valores de $|d(\frac{a_1,a_2}{a})^i|$. Descarte toda instância i tal que $|d(\frac{a_1,a_2}{a})^i| = 0$ e decremente o valor de n a cada descarte. Ocorrendo empates em valores de $|d(\frac{a_1,a_2}{a})^i|$, calcule a média das classificações dos itens envolvidos no empate e, a cada um deles, atribua a classificação média calculada.

3. A cada classificação $c_i, 1 \leq i \leq n$, atribua um sinal igual àquele do termo $d(\frac{a_1,a_2}{a})^i$ e obtenha a classificação com sinal c_i'.

4. Finalmente, calcule a soma dada por $W = \sum_{i=1}^{k} c_i'$.

Os resultados destes cálculos para as instâncias consideradas são mostrados nas colunas (4–7) da Tabela 4.19.

A hipótese nula do teste estatístico de Wilcoxon consiste em recusar a dominância de algum dos algoritmos confrontados. No caso presente, ela consiste em considerar que P e ND apresentam desempenhos semelhantes. Em contrapartida, a chamada hipótese alternativa supõe que existe uma superioridade de ND em relação a P. Neste caso, para que a hipótese nula seja rejeitada a um certo nível de confiabilidade α, é necessário constatarmos que $W > W'$ onde, para $n > 10$, o valor crítico W' pode ser aproximado pela expressão

$$W' = Z(\alpha)\sqrt{n(n+1)(2n+1)/6} \tag{4.17}$$

sendo $Z(\alpha)$ o ponto da curva de distribuição normal de forma que $(100 \times \alpha)\%$ da área sob a curva fica à sua esquerda. Desta forma, se adotarmos uma margem de confiança de 90% (i.e., $\alpha = 0.10$, implicando $Z(0.10) = 1.282$ para testes direcionais), obtemos $W' = 169.96$. Este resultado nos diz que se a hipótese nula é verdadeira (e as premissas do teste de Wilcoxon são válidas), então em apenas 10% das tentativas W seria superior

[13] Estes foram usados para cálculo das médias correspondentes aos algoritmos PHYBRID(CD) e NDHYBRID(CD) mostradas no gráfico da Figura 4.12.

a W'. Mas, como $W = 181 > 169,96 = W'$, esta hipótese é rejeitada com um nível de confiabilidade de 90%. Ou seja, nós podemos inferir, com uma margem de 90% de confiança, que NDHYBRID(CD) tem melhor desempenho que PHYBRID(CD).

Tabela 4.19: Teste de Wilcoxon para comparação entre PHYBRID(CD) e NDHYBRID(CD).

label	$r\left(\frac{B}{BC}\right)^i$	$r\left(\frac{ND}{BC}\right)^i$	$d\left(\frac{B,ND}{BC}\right)^i$	$\lvert d\left(\frac{B,ND}{BC}\right)^i \rvert$	c_i	c'_i
dim.queen8_8	127.95	106.40	21.56	21.56	33	33
dim.miles1000	63.26	70.60	-7.35	7.35	22	-22
dim.queen7_7	155.34	131.01	24.33	24.33	35	35
dim.DSJC125.9	48.50	53.87	-5.37	5.37	17	-17
mat.can73	70.73	64.55	6.18	6.18	19	19
mat.lund_a	83.14	86.88	-3.73	3.73	14	-14
mat.L125.bcsstk05	1.39	26.26	-24.87	24.87	36	-36
mat.L125.dwt__193	25.76	23.75	2.01	2.01	10	10
mat.L125.fs_183_1	73.79	79.29	-5.50	5.50	18	-18
mat.bcsstk04	3.49	18.38	-14.89	14.89	27	-27
mat.arc130	44.19	43.22	0.97	0.97	6	6
mat.L100.steam2	50.62	51.48	-0.86	0.86	5	-5
mat.L120.fidap025	21.58	20.83	0.75	0.75	4	4
mat.L120.cavity01	62.20	54.09	8.12	8.12	23	23
mat.L120.fidap021	23.77	22.76	1.01	1.01	7	7
mat.L120.rbs480a	30.98	29.27	1.71	1.71	8	8
mat.L120.wm2	76.70	58.64	18.07	18.07	30	30
mat.L100.rbs480a	73.14	55.79	17.35	17.35	29	29
mat.L80.wm2	87.96	66.94	21.02	21.02	31	31
mat.L100.wm3	79.03	57.54	21.49	21.49	32	32
mat.L120.e05r0000	66.41	61.36	5.05	5.05	16	16
mat.L100.wm1	58.58	68.54	-9.96	9.96	24	-24
mat.L120.fidap022	36.02	59.03	-23.00	23.00	34	-34
mat.L120.fidap001	15.57	15.17	0.40	0.40	2	2
mat.L100.e05r0000	73.01	74.98	-1.97	1.97	9	-9
mat.L120.fidapm02	11.87	11.70	0.16	0.16	1	1
mat.L100.fidap001	47.24	44.49	2.76	2.76	12	12
mat.L100.fidap022	42.84	43.41	-0.57	0.57	3	-3
mat.L80.fidap001	88.07	88.07	0.00	0.00	–	–
mat.L80.fidap022	77.88	93.26	-15.38	15.38	28	-28
mat.L100.fidap002	74.11	67.79	6.32	6.32	20	20
mat.L120.fidap002	13.78	16.54	-2.77	2.77	13	-13
miplib.khb05250.p	163.86	122.74	41.12	41.12	37	37
miplib.l152lav.p	77.95	65.99	11.97	11.97	25	25
miplib.lp4l.p	66.07	68.40	-2.34	2.34	11	-11
miplib.air03.p	71.69	66.89	4.80	4.80	15	15
miplib.misc03.p	88.12	81.08	7.04	7.04	21	21
miplib.misc07.p	84.41	71.13	13.28	13.28	26	26
W ($\sum$ S.Rank)						181

Bibliography

[1] T. P. Bagchi. Pareto-optimal solutions for multi-objective production scheduling problems. *Evolutionary Multi-Criterion Optimization. Lecture Notes in Computer Science*, 1993/2001:458–471, 2001.

[2] E. Balas and N. Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21:19–46, 1981.

[3] E. Balas and C. C. de Souza. The vertex separator problem: a polyhedral investigation. *Mathematical Programming*, 103:583–608, 2005.

[4] E. Balas and C. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization. *Mathematical Programming Study*, 12:37–60, 1980.

[5] J.E. Beasley. *Modern Heuristic Techniques*, chapter 6. Blackwell Scientific Press, Oxford, 1993.

[6] F. Calheiros, A. Lucena, and C. C. de Souza. Optimal rectangular partitions. *Networks*, 41:51–67, 2003.

[7] V. Cavalcante and C. C. de Souza. Lagrangian relaxation and cutting planes for the vertex separator problem. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE)*, pages 471–482, 2007.

[8] V. Cavalcante, C. C. de Souza, and A. Lucena. A relax-and-cut algorithm for the set partitioning problem. *Computers Operations Research*, 35:1963–1981, 2008.

[9] A.S. Cunha. *Árvores ótimas em grafos: modelos, algoritmos e aplicações*. PhD thesis, Engenharia de Sistemas e Computação, UFRJ, Rio de Janeiro, Maio 2006.

[10] A.S. Cunha and A. Lucena. Lower and upper bounds for the degree-constrained minimum spanning tree problem. *Networks*, 50(1), April 2007.

[11] C. C. de Souza and E. Balas. The vertex separator problem: algorithms and computations. *Mathematical Programming*, 103:609–631, 2005.

[12] M.F.F. Filho. *GRASP e Busca Tabu Aplicados a Problemas de Programação de Tarefas em Máquinas Paralelas*. PhD thesis, Departamento de Engenharia de Sistemas, Facauldade de Engenharia Elétrica e Ciência da Computação , UNICAMP, October 2007.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W.H. Freeman and Co., New York, 1979.

[14] B.L. Golden and W. R. Stewart. *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, chapter Empirical analysis of heuristics. John Wiley and Sons, Chichester, NY, Brisbane, Toronto, 1985.

[15] M. Guignard. Lagrangean relaxation. In M.A. Lòpez-Cerdá and I. García-Jurado, editors, *Top*, volume 11, pages 151–200. Springer Berlin/ Heidelberg, 2003.

[16] G. Kliewer and L. Timajev. Relax-and-cut for capacitated network design. In *Proceedings of 13th Annual European Symposium on Algorithms (ESA)*, volume 3669, pages 47–58, Mallorca, Spain, October 2005.

[17] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. In *COAL Bulletin*, volume 21. Mathematical Programming Society, 1992.

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

[19] C. Martinhon, A. Lucena, and N. Maculan. Stronger k-tree relaxations for the vehicle routing problem. *European Journal on Operational Research*, 158:56–71, 2004.

[20] C.E. Ferreira R. Borndörfer and A. Martin. Decomposing matrices into blocks. *SIAM Journal on optimization*, 9:236–269, 1998.

[21] R.Z. Ríos-Mercado and J. F. Bard. An enhanced tsp-based heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, 5:53–70, 1999.

Capítulo 5

Conclusões e trabalhos futuros

O principal objetivo deste trabalho de pesquisa foi averigüar a aplicabilidade do uso combinado de relaxação Lagrangiana e Combinatória Poliédrica no contexto de problemas de Programação Inteira 0–1. Com este fim, foram estudados o problema de particionamento de conjuntos (SPP) e o problema do separador de vértices (VSP) e, para ambos, algoritmos *relax-and-cut* foram desenvolvidos e *frameworks* para solução concebidos. Em seguida, experimentos computacionais foram realizados e os resultados obtidos foram confrontados com aqueles reportados para os algoritmos de melhor desempenho disponíveis na literatura.

Basicamente duas formas de uso de algoritmos *relax-and-cut* foram empregadas: `(i)` como métodos heurísticos que, a exemplo dos algoritmos Lagrangianos clássicos, conseguem estimar a qualidade das soluções primais produzidas, comparando-os com os limitantes duais que, neste caso, devido à geração de cortes, são potencialmente mais fortes e `(ii)`, como uma ferramenta de pré-processamento para algoritmos tradicionais de Programação Inteira baseados em planos de corte, sendo capaz de gerar desigualdades válidas para o problema tratado que, espera-se, contribuam para acelerar a convergência destes últimos. De fato, experimentos feitos neste trabalho mostraram que, em muitos casos, esta hibridização dos algoritmos é mais eficiente do que outras abordagens exatas disponíveis para resolver os problemas estudados.

Para o problema de particionamento de conjuntos, os resultados computacionais obtidos evidenciaram que o *framework* proposto é competitivo com os demais algoritmos Lagrangianos testados. Em particular, a separação de desigualdades do tipo *clique* teve um impacto bastante positivo na melhoria dos limitantes duais produzidos pela relaxação Lagrangiana. Como conseqüência, várias instâncias foram resolvidas pelo algoritmo *relax-and-cut* à otimalidade (66% das instâncias testadas) e, em particular, uma instância não resolvida pelo `XPRESS` em 10 horas de CPU teve sua otimalidade provada em pouco mais de 7 minutos usando o algoritmo proposto aqui. Crucial para a obtenção destes resulta-

dos foi a verificação de dominância no gerenciamento das desigualdades identificadas ao longo da execução do algoritmo. Além disso, notadamente para instâncias onde o XPRESS encontrou dificuldades de computação, a hibridização proposta para o SPP mostrou ser bastante efetiva, diminuindo os tempos para solução dos problemas.

Possíveis extensões deste trabalho sobre o SPP incluem: (a) a introdução de procedimentos mais agressivos de pré-processamento de instâncias do mundo real, a exemplo de [1, 10, 14]; (b) a separação de novas classes de desigualdades para o problema e, (c) a melhoria da nossa heurística primal com base em idéias implementadas em [1].

No estudo realizado sobre o problema do separador de vértices várias configurações de algoritmos *relax-and-cut* foram desenvolvidas e testadas. Todas as versões implementadas se basearam em planos de corte derivados de desigualdades válidas para o VSP apresentadas em [3]. Os resultados computacionais revelaram que, em geral, os algoritmos *relax-and-cut* conseguiam produzir soluções ótimas ou (quase ótimas) muito rapidamente. Entretanto, para poucas instâncias, o *gap* de integralidade foi fechado sem auxílio da hibridização proposta com o algoritmo *branch-and-cut* descrito em [16]. Neste caso, os resultados computacionais atestaram, entre outras coisas, que 92% das instâncias com maior densidade foram resolvidas mais rapidamente usando nosso *framework* do que pela melhor configuração do algoritmo implementado por de Souza e Balas, a melhor abordagem exata atualmente disponível para o VSP. No caso de grafos menos densos, os resultados mostraram que nossa abordagem é competitiva com as demais testadas. Além destes, resultados obtidos para instâncias de problemas em aberto provenientes da MIPLIB atestaram o bom desempenho da nossa heurística Lagrangiana a qual, na maioria dos casos, melhorou o valor conhecido para aquelas instâncias até o presente momento.

Vale ressaltar também que, no trabalho desenvolvido com o VSP, é sugerida uma forma distinta das usualmente descritas na literatura para utilização de planos de corte combinados com algoritmos Lagrangianos. Como uma contribuição teórica, foi mostrado que o problema de determinar a existência de uma desigualdade CD violada pode ser resolvido polinomialmente ou é $\mathcal{NP}$-completo, dependendo da relaxação adotada. Este resultado justifica o emprego de nossa heurística para a separação de desigualdades daquela classe.

Claramente, o aprimoramento dos algoritmos aqui desenvolvidos, através da integração de novas desigualdades válidas e do desenvolvimento de novas heurísticas primais, surgem como uma direção de pesquisa imediata. Além destes, futuras direções incluem estudos de como adaptar de forma eficiente outros algoritmos Lagrangianos ao esquema proposto pelos algoritmos *relax-and-cut* como, por exemplo, os métodos de Feixe (*bundle*) e o algoritmo do Volume. Finalmente, uma alternativa natural de pesquisa seria o desenvolvimento de novos algoritmos *relax-and-cut* para outros problemas de otimização discreta.

Appendix A

Lagrangian relaxation and cutting planes for Set Partitioning

Este trabalho foi publicado nos Anais do *XIII Congreso Latino-Iberoamericano de Investigación Operativa (CLAIO)*, Montevideo - Uruguai, em novembro de 2006 (artigo 158).

Relax-and-Cut (R&C) algorithms offer an alternative to strengthen Lagrangian relaxation bounds by dynamically selecting and dualizing inequalities (cuts) within a Lagrangian Relaxation framework. This paper proposes a Relax-and-Cut for the Set Partitioning Problem (SPP). Usually, for SPP instances with integrality gap, a variant of the classical Lagrangian relaxation (LR) is used where a knapsack constraint is amended to the model. Our computational results showed that the Relax-and-Cut algorithm outperforms the latter approach both in terms of lower bounds and of CPU time. Decisive in achieving this performance was the pool management proposed that, to our knowledge, is a novelty for Relax-and-Cut algorithms.

A.1 Introduction

The term Relax-and-Cut (R&C) was coined to denote a class of Lagrangian Relaxation algorithms where inequalities may only be explicitly dualized when they become violated at a Lagrangian relaxation solution (see [6, 8]).These algorithms appear as a promising alternative approach to strengthen Lagrangian relaxation bounds. They use a dynamic inequality dualization scheme that renders viable the application of Lagrangian Relaxation to models with an exponential number of inequalities. Indeed, attempts to do that for the TSP [1] date from the early 1980's.

In this study, a R&C algorithm is proposed to the Set Partitioning Problem (SPP).

A general description of the R&C framework is given in section A.2. A R&C algorithm for SPP is proposed in section A.3 where details are given for the Lagrangian heuristic used and for the cut pool management policy implemented. Section A.4 reports on the computational results obtained for test sets used. Finally, section A.5 presents some conclusions and discusses a possible extension to this study.

A.2 Relax-and-Cut Algorithms

Denote by X a subset of $\mathbb{B}^n = \{0,1\}^n$ and let

$$Z = \min \{cx : Ax \leq b,\ x \in X\} \tag{A.1}$$

be a formulation for a $\mathcal{NP}$-hard combinatorial optimization problem. In association with (A.1) one has $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$, where m and n are positive integral values representing, respectively, the number of constraints and the number of variables involved. Assuming that m is an exponential function of n, let Z' denote the formulation obtained after removing constraints $Ax \leq b$ from (A.1). Also, assume that Z' can be solved in polynomial or pseudo-polynomial time in the problem size. A Lagrangian relaxation of (A.1) is obtained by bringing the term $\lambda(Ax - b)$ into the objective function of Z', where $\lambda \in \mathbb{R}^m_+$ is the corresponding vector of Lagrange multipliers. The resulting *Lagrangian relaxation Problem* (LRP) is

$$Z(\lambda) = \min \{cx + \lambda(Ax - b) :\ x \in X\} = \min \{(c + \lambda A)x - \lambda b :\ x \in X\}. \tag{A.2}$$

It is a known fact that $Z(\lambda) \leq Z$ and, therefore, the tightest possible lower bound on Z, attainable through $LRP(\lambda)$, is given by an optimal solution to the *Lagrangian dual problem* (LDP) $Z_D = \max_{\lambda \in \mathbb{R}^m_+} \{\min \{(c + \lambda A)x - \lambda b :\ x \in X\}\}$. Several methods exist to compute the LDP. Among these, due to its simplicity and the acceptable results it returns, the subgradient method (SM) is the most widely used. Clearly, if there are exponentially many inequalities in (A.1), the use of traditional Lagrangian relaxation becomes impracticable. Alternatively, the R&C scheme proposes a dynamic strategy dualize inequalities. In the literature two strategies to implement R&C algorithms are discussed. They differ, basically, on the moment at which the new inequalities are dualized. In our implementation, we adopt the Non Delayed Relax-and-Cut strategy (see [7]). Thus, one identify inequalities that are violated at $\bar{x}^k$, an optimal solution to (A.2) at iteration k of SM. To do so, likewise polyhedral cutting-plane generation, a separation problem must be solved at every iteration of SM. Thus, one tries to find at least one inequality violated by the current LRP solution. The inequalities thus identified are candidates to be dualized. It is worth noting that separation problems arising in R&C algorithms may be easier than their polyhedral cutting-plane algorithm counterparts. That applies since LRP normally has integral valued solutions (cf. [8]).

A.3 A Relax-and-Cut to the Set Partitioning Problem

SPP is a NP-hard problem generically formulated as below, where A is a $(0,1)$ $m \times n$ matrix, c is a n-dimensional vector and e_m is the m-dimensional vector with all entries equal to one.

$$Z_{SPP} = \min \{cx : Ax = e_m,\ x \in \mathbb{B}^n\}, \tag{A.3}$$

Two other NP-hard problems are closely related to SPP: the Set Packing Problem (SSP) and the Set Covering Problem (SCP).Thus, classes of valid inequalities for SPP are typically derived from SSP and SCP [2, 5, 11]. Among these, clique inequalities are often cited as the most practically relevant to oslve SPPs exactly. To study the facial structure of the SPP polytope we associate an intersection graph, $G_A = (N, E)$, to the columns of the SPP input matrix A where (see [2, 5] for details): *(1)* for $j \in N = \{1, 2, \ldots, n\}$, the node j is associated to the column a^j of A, and, *(2)* there is an edge $(i, j) \in E$, for $i, j \in N$ and $i \neq j$, if the columns a^i and a^j of A are nonorthogonal.

Every feasible solution to SPP corresponds to a subset $S \subseteq N$ of nodes, where each feasible set S is an independent set of G_A. Also, let $V \subseteq N$ be a set in G_A such that V induces a complete subgraph in G_A. Then, every S meets V in at most one element. Thus, as each node of N represents a variable in SSP, every solution to the SPP satisfies the inequality $\sum_{v \in V} x_v \leq 1$. That inequality defines a facet of SSP if and only if V is a maximal clique of G_A [2, 5, 10, 11]. However, the number of maximal cliques grows exponentially with the graph size. Additionally, finding the most violated such inequalities is hard. Thus, a heuristic separation routine was implemented that, at every SM iteration, looks for at least one maximal clique not satisfied by the current solution $\bar{x}^k$ of LRP. Our heuristic is a simple modification of the greedy algorithm in [9] where nodes are selected in the decreasing order of their degrees. The intersection graph $G_A = (N', E)$ passed as parameter to the heuristic is build up with only the subset $N' = \{j \in N : \bar{x}^k_j = 1\}$.

Lagrangian Relaxations. Re-writing (A.3) in summation notation and dualizing all equality constraints in a Lagrangian fashion we obtain

$$\text{LRP}(\lambda) \ = \ \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i : x_j \in \{0,1\},\ j = 1, \ldots, n\} \tag{A.4}$$

where $\bar{c}_j$ is the Lagrangian cost of x_j given by $\bar{c}_j = (c_j - \sum_{i=1}^{m} \lambda_i a_{ij})$, $\lambda \in \mathbb{R}^m$ is the Lagrangian multiplier vector attached to the constraint set dualized, $a_{ij} \in \{0,1\}$ corresponds to the coefficient at position ij in $A^{m \times n}$ and c_j is the cost associated to the binary variable x_j. It is straightforward to verify that (A.4) may be solved by simple inspection in linear time on n. However, it is possible to add to (A.4) redundant constraints in the

original SPP formulation (A.1), which may strengthen the lower bound LRP(λ) obtained. We did that by introducing constraint $\sum_{j=1}^{n} x_j \leq m$ to (A.4), yielding

$$\text{LRP}(\lambda) = \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i : \sum_{j=1}^{n} x_j \leq m,\ x \in \{0,1\}^n\} \tag{A.5}$$

Notice that, sorting the Lagrangian costs, this LRP can be solved by inspection and has complexity $O(n \log n)$. Another possible Lagrangian relaxation is computationally more expensive than (A.5), but can produce better dual bounds. It adds the knapsack constraint $\sum_{j=1}^{n} w_j x_j = m$, where w_j denotes the number of rows that can be covered by variable x_j. The resulting LRP produced is

$$\text{LRP}(\lambda) = \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i : \sum_{j=1}^{n} w_j x_j = m, x \in \{0,1\}^n\} \tag{A.6}$$

This redundant constraint is valid to any feasible solution of (A.1) . In fact, the lower bound yield by solving the LDP may be even better than the one achieved by the linear relaxation of (A.1) [3, 10].

Adding Clique inequalities in a Relax-and-Cut framework. Our separation routine is called at every SM iteration. The original SPP and all the relaxations described above are modified in order to incorporate the clique inequalities identified. These inequalities can be written as $\sum_{j=1}^{n} d_{kj} x_j \leq 1,\ k = 1, \ldots, |K|$, where K is the set of cliques considered at the current SM iteration and the d_{kj} are the binary coefficients in the clique inequality matrix $D^{|K| \times n}$. Indeed, relaxation (A.5) can be modified to the expression below, where $\beta \in R_{+}^{|K|}$ is the vector of multipliers associated to the clique inequalities and $\bar{c}_j = (c_j - \sum_{i=1}^{m} \lambda_i a_{ij} + \sum_{k=1}^{|K|} \beta_k d_{kj})$, with $j = 1, \ldots, n$.

$$\text{LRP}_{\leq m}^{\lambda,\beta} = \min \{\sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \lambda_i - \sum_{k=1}^{|K|} \beta_k : \sum_{j=1}^{n} x_j \leq m,\ x \in \{0,1\}^n\} \tag{A.7}$$

Cut pool management. It is necessary to establish a policy to manage the pool of valid inequalities yielded by the separation routine. With this aim, dominance relationships (see [10] for definitions) between inequalities were taken into account. New valid inequalities are inserted if the following conditions are met: `(a)` they are not identical to any cut in the pool neither to any inequality from the original model and `(b)` they are not dominated by any cut in the pool neither to any inequality from the original model. Inequalities are deleted only if they are dominated by any new valid inequality identified and provided they do not belong to the original model.

No reference to pool management in R&C context seems to be made before this work. However, some comparative tests showing the pool management benefits obtained are described in section A.4.

Getting upper bounds to the SPP. A Lagrangian heuristic was developed to construct feasible solutions. It is a simple greedy algorithm that sorts the variables according to the LRP current solution and, at each iteration, select/set a variable to 1 based on that sorting. Notice that the algorithm can fail (no backtracking is allowed). In this case, no feasible solution is generated.

A.4 Computational Results

The main goal in this experimentation is to determine whether or not the use of cutting planes in combination with Lagrangian Relaxation pays off. The tests were performed on a AMD Athlon XP 2600+ with 512 MB of RAM. The code was written in `C++`. LEDA 4.4.1 also used to help with graph structures manipulation.

Test Sets. We worked on two test sets. The first, **I1**, is made of real-world instances available at `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`. They originate from the airline crew scheduling problem [5]. Among them, we select all those having integrality gaps, but the instance `us01`, with 1053137 columns, that was not tested due to memory limitations. The second dataset, **I2**, is composed of SCP instances cited in [4] as being hard ones to solve. To each SCP instance we concatenated an identity matrix to guarantee the existence of feasible SPP solution. The original instances have unitary costs and we decided to do the same for the additional columns.

Parameters. The following settings were used for the basic parameters of the subgradient algorithm: *(1)* α^k, the constant value used to multiply the *step size* at every SM k is initially set to 2 and multiplied by 0.75 every 100 consecutive iterations without improvement on the lower bound; *(2)* the primal heuristic is called at every SM iteration. In the first iteration, if the heuristic fails, it is called again with V sorted in nondecreasing order; *(3)* the algorithm stops when the limit of 4000 SM iterations is reached or when $\alpha^k \leq 0.00001$, what occurs first.

Results. Table A.1 shows the results obtained by the LRs ((A.5) and (A.6)) and the R&C described by (A.7). Uupper and lower bounds (rounded up) and the time required to obtain them are reported. The first three columns show the number of rows, columns and density, respectively, of each instance.

The optimal integer solution values and optimal LP values reported are known for all instances in set **I1**. For the four smaller instances in **I2**, the optimal values reported were obtained by the solver XPRESS Optimizer 14.27. Experiments with the two largest instances were discarded because no optimal LP solution was found after one hour of

execution of the XPRESS solver.

The results in table A.1 reveal that the LR having the extra knapsack constraint (A.6), was able to close the gaps of only five instances. However, it had worse performance than LR (A.5) for several instances. Also, the LRP solutions produced by (A.6) have, typically, few variables with value one. As a result, the use of cutting planes together with a knapsack constraint was discarded. This is because the intersection graph associated to a solution of the LRP would have small cliques thus leading to weak inequalities. Considering the columns relative to R&C, we see that 20 instances were solved. For 38 out of the 42 instances, the lower bounds obtained by R&C are at least as good as those from the others algorithms. and, for 37 cases they are strictly better than the LP bounds.

An contribution of this paper to R&C algorithms, refers to the management of the cut pool. In cutting-plane algorithms for ILP, the same cut is never identified twice and the usage of dominated cuts is avoided. In LR, in different SM iterations, LRPs may have the same solution. Since the separation routines are deterministic, the same cuts would be generated again. To our knowledge, in no early implementation of the R&C algorithm the authors pay attention to this fact.

Table A.2 compares the results obtained with and without the dominance check. The tests were ran on a subset of instances arbitrarily chosen from each subgroup. The size of the cut pool was limited to 50.000 cuts. Notice that, when the dominance check is turned on, the number of cuts in the pool at the end of the execution is drastically smaller. Also, substantial savings in running time are attained, despite the need of checking for dominance rules. The *Ratio* subcolumns give the gain relative to both these measures. Observe that often dominance check also leads to better lower bounds. It is also worth mentioning that convergence problems (see *lb* column) occur for some instances in these tests when the dominance check was off.

Finally, by examining the columns relative to the R&C variants in table 3.1, one can see that our Lagrangian heuristic is able to find a feasible solution in all cases. With this respect, our Lagrangian heuristic is more effective than the algorithm proposed in [5] who failed for several instances.

A.5 Conclusions and future works

In this paper a R&C algorithm was developed for the Set Partitioning problem. Computational results were obtained for instances from the literature that have integrality gaps. For comparison purposes, two other Lagrangian algorithms were implemented. One where all SPP original constraints are dualized (A.5) and the other amended by a knapsack constraint which is kept in the Lagrangian subproblem (A.6). Since, for the latter case, the subproblem is solved exactly via dynamic programming, in theory, the lower bound re-

Instance				Optimal	LP	LR (A.5)			LR (A.6)			R&C (A.7)		
label	m	n	$d(\%)$			ub	lb	$t(s)$	ub	lb	$t(s)$	ub	lb	$t(s)$
nw03	59	43749	14.10	24492	24447	24558	24447	172.6	34917	24445	247.3	24492	**24492**	120.8
nw06	50	6774	28.54	7810	7640	9886	7640	21.1	50670	7628	33.4	13582	7727	26.3
nw11	39	8820	16.64	116256	116254.5	116259	116255	26.2	116265	116255	35.4	116256	**116256**	4.2
nw13	51	16043	12.78	50146	50132	50166	50132	54.7	53462	50130	78.9	50146	50134	57.4
nw17	61	118607	13.96	11115	10875.7	37938	10853	517.2	64476	10839	698.9	37938	10831	536.4
nw18	124	10757	6.82	340160	338864.3	383664	338811	35.7	548540	338579	107.7	387694	338950	55.4
nw20	22	685	24.70	16812	16626	16812	16626	1.5	18558	16624	1.3	16812	**16812**	1.0
nw21	25	577	24.89	7408	7380	7436	7380	1.2	7448	7380	1.1	7408	**7408**	0.1
nw22	23	619	23.87	6984	6942	7060	6942	1.4	7158	6942	1.1	6984	**6984**	0.2
nw23	19	711	24.80	12534	12317	12534	12317	1.5	13904	12317	1.2	12534	**12534**	1.7
nw24	19	1366	33.20	6314	5843	6462	5843	3.2	6462	5922	2.3	6554	6295	4.2
nw25	20	1217	30.16	5960	5852	6596	5852	2.8	7366	5852	2.2	5960	**5960**	0.7
nw26	23	771	23.77	6796	6743	6804	6743	1.7	8180	6743	1.5	6796	**6796**	0.6
nw27	22	1355	31.52	9933	9877.5	9972	9878	3.1	9933	**9933**	1.2	9933	**9933**	0.7
nw28	18	1210	39.27	8298	8169	8298	8169	2.7	8298	**8298**	0.5	8298	**8298**	0.6
nw29	18	2540	31.04	4274	4185.9	4814	4185	6.4	4856	4190	4.8	4814	4229	7.5
nw30	26	2653	29.63	3942	3726.8	4294	3727	6.9	4294	3737	7.5	3942	**3942**	5.8
nw31	26	2662	28.86	8038	7980	8038	7980	6.8	8822	8009	7.5	8038	8027	7.9
nw32	19	294	24.29	14877	14570	14877	14570	0.6	14877	14572	0.5	14877	**14877**	0.6
nw33	23	3068	30.76	6678	6484	7746	6484	7.9	7968	6484	7.6	7968	6536	8.6
nw34	20	899	28.06	10488	10453.5	10713	10454	2.0	10488	**10488**	0.7	10488	**10488**	0.1
nw35	23	1709	26.70	7216	7206	7216	7206	4.1	7216	**7216**	1.1	7216	**7216**	0.7
nw36	20	1783	36.90	7314	7260	7502	7260	4.3	7550	7265	3.4	7514	7259	5.7
nw37	19	770	25.82	10068	9961.5	10068	9962	1.6	10068	**10068**	0.4	10068	**10068**	0.1
nw38	23	1220	32.33	5558	5553	5558	5552	2.9	8402	5552	2.4	5558	**5558**	0.5
nw39	25	677	26.55	10080	9868.5	10545	9869	1.5	10512	10056	1.4	10080	**10080**	0.7
nw40	19	404	26.95	10809	10658.3	10809	10659	0.8	12180	10666	0.7	10809	**10809**	0.2
nw41	17	197	22.10	11307	10972.5	11307	10973	0.4	12897	11084	0.3	11307	**11307**	0.1
nw42	23	1079	26.32	7656	7485	7910	7485	2.4	10826	7485	2.1	7656	7636	3.4
nw43	18	1072	25.18	8904	8897	8904	8897	2.5	11528	8897	1.8	8904	**8904**	1.0
us04	163	28016	6.52	17854	17731.7	17865	17722	106.3	17865	17732	361.1	17912	17747	119.6
aa01	823	8904	1.00	56138	55535.4	57528	55442	29.2	377241	51140	516.8	57528	55137	137.7
aa03	825	8627	1.00	49649	49616.4	50747	49606	28.2	346449	47159	500.0	50747	49581	67.4
aa04	426	7195	1.70	26402	25877.6	29888	25843	22.2	214258	25724	223.6	29888	25867	80.6
aa05	801	8308	0.99	53839	53735.9	53950	53724	26.7	370429	51504	471.8	54909	53644	83.0
aa06	646	7292	1.10	27040	26977.2	27293	26974	22.7	167944	26851	330.0	27293	26987	44.2
kl01	55	7479	13.67	1086	1084	1099	1084	22.6	1107	1084	38.8	1089	1085	31.3
kl02	71	36699	8.16	219	215.3	221	216	132.0	229	216	233.7	220	216	145.8
cyc06	240	432	1.16	112	48	120	48	1.0	164	48	6.8	120	86	35.3
cyc07	672	1120	0.45	352	112	352	112	3.2	472	112	47.3	352	225	239.5
cyc08	1792	2816	0.18	1024	256	1036	256	11.1	1318	256	312.5	1048	1024	841.8
cyc09	4608	6912	0.07	2816	576	2844	576	43.2	3404	576	2113.8	2844	2810	3596.9

Table A.1: Computational results for SPP instances.

turned by that Lagrangian algorithm may exceed the LP relaxation bound for (A.3). The same applies for our R&C algorithm. However, as our results demonstrate, the the R&C algorithm is not only faster than the Lagrangian algorithm for (A.6) but also generates better lower bounds. Also, out of the 42 instances in the test set, 20 were solved exactly by the R&C algorithm. These results were attained through an implementation in which we carefully dealt with dominance relations among the generated cuts. To our knowledge the treatment of dominance in R&C algorithms is a novelty.

In our implementation, the separation was restricted to the class of clique inequalities. Thus, a natural further implementation would be to separate other classes of valid inequalities for SPP.

Acknowledgments. Abilio Lucena was supported by grants 300149/94-8 (CNPq) and E26/71.906/00 (FAPERJ). Cid de Souza was supported by grants 307773/2004-3 (CNPq) and 03/09925-5 (FAPESP).

Instance	Dominance OFF			Dominance ON			Ratio (%)	
	cuts	t(s)	lb	cuts	t(s)	lb	cuts	time
nw24	8053	6.8	6036	533	4.2	6295	6.62	61.76
nw33	4142	11.5	6536	376	8.6	6536	8.76	74.78
nw41	6718	2.6	11298	37	0.1	11307	0.55	3.85
us04	36664	133.9	15952	3511	119.6	17747	9.58	89.32
aa06	50000	77.6	26714	1881	44.2	26987	3.76	56.96
kl01	50000	55.9	1079	2837	31.3	1085	5.67	55.99
cyc06	50000	41.4	74	375	35.3	86	0.75	85.27

Table A.2: *Relax and Cut* with and without dominance check usage.

Bibliography

[1] E. Balas and N. Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21:19–46, 1981.

[2] R. Borndörfer. *Aspects of Set Packing, Partitioning and Covering*. PhD thesis, Faculty of Mathematics at Tech.University of Berlin, 1998.

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

[4] T. Grossman and A. Wool. Computacional experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101:81–92, 1997.

[5] K.L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6), June 1993.

[6] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. In *COAL Bulletin*, volume 21. Mathematical Programming Society, 1992.

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

[8] C. Martinhon, A. Lucena, and N. Maculan. Stronger k-tree relaxations for the vehicle routing problem. *European Journal on Operational Research*, 158:56–71, 2004.

[9] G.L. Nemhauser and G. Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *J.Opl. Res. Soc.*, 43(5):443–457, 1992.

[10] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.

[11] M. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(199-215), 1973.

Appendix B

Lagrangian relaxation and cutting planes for the vertex separator problem

Este trabalho foi publicado nos anais do *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE)*, Zhejiang University, Hangzhou - China, em abril de 2007 (*Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, Volume 4614/2007, pages 471-482, ISSN 0302-9743 (versão impressa) 1611-3349 (versão digital), ISBN 978-3-540-74449-8).

In this paper we propose an algorithm for the vertex separator problem (VSP) based on Lagrangian Relaxation and on cutting planes derived from valid inequalities presented in [2]. The procedure is used as a preprocessing for the branch-and-cut (B&C) algorithm implemented for VSP in [6], aiming to generate an initial pool of cutting planes and a strong primal bound for the latter. Computational experiments show that the exact method obtained in that way outperforms the pure B&C algorithm recently introduced by de Souza and Balas in [6]. Besides, we show that the Lagrangian phase is a very effective heuristic for the VSP, often producing optimal solutions extremely fast.

B.1 Introduction

A *vertex separator* in an undirected graph is a subset of the vertices, whose removal disconnects the graph in at least two nonempty connected components. Recently, Balas and de Souza [2, 6] studied the vertex separator problem (VSP) which can formally be stated as follows.

`INSTANCE:` a connected undirected graph $G = (V, E)$, with $|V| = n$, an integer $1 \leq b \leq n$ and a cost c_i associated with each vertex $i \in V$.
`PROBLEM:` find a partition of V into disjoint sets A, B, C, with A and B nonempty, such that *(i)* E contains no edge (i, j) with $i \in A$, $j \in B$, *(ii)* $\max\{|A|, |B|\} \leq b$, *(iii)* $\sum_{j \in C} c_j$ is minimized.

The sets A and B are called the *shores* of the separator C. A separator C that satisfies (i) but violates (ii) is termed *infeasible*; one that satisfies (i) and (ii) is *feasible*; and a separator that satisfies (i), (ii), (iii) is optimal. Unless otherwise specified, the term separator is used here to denote a feasible one. The VSP is $\mathcal{NP}$-hard and has widespread applicability in network connectivity. We refer to [2] for further discussion on applications, including one in Linear Algebra.

Based on the Integer Programming (IP) model and the strong valid inequalities introduced by Balas and de Souza, we propose an algorithm that combines Lagrangian relaxation with cutting plane techniques to solve the VSP. Our method belongs to a class of Lagrangian Relaxation algorithms where constraints of certain families of inequalities may only be explicitly dualized when they become violated at some Lagrangian relaxation solution (see [5, 7, 9, 11]). These so-called Relax-and-Cut (R&C) algorithms appear as a promising alternative approach to strengthen Lagrangian relaxation bounds.

Related Work. Relax-and-Cut algorithms are presented in several recent works in literature [5, 7, 8, 9, 10, 11]. These algorithms use a dynamic inequality dualization scheme that renders viable the application of Lagrangian Relaxation to models with an exponential number of inequalities. Indeed, a similar approach for the traveling salesman problem [1] date from the early 80's.

The VSP described in this paper was first defined in [2] by Balas and de Souza were a polyhedral investigation of the VSP is conducted. They introduced several classes of strong valid inequalities for the polytope associated to the problem. The same authors reported extensive computational experiments with a branch-and-cut (B&C) algorithm based on these inequalities in [6]. A similar problem is discussed in [12] by Borndörfer *et al.* This problem can be considered a generalization of the VSP in the sense that the partitioning can be done in more than two sets of vertices. However, contrarily to the VSP, solutions where one of the shores remains empty are allowed.

Our contribution. The contribution of this paper is twofold and is related to the different usages we made of the R&C algorithm. First, we consider the performance of the B&C algorithm developed in [6] when the R&C is used as a preprocessing phase. We found out that the resulting method is a very powerful technique to tackle hard instances of VSP, outperforming the pure B&C algorithm in most of the benchmark instances. Thus, our algorithm can be viewed as the best exact method available so far for the VSP. Also, the R&C algorithm can be used as a heuristic for the VSP. We show that it

produces solutions of very high quality for the problem, often optimal ones. Besides, our Lagrangian heuristic is much faster than the Linear Programming heuristic proposed in [6].

The paper is organized as follows. Section B.2 presents the results in [2, 6] that are relevant to our work, namely, an IP formulation of VSP and a class of valid inequalities used as cutting planes in the R&C algorithm. Section B.3 briefly reviews the Lagrangian relaxation technique and the subgradient method (SM). A general description of the R&C framework is also given and the specific algorithm we developed for the VSP is described in section B.4, including details of our Lagrangian heuristic for the problem. Section B.5 reports on the computational results obtained for test instances gathered from the literature. Finally, in section B.6, we draw some conclusions and discuss possible extensions of this study.

B.2 An IP model for VSP and a class of valid inequalities

We now describe the mixed IP model presented in [2, 6] on which our Lagrangian relaxation is based. For every vertex $i \in V$, two binary variables are defined: $u_{i1} = 1$ if and only if $i \in A$ and $u_{i2} = 1$ if and only if $i \in B$. For $S \subseteq V$ and $k \in \{1, 2\}$, let $u_k(S)$ denote $\sum(u_{ik} : i \in S)$, and $u(S) = u_1(S) + u_2(S)$. An IP formulation for the VSP is given by

$$\max \sum_{i \in V} c_i(u_{i1} + u_{i2})$$

$$u_{i1} + u_{i2} \leq 1, \qquad \forall\, i \in V \tag{B.1}$$

$$u_{i1} + u_{j2} \leq 1,\ u_{j1} + u_{i2} \leq 1, \qquad \forall\, (i, j) \in E \tag{B.2}$$

$$u_1(V) \geq 1, \tag{B.3}$$

$$u_2(V) \leq b, \tag{B.4}$$

$$u_1(V) - u_2(V) \leq 0, \tag{B.5}$$

$$u_{i2} \geq 0,\ u_{i1} \in \{0, 1\}, \qquad \forall\, i \in V. \tag{B.6}$$

Inequality (B.1) forces every vertex to belong to at most one shore. Inequalities (B.2) prohibits the extremities of an edge to be on distinct shores. Inequalities (B.3) to (B.5) limit the size of the shores and, at the same time, reduce the symmetry of the model by forcing the size of shore A to be bounded by that of shore B. As observed in [6], if the u_{i1} variables are integer for all $i \in V$, the integrality of the u_2 variables can be dropped from

the formulation. Though this observation is not taken into account by our Lagrangian relaxation, it is relevant for IP solvers.

Balas and de Souza [2] call a valid inequality for VSP *symmetric* if, for all $j \in V$, the coefficients of the variables u_{j1} and u_{j2} in the inequality are the same. Besides, they show that vertex separators are intimately related to vertex dominators. A vertex dominator is a subset of vertices of the graph such that all the remaining vertices are adjacent to at least one of them. The dominator is said to be connected if the subgraph induced by its vertices is connected. Balas and de Souza then stated the following property: every separator and every connected dominator have at least one vertex in common. From this observation, they derived a class of symmetric inequalities associated with connected dominators, the so-called `CD` inequalities. If $S \subset V$ is a connected dominator, the `CD` inequality for S is given by

$$u(S) \leq |S| - 1. \tag{B.7}$$

Inequality (B.7) is clearly valid for the VSP polytope P, where $P = \text{conv}\{u \in \{0,1\}^{2|V|} : u$ satisfies (B.1)–(B.6)$\}$. It is non dominated only if S is *minimal* (with respect to vertex removal). However, necessary and sufficient conditions for `CD` inequalities to define facets are not known in general. But, in [6], they are shown to be very effective in computations.

According to [6], for unit costs, one can adapt the separation routine to search for more stringent `CD` inequalities. These inequalities are valid for all vectors $u \in P$ satisfying $u(V) \geq z_{LB} + 1$, but chop off several feasible solutions with smaller costs. Their usage preserves optimality and is conditioned to the existence of a lower bound z_{LB}. We call them *conditional cuts*, in an analogy to what is done for the set covering problem in [3]. For the VSP, these cuts are obtained computing $\alpha = max\{z_{LB} - b + 1, 1\}$ and searching minimal dominators that cover at least $k = |V| - \alpha + 1$ vertices (k-dominators). Thus, given a lower bound z_{LB} for the optimum, the separation routine can be changed to identify minimal connected dominators that *cover* at least k vertices. Conditional cuts are used both in the B&C algorithm in [6] and in our implementation.

B.3 Relax-and-Cut (R&C) algorithms

We now review the basics on Lagrangian relaxation and relax-and-cut algorithms that are relevant to us. Denote by X a subset of $\mathbb{B}^n = \{0,1\}^n$ and let

$$Z = \max \{cx : Ax \leq b,\ x \in X\} \tag{B.8}$$

be a formulation for a $\mathcal{NP}$-hard combinatorial optimization problem. In association with (B.8) one has $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$, where m and n are positive integral values representing, respectively, the number of constraints and the number of variables involved.

Assuming that m is an exponential function of n, let Z' denote the formulation obtained after removing constraints $Ax \leq b$ from (B.8). Also, assume that Z' can be solved in polynomial or pseudo-polynomial time in the problem size.

A Lagrangian relaxation of (B.8) is obtained by bringing the term $\lambda(b - Ax)$ into the objective function of Z', where $\lambda \in \mathbb{R}^m_+$ is the corresponding vector of Lagrange multipliers. The resulting *Lagrangian relaxation Problem* (LRP) is

$$Z(\lambda) = \max \{cx + \lambda(b - Ax) : \ x \in X\} = \max \{(c - \lambda A)x + \lambda b : \ x \in X\}. \tag{B.9}$$

It is a known fact that $Z(\lambda) \geq Z$ and, therefore, the tightest possible upper bound on Z, attainable through $LRP(\lambda)$, is given by an optimal solution to the *Lagrangian dual problem* (LDP) $Z_D = \min_{\lambda \in \mathbb{R}^m_+} \{\max \{(c - \lambda A)x + \lambda b : \ x \in X\}\}$. In the literature, several methods exist to compute the LDP. Among these, due to its simplicity and the acceptable results it returns, the subgradient method (SM) is the most widely used [4]. A brief review of that method follows since the R&C algorithm we suggest here for the VSP is deeply based on SM.

SM is an iterative algorithm which solves a succession of LRPs like the one in (B.9). It starts with a feasible vector λ^0 of Lagrangian multipliers and, at iteration k, generates a new feasible vector λ^k of multipliers and an associated LRP.

At iteration k, let $\bar{x}^k$ be an optimal solution to (B.9) with cost $Z(\lambda^k)$ and let Z^k_{LB} be a known lower bound on (B.8). An associated subgradient vector (for the m relaxed constraints) is then computed as $g^k_i = (b_i - a_i\bar{x}^k), \ i = 1, 2, \ldots, m$. That vector is then used to update λ^k. To that order, a *step size* θ^k must be computed first. Typically, for a real valued parameter $\pi^k \in (0, 2]$, formula

$$\theta^k = \frac{\pi^k(Z(\lambda^k) - Z^k_{LB})}{\sum_{i=1}^m (g^k_i)^2} \tag{B.10}$$

is used [4]. Finally, once θ^k is obtained, λ^k is updated as

$$\lambda^{k+1}_i = \max \{0; \lambda^k_i - \theta^k g^k_i\}, \ i = 1, 2, \ldots, m. \tag{B.11}$$

Notice that the straightforward use of formulas (B.10-B.11) may become troublesome when a *huge* number of dualized inequalities exist. An alternative is then to modify SM according to the R&C scheme discussed below.

In the literature two strategies to implement R&C algorithms are discussed. They differ, basically, on the moment at which the new inequalities are dualized. In a Delayed Relax-and-Cut (DRC), several executions of SM are made and the cuts found during one such execution are added only at the beginning of the next one. In a Non Delayed Relax-and-Cut (NDRC), typically a single SM run is done and cuts are dualized along the

iterations as they are found. See [5, 7, 9, 10, 11] for details. In a comparison carried out in [10], NDRC performed better than DRC. Therefore, in our implementation, we adopt the NDRC strategy.

Clearly, if there are exponentially many inequalities in (B.8), the use of traditional Lagrangian relaxation becomes impracticable. Alternatively the R&C scheme proposes a dynamic strategy to dualize inequalities. In this process, one should be able to identify inequalities that are violated at $\bar{x}^k$. To do so, likewise polyhedral cutting-plane generation, a separation problem must be solved at every iteration of SM. Thus, one tries to find at least one inequality violated by the current LRP solution. The inequalities thus identified are candidates to be dualized. It is worth noting that separation problems arising in R&C algorithms may be easier than their polyhedral cutting-plane algorithm counterparts. That applies since LRP normally has integral valued solutions (cf. [11]).

B.4 A relax-and-cut algorithm for the VSP

Different Lagrangian relaxations can be devised from the formulation given in Section B.2. We decided to start with a simple one in which the constraint sets (B.1) and (B.2) are dualized by means of the vector multipliers $\lambda \in \mathbb{R}_+^{|V|}$, $\beta^1 \in \mathbb{R}_+^{|E|}$ and $\beta^2 \in \mathbb{R}_+^{|E|}$, respectively. The resulting LRP is given by

$$\begin{aligned} \mathrm{LRP}(\lambda, \beta^1, \beta^2) = \max \{ & \sum_{i \in V} (\bar{c}_{i1} u_{i1} + \bar{c}_{i2} u_{i2} + \lambda_i) + \sum_{\substack{(i,j) \in E \\ i<j}} (\beta^1_{i,j} + \beta^2_{i,j}) : \\ & u_{kl},\ k \in V,\ l = 1,2 \text{ satisfy (B.3)–(B.6)} \} \end{aligned} \tag{B.12}$$

where $\bar{c}_{k1} = c_k - \lambda_k - \sum_{\substack{(k,j) \in E \\ k<j}} \beta^1_{k,j} - \sum_{\substack{(i,k) \in E \\ i<k}} \beta^2_{i,k}$ and $\bar{c}_{k2} = c_k - \lambda_k - \sum_{\substack{(i,k) \in E \\ i<k}} \beta^1_{i,k} - \sum_{\substack{(k,j) \in E \\ k<j}} \beta^2_{k,j}$ are the Lagrangian costs of, respectively, u_{k1} and u_{k2}. Notice that (B.12) can be solved in $O(|V| \log |V|)$ time by sorting the variables according to their Lagrangian costs and after performing a few simple calculations.

On the other hand, the computation of good primal bounds is important for the computation of the step size (B.10) in the SM and to assess the duality gap along the iterations of the algorithm. To compute lower bounds for the VSP, we devise a simple greedy heuristic. Initially, the set L containing the vertices that are candidates to be part of the shores is built. This excludes the universal vertices (those which are adjacent to all other vertices) since, obviously, they must be in all separators. The heuristic chooses arbitrarily two nonadjacent vertices of L and assigns them to different shores so that, in the end, they will not be empty. It proceeds by assigning vertices to shores, prioritizing the assignments corresponding to the variables with higher weighted Lagrangian costs. The weighting method, chosen after some experimentation, is implemented by multiplying

the original cost of the variable associated to a vertex v by the degree of v ($\delta(v)$), as seen in step 5. All the assignments are made so as to maintain the compatibility and to respect the maximum size of the shores. As a final step, a local search subroutine is ran in an attempt to improve on the solution produced by the heuristic. These steps are summarized below.

Lagrangian heuristic $(G = (V, E), \overline{c})$
1. $L \leftarrow V \backslash \{$universal vertices in $G\}$;
2. $v \leftarrow \{$any vertex in $L\}$;
3. Initialize shore A: $A \leftarrow \{v\}$ and $L \leftarrow L \backslash \{v\}$;
4. Initialize shore B: $B \leftarrow \{$first vertex in $L \backslash \mathrm{Adj}(v)\}$ and $L \leftarrow L \backslash B$;
5. `for` $k = 1, 2$ `do`:
   `for` all $i \in L$, compute $w_{u_{ik}} \leftarrow \overline{c}(u_{ik}) * \delta(i)$;
   Let S_k be the list of variables u_{ik} sorted nonincreasingly by $w_{u_{ik}}$;
6. `while` $|A| < b$ `or` $|B| < b$ `do`
   $f_1 \leftarrow \{$vertex corresponding to the first variable in $S_1\}$;
   $f_2 \leftarrow \{$vertex corresponding to the first variable in $S_2\}$;
   `if` $\overline{c}(u_{f_1,1}) > \overline{c}(u_{f_2,2})$ `then`
     $A \leftarrow A \cup \{f_1\}$; $S_1 \leftarrow S_1 \backslash \{u_{f_1,1}\}$;
     `for` all $j \in \mathrm{Adj}(f_1)$ `do` $S_2 \leftarrow S_2 \backslash \{u_{j,2}\}$;
   `else`
     $B \leftarrow B \cup \{f_2\}$; $S_2 \leftarrow S_2 \backslash \{u_{f_2,2}\}$;
     `for` all $j \in \mathrm{Adj}(f_2)$ `do` $S_1 \leftarrow S_1 \backslash \{u_{j,1}\}$;
   `if` $|A| = b$, $\overline{c}(u_{f_1,1}) \leftarrow -\infty$; /* avoids new vertices in A */
   `if` $|B| = b$, $\overline{c}(u_{f_2,2}) \leftarrow -\infty$; /* avoids new vertices in B */
7. Compute the separator: $C \leftarrow V \backslash \{A \cup B\}$
8. `Local Search`(G, A, B, C, c)
9. `return` (A, B, C)

We now turn to the improvement heuristic. The local search starts by enlarging the separator with as many vertices of the shores belonging to its adjacency as possible (steps 1–5). Then vertices are transferred from the separator back to the shores in step 6 in an arbitrary order. However, the choice of the destination shore is made so as to increase the chances of future moves from the separator to the shores. This is evaluated via the simple computations in steps 6.i to 6.m. The overall complexity of the Lagrangian heuristic, including the local search procedure, is $O(|V| \log |V| + |E|)$.

Local Search (G, A, B, C, c) *(initializations)*
1. Let A_C be the vertices in A that have neighbors in C;
2. Let B_C be the vertices in B that have neighbors in C;
3. `if` $A = A_C$ `then` $A_C \leftarrow A_C \backslash \{$arbitrarily chosen vertex of A$\}$;
4. `if` $B = B_C$ `then` $B_C \leftarrow B_C \backslash \{$arbitrarily chosen vertex of B$\}$;
5. $A' \leftarrow A \backslash A_C$; $B' \leftarrow B \backslash B_C$; $C' \leftarrow C \cup A_C \cup B_C$;

Local Search (G, A, B, C, c) *(main loop)*
6. `for` every vertex $v \in C'$ `do`:
6.a `if` $|A'| = b$ and $|B'| = b$ `then break`;
6.b `if` $|\text{Adj}(v) \cap A'| \neq \emptyset$ and $|\text{Adj}(v) \cap B'| \neq \emptyset$, `then continue`;
6.c $C' \leftarrow C' \backslash \{v\}$;
6.d `if` $\text{Adj}(v) \subset C'$ `then`
6.e `if` $|A'| = b$ `then` $B' \leftarrow B' \cup \{v\}$;
6.f `else`
6.g `if` $|B'| = b$ `then` $A' \leftarrow A' \cup \{v\}$;
6.h `else`
6.i $n_A \leftarrow 0; \quad n_B \leftarrow 0;$
6.j `for` all $w \in \text{Adj}(v)$ `do`
6.k $n_A \leftarrow n_A + |\text{Adj}(w) \cap A|; \quad n_B \leftarrow n_B + |\text{Adj}(w) \cap B|;$
6.l `if` $n_A > n_B$ `then` $A' \leftarrow A' \cup \{v\}$;
6.m `else` $B' \leftarrow B' \cup \{v\}$;
6.n `else`
6.o `if` $\text{Adj}(v) \subset A' \cup C'$ and $|A'| < b$ `then` $A' \leftarrow A' \cup \{v\}$;
6.p `else` /* $\text{Adj}(v) \subset B' \cup C'$ */
6.q `if` $|B'| < b$ `then` $B' \leftarrow B' \cup \{v\}$;
7. `if` $\sum_{i \in C} c_i > \sum_{i \in C'} c_i$ `then` $A \leftarrow A'$, $B \leftarrow B'$, $C \leftarrow C'$.

Another ingredient of our R&C algorithm is a fast separation routine that looks for violated CD inequalities at $\bar{u} = (\bar{u}_1, \bar{u}_2)$, the optimal solution of the current LRP in (B.12). A high level description of our procedure is given below where, for any $S \subset V$, $\text{Adj}(S)$ refers to the set of all vertices in $V \backslash S$ which are adjacent to at least one vertex in S. The routine starts by constructing the subgraph $G_{\bar{u}} = (W, F)$ of the input graph $G = (V, E)$ which is induced by the vertices $i \in V$ with $\bar{u}_{i1} + \bar{u}_{i2} \geq 1$. It is easy to see that, if W is a dominator and $G_{\bar{u}}$ is connected then the CD inequality associated to W is violated by $\bar{u}$. Unfortunately, the converse is not true in general but holds when constraints (B.1) are satisfied. Thus, our routine can be viewed as a heuristic. Step 5 of the algorithm tries to strengthen the inequality since the minimality of the dominator is a necessary condition for a CD inequality to be facet defining. It checks if the removal of a limited number of vertices preserves the connectedness of the graph induced by W and the dominance property. Clearly, in this way, the separation routine can be implemented to have $O(|V| + |E|)$ time complexity.

CD-Separation(G)
1. Construct $G_{\bar{u}} = (W, F)$;
2. Determine n_{CC}, the number of connected components of $G_{\bar{u}}$;
3. `if` $n_{CC} = 1$ `then` /* $G_{\bar{u}}$ is connected */
4. `if` $V \subseteq (W \cup \mathrm{Adj}(W))$ `then` /* W is a dominator of V */
5. Turn W into a minimal `CD`;
6. `return` the `CD` inequality $u(W) \leq |W| - 1$;
7. `else return FAIL`; /* no new cut is returned for dualization */

In our R&C algorithm the separation procedure is called at each SM iteration. Since we implemented two greedy ways to obtain minimal `CD` inequalities, at most two cuts are produced at each SM iteration. Every new cut separated is stored in a pool and dualized in a Lagrangian fashion. The relaxation in (B.12) is then modified to incorporate this constraint. As a result, the term $\sum_{k=1}^{|pool|} \mu_k(|S_k| - 1 - u(S_k))$ is added to the cost function of (B.12), where $\mu \in \mathbb{R}_+^{|pool|}$ is the vector of multipliers of the `CD` inequalities that are currently dualized. The management of the cut pool is quite simple: a new inequality is inserted only if it is not identical to another inequality already in the pool or in the original formulation.

When the R&C algorithm stops, the conditional cut generator (CCG) is executed. This procedure tries to transform every `CD` inequality in the pool into a more restrictive conditional cut, according to the discussion in Section B.2. For that, z_{LB}, the best lower bound at the end of the R&C algorithm, is used.

In our implementation, the R&C algorithm and CCG form a preprocessing phase for the B&C described in [6]. The output from this phase is a set of, possibly conditional, `CD` cuts which are added *a priori* to the model given as input of B&C. Moreover, B&C is also given the values of z_{LB}, which may help to prune the enumeration earlier, and of $\alpha = z_{LB} - b + 1$ which, according to Section B.2, is applied to limit the size of the smallest shore in equation (B.3). We denote by R&C&B&C, the algorithm resulting from this combination of R&C and B&C. Figure B.1 depicts the flow diagram of R&C&B&C.

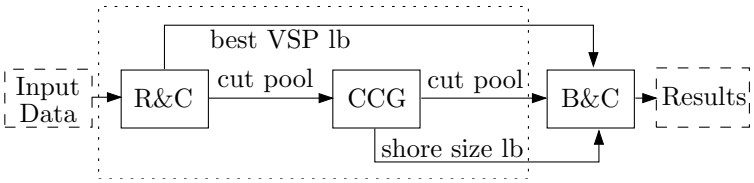

Figure B.1: Flow Diagram of R&C&B&C.

B.5 Computational Results

In this section we report the computational tests carried out with the R&C algorithm implemented for the VSP. Tests were ran on a Pentium IV machine having 1GB of RAM and operating under Linux OS. The codes are written in `C` and `C++`, using resources of the Standard Template Library. XPRESS Optimizer 16.01.05 was used as the IP solver.

Our experiments were made on a subset of instances taken from [6]. Initially, from the more than 140 instances, we select those whose graph densities were higher than 40%. Among those, we kept only the ones which required more than a minute of CPU time to

be solved by the branch-and-bound (B&B) algorithm of XPRESS (default configuration). We end up with 31 instances for our tests, all of which, with cost vector equals to the sum vector. Instances from [6] can be downloaded from `www.ic.unicamp.br/~cid/Problem-instances/VSP.html`.

The motivation to select only high density graphs comes from : `(a)` these are the most difficult cases for standard B&B algorithms; `(b)` experimental results reported in [6] revealed some difficulties of the LP solver when high density cuts are added to the problem matrix, resulting on a severe degradation of the performance of cutting-plane algorithms and `(c)` in high density graphs, connected dominators tend to be small and, therefore, the corresponding `CD` inequalities are of low density. Thus, the use of cutting-plane algorithms is justifiable and likely to be more effective for high density graphs.

The following settings were used for the basic parameters of the subgradient algorithm: `(a)` in equation (B.10), π is initially set to 2 and multiplied by 0.5 each 100 consecutive iterations without improvement on the upper bound; `(b)` the Lagrangian heuristic is called at each SM iteration; `(c)` the local search heuristic is called only when the cost of the current solution differs from those of the ones previously found. Cost repetition is easily identified in our case since there are only $O(|V|)$ possible values for the cost function; `(d)` the algorithm stops when the limit of 2000 SM iterations is reached or when, $\pi^k \leq 0.00001$ in equation (B.10), what occurs first.

Computational experiments were performed aiming to assess three aspects of our approach: `(i)` the quality of the dual bounds produced by our R&C algorithm; `(ii)` the quality of the primal bounds produced by our Lagrangian heuristic; `(iii)` the effectiveness of using our approach as a preprocessing tool that provide an IP solver with a tighter formulation.

The latter aspect deserves some discussion. First, it should be noticed that the IP model from section B.2 is rather weak. In the linear relaxation, setting all variables to 1/2 satisfies all the constraints for any sufficiently large value of b (which is the case for all instances in our dataset). This gives the worst dual bound one could come up with: n ! Thus, poor dual bounds are expected unless strong cuts are added to the formulation. Results reported in [6] show that `CD` inequalities fulfill this requirement. However, a drawback to use such inequalities comes from the fact that the corresponding separation problem is $\mathcal{NP}$-hard in general. The authors had then to resort to a heuristic procedure to perform the task. Their heuristic is of quadratic-time complexity, therefore, more expensive than our linear-time complexity heuristic used to separate integral points.

Moreover, inspecting the behavior of the B&C algorithm developed in [6], which we had access to, we noticed that a couple of `CD` inequalities needed to be separated and added to the model before good dual bounds are computed. Thus, it would be very helpful if we could quickly generate a set of initial `CD` cuts. That is precisely the role to be

played by the R&C algorithm in the preprocessing phase. Besides this, the lower bound from the Lagrangian heuristic can be used as an incumbent to accelerate the pruning of the search tree.

Four algorithms were tested, namely: our R&C, the standard B&B from XPRESS, the B&C from [6] and the hybrid method, denoted by R&C&B&C, that uses R&C as a preprocessing for the B&C algorithm. In R&C&B&C, the B&C algorithm starts with the IP model strengthened by: `(a)` the `CD` cuts and `(b)` the incumbent solution returned by the preprocessing phase and `(c)` a constraint stating that universal nodes in the input graph must belong to the separator. The results of our experiments are compiled in table B.1. Five groups of columns are present in the table. The first describes the instance characteristics: name, density (d), the number of vertices (n) and edges (m) and the optimum (`Opt`). The four remaining groups give results of the different algorithms: `ub` (`lb`) for upper (lower) bounds, `t` for the computation time and `tb` for the time needed to find the best primal bound. Column head `Pool` for R&C gives the number of violated cuts added to the model before the execution of B&C.

We observe that, though the instances `dim.DSJC125.5` and `miplib.misc07` were initially selected to be part of our dataset, their results are not shown in table B.1. This exclusion took place cause none of the tested algorithms computed `dim.DSJC125.5` and only R&C&B&C was able to solve `miplib.misc07` within the fixed time limit of 30 minutes for each run. Our approach needed 1658 seconds to prove optimality and the Lagrangian heuristic found its best solution with cost 115 (only one unit away from the optimum) in just 0.23 seconds !

We first analyse the dual bounds generated by R&C. If no cuts were used, the theoretical bound for the Lagrangian relaxation would be n as discussed earlier. Comparing column `ub` for R&C with n, we see that, by adding `CD` cuts, R&C is able to produce better bounds than the trivial one in 24 out of the 29 instances. This suggests that the preprocessing phase in R&C&B&C pays off as far as dual bounds are concerned. However, comparing columns `ub` for R&C and `Opt`, one sees that huge integrality gaps are still to be closed. Instance `mat.L80.fidapm02` was the only one that was solved to optimality by R&C.

On the primal side, excellent results were achieved. As seen in column `lb` of R&C, in 23 out of 29 cases our simple Lagrangian heuristic found an optimal solution. The average error of the heuristic was of only 0.94% and the maximum error was 13.0% for instance `miplib.l152lav.p`. For a better appreciation of the performance of the Lagrangian heuristic (LR-H), we compare the columns `tb(s)` from R&C and from B&C. This comparison can be visualized by inspecting the histogram in figure B.2, where we restricted ourselves to the cases for which LR-H found the optimum. This histogram reveals that LR-H finds optimal solutions much quicker than the LP based heuristic (LP-

H). Besides, it shows that in 82% of the cases, the optimum was found in less than 0.01 seconds and, for all instances, LR-H reached the optimum in less than 0.5 seconds.

Instance					R&C					B&B		B&C			R&C&B&C	
name	d (%)	n	m	Opt	ub	lb	tb(s)	t(s)	Pool	t(s)	node	t(s)	node	tb(s)	t(s)	node
dim.miles1000	0.40	128	6560	110	122	109	<0.01	3.09	830	85.48	467	**15.51**	9	9.84	17.98	9
dim.DSJC125.9	0.90	125	14047	22	63	**22**	<0.01	4.4	1116	> 1800.00	72631	**890.96**	29809	13.85	1017.17	27935
mat.L125.fs_183_1	0.44	125	6909	98	135	96	0.05	2.04	16	225.19	2354	29.42	24	15.31	**27.88**	33
mat.bcsstk04	0.68	132	11968	84	91	**84**	<0.01	3.74	912	> 1800.00	17584	74.77	61	5.75	**11.23**	3
mat.arc130	0.93	130	15656	88	112	**88**	0.01	5.5	1138	**177.95**	127	225.36	83	12.77	237.41	83
mat.L120.cavity01	0.42	120	6064	99	117	**99**	0.13	3.24	345	239.83	2385	**9.15**	9	4.59	10.3	11
mat.L120.fidap021	0.43	120	6236	98	116	**98**	<0.01	2.13	349	98.07	776	12.56	15	3.22	**4.32**	11
mat.L120.rbs480a	0.46	120	6666	88	105	**88**	0.04	2.34	505	1193.02	14220	186.34	343	44.1	**82.95**	116
mat.L100.rbs480a	0.52	100	5200	73	82	**73**	<0.01	2.3	537	261.47	4831	19.28	57	0.30	**15.07**	55
mat.L120.e05r0000	0.59	120	8474	90	107	**90**	<0.01	2.82	539	246.59	1299	8.50	9	0.49	**6.94**	5
mat.L100.wm1	0.60	100	6012	74	102	73	0.4	1.63	49	71.14	974	19.67	21	11.17	**15.21**	21
mat.L120.fidap022	0.60	120	8734	84	93	**84**	<0.01	2.95	634	1008.47	10797	**32.66**	45	0.52	81.2	133
mat.L100.wm2	0.61	100	6178	76	103	**76**	<0.01	1.69	6	188.39	1925	12.74	14	8.66	**8.52**	14
mat.L100.fidapm02	0.62	100	6280	69	73	**69**	<0.01	2.04	663	426.09	6695	5.84	7	0.35	**4.27**	7
mat.L120.fidap001	0.63	120	9084	82	93	**82**	<0.01	3.76	464	> 1800.00	30274	**30.77**	29	15.6	54.81	5
mat.L100.e05r0000	0.64	100	6406	70	83	**70**	<0.01	1.81	353	209.06	3497	6.02	9	0.33	**4.06**	3
mat.L80.fidapm02	0.65	80	4196	53	**53**	**53**	<0.01	1.21	503	131.64	11323	3.48	11	1.88	**2.36**	5
mat.L120.fidapm02	0.65	120	9372	86	97	**86**	0.1	2.96	537	723.72	6602	27.73	21	5.60	**7.19**	5
mat.L100.fidap001	0.68	100	6858	64	81	**64**	<0.01	2.62	285	1185.9	55503	20.3	63	4.29	**9.47**	35
mat.L100.fidap022	0.68	100	6818	62	78	**62**	0.01	2.37	369	965.03	36353	43.63	131	4.96	**31.23**	107
mat.L80.fidap022	0.76	80	4886	41	75	**41**	<0.01	1.81	186	277.68	21621	11.91	135	0.24	**3.63**	137
mat.L100.fidap027	0.81	100	8128	69	85	**69**	<0.01	2.55	248	83.70	655	5.90	5	3.51	**4.14**	1
mat.L100.fidap002	0.82	100	8214	66	89	**66**	<0.01	2.52	184	203.21	3381	**4.56**	3	0.88	4.82	5
mat.L120.fidap002	0.82	120	11892	68	107	**68**	0.01	3.58	225	1439.09	12803	34.29	51	4.68	**8.74**	5
mat.L120.fidap027	0.85	120	12222	83	101	**83**	0.01	3.74	414	226.79	3758	12.36	7	7.79	**9.86**	5
miplib.l152lav.p	0.40	97	3829	61	98	54	<0.01	1.58	122	639.63	39489	73.24	292	50.87	**56.76**	245
miplib.lp4l.p	0.46	85	3373	50	82	46	0.01	1.37	94	411.68	34485	43.31	358	40.98	**36.48**	332
miplib.air03.p	0.61	124	9502	75	125	74	2.46	2.87	241	496.60	6373	158.67	124	68.64	**115.75**	113
miplib.misc03.p	0.63	96	5884	52	88	**52**	0.05	2.42	802	1387.42	55610	130.18	4411	8.73	**123.16**	3227
$\sum$										> 18002.84	458792	2149.11	36156		2012.91	32666

Table B.1: Computational results for VSP instances

Finally, we analyse the behavior of R&C&B&C and compare its performance with that of B&C and B&B. In our analysis, percentages are computed relative to the number of instances tested that could be solved by at least one of the approaches, say, 30. B&B is by far the worst alternative but curiously reached the optimum much sooner than its competitors in instance `mat.arc130`. Back to the other algorithms, one sees that R&C&B&C is faster than B&C in 23 out of the 30 instances (77%). Besides, the total time spent by B&C in the 29 instances solved by both methods (see last row of table B.1) is 6.8% larger than that of R&C&B&C. As for the number of nodes explored during the enumeration, one can see that R&C&B&C visits less nodes than B&C, with a total reduction of 9.6% over these 29 instances. Overall, we can say that R&C&B&C is better than B&C code of [6], beating the latter in the majority of the tested instances.

B.6 Conclusions and future works

In this paper we developed an R&C algorithm for the VSP and tested it on a set of instances from the literature. The primal bounds produced by a simple Lagrangian heuristic

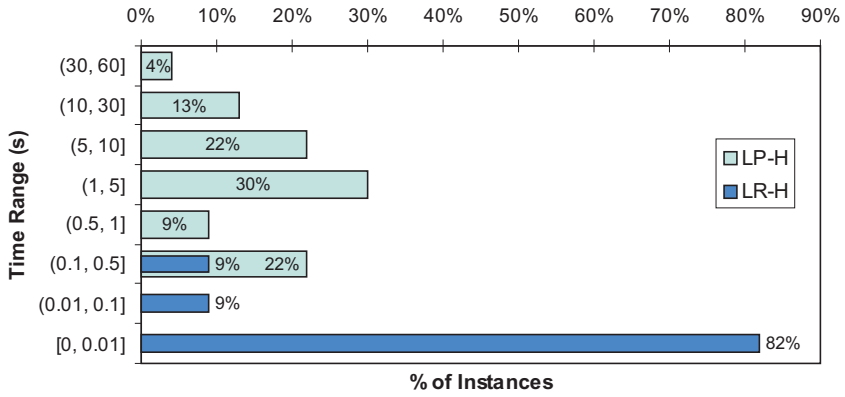

Figure B.2: Time to optimum for Lagrangian (LR-H) and LP-based (LP-H) heuristics.

proved to be very effective, rapidly reaching the optimum in many cases. Though the dual bounds were not as good, the R&C algorithm proved to be an excellent tool for preprocessing, quickly providing inequalities to strengthen the initial IP model, which allow the B&C algorithm of [6] to solve the large majority of the tested instances faster than it does when working alone. To our knowledge, this turns our R&C&B&C algorithm the best exact method available for the VSP to date. Future works include the study of different relaxations, better primal heuristics and the use of other known inequalities for the VSP polytope discussed in [2] as cutting planes in the R&C algorithm.

Bibliography

[1] E. Balas and N. Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21:19–46, 1981.

[2] E. Balas and C. C. de Souza. The vertex separator problem: a polyhedral investigation. *Mathematical Programming*, 103:583–608, 2005.

[3] E. Balas and C. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization. *Mathematical Programming Study*, 12:37–60, 1980.

[4] J.E. Beasley. *Modern Heuristic Techniques*, chapter 6. Blackwell Scientific Press, Oxford, 1993.

[5] F. Calheiros, A. Lucena, and C. C. de Souza. Optimal rectangular partitions. *Networks*, 41:51–67, 2003.

[6] C. C. de Souza and E. Balas. The vertex separator problem: algorithms and computations. *Mathematical Programming*, 103:609–631, 2005.

[7] M. Guignard. Lagrangean relaxation. In M.A. Lòpez-Cerdá and I. García-Jurado, editors, *Top*, volume 11, pages 151–200. Springer Berlin/ Heidelberg, 2003.

[8] G. Kliewer and L. Timajev. Relax-and-cut for capacitated network design. In *Proceedings of 13th Annual European Symposium on Algorithms (ESA)*, volume 3669, pages 47–58, Mallorca, Spain, October 2005.

[9] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. In *COAL Bulletin*, volume 21. Mathematical Programming Society, 1992.

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

[11] C. Martinhon, A. Lucena, and N. Maculan. Stronger k-tree relaxations for the vehicle routing problem. *European Journal on Operational Research*, 158:56–71, 2004.

[12] C.E. Ferreira R. Borndörfer and A. Martin. Decomposing matrices into blocks. *SIAM Journal on optimization*, 9:236–269, 1998.

Referências Bibliográficas

[1] A. Atamtürk, G.L. Nemhauser, and M.W.P. Savelsbergh. A combined Lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems. *Journal of Heuristics*, 1:247–259, 1996.

[2] E. Balas and N. Christofides. A restricted Lagrangean approach to the traveling salesman problem. *Mathematical Programming*, 21:19–46, 1981.

[3] E. Balas and C. C. de Souza. The vertex separator problem: a polyhedral investigation. *Mathematical Programming*, 103:583–608, 2005.

[4] E. Balas and M.W. Padberg. *Combinatorial Optimization*, chapter Set Partitioning - A Survey. John Wiley and Sons, Chichester, NY, Brisbane, Toronto, 1979.

[5] E. Balas and P. Toth. *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, chapter 10, Branch and bound methods. John Wiley and Sons, Chichester, NY, Brisbane, Toronto, 1985.

[6] M.L. Balinski and R.E. Quandt. On an integer program for a delivery problem. *Operations research*, 12:300–304, 1964.

[7] F. Barahona and R. Anbil. The volume algorithm:producing primal solutions with the subgradient method. *Mathematical Programming*, 87:385–399, 2000.

[8] J.E. Beasley. *Modern Heuristic Techniques*, chapter 6. Blackwell Scientific Press, Oxford, 1993.

[9] A. Belloni and A. Lucena. Lagrangean heuristics to linear ordering. Personal Communication, October 2001.

[10] R. Borndörfer. *Aspects of Set Packing, Partitioning and Covering*. PhD thesis, Faculty of Mathematics at Tech.University of Berlin, 1998.

[11] C.Y Byun. *Lower Bounds for Large Scale Set Partitioning Problems*. PhD thesis, Faculty of Mathematics at Tech. University of Berlin, Berlin, January 2001.

[12] F. Calheiros, A. Lucena, and C. C. de Souza. Optimal rectangular partitions. *Networks*, 41:51–67, 2003.

[13] F.C. Calheiros. Partições retangulares Ótimas: Algoritmos Lagrangeanos e planos de corte. Master's thesis, Instituto de Computação, Universidade Estadual de Campinas (UNICAMP), 2001.

[14] P.C. Chu and J.E. Beasley. Constraint handling in genetic algorithms: The set partitioning problem. *Journal of Heuristics*, 11:323–357, 1998.

[15] A.S. Cunha. *Árvores ótimas em grafos: modelos, algoritmos e aplicações*. PhD thesis, Engenharia de Sistemas e Computação, UFRJ, Rio de Janeiro, Maio 2006.

[16] C. C. de Souza and E. Balas. The vertex separator problem: algorithms and computations. *Mathematical Programming*, 103:609–631, 2005.

[17] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. *Working Paper*, 1993.

[18] L. Escudero, M. Guignard, and K. Malik. A Lagrangean relax and cut approach for the sequential ordering with precedence constraints. *Annals of Operations Research*, 50:219–237, 1994.

[19] C.E. Ferreira and Y. Wakabayashi. *Combinatória Poliédrica e Planos-de-Corte Faciais*. X Escola de Computação, Campinas, Julho 1996.

[20] M.L. Fisher. The Lagrangean relaxation method for solving integer programming problems. *Management Science*, 27, January 1981.

[21] M.R. Garey and D.S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W.H. Freeman and Co., New York, 1979.

[22] R.S. Garfinkel and G.L. Nemhauser. *Integer Programming*, volume 1972. John Wiley and Sons, NY, London, Sydney, Toronto, 1972.

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

[24] M. Guignard. Lagrangean relaxation. In M.A. Lòpez-Cerdá and I. García-Jurado, editors, *Top*, volume 11, pages 151–200. Springer Berlin/ Heidelberg, 2003.

[25] B. Guta. *Subgradient Optimization Methods in Integer Programming with an Application to a Radiation Therapy Problem*. PhD thesis, Mathematik , Universitätsbibliothek Kaiserslautern, September 2003.

[26] M. Held and R.M. Karp. The traveling salesman problems and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[27] M. Held and R.M. Karp. The traveling salesman problems and minimum spanning trees: Part ii. *Mathematical Programming*, 1:6–25, 1971.

[28] J.B. Hiriarty-Urruty and C. Lémarechal. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundel Methods*. Springer Verlag, 1993.

[29] K.L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6), June 1993.

[30] A. Klose. A Lagrangean relax-and-cut approach for the two-stage capacitated facility location problem. *European Journal of Operational Research*, 126:408–421, 2000.

[31] J.T. Linderoth, E.K. Lee, and M.W.P. Savelsbergh. A parallel, linear programming based heuristic for large scale set partitioning problems. *INFORMS J. On Computing*, 13:1–19, 2001.

[32] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. In *COAL Bulletin*, volume 21. Mathematical Programming Society, 1992.

[33] A. Lucena. Tight bounds for the steiner problem in graphs. In *Proceedings of NETFLOW93*, pages 147–154, 1993.

[34] A. Lucena. Relax and cut algorithms. In *IV ALIO-EURO Meeting*, Pucon, 2002.

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

[36] A. Lucena and J.E. Beasley. *Advances in Linear and Integer Programming*, chapter 5, Branch and Cut Algorithms. Oxford University Press, Oxford, 1996.

[37] G. Reinelt M. Jünger and S. Thienel. *Combinatorial Optimization*, chapter Practical Problem Solving with Cutting Plane Algorithms in Combinatorial. John Wiley and Sons, 1995.

[38] R.E. Marsten and F. Shepardson. Exact solution of crew scheduling problems using the set partitioning mode: Recent successful applications partitioning. *Networks*, 11:165–177, 1981.

[39] C. Martinhon, A. Lucena, and N. Maculan. A relax and cut algorithm for the vehicle routing problem. *Personal Communication*, 2000.

[40] C. Martinhon, A. Lucena, and N. Maculan. Stronger k-tree relaxations for the vehicle routing problem. *European Journal on Operational Research*, 158:56–71, 2004.

[41] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.

[42] C.E. Ferreira R. Borndörfer and A. Martin. Decomposing matrices into blocks. *SIAM Journal on optimization*, 9:236–269, 1998.

[43] C. Revelle, D. Marks, and J.C. Liebman. An analysis of private and public sector location models. *Operations Research*, 16:692–707, 1970.

[44] J.G. Root. An application of symbolic logic to a selection problem. *Operations Research*, 12:519–526, 1964.

[45] L.A. Wolsey. *Integer Programming*. John Wiley and Sons, Chichester, 1998.