ALGORITMOS EXATOS E HEURÍSTICOS PARA O PROBLEMA DA DIVERSIDADE MÁXIMA

Lucas Vinicius Amaral de Oliveira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Abílio Pereira de Lucena Filho

Rio de Janeiro
Agosto de 2017

ALGORITMOS EXATOS E HEURÍSTICOS PARA O PROBLEMA DA DIVERSIDADE MÁXIMA

Lucas Vinicius Amaral de Oliveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Abílio Pereira de Lucena Filho, D.Sc.

Prof. Luidi Gelabert Simonetti, D.Sc.

Prof. Yuri Abitbol de Menezes Frota, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2017

Oliveira, Lucas Vinicius Amaral de

Algoritmos Exatos e Heurísticos para o Problema da Diversidade Máxima/Lucas Vinicius Amaral de Oliveira. – Rio de Janeiro: UFRJ/COPPE, 2017.

X, 54 p. 29, 7cm.

Orientador: Abílio Pereira de Lucena Filho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2017.

Referências Bibliográficas: p. 51 – 54.

1. Problema da Máxima Diversidade. 2. Non-Delayed Relax-and-Cut. 3. Branch-and-Cut. I. Lucena Filho, Abílio Pereira de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedicado à minha família.

Agradecimentos

Agradeço à Deus.
Agradeço à minha família, que sempre torce por mim. Agradeço à minha esposa, Dayane, pelo carinho e grande incentivo para que eu terminasse esta pesquisa. Aos amigos que fiz na minha estadia no Rio de Janeiro, que estiveram presentes nos momentos de descontração. A Matheus, agradeço o companheirismo e a amizade em todos os momentos.
Sou grato ao meu orientador, prof. Abílio Lucena, pelos ensinamentos e comentários feitos para que eu melhorasse como pesquisador. Quero agradecer aos professores Luidi Simonetti e Yuri Menezes por aceitarem fazer parte da banca de minha dissertação. A todos os professores do PESC, agradeço pela orientação nas aulas e também pela prestatividade fora delas.
Agradeço ao professor Leandro Coelho da Université Laval, pelo acolhimento e orientação durante todo o período que passei no Canadá, e aos colegas pesquisadores que conheci nesse período.
Quero agradecer à CAPES, pela bolsa concedida para realização desta pesquisa.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ALGORITMOS EXATOS E HEURÍSTICOS PARA O PROBLEMA DA DIVERSIDADE MÁXIMA

Lucas Vinicius Amaral de Oliveira

Agosto/2017

Orientador: Abílio Pereira de Lucena Filho

Programa: Engenharia de Sistemas e Computação

Neste trabalho, propomos algoritmos Branch-and-Cut e heurísticas Lagrangeanas para o Problema da Máxima Diversidade. Utilizamos uma formulação linear padrão fortalecida por desigualdades válidas para o problema. Estas são dualizadas dinamicamente nas heurísticas e utilizadas como cortes nos algoritmos exatos. Para tanto, propomos alguns procedimentos heurísticos para separá-las de maneira eficiente. Adicionalmente, realizamos testes computacionais e comparamos nossos algoritmos exatos e heurísticos com aqueles existentes na literatura para o problema.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EXACT AND HEURISTIC METHODS FOR THE MAXIMUM DIVERSITY PROBLEM

Lucas Vinicius Amaral de Oliveira

August/2017

Advisor: Abílio Pereira de Lucena Filho

Department: Systems Engineering and Computer Science

In this work, we suggest Branch-and-Cut algorithms and Lagrangian heuristics for the Maximum Diversity Problem. We use a standard formulation reinforced by some valid inequalities for the problem. These are dynamically dualized in the heuristics and used as cuts in the exact methods. To separate them efficiently, we propose some heuristic procedures. In addition, we perform computational tests and compare our exact and heuristics methods with those found in the literature for the problem.

Sumário

Lista de Tabelas

Capítulo 1

Introdução

Muitos problemas práticos presentes nas mais diversas áreas do cotidiano(indústria, transportes, agricultura, educação e saúde, dentre outras) podem ser representados por um problema de Otimização Matemática (OM). Nesse tipo de representação, o problema original é expresso através da maximização/minimização de uma função de várias variáveis, sujeita a um conjunto de restrições. Se a representação do problema for suficientemente acurada, este é então resolvido indiretamente através da solução do problema matemático que o representa [1].

Um problema de Otimização Combinatória (OC) é um caso particular de um problema de OM, onde se exige que algumas ou mesmo todas as variáveis assumam apenas valores inteiros. Neste trabalho, investigamos um OC pouco conhecido e discutido na literatura, mas que apresenta uma vasta gama de aplicações.

Para descrevermos o problema, assuma que um conjunto de itens $N = \{1, \ldots, n\}$ é dado juntamente com uma medida da *diversidade* existente entre cada um de seus pares, $\{c_{\{i,j\}} \in \mathbb{R}^+ : i, j \in N, i \neq j\}$. O Problema da Máxima Diversidade (do inglês *Maximum Diversity Problem* (MDP)) [2–5] consiste em identificar o subconjunto $S \subset N$ de k itens, $2 \leq k \leq |N| - 1$, com a maior soma possível de diversidades $\sum_{i,j \in S, i \neq j} c_{\{i,j\}}$.

O MDP foi formalmente introduzido por KUO *et al.* [2] onde uma denominação apropriada foi dada a ele e conexões foram estabelecidas com trabalhos anteriores relacionados ao tema. Adicionalmente, [2] também mostrou que o MDP é NP-Difícil. De fato, como mostrado em [2], o Problema da Clique [6], que é comprovadamente NP-Difícil, pode ser reduzido ao MDP.

O MDP pode também ser encontrado na literatura sob outras denominações: *maxsum dispersion* [7], *MAX-AVG dispersion* [8], *edge-weighted clique* [9], *remote-clique* [10], *maximum edge-weighted subgraph* [11], e *dense k-subgraph* [12].

1.1 Aplicações

Aplicações para o MDP foram sugeridas nas mais diversas áreas do conhecimento e do cotidiano. KUO *et al.* [2] citam como exemplo a amostragem representativa de indivíduos com características diversas (país de origem, etnia, sexo, religião). Nesse caso, geralmente, deseja-se escolher um determinado número de indivíduos de um dado conjunto, que apresentem a maior diversidade de características possível entre si. Algumas universidades americanas utilizaram o MDP para promover uma maior diversidade de seu corpo estudantil [13]. Estas consideram, além do desempenho nas provas e da excelência curricular, fatores que reputam como importantes para promover diversidade. Da mesma forma, o congresso americano também chegou a analisar estratégias para incentivar a diversidade de imigrantes numa reforma das leis de imigração [14].

DUARTE e MARTÍ [15] descrevem aplicações do MDP em áreas como ecologia e engenharia genética. Na área da ecologia, numa aplicação envolvendo a preservação de sistemas ecológicos, as melhores condições de sobrevivência do sistema são alcançadas quando se maximiza a diversidade das espécies nele presentes. Na engenharia genética, alguns problemas de desenvolvimento de novas espécies podem ser formulados como um problema de MDP, onde se deseja controlar a diversidade do estoque de reprodução.

Ainda em [15], foi apresentada uma nova aplicação do MDP envolvendo algoritmos evolucionários. Nesse tipo de algoritmo, manter a diversidade genética da população de soluções é uma tarefa importante, porém, complexa. Os autores propuseram então a utilização de um algoritmo MDP para a atualização da população. Nessa atualização, um subconjunto de soluções maximalmente diversas entre si é escolhido a partir de um conjunto maior de candidatos de boa qualidade.

Uma outra aplicação para o MDP está associada à problemas de localização de estabelecimentos comerciais concorrentes [16]. Nessa abordagem, locais para a construção de um certo número desses estabelecimentos devem ser escolhidos a partir de um conjunto maior de locais candidatos. Nesse caso, o subconjunto de locais com a maior distância mútua total entre seus elementos deve ser escolhido.

1.2 Métodos de solução disponíveis

Trabalhos utilizando abordagens exatas para resolver o MDP são escassos na literatura. MARTÍ *et al.* [3] sugerem um algoritmo do tipo *Banch-and-Bound* [17]que explora uma árvore de ramificação não binária e livre de repetições. Dessa maneira conseguem resolver instâncias do problema com até 150 itens. Não encontramos outros trabalhos com enfoque exato na literatura. A existência de um único algo-

ritmo exato para resolver um problema que já vem sendo investigado na literatura há bastante tempo é um indicativo da dificuldade prática de se obter êxito nessa empreitada.

Abordagens heurísticas para resolver o MDP, por sua vez, vêm sendo investigadas desde que o problema foi originalmente proposto. As mais recentes foram testadas em instâncias com até 5 mil itens. Discutimos algumas delas a seguir, apresentando, de forma sucinta, suas principais características. PALUBECKIS [18] sugeriu uma heurística baseada em *Tabu Search* [19] denominada *Iterated Tabu Search* (ITS). O procedimento se inicia com uma solução gerada aleatoriamente, que chamaremos de solução pivô. Partindo desta solução e a cada iteração do algoritmo, o método *Tabu Search* é empregado para tentar melhorá-la. Um procedimento de perturbação é aplicado para assim se obter uma nova solução pivô. Se um critério de parada é atingido, o procedimento é concluído. Caso contrário, uma nova iteração é iniciada para a solução pivô em mãos. A melhor solução pivô obtida durante todo esse processo é armazenada e oferecida como resposta.

DUARTE e MARTÍ [15] propuseram duas heurísticas para o MDP. Uma baseada em *Tabu Search* e outra baseada no método *Greedy Randomized Adaptive Search Procedure* (GRASP) [20]. Em cada caso, os autores investigaram também algumas variantes de suas heurísticas e fizeram comparações entre os métodos que propõem e alguns outros métodos encontrados na literatura. Nestas, o método que mais se destacou foi uma das variantes da heurística *Tabu Search* proposta em [15].

GALLEGO *et al.* [21] sugerem uma heurística híbrida para o MDP. O método combina *Tabu Search* e GRASP em um ambiente de *Scatter Search* [22]. A técnica se baseia em algoritmos genéticos. Nesse tipo de algoritmo, uma população de soluções é mantida ao longo do procedimento. Novas soluções são construídas através da combinação de elementos pertencentes àquela população. Uma busca local é aplicada para melhorar tais soluções. Ao final de sua aplicação, a melhor solução obtida pelo método é oferecida como resposta.

BRIMBERG *et al.* [16] investigaram algumas heurísticas baseadas em *Variable Neighborhood Search* (VNS) [23], originalmente propostas para o *Heaviest k-subgraph Problem* (HSP). Esse problema é uma generalização do MDP definido sobre grafos. Isso implica em dizer que qualquer algoritmo proposto para o HSP também pode ser utilizado para resolver o MDP. Uma das variantes apresentadas em [16] é uma versão "distorcida" do VNS. Nela, um "passo" pode dar origem a uma solução com qualidade ligeiramente inferior à solução em mãos, o que não é usual na aplicação padrão do método. Outra variante investigada combina uma heurística construtiva seguida da aplicação do VNS.

MARTÍ *et al.* [4] compararam computacionalmente várias heurísticas sugeridas na literatura. Além disso, reuniram naquela referência uma série de instâncias

utilizadas na literatura, criando assim uma biblioteca de instâncias "benchmark" para o MDP, a MDPLIB, disponível em http://www.optsicom.es/mdp/. Dentre as heurísticas testadas em [4], as de melhor desempenho foram [16] e [18], que utilizam respectivamente VNS e *Tabu Search*.

Recentemente, WANG *et al.* [24] introduziram um Algoritmo Memético [25] que utiliza *Tabu Seach*. O algoritmo mantém uma população de soluções e realiza procedimentos de combinação de soluções e melhoramento. Inicialmente, é criada uma população de soluções. A cada iteração, os elementos da população são combinados para gerar novas soluções, e o *Tabu Seach* é então empregado para tentar melhorá-las. Uma regra de atualização de população é utilizada para decidir se uma nova solução deve ou não entrar na população. Em caso positivo, a mesma é utilizada para decidir qual solução deve sair da população e dar lugar à nova. Se um determinado número de iterações consecutivas são executadas sem que haja modificação da população, um procedimento é ativado para construir uma nova população.

Além desta introdução, a dissertação contém 4 capítulos adicionais. No Capítulo 2 apresentamos uma formulação linear padrão para o MDP. No mesmo capítulo, discutimos algumas desigualdades válidas para reforçar essa formulação. Estas são utilizadas como planos de corte em algoritmos exatos ou dualizadas dinamicamente em heurísticas Lagrangeanas e são separadas heuristicamente. No Capítulo 3 fazemos uma breve descrição dos algoritmos *Relax-and-Cut* e propomos algumas heurísticas para o MDP baseadas nesta técnica. Apresentamos também resultados computacionais comparando nossos algoritmos com algumas heurísticas da literatura. No Capítulo 4 fazemos uma breve descrição de algoritmos *Branch-and-Cut* e propomos alguns algoritmos deste tipo para o MDP. Ainda neste capítulo, comparamos computacionalmente nossos algoritmos exatos com aquele encontrado na literatura. Finalmente, no Capítulo 5, discutimos nossos resultados como um todo e oferecemos algumas conclusões sobre o nosso trabalho.

Capítulo 2

Formulação do Problema

O MDP pode ser definido em termos de um grafo completo $G = (V, E)$ com um conjunto de vértices V e um conjunto de arestas E. Nesse caso, uma correspondência biunívoca deve existir entre os itens de N e os vértices de V. Além disso, o peso da aresta $e = \{i, j\} \in E$ deve ser fixado em c_e. Dessa forma, soluções viáveis para o MDP são definidas por cliques de G com k vértices (k-cliques de G). Uma clique, vale ressaltar, é definida por um conjunto de vértices que induz um subgrafo completo de G. No nosso caso, como G é completo, todo subgrafo de G é completo. O *peso* da clique é definido como a soma dos pesos das arestas de seu subgrafo induzido. Assim, uma solução ótima para o MDP é dada pela k-clique de G com o maior peso possível.

Para formular o MDP como discutido acima, variáveis $\{0 \leq x_i \leq 1 : i \in V\}$ são associadas aos vértices de G e uma região poliedral $\mathcal{R}_0$ é definida por:

$$\sum_{i \in V} x_i = k \tag{2.1}$$

$$0 \leq x_i \leq 1, \quad i \in V. \tag{2.2}$$

Uma formulação booleana quadrática para o MDP é então dada por

$$\{\max \sum_{e=\{i,j\} \in E} c_e x_i x_j : \mathbf{x} \in \mathcal{R}_0 \cap \mathbb{B}^{|V|}\}. \tag{2.3}$$

2.1 Reformulação Linear Inteira

Utilizamos um procedimento padrão para linearizar (2.3). Para tanto, variáveis $\{y_e \in \mathbb{R}_+ : e = \{i, j\} \in E\}$ são introduzidas à formulação e o produto $x_i x_j$ é substituído por uma variável y_e correspondente e por desigualdades válidas que impõem a relação $y_e = x_i x_j$. Dessa forma, obtemos uma nova região poliedral $\mathcal{R}_1$

definida por

$$\sum_{i \in V} x_i = k \tag{2.4}$$

$$y_e \leq x_l, \quad e = \{i, j\} \in E, l = i \vee l = j, \tag{2.5}$$

$$y_e \geq x_i + x_j - 1, \quad e = \{i, j\} \in E, \tag{2.6}$$

$$0 \leq x_i \leq 1, \quad i \in V, \tag{2.7}$$

$$y_e \geq 0, \quad e \in E, \tag{2.8}$$

e uma formulação linear para o MDP é então dada por

$$\{\max \sum_{e \in E} c_e y_e : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_1 \cap (\mathbb{B}^{|V|}, \mathbb{R}_+^{|E|})\}. \tag{2.9}$$

Esta formulação pode ser reforçada através de algumas desigualdades válidas propostas na literatura. Aquelas que utilizamos neste trabalho são apresentadas na subseção que se segue. Antes disso, no entanto, vale aqui ressaltar que o problema de programação linear

$$\{\max \sum_{e \in E} c_e y_e : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_1\}, \tag{2.10}$$

obtido da formulação (2.9) ao "relaxarmos" suas restrições de integralidade é chamado de "relaxação linear" e fornece um limite superior para a mesma. Vale também ressaltar que ao reforçarmos a formulação (2.9) estamos de fato reforçando sua relaxação linear, de tal forma que melhores (menores) limites superiores são eventualmente fornecidos por ela. Como veremos no Capítulo 4, os algoritmos de solução exata que desenvolvemos para o MDP são baseados no uso de relaxações lineares reforçadas por desigualdades válidas.

2.2 Restrições Estrela

Restrições Estrela se originam da aplicação da *Reformulation Linearization Technique* (RLT) de Sherali and Admans [26] sobre a equação (2.4). Elas são obtidas multiplicando-se aquela equação por qualquer x_i, $i \in V$, e linearizando-se a equação quadrática resultante. Para tanto, deve-se ter em mente que $x_i = x_i x_i$ se aplica em nosso caso. Após algumas manipulações algébricas, Restrições Estrela são descritas como

$$\sum_{e \in \delta(i)} y_e = (k-1)x_i, \quad \forall i \in V, \tag{2.11}$$

onde $\delta(i) \subset E$ é o conjunto das arestas incidentes em $i \in V$.

Como é possível observar, a RLT gera novas restrições não lineares válidas para

um problema multiplicando-se algumas das restrições de sua formulação por um fator de grau $d \in \{0, \ldots, n\}$. A aplicação da RLT que utiliza um fator de grau d é chamada de RLT de nível d ou RLTd. As Restrições Estrela são criadas utilizando-se RLT de nível 1, dado que fatores de grau 1 foram aplicados, ou sejam, variáveis x. As restrições resultantes são linearizadas e normalmente aumentam a dimensão do problema através da inserção de novas variáveis, o que não acontece com o nosso problema.

Ao longo de nossa investigação, testamos a aplicação de RLT de nível 2, multiplicando-se as restrições (2.4) por variáveis $y_e, e \in E$. Para tanto, variáveis de grau 3, z_{ijl}, i, j e $l \in V$, foram geradas, juntamente com desigualdades para assegurar a relação $z_{ijl} = x_i x_j x_l$. A sua utilização, no entanto, foi eventualmente descartada por não se justificar em termos de ganhos computacionais.

2.3 Politopo Booleano Quadrático

PADBERG [27] investigou o Politopo Booleano Quadrático , QP^n, no contexto da linearização de um problema 0-1 quadrático irrestrito. Três famílias de desigualdades válidas para QP^n são investigadas em [27] e aqui consideraremos casos particulares de duas delas.

Desigualdades Clique: Para $S \subset V$, com $|S| \geq 2$, e um valor inteiro α, $1 \leq \alpha \leq |S| - 1$, denote por $x(S)$ a soma das variáveis x correspondentes aos vértices de S. Da mesma forma, denote por $E(S)$ o conjunto das arestas com as duas extremidades em S e denote por $y(E(S))$ a soma das variáveis y correspondentes. *Desigualdades Clique* são então definidas como

$$\alpha x(S) - y(E(S)) \leq \frac{\alpha(\alpha + 1)}{2} \tag{2.12}$$

e são válidas para QP^n. Se $|S| \geq 3$ e $1 \leq \alpha \leq |S| - 2$ se aplicam, elas definem facetas de QP^n (ver [27] para detalhes). Deste ponto do texto em diante, passaremos a nos referir às Desigualdades Clique como *GClique*.

Desigualdades Corte: Considere $S \subset V$, com $|S| \geq 1$, e $T \subseteq V \setminus S$, com $|T| \geq 2$. Denote por $(S : T)$ o $S - T$ *corte*, i.e., o conjunto de arestas com uma extremidade em S e outra em T. De forma análoga, denote por $y(S : T)$ a soma das variáveis $\mathbf{y}$ correspondentes às arestas do corte $(S : T)$. *Desigualdades Corte* são então descritas como

$$-x(S) - y(E(S)) + y(S : T) - y(E(T)) \leq 0 \tag{2.13}$$

e definem facetas para QP^n (ver [27] para detalhes). Deste ponto do texto em diante, passaremos a nos referir às Desigualdades Corte como *GCut* (exceto em alguns casos particulares, que identificaremos quando necessário).

2.4 Algoritmos de Separação

Nos algoritmos exatos e heurísticos propostos para o MDP nesta dissertação, as desigualdades do Politopo Booleano Quadrático, descritas anteriormente, são utilizadas apenas quando se fazem necessárias. São então utilizadas como cortes, no primeiro caso, ou dualizadas dinamicamente, no segundo. Para tanto, a utilização de um algoritmo de separação [1] é mandatória. Algoritmos de separação são específicos para cada família de desigualdades. Eles tomam como entrada a solução de uma relaxação, linear ou lagrangiana, de uma dada formulação e identificam se alguma desigualdade da família em questão é violada por aquela solução.

Algoritmos de separação podem ser exatos ou heurísticos. Na separação exata, tem-se a garantia de que, se alguma desigualdade pertencente a família for violada, a mesma será identificada. O mesmo pode não ocorrer ao se utilizar um algoritmo de separação heurístico. No entanto, em alguns casos, a separação exata é de alto custo computacional, e o uso de separação heurística se justifica.

A seguir apresentamos os algoritmos de separação que utilizamos para as famílias de desigualdades mencionadas acima. Para tanto, assuma que $G' = (V', E')$ é o grafo suporte associado à solução de uma relaxação linear ou lagrangiana de uma formulação do MDP. Este é um dado de entrada para nossos algoritmos de separação, que são utilizados tanto em heurísticas lagrangianas quanto em métodos exatos.

Nos nossos métodos exatos, dada uma relaxação linear, $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, válida para o MDP, E' é então definido pelas arestas de G para as quais $\overline{y}_e > 0$ se aplica, enquanto V' corresponde aos vértices de G para os quais $x_i > 0$ se aplica. Nas nossas heurísticas Lagrangianas, o grafo G' é obtido a partir da solução do *Subproblema Lagrangiano*, como veremos no Capítulo 3. Nos dois casos, à cada vértice $i \in V'$ no grafo suporte, é atribuído um custo $\overline{x}_i$. Da mesma forma, à cada aresta $e \in E'$ é atribuído um custo $\overline{y}_e$.

2.4.1 Separação Heurística de Desigualdades Clique

Um pseudo-código para a separação heurística das GClique é apresentado no Algoritmo 1. No algoritmo, o procedimento $CliqueCost(S)$ retorna o grau de violação da GClique dada pelo conjunto S e tomando $\alpha = 1$. O método se inicia a partir de uma clique formada por um único vértice e, através de uma inserção sequencial de vértices, aumenta gradativamente a dimensão da clique e o grau de violação da

GClique representada pela mesma.

Este procedimento é efetuado $|V'|$ vezes, sendo iniciado em cada uma delas por um vértice distinto de V'. Inicialmente, GClique violadas são identificadas e armazenadas. Então, para cada vértice de V', a desigualdade armazenada de maior violação, dado que uma existe, é utilizada, seja como plano de corte nos nossos algoritmos exatos Branch-and-Cut, ou dualizada dinamicamente nas nossas heurísticas Lagrangeanas.

Algoritmo 1 Separação de GClique

```
1:  procedure GenGClique( i, V')
2:      S = {i}
3:      U = V' \ {i}
4:
5:      while U ≠ ∅ do
6:          escolha algum j ∈ U
7:          U = U \ {j}
8:          c = CliqueCost(S)
9:          c1 = CliqueCost(S ∪ {j})
10:         if c1 > c then
11:             S = S ∪ {j}
12:     if |S| < 3 ou CliqueCost(S) ≤ 1 then
13:         S = {}
```

2.4.2 Separação Heurística de Desigualdades Corte

Apresentamos a seguir dois algoritmos de separação das *GCut*. As versões são ligeiramente distintas e se justificam naturalmente pela forma como são utilizadas, no Capítulo 4. O primeiro implementa uma "separação lenta" e o segundo uma "separação rápida" das desigualdades. Em ambos, o procedimento $CutCost(S, T)$ retorna o grau de violação da GCut definida pelo corte $(S : T)$.

O Algoritmo 2 define um pseudo-código para a separação lenta das *GCut*. O método se inicia a partir de um corte definido por apenas dois vértices, um pertencente a S e o outro pertencente a T. A cada iteração, $|S \cup T|$ aumenta em uma unidade, sob um objetivo "guloso" de obter a *GCut* "mais violada" à cada iteração. O procedimento é realizado $|V'|(|V'| - 1)/2$ vezes, iniciando-se, em cada caso, por pares distintos de vértices de G'.

O Algoritmo 3 define um pseudo-código para a separação rápida das *GCut*. O algoritmo se inicia a partir de um "corte" inicializado por um único vértice e, deste ponto em diante, procede de forma similar ao algoritmo anterior. O procedimento é realizado $|V'|$ vezes, uma para cada vértice distinto do grafo suporte G'.

Nos dois algoritmos, primeiro são identificadas e armazenadas as *GCut* violadas.

Algoritmo 2 Separação lenta de GCut

```
1: procedure GenGCut( i, j, V')
2:     S = {i}
3:     T = {j}
4:     U = V' \ {i, j}
5:
6:     while U ≠ ∅ do
7:         escolha algum l ∈ U
8:         U = U \ {l}
9:         c = CutCost(S, T)
10:        c1 = CutCost(S ∪ {l}, T)
11:        c2 = CutCost(S, T ∪ {l})
12:        if c2 ≥ c1 e c2 > c then
13:            T = T ∪ {l}
14:        else if c1 > c then
15:            S = S ∪ {l}
16:    if |T| < 2 ou CutCost(S, T) ≤ 0 then
17:        S = {}
18:        T = {}
```

Então, para cada vértice de V', a *GCut* armazenada de maior violação, se existir, é usada como plano de corte nos nossos algoritmos exatos Branch-and-Bound ou dualizada dinamicamente, nas nossas heurísticas Lagrangeanas.

2.4.3 Casos especiais

Nesta dissertação, a separação exata das Desigualdades Corte é feita para dois casos particulares. A saber, para as desigualdades *3Cut* definidas por $S \subset V$, $|S| = 1$ e $|T| = 2$, que podem ser separadas em tempo $O(|V|^3)$, e para as desigualdades *4Cut* definidas por $S \subset V$, $|S| = 1$ e $|T| = 3$, que podem ser separadas em tempo $O(|V|^4)$.

Utilizando o grafo $G' = (V', E')$ definido anteriormente, a separação se dá como descrito a seguir. Inicialmente, são enumeradas de forma exaustiva todas as desigualdades, de 3 vértices para as 3Cut e de 4 vértices para as 4Cut, existentes em G'. Ao fazer isso, aquelas que são violadas são armazenadas. Posteriormente, para cada vértice do grafo, a desigualdade de maior violação, caso exista uma, é utilizada como plano de corte ou dualizada, dependendo do caso.

Casos especiais adicionais das Desigualdades Clique foram estudadas nas fases iniciais desta pesquisa, mas foram eventualmente descartados. A razão para isso foi o baixo custo/benefício computacional que apresentaram. Resultados preliminares e aqueles que apresentamos no Capítulo 4 indicam que as Desigualdades Corte têm melhor desempenho que as Desigualdades Clique.

Algoritmo 3 Separação rápida de GCut

```
1:  procedure GenGCut( i, V')
2:      S = {i}
3:      T = {}
4:      U = V' \ {i}
5:
6:      while U ≠ ∅ do
7:          escolha algum j ∈ U
8:          U = U \ {j}
9:          c = CutCost(S, T)
10:         c1 = CutCost(S ∪ {j}, T)
11:         c2 = CutCost(S, T ∪ {j})
12:         if c2 ≥ c1 e c2 > c then
13:             T = T ∪ {j}
14:         else if c1 > c then
15:             S = S ∪ {j}
16:     if |T| < 2 ou CutCost(S, T) ≤ 0 then
17:         S = {}
18:         T = {}
```

Capítulo 3

Heurísticas Lagrangeanas

Procedimentos Lagrangeanos análogos a algoritmos de planos de cortes, foram introduzidos simultaneamente em [28] e [29]. Posteriormente, em [30], o termo "Relax-and-Cut", introduzido em [29], foi utilizado para denotar tanto o procedimento sugerido em [28], que passou a ser chamado "Non-Delayed Relax-and-Cut" (NDRC), quanto aquele sugerido em [29], que ao invés de sua denominação original Relax-and-Cut, foi chamado de "Delayed Relax-and-Cut" (DRC). Resultados computacionais apresentados em [30] mostraram um melhor desempenho dos algoritmos NDRC, quando comparados a algoritmos DRC correspondentes.

Relaxação Lagrangeana em geral e algoritmos Relax-and-Cut, em particular, são muito utilizados para se obter bons limites duais e primais para problemas de OC. Em especial, no caso específico dos algoritmos Relax-and-Cut, os problemas tratados são não apenas NP-Difíceis, mas costumam envolver também formulações com um número exponencialmente grande de desigualdades. O texto a seguir descreve o funcionamento geral de um algoritmo NDRC e se baseia em [30].

3.1 Algoritmos NDRC

Dada uma formulação para um problema NP-difícil de OC, assuma que um número exponencialmente grande de desigualdades fazem parte da mesma. Assuma também que essa formulação pode ser descrita genericamente como

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

onde $x \in \mathbb{B}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{mxn}$ e $X \subseteq \mathbb{B}^n$, onde m e n assumem valores inteiros positivos. Como é usual em relaxação Lagrangeana, assuma que

$$\min\{cx : x \in X\} \tag{3.2}$$

é um problema fácil de resolver (ou seja, pode ser resolvido à otimalidade em tempo polinomial). Por outro lado, no que é pouco usual em termos de relaxação Lagrangeana, assuma que m aumenta exponencialmente com n (ou seja, assuma que a família de desigualdades a dualizar contém exponencialmente muitos elementos). Apesar dessa enorme barreira, vamos proceder à dualização das desigualdades

$$\{a_i x \leq b_i : i = 1, 2, \ldots, m\}, \tag{3.3}$$

na forma Lagrangeana usual, assumindo ser isso possível, quando, de fato, não o é.

Devido ao potencialmente enorme número de desigualdades a dualizar, fazemos essa dualização, inicialmente, sem levar em conta suas implicações práticas. Sendo $\lambda \in \mathbb{R}^m_+$, os multiplicadores de Lagrange associados às desigualdades dualizadas, um limite inferior para (3.1) é então dado pelo valor da solução do Subproblema Lagrangeano

$$z_\lambda = \min\{(c + \lambda A)x - \lambda b : x \in X\} \tag{3.4}$$

denotado aqui por SL(λ). Nesse caso, o melhor limite inferior possível para (3.1), é obtido ao se resolver o Problema Dual Lagrangeano (PDL)

$$z_d = \max_{\lambda \in \mathbb{R}^m_+} \{z_\lambda\}. \tag{3.5}$$

O PDL é tipicamente resolvido de forma iterativa, atualizando-se os multiplicadores λ até que o valor z_d seja eventualmente obtido.

Para melhor explicar a forma de tratamento das exponencialmente muitas desigualdades $Ax \leq b$, apresentamos a seguir uma breve descrição do Método do Subgradiente (MS), seguida da adaptação do mesmo para o NDRC, como proposto em [28].

3.1.1 O Método do Subgradiente

O MS é utilizado para resolver o PDL de forma iterativa através da atualização sistemática dos multiplicadores de Lagrange. Em uma dada iteração de MS, para um vetor viável de multiplicadores de Lagrange λ, o SL(λ) correspondente é resolvido e uma solução $\overline{x}$, ótima para ele, de valor z_λ, é obtida. Seja z_{ub} um limite superior conhecido para (3.1). Denotando-se por $g \in \mathbb{R}^m$ o vetor de subgradientes no ponto $\overline{x}$, temos então que

$$g_i = (b_i - a_i \overline{x}), i = 1, 2, \ldots, m. \tag{3.6}$$

O MS é iniciado com um vetor λ qualquer, viável. Na literatura, os multiplicadores de Lagrange, λ, são normalmente atualizados após o cálculo de um tamanho

de passo

$$\theta = \frac{\epsilon(z_{ub} - z_\lambda)}{\sum_{i=1,\ldots,m} g_i^2}, \tag{3.7}$$

onde ϵ é um número real assumindo valores em $(0, 2]$. Feito isso, procedemos à atualização desses multiplicadores através da fórmula

$$\lambda_i \equiv \max\{0; \lambda_i - \theta g_i\} \tag{3.8}$$

e passamos então para a iteração seguinte de MS.

Uma implementação prática do MS exige a redução gradativa do valor de ϵ. Para tanto, é necessário definir o número máximo de iterações de MS, *maxiter*, a serem realizadas. Então, a cada *maxhalf* iterações consecutivas sem que haja melhora no melhor limite dual já obtido, o valor de ϵ é reduzido pela metade. Normalmente, *maxhalf* é definido em função de *maxiter*. Adotamos *maxhalf* = *maxiter*/20 como padrão nos nossos algoritmos.

Nesse ponto, é importante ressaltar que, diante de um número exponencialmente grande de desigualdades dualizadas, a aplicação direta das fórmulas (3.7) e (3.8) seria uma tarefa de realização impraticável. Desse modo, para permitir a utilização de λ, vamos proceder a algumas modificações de MS.

3.1.2 Uma adaptação de MS ao NDRC

As desigualdades (3.3), numa iteração corrente de MS, podem ser divididas em 3 grupos distintos. O primeiro deles é formado pelas desigualdades violadas por $\overline{x}$. O segundo contém as desigualdades que, no momento, têm multiplicadores não nulos associados a elas. Note que, da forma como foram definidos esses dois grupos, é possível que uma mesma desigualdade faça parte de ambos, simultaneamente. As demais desigualdades de (3.3), ou seja, aquelas que não fazem parte dos dois grupos anteriores, definem o nosso terceiro grupo. Estas, por sua vez, não são violadas por $\overline{x}$ e têm multiplicadores nulos na iteração corrente de MS. Daqui em diante, vamos nos referir aos três grupos que acabamos de definir como grupos 1, 2 e 3, respectivamente.

Na aplicação tradicional de relaxação Lagrangeana, quando m é uma função polinomial de n, BEASLEY [31] fez uma observação bastante interessante. Ele constatou que ao fixar-se, arbitrariamente, em qualquer iteração do método, $g_i = 0$ para toda componente i de g onde $g_i \leq 0$ e $\lambda_i = 0$, se obtém uma boa convergência prática de MS, para o limite (3.5). Em nosso contexto, essa sugestão é equivalente a fixarmos em 0 todas as componentes de g associadas a desigualdades do grupo 3.

Apesar de assumirmos a existência de um número exponencialmente grande de desigualdades em (3.3), vamos seguir a sugestão de Beasley. Uma justificativa para

isso, vem das duas observações que se seguem. Na primeira delas, note que, mesmo sem efetuarmos as modificações propostas por [31], pela fórmula de atualização (3.8) os multiplicadores associados às desigualdades do grupo 3 permaneceriam nulos ao final de (3.8). Em função disso, essas desigualdades são chamadas de "inativas". É evidente que uma desigualdade inativa, na iteração corrente de MS, não irá contribuir diretamente para a função objetivo Lagrangeana. Tal fato nos leva a uma segunda observação. A de que, em geral, para as desigualdades do grupo 3, o número de componentes de g assumindo valores estritamente negativos tende a ser enorme. Se essas componentes fossem utilizados em (3.7), o valor obtido para θ tenderia a 0. Ou seja, o multiplicador de Lagrange para qualquer uma das desigualdades dualizadas permaneceria virtualmente inalterado, após sua atualização. Isso, por sua vez, traria problemas de convergência para o método. Entretanto, ao seguirmos a sugestão de [31], conseguimos evitar esta dificuldade. Conseguimos também definir uma forma conveniente de tratar um número exponencialmente grande de desigualdades candidatas a dualização.

É evidente que, sob a classificação de desigualdades proposta acima, é possível (e também provável) que uma desigualdade mude sua classificação, ao longo das iterações de MS. Da mesma forma, é evidente que apenas aquelas desigualdades classificadas nos grupos 1 e 2 podem efetivamente contribuir para os custos Lagrangeanos ($c + \lambda A$), em uma dada iteração de MS. Assim sendo, as desigualdades dos grupos 1 e 2 são denominadas "desigualdades ativas". Em oposição a isso, as desigualdades do grupo 3 são chamados "inativas".

Em resumo, para atualizar os multiplicadores λ em um algoritmo NDRC, apenas as desigualdades ativas são utilizadas nas fórmulas (3.7) e (3.8), à cada iteração de MS.

Um detalhe importante no esquema de classificação sugerido acima, é a identificação das restrições violadas por $\overline{x}$ na iteração corrente de MS. Recorde que, de acordo com esta classificação, tais restrições pertencem ao grupo 1. Identificá-las, é então um problema análogo ao Problema de Separação, associado aos algoritmos de planos de corte. No entanto, no nosso caso específico, tal problema tende a ser mais fácil de resolver. Isso porque, num contexto Lagrangeano, $\overline{x}$ tende a assumir apenas valores inteiros.

3.2 Algoritmos NDRC para o MDP

Ao longo desta pesquisa, diversas versões do algoritmo NDRC para o MDP foram propostas e avaliadas. Tais versões diferem entre si pelas famílias de desigualdades utilizadas como candidatas à dualização e pelos algoritmos de identificação de desigualdades violadas aplicados a elas. Tomando por base alguns resultados preli-

minares, versões do algoritmo que se mostraram menos atraentes foram descartadas enquanto versões mais promissoras foram mantidas. Dentre as versões do NDRC que investigamos para o MDP, optamos por mostrar aqui cinco delas: NDRC_S, NDRC_3Cut, NDRC_4Cut, NDRC_GCut e NDRC_GCut_F.

Todas as versões do NDRC aqui descritas utilizam a formulação que apresentaremos a seguir. Iremos nos referir a ela, daqui em diante, como "formulação base". Esta, por sua vez, é reforçada com famílias de desigualdades distintas em cada versão. A saber, na versão NDRC_S apenas a formulação base é utilizada, sem reforços adicionais. Nas versões NDRC_3Cut e NDRC_4Cut a formulação base é reforçada pelas famílias de desigualdade 3Cut e 4Cut, respectivamente. Finalmente, tanto em NDRC_Gcut quanto em NDRC_Gcut_F, a formulação base é reforçada pelas desigualdades GCut. Entretanto, estas duas versões diferem entre si na forma de se obter o grafo suporte G', utilizado no algoritmo de separação (mais detalhes nas Seções 3.3 e 3.5).

Para descrevermos as formulações de MDP que utilizamos, considere uma região poliedral $\mathcal{R}_2$, definida pela interseção de $\mathcal{R}_1$ (definida na Seção 2.1) com as Restrições Estrela (2.11). Nossos algoritmos NDRC para o MDP utilizam então uma formulação base para o problema, definida por

$$\{\max \sum_{e \in E} c_e y_e : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_2 \cap (\mathbb{B}^{|V|}, \mathbb{R}_+^{|E|})\} \tag{3.9}$$

e se inspiram no algoritmo NDRC proposto em [32], para o Problema da Mochila Quadrática (QKP).

Para gerar limites duais Lagrangeanos a partir de (3.9), as restrições (2.5),(2.6) e (2.11) são dualizadas no estilo Lagrangeano Tradicional (LT). Então, de acordo com nossas observações anteriores, associamos matrizes de multiplicadores Λ^1 e Λ^2, e um vetor de multiplicadores, λ^{star}, respectivamente às restrições (2.5), (2.6) e (2.11).

O algoritmo NDRC_3Cut utiliza a formulação (3.9) reforçada pelas desigualdades 3Cut, um dos casos especiais das GCuts, (2.13). Tais desigualdades são dualizadas dinamicamente no estilo NDRC, com o uso de um vetor de multiplicadores Γ. Desigualdades 3Cut são portanto dualizadas sob demanda, à medida que se tornam violadas na solução de um SL. Sempre que isso ocorre, o valor corrente do seu respectivo multiplicador pode ser nulo ou não nulo. Se for não nulo, a desigualdade foi violada inicialmente na solução de um SL anterior e continuou *ativa* até a iteração corrente do MS. Do contrário, pelo menos por enquanto, assumiremos que a desigualdade está sendo violada pela primeira vez. No entanto, de uma forma ou de outra, se uma desigualdade 3Cut é violada pela solução do SL corrente, a ela será atribuído um multiplicador não nulo ao final da iteração corrente do MS. Uma desigualdade dualizada dinamicamente é então dita ativa sempre que seu multiplicador

correspondente tem um valor não nulo. Deve-se notar que uma desigualdade pode se tornar *inativa* na iteração seguinte do MS e, da mesma forma, se tornar novamente ativa, posteriormente, no decorrer do procedimento (veja [30], para detalhes).

Nos algoritmos NDRC_4Cut, NDRC_Gcut e NDRC_Gcut_F as desigualdades utilizadas para fortalecer a formulação base, ou seja, as desigualdades 4Cut e GCut, são tratadas de maneira similar àquela descrita acima para as desigualdades 3Cut. São então dualizadas dinamicamente, em um modelo NDRC. A família GCut contém exponencialmente muitas desigualdades e sua utilização num estilo Lagrangeano Tradicional seria então inviável. Justifica-se, portanto, sua utilização ao estilo NDRC. As 3Cut e 4Cut, apesar de não envolverem exponencialmente muitas desigualdades, contêm um número excessivo de desigualdades, o que também justifica dualizá-las ao estilo NDRC.

De uma maneira geral, o número de desigualdades ativas tende a ser "pequeno" em qualquer iteração do MS. Dessa forma, desigualdades ativas desempenham um papel fundamental na atualização dos multiplicadores de Lagrange (veja [30], para detalhes). Por um lado, todos os subgradientes associados às restrições RLP (2.11), que são poucas, devem sempre ser utilizados nessas operações, sejam essas restrições ativas ou não. Por outro, no entanto, dentre as desigualdades dinamicamente dualizadas, apenas as ativas devem se consideradas para efeito de atualização dos multiplicadores. Assim procedendo (veja [30], para detalhes), se evita a armadilha do esforço computacional elevado ao se efetuar essa operação.

Em complemento aos multiplicadores $\Lambda = (\Lambda^1, \Lambda^2, \lambda^{star})$, lembramos que Γ define os multiplicadores associados às famílias de desigualdades 3Cut, 4Cut e GCut. Note que a dimensão de Γ pode ser exponencial. No entanto, a maioria dos coeficientes de Γ corresponde a desigualdades inativas e estas têm, portanto, valor zero. Assim sendo, nas regras de atualização de multiplicadores, pouquíssimas desigualdades associadas a multiplicadores Γ, somente as ativas, são efetivamente consideradas.

Denotando por $\mathcal{R}_3$ a região poliedral definida por (2.4), (2.7) e (2.8), o SL que iremos resolver a cada iteração do MS é descrito por

$$v(\Lambda, \Gamma) = \{\max \sum_{i \in V} \overline{q}_i x_i + \sum_{e \in E} \overline{c}_e y_e + Const(\Lambda, \Gamma) : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_3 \cap (\mathbb{B}^V, \mathbb{R}_+^E)\}, \quad (3.10)$$

onde $\overline{q}$ e $\overline{c}$ são os custos Lagrangeanos que se aplicam, enquanto $Const(\Lambda, \Gamma)$ é uma constante que depende de Λ e Γ.

3.3 Resolvendo o Subproblema Lagrangeano (SL)

Duas diferentes abordagens para resolver SL são discutidas a seguir. Como veremos, além de gerar limites duais para o MDP elas também definem soluções primal

viáveis para o problema. Dado que utilizamos informação dual para gerar as soluções primais, estas tendem a ser de boa qualidade.

3.3.1 Abordagem I

Tomando (3.10) como ponto de partida, dois subproblemas independentes são resolvidos para gerar limites duais válidos para o MDP. Um é definido pelas variáveis $\mathbf{x}$, enquanto o outro é definido por variáveis $\mathbf{y}$. O primeiro subproblema é resolvido fixando-se em 1 as k variáveis x com maior valor $\overline{q}$. As demais variáveis em x são fixadas em 0. O segundo problema é resolvido atribuindo-se o valor 1 a toda variável $\mathbf{y}_e$ com $\overline{c}_e > 0$. Finalmente, adicionando-se $Const(\Lambda, \Gamma)$ aos valores ótimos dos dois subproblemas que acabamos de descrever, obtém-se então um limite dual válido para o MDP. Como um benefício adicional desse esquema, $\overline{x}$ define uma solução viável para o MDP, de valor $\sum_{e=i,j \in E} c_e \overline{x}_i \overline{x}_j$.

Limites duais melhores que os que acabamos de discutir podem ser obtidos utilizando-se a abordagem que se segue.

3.3.2 Abordagem II

Vamos formular primeiro um SL auxiliar que é definido apenas nas variáveis $\mathbf{x}$. Para tanto, considere as desigualdades

$$y_e \leq \frac{x_i + x_j}{2} \quad \forall e = \{i, j\} \in E, \tag{3.11}$$

obtidas após a soma e divisão por 2 das duas desigualdades definidas em (2.5) para cada aresta de G.

Seguindo o mesmo raciocínio utilizado em [32] e repartindo as contribuições das desigualdades (3.11) entre i e j, "pesos Lagrangeanos auxiliares", $\pi_i = \overline{q}_i + \sum_{e \in \delta(i)} \max\{\frac{\overline{c}_e}{2}, 0\}$, são então associados à cada variável x_i, $i \in V$. A justificativa para assim proceder é que só seria aplicável um custo Lagrangeano $\max\{\overline{c}_e, 0\}$, onde $e = \{i, j\}$, se x_i e x_j assumem simultaneamente valor 1 na solução do SL. Ao se dividir entre elas o valor de $\max\{\overline{c}_e, 0\}$, em partes iguais, chegamos a um subproblema definido unicamente nas variáveis x, ou seja,

$$Const(\Lambda, \Gamma) + \{\max \pi \mathbf{x} : \mathbf{x} \in \mathbb{B}^{|V|} \text{ e satisfaz } (2.4)\}. \tag{3.12}$$

Este, garantidamente, nos fornece um limite superior válido para o MDP e é resolvido por inspeção.

Assumindo que $\overline{\mathbf{x}}$ é uma solução ótima para (3.12), valores correspondentes para y_e, $e = \{i, j\} \in E$, são obtidos da seguinte forma:

- $\overline{y}_e = 1$, se $\overline{x}_i = \overline{x}_j = 1$.
- $\overline{y}_e = \frac{1}{2}$, se $\overline{x}_i = 1$ e $\overline{x}_j = 0$ ou $\overline{x}_i = 0$ e $\overline{x}_j = 1$.
- $\overline{y}_e = 0$, se $\overline{x}_i = \overline{x}_j = 0$.

É possível melhorar ainda mais os limites duais definidos por (3.12). Para tanto, note que as restrições (2.11) impõem que quando $x_i = 1$, exatamente $(k-1)$ variáveis y_e, $e \in \delta(i)$, devem assumir o valor 1. Dessa forma, podemos tomar π_i como a soma de $\overline{q}_i$ com os $(k-1)$ maiores $\{\frac{\overline{c}_e}{2}\colon\ e \in \delta(i)\}$. Consequentemente, ao contrário do que prevalecia anteriormente, pode-se eventualmente obter contribuições para π_i advindos de variáveis y_e com $\overline{c}_e < 0$, fortalecendo portanto os limites anteriores. Da mesma forma, se existirem mais do que $(k-1)$ arestas assumindo custos Lagrangeanos positivos em $\{\overline{c}_e : e \in \delta(i)\}$, o valor de π_i é reduzido ao tomarmos apenas as contribuições das $(k-1)$ arestas de maior custo.

Note que, mesmo com a modificação que acabamos de sugerir, o subproblema a ser resolvido, ou seja, (3.12), se mantém o mesmo e sua solução é obtida por inspeção. Valores para y, por sua vez, são obtidos de maneira distinta daquela descrita anteriormente. Para obtê-los, seja $\mathcal{E}_i \subseteq \delta(i)$, $i \in V$, o conjunto das $(k-1)$ arestas escolhidas para compor o peso Lagrangeano auxiliar π_i. Para facilitar a descrição, uma vez obtida uma solução $\overline{x}$ para (3.12), redefinimos $\mathcal{E}_i = \emptyset$ se $x_i = 0$. Assim procedendo, valores correspondentes para y_e, $e = \{i, j\} \in E$, são obtidos da seguinte forma:

- $\overline{y}_e = 1$, se $e \in \mathcal{E}_i$ e $e \in \mathcal{E}_j$.
- $\overline{y}_e = \frac{1}{2}$, se $e \in \mathcal{E}_i$ e $e \notin \mathcal{E}_j$ ou $e \notin \mathcal{E}_i$ e $e \in \mathcal{E}_j$.
- $\overline{y}_e = 0$, se $e \notin \mathcal{E}_i$ e $e \notin \mathcal{E}_j$.

O subproblema (3.12), na prática, é resolvido ordenando-se de forma crescente os valores $\{\pi_i : i \in V\}$. As k variáveis de maior valor são fixadas em 1 e as demais em 0. Ao longo de nossa investigação, observamos que essa ordenação é um forte indicativo da "probabilidade" de uma variável x_i fazer parte de uma solução ótima, principalmente quando o limite dual dado por (3.12) é de boa qualidade. No Capítulo 4 mostramos como essa propriedade é explorada nos algoritmos exatos propostos neste trabalho.

3.3.3 Implicação

Um importante implicação da utilização da Abordagem II diz respeito ao esforço computacional gasto nos algoritmos de separação. Como foi dito anteriormente, nossos algoritmos NDRC utilizam os algoritmos de separação descritos no Capítulo

2 para identificar desigualdades violadas candidatas a dualização. A dimensão do grafo suporte, $G' = (V', E')$, associado à solução de um SL, é então um indicativo do esforço computacional a ser despendido na separação das desigualdades.

Seja $(\overline{x}, \overline{y})$ a solução do SL, correspondente à Abordagem II. Nesse caso, o grafo suporte pode ser definido de duas formas distintas. Na primeira, E' é definido pelas arestas de G para as quais $\overline{y}_e > 0$ se aplica, enquanto V' corresponde aos vértices G que aparecem em alguma extremidade das mesmas. Em outras palavras, G' é o subgrafo de G induzido pelas arestas E'. Note que toda aresta de E' tem uma extremidade com valor 1 na solução $\overline{x}$, e pode ou não ter extremidade com valor 0. Dessa forma, G' tende a ser de pequena dimensão, com k e $k(k-1)/2$ servindo de referência para $|V'|$ e $|E'|$, respectivamente.

Na segunda forma de se obter o grafo suporte G', V' é definido pelos vértices de G para os quais $\overline{x}_i = 1$ se aplica, enquanto E' é definido pelas arestas de G com as duas extremidades em V'. Em outras palavras, G' é o subgrafo de G induzido pelos vértices V'. Assim, o tamanho de G' é definido unicamente pelo valor de k, onde $|V'| = k$.

Na primeira forma de se obter o grafo suporte, toda a informação presente na solução do PRL, de certo modo, é passada para o grafo. Nos referiremos ao grafo G' obtido desta maneira por grafo suporte completo. Na segunda forma, por sua vez, parte da informação é deixada de fora, a saber, algumas das arestas associadas a $\overline{y}_e = 1/2$ com uma das extremidades fora do grafo G'. Nos referiremos ao grafo suporte obtido desta maneira por grafo suporte reduzido. É fácil notar que a separação utilizando o grafo suporte reduzido utiliza menos recursos computacionais que a separação utilizando o grafo suporte completo. Na Seção 3.5 mostramos a diferença de desempenho em separar *Gcut* utilizando os dois tipos de grafo suporte.

3.4 Busca Local

Uma vez obtida uma solução viável para o MDP, esta é normalmente submetida a um procedimento de busca local que visa obter uma melhor solução a partir de uma solução conhecida. Ao longo desta pesquisa, testamos vários desses procedimentos. Nesta seção, descrevemos o procedimento de busca local que foi adotado em nossos algoritmos. Ele consiste numa busca local que mescla o procedimento proposto em [16], numa heurística VNS, com o procedimento *Tabu Search*, sugerido em [24].

3.4.1 Tabu Search

Tabu Search (TS) é um procedimento iterativo que tenta melhorar uma solução viável que se tem acesso. Ele opera na vizinhança daquela solução e utiliza um

operador de *troca* para substituir variáveis pertencentes à solução disponível por variáveis não pertencentes. Trocas são realizadas à cada iteração, e tem como objetivo obter a melhor solução na vizinhança investigada. Note que, dessa forma, uma solução de pior qualidade pode ser obtida como resultado de uma operação de troca, desde que esta seja a melhor troca disponível na iteração corrente. O método utiliza ainda uma "memória de curto prazo", que previne a reincidência de soluções obtidas recentemente. Como consequência, TS é capaz de fugir de ótimos locais de baixa qualidade.

Descrevemos a seguir o procedimento TS proposto para o MDP em [24]. Este se utiliza de dois artifícios para obter ganhos de eficiência. O primeiro reduz o número de variáveis candidatas à troca. O segundo, permite que o cálculo do custo de uma solução seja feito de forma mais eficiente.

O algoritmo TS recebe como entrada um vetor solução $x \in \mathbb{B}^n$ e seu custo $f(x) = \sum_{e=\{i,j\}\in E} c_e x_i x_j$. As variáveis de $x = \{x_1, x_2, \ldots, x_n\}$ são divididas em dois subconjuntos, U e Z. O subconjunto U contém as variáveis com valor 1 e o subconjunto Z contém as variáveis com valor 0. O operador de *troca* substitui uma variável pertencente a U por uma pertencente a Z. Note que o conjunto U remete à uma solução do MDP definido em grafos. Seus elementos induzem uma k-clique em G e definem uma solução viável para o problema. Dessa forma, após uma operação de troca, como a cardinalidade dos conjuntos permanece inalterada, a viabilidade da solução é mantida.

À cada iteração (ou passo), o método parte de uma solução $\overline{x}$ e a transforma numa nova solução, que também denotaremos por $\overline{x}$. Um passo é executado efetuando-se uma operação de troca entre um par de variáveis distintas, originárias dos conjuntos U e Z. Como $|U| = k$ e $|Z| = n - k$, $k(n-k)$ pares distintos de candidatos à troca devem ser considerados. Para escolher o melhor deles, é necessário calcular o custo de todas as soluções existentes na "1-vizinhança" de $\overline{x}$.

A complexidade de se escolher a melhor solução na 1-vizinhança definida acima é $O(k^2)$, o que pode ser considerado computacionalmente caro. Essa operação de troca, no entanto, altera a solução original apenas marginalmente, já que uma única variável é substituída no decorrer dela. Em função disso, [24] sugere o procedimento de troca eficiente que discutiremos a seguir.

Inicialmente, um vetor Δ é utilizado para armazenar o *custo de troca* de cada variável. Este é definido pela variação no custo da solução, provocada pela operação de mover uma variável x_i de seu subconjunto atual para o outro subconjunto. Cada coeficiente Δ_i, $i \in V$, é definido pela formula:

$$\Delta_i = \begin{cases} -\sum_{e=\{i,j\}:x_j\in U} c_{ij}, & \text{se } x_i \in U \\ \sum_{e=\{i,j\}:x_j\in U} c_{ij}, & \text{se } x_i \in Z \end{cases}$$

A variação resultante da troca de $x_i \in U$ por $x_j \in Z$ é então dada por:

$$\omega_{ij} = \Delta_i + \Delta_j - c_{ij}.$$

Consequentemente, o valor da nova solução é definido por $f(x)+\omega_{ij}$. A melhor troca a efetuar corresponde então ao par de itens i, j de maior ω_{ij}. Após a realização da mesma, os coeficiente de Δ são atualizados por:

$$\Delta_k = \begin{cases} -\Delta_i + c_{ij}, \text{ para } k = i \\ -\Delta_j + c_{ij}, \text{ para } k = i \\ \Delta_k + c_{ik} - c_{kj}, \text{ para } k \neq i \wedge k \neq j, x_k \in U \\ \Delta_k - c_{ik} + c_{kj} \text{ para } k \neq i \wedge k \neq j, x_k \in Z) \end{cases}$$

Quanto maior for o custo de troca Δ_k correspondente a um vértice, mais provável será a participação desse vértice numa troca de boa qualidade. Ou seja, o processo de escolha pela melhor troca pode ser acelerado restringindo-o apenas às trocas entre candidatos de boa qualidade. Para tanto, os melhores candidatos de U e Z são respectivamente selecionados para compor as listas LCU e LCZ. Trocas ficam restritas então apenas a variáveis pertencentes e essas listas.

As dimensões das listas LCU e LCZ, definidas por um parâmetro tlc, impactam diretamente no tempo de solução do algoritmo e na qualidade das soluções encontradas. Um valor muito pequeno para tlc pode deixar de fora bons candidatos e acabar não levando a melhores soluções. Já um valor muito grande pode incluir candidatos de baixa qualidade e levar a um custo computacional excessivo sem um ganho de qualidade que o justifique.

Um vetor T é utilizado como uma "memória de curto prazo". O vetor armazena a iteração a partir da qual cada variável estará disponível para participar de uma operação de troca. Todas as entradas do vetor são inicializadas em 0, indicando que todas as variáveis estão disponíveis para troca na primeira iteração. Sempre que uma troca é efetuada, digamos entre as variáveis x_i e x_j, as entradas $T[i]$ e $T[j]$ são atualizadas de acordo com o seguinte esquema: 1) dois números inteiros r_i e r_j são escolhidos aleatoriamente dentro de um intervalo $[a, b]$, pré-definido; 2) r_i e r_j são somados ao número da iteração atual e atribuídos a $T[i]$ e $T[j]$, respectivamente. Dessa forma, a solução na troca anterior se torna inatingível por, no mínimo, a iterações.

Como mencionado anteriormente, à cada iteração do método é efetuada uma troca, mesmo que isso leve a uma solução de qualidade inferior à da solução corrente. Isso pode ocorrer quando a solução corrente define ótimo local, ou seja, uma solução de qualidade igual ou superior à todas as soluções na vizinhança investigada. Esse comportamento dá ao algoritmo a capacidade de fugir de ótimos locais, mas demanda

a necessidade de se armazenar, em separado, a melhor solução obtida. No algoritmo, as entradas x^* e $f(x^*)$ têm o papel de armazenar respectivamente a melhor solução até então obtida e o seu custo associado. Sempre que o custo de uma nova solução supera $f(x^*)$, esta é armazenada em x^* e $f(x^*)$ é atualizado em função disso. O Algoritmo 4 descreve o pseudo-código do TS proposto em [24].

Algoritmo 4 Tabu Search

procedure TS(s, $f(s)$)
    Inicializar vetor Δ, lista tabu T com valor 0, os conjuntos U e Z com as variáveis com valor 1 e 0 em s, respectivamente. Iter=0, IterSemMelhora = 0, $s^* = s$, $f(s^*) = f(s)$.

    while $IterSemMelhora < \varphi$ **do**
        Inicializar conjuntos LCU e LCZ a partir de U e Z de acordo com o valor de Δ
        Identificar os índices i^*_{nt} e j^*_{nt} das variáveis não tabu pertencentes a LCU e LCZ que leva ao maior valor de ω. Identificar i^*_t e j^*_t para variáveis tabu.

        if $\omega_{i^*_t j^*_t} > \omega_{i^*_{nt} j^*_{nt}}$ e $f(s) + \omega_{i^*_t j^*_t} > f(s^*)$ **then**
            $i^* = i^*_t, j^* = j^*_t$
        else
            $i^* = i^*_{nt}, j^* = j^*_{nt}$
        $s_{i^*} = 0, s_{j^*} = 1, f(s) = f(s) + \omega_{i^* j^*}$
        $U = U \setminus \{s_{i^*}\} \cup \{s_{j^*}\}, Z = Z \setminus \{s_{j^*}\} \cup \{s_{i^*}\}$
        $T[i] = Iter + rand(a, b), T[j] = Iter + rand(a, b)$
        Atualizar Δ

        if $f(s) > f(s^*)$ **then**
            $s^* = s$, $f(s^*) = f(s)$
            $IterSemMelhora = 0$
        else
            $IterSemMelhora = IterSemMelhora + 1$

        $Iter = Iter + 1$
    return s^*

3.4.2 Busca local VNS

O algoritmo introduzido em [16] leva em consideração um conjunto de vizinhanças, denotadas por $\{\mathcal{N}_p : p = 1, \ldots, p_{max}\}$. Para definir qualquer uma dessas vizinhanças, é necessário definir primeiro uma função métrica para o espaço de soluções do MDP.

Seja $\mathscr{S} = \{S | S \subseteq V, |S| = k\}$ uma caracterização do espaço de soluções do MDP e considere a função métrica ρ que expressa a *distância* entre qualquer par de

soluções viáveis, digamos $S, S' \in \mathscr{S}$. Esta distância é dada por

$$\rho(S, S') = \rho(S', S) = |S \setminus S'|.$$

A vizinhança $\mathcal{N}$, expressa implicitamente pela função métrica ρ, é definida como:

$$\mathcal{N}_p(S) = \{S' | S' \in \mathscr{S}, \rho(S', S) = p\}, \quad p = 1, \ldots, p_{max}.$$

Note que p_{max} é limitado pela distância máxima permitida para qualquer par de soluções existentes no espaço de soluções. Nesse caso, $p_{max} = \min\{k, |V| - k\}$ se aplica.

O procedimento de busca local baseado na heurística VNS sugerida em [16] é descrito através do pseudo-código apresentado no Algoritmo 5. O procedimento toma como entrada uma solução viável para o MDP e tenta obter uma sequência de soluções monotonicamente melhores. Com esse objetivo, realiza operações de *variação* e *melhora* sobre a solução corrente do MDP.

Assumindo que $\mathcal{N}_p$ é a vizinhança sob investigação, uma operação de variação define a próxima vizinhança a ser investigada, seja para $\mathcal{N}_{p+1}$ ou $\mathcal{N}_{p_{min}}$. A primeira vizinhança é escolhida se $(p + 1) \leq p_{max}$ e a operação de melhora efetuada em $\mathcal{N}_p$ falha. Alternativamente, $\mathcal{N}_{p_{min}}$ é escolhida se a operação de melhora é bem sucedida. Finalmente, se $(p + 1) > p_{max}$ a busca local é encerrada.

A operação de melhora utiliza o procedimento TS descrito na Seção 3.4.1. A operação de variação é realizada por um procedimento de *shake*. Este toma como entrada um inteiro p e uma solução viável S, para o MDP. O *shake* retorna uma solução, $S' \in \mathcal{N}_p(S)$, que corresponde a uma troca simultânea de p vértices de S por p vértices de $V \setminus S$.

Algoritmo 5 Variable Neighborhood Search

```
1:  procedure VNS(S, p_min, p_max)
2:      s* = TS(S, f(S))
3:
4:      p = p_min
5:      while p ≤ p_max do
6:          s = shake(s*, p)
7:          s = TS(s, f(s))
8:          if f(s) > f(s*) then
9:              s* = s
10:             p = p_min
11:         else
12:             p = p + 1
13:     return s*
```

Em nossos experimentos computacionais sempre utilizamos $p\_min = p\_max =$

1. Exceder esse valor para p_max se mostrou ineficiente e, portanto, foi evitado. Também utilizamos os seguintes parametros para TS: $tlc = \sqrt{n}$, $\varphi = 6 * k$, $a = 15$ e $b = 25$. Tais valores foram sugeridos em [24], em função de uma análise de sensibilidade lá conduzida.

3.5 Testes computacionais

Diferentes algoritmos NDRC foram desenvolvidos e testados para o MDP ao longo desta pesquisa. Nestes experimentamos: a dualização de Desigualdades Clique; a dualização de casos especiais destas contendo de 3 a 10 vértices, separados de maneira exata ou heurística; a dualização de casos especiais das Desigualdades Corte contendo de 5 a 10 vértices, separados de maneira exata ou heurística. Também experimentamos a separação de *Gcut* e *Gclique* ortogonais entre si, bem como dualização simultânea de mais de uma desigualdade de cada tipo, à cada iteração. A maioria desses experimentos foram mal sucedidos e suas propostas foram eventualmente descartadas. Dentre as restantes, as de melhor desempenho foram mantidas e melhoradas. A seguir apresentamos resultados computacionais para algumas delas.

Em particular, cinco algoritmos NDRC são analisados: NDRC_S, NDRC_3Cut, NDRC_4Cut, NDRC_GCut e NDRC_GCut_F. Uma breve descrição de cada um deles é feita a seguir:

- NDRC_S: utiliza diretamente a formulação (3.9), sem o reforço de qualquer uma das desigualdades válidas que discutimos no Capítulo 2 para o MDP. O algoritmo, no entanto, dualiza dinamicamente as restrições (2.5) e (2.6), separando-as de maneira exata, por inspeção.

- NDRC_3Cut: utiliza a formulação (3.9) reforçada pelas desigualdades 3Cut. Para separação, utiliza o algoritmo descrito no Capítulo 2 especializado para 3Cut.

- NDRC_4Cut: utiliza a formulação (3.9) reforçada pelas desigualdades 4Cut. Para separação, utiliza o algoritmo descrito no Capítulo 2 especializado para 4Cut.

- NDRC_Gcut: utiliza a formulação (3.9) reforçada pelas desigualdades GCut, (2.13). Para separação, utiliza o algoritmo de separação rápida para GCut, proposto no Capítulo 2. Como entrada para o mesmo, utiliza o grafo suporte reduzido.

- NDRC_Gcut_F: mesmo esquema descrito acima para NDRC_Gcut, diferindo deste apenas pelo uso do grafo suporte completo, como entrada para o algoritmo de separação.

Nossos algoritmos NDRC foram testados como heurísticas Lagrangeanas para o MDP e comparados com as melhores heurísticas existentes para o problema. A saber: *Integrated Tabu Search* (ITS) ([15]), *Variable Neighbohood Search* (VNS) ([16]), *Tuned Iterated Greedy* (TIG) ([33]), *Learneble Tabu Search with Estimation of Distribution Algorithm* (LTS-EDA) ([34]) e *Tabu Search/Memetic Algorithm* (TS/MA) ([24]).

Os seguintes conjuntos de instâncias, já utilizados em comparações prévias feitas na literatura, definiram a nossa base de testes:

- SOM-b: Conjunto de 20 instâncias introduzido em [35]. As dimensões envolvidas são as seguintes: n=100, k=10, 20, 30 e 40; n=200, k=20, 40, 60 e 80; n=300, k = 30, 60, 90 e 120; n=400, k=40, 80, 120 e 160; n=500, k=50, 100, 150 e 200. Diversidades entre cada par de itens foram geradas aleatoriamente como um número inteiro no intervalo $[0, 9]$.

- GKD-c: Conjunto de 20 instâncias com n=500 e k=50 introduzido em [15]. Itens são definidos como pontos no $\mathbb{R}_+^{10}$ com coordenadas geradas aleatoriamente no intervalo de 0 a 10. Diversidades são definidas pelas distâncias Euclidianas entre cada par de itens/pontos.

- MDG-a (1-20): Conjunto de 20 instâncias introduzido em [15]. Instâncias têm $n = 500$ e $k = 50$ e diversidades geradas aleatoriamente no intervalo $[0, 10]$.

- MDG-b: Conjunto de 40 instâncias introduzido em [15]. Vinte instâncias têm $n = 500$ e $k = 50$, e as demais têm $n = 2000$ e $k = 200$. Diversidades são geradas aleatoriamente no intervalo $[0, 10]$

- Type1_22 (MDG-a 21-40): Conjunto de 20 instâncias introduzido em [15]. Estas têm $n = 2000$ e $k = 200$, e diversidade gerada aleatoriamente no intervalo $[0, 10]$.

- b2500: Conjunto de 10 instâncias definidas inicialmente para o problema Unconstrained Binary Quadratic Programming (UBQP). Têm $n = 2500$ e $k = 1000$ e diversidades geradas aleatoriamente no intervalo $[-100, 100]$.

As instância SOM-b, GKD-c, MDG-a (Type1_22) e MDG-b estão armazenadas em http://www.optsicom.es/mdp e as instâncias b2500 em http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Nossos algoritmos foram implementados em C++ e testados em um computador equipado com um processador Intel Xeon X5675, rodando em *thread* única a 3.06GHz, com 48 GB de memória RAM disponível. Para cada algoritmo, os parâmetros do MS foram definidos de forma a permitir a realização de todas as

iterações previstas para o método, dentro de um limite de tempo pré-estabelecido. Agindo de forma diferente, a convergência do método poderia ser comprometida. A seguir indicamos as configurações do MS utilizadas em cada caso.

Nos resultados das Tabelas 3.1 e 3.2, utilizamos os algoritmos NDRC_S, NDRC_-3Cut e NDRC_4Cut sob os seguintes parâmetros para o MS: maxiter=3000, $\epsilon = 2$ e maxhalf=150. Para os algoritmos NDRC_Gcut e NDRC_Gcut_F os parâmetros utilizados foram os seguintes: maxiter=3000, $\epsilon = 0.125$ e maxhalf=150.

Os resultados das Tabelas 3.3 3.4 e 3.5 estão associados às instâncias ,Type1_22 e b2500, utilizando os seguintes parâmetros para nossos algoritmos: maxiter=20, $\epsilon = 2$ e maxhalf=2. Os resultados apresentados para os demais algoritmos foram retirados diretamente de [24].

Nos resultados da Tabela 3.6, o algoritmo NDRC_S operou sob os seguintes parâmetros: maxiter=800, $\epsilon = 2$ e maxhalf=40. Para o algoritmo NDRC_3Cut utilizamos: maxiter=400, $\epsilon = 2$ e maxhalf=20; para o algoritmo NDRC_4Cut utilizamos: maxiter=200, $\epsilon = 2$ e maxhalf=10; e finalmente, para o algoritmo NDRC_Gcut utilizamos: maxiter=400, $\epsilon = 0.125$ e maxhalf=20.

As Tabelas 3.1 e 3.2 indicam os limites duais obtidos pelos algoritmos NDRC_S, NDRC_3Cut, NDRC_4Cut, NDRC_GCut e NDRC_GCut_F para 4 conjuntos distintos de instâncias, de dimensões variadas. Nas tabelas, a coluna LB fornece o valor da melhor solução conhecida para a instância. Em todos os casos testados, nossos algoritmos conseguiram chegar a elas e, dessa maneira, omitimos das tabelas a apresentação individual desse resultado. A coluna UB indica os limites superiores obtidos. A coluna Gap apresenta a diferença percentual entre valores correspondentes de UB e LB, tomando o limite inferior como referência. Finalmente, a coluna Time indica o tempo de CPU, em segundos. Para obtenção dos resultados, um limite de tempo de 7200 segundos foi imposto. Em todas as tabelas, os melhores resultados são apresentados em negrito.

Na Tabela 3.1, o algoritmo NDRC_GCut obtém os melhores limites duais para a maioria das instâncias. No entanto, como a separação de GCut é realizada de maneira heurística no algoritmo, nas instâncias em que o valor de k é pequeno, os limites duais encontrados se mostraram mais fracos que aqueles obtidos pelos algoritmos NDRC_3Cut e NDRC_4Cut, que dualizam diretamente as desigualdades 3Cut ou 4Cut, respectivamente, estas separadas de maneira exata. Podemos observar que a qualidade dos limites duais obtidos é influenciada não apenas pela dimensão das instâncias, mas também por seus tipos. Para todos os algoritmos testados, os valores da coluna Gap são muito menores para as instâncias GKD-c do que para as instâncias MDG-a e MDG-b, apesar de suas dimensões serem as mesmas. Por outro lado, instâncias com dimensões muito inferiores às de GKD-c, como aquelas em SOM-B_2 e SOM-b_5, apresentam valores de Gap muito mais altos. Uma conclusão

que se pode tirar desses resultados é que nossos algoritmos NDRC funcionam bem como heurísticas Lagrangeanas, mas se comportam mal como algoritmos exatos.

Instâncias	n	k	LB	NDRC_S			NDRC_3Cut			NDRC_4Cut			NDRC_GCut		
				UB	Gap	Time	UB	Gap	Time	UB	Gap	Time	UB	Gap	Time
SOM-b_1	100	10	333	402,99	21,0	10,8	389,61	17,0	8,2	386,98	**16,2**	8,5	394,78	18,6	9,0
SOM-b_2		20	1195	1605,45	34,3	17,8	1454,15	21,7	15,2	1399,93	17,1	15,6	1385,90	**16,0**	24,8
SOM-b_3		30	2457	3431,04	39,6	23,2	2808,24	14,3	25,2	2884,36	17,4	23,4	2668,87	**8,6**	64,9
SOM-b_4		40	4142	5788,74	39,8	31,7	4566,26	10,2	43,5	4822,57	16,4	40,6	4398,95	**6,2**	121,7
SOM-b_5	200	20	1247	1702,38	36,5	50,4	1631,70	30,9	37,8	1629,86	**30,7**	37,6	1657,04	32,9	43,9
SOM-b_6		40	4450	6598,18	48,3	85,5	5857,17	31,6	77,7	5844,39	31,3	77,0	5464,21	**22,8**	179,2
SOM-b_7		60	9437	14043,68	48,8	112,5	11482,75	21,7	212,4	11578,62	22,7	160,0	10564,99	**12,0**	583,2
SOM-b_8		80	16225	23698,08	46,1	134,7	18533,30	14,2	420,8	19410,75	19,6	346,3	17385,52	**7,2**	1204,0
SOM-b_9	300	30	2694	3896,36	44,6	113,1	3715,09	37,9	92,3	3737,32	38,7	85,2	3692,58	**37,1**	125,4
SOM-b_10		60	9689	14989,81	54,7	231,1	13474,51	39,1	234,9	13498,25	39,3	229,4	12130,60	**25,2**	705,0
SOM-b_11		90	20743	31922,50	53,9	319,0	26124,78	25,9	787,7	26772,14	29,1	604,6	23416,29	**12,9**	2441,4
SOM-b_12		120	35881	53557,55	49,3	351,1	41856,06	16,7	1443,1	44938,62	25,2	1587,3	38630,73	**7,7**	4363,6
SOM-b_13	400	40	4658	6998,38	50,2	208,2	6670,32	43,2	175,8	6597,55	**41,6**	167,5	6597,76	**41,6**	273,4
SOM-b_14		80	16956	26790,35	58,0	327,1	23816,12	40,5	490,4	24388,40	43,8	514,4	21205,26	**25,1**	1940,7
SOM-b_15		120	36317	57006,55	57,0	434,8	46633,45	28,4	1601,9	48615,12	33,9	1631,8	41128,97	**13,2**	5956,4
SOM-b_16		160	62487	95206,34	52,4	602,8	74306,22	18,9	4123,1	82734,60	32,4	4817,4	67836,07	**8,6**	7202,4
SOM-b_17	500	50	7141	10986,99	53,9	299,8	10449,67	46,3	306,1	10378,70	45,3	283,3	10248,40	**43,5**	505,3
SOM-b_18		100	26258	41973,37	59,8	526,4	37285,03	42,0	1004,1	36791,89	40,1	1027,8	32588,78	**24,1**	4228,9
SOM-b_19		150	56572	89218,11	57,7	793,9	73197,88	29,4	4237,5	74931,18	32,5	3750,2	63735,42	**12,7**	7204,0
SOM-b_20		200	97344	148936,19	53,0	1301,2	116716,70	19,9	7200,1	140595,95	44,4	7201,2	104921,40	**7,8**	7204,9
	Média				47,9	298,8		27,5	1126,9		30,9	1130,5		**19,2**	2219,1
GKD-c_1	500	50	19485,18	21547,45	10,6	185,0	20827,63	6,9	228,5	20653,68	6,0	246,1	20321,28	**4,3**	375,1
GKD-c_2			19701,53	21707,73	10,2	184,6	20967,65	6,4	236,9	20817,14	5,7	282,5	20353,38	**3,3**	434,6
GKD-c_3			19547,20	21537,47	10,2	179,3	20729,59	6,0	234,0	20638,32	5,6	286,2	20308,98	**3,9**	353,7
GKD-c_4			19596,46	21531,30	9,9	170,6	20790,29	6,1	221,0	20731,10	5,8	288,0	20298,94	**3,6**	343,9
GKD-c_5			19602,62	21582,58	10,1	188,5	20720,48	5,7	214,7	20813,70	6,2	249,3	20424,97	**4,2**	318,4
	Média				10,2	181,6		6,2	227,0		5,8	270,4		**3,9**	365,1
MDG-a_1	500	50	7833,83	11644,25	48,6	246,4	11240,18	43,5	322,8	11174,37	42,6	413,4	11060,37	**41,2**	484,1
MDG-a_2			7771,66	11649,45	49,9	231,6	11148,76	43,5	328,4	11178,15	43,8	374,5	10953,45	**40,9**	488,3
MDG-a_3			7759,36	11652,63	50,2	259,0	11245,86	44,9	324,8	11179,13	44,1	378,5	11048,33	**42,4**	496,3
MDG-a_4			7770,24	11649,10	49,9	287,9	11156,28	43,6	329,1	11179,98	43,9	335,7	11083,11	**42,6**	437,3
MDG-a_5			7755,23	11631,04	50,0	298,8	11235,27	44,9	329,5	11176,46	44,1	333,6	11029,28	**42,2**	465,5
	Média				49,7	264,7		44,1	326,9		43,7	367,1		**41,9**	474,3
MDG-b_1	500	50	778030,57	1164246,48	49,6	209,8	1115096,64	43,3	220,2	1117370,05	43,6	325,7	1105090,55	**42,0**	445,4
MDG-b_2			779963,54	1163787,62	49,2	287,8	1123278,14	44,0	222,8	1116504,72	43,1	324,0	1107231,45	**42,0**	423,3
MDG-b_3			776768,17	1163565,65	49,8	260,5	1123266,03	44,6	209,9	1117056,78	43,8	303,6	1103563,50	**42,1**	442,4
MDG-b_4			775394,47	1164016,85	50,1	288,9	1123763,08	44,9	216,0	1117346,83	44,1	309,0	1111575,60	**43,4**	398,8
MDG-b_5			775610,96	1165084,28	50,2	293,3	1115427,06	43,8	224,1	1119055,58	44,3	309,3	1106209,14	**42,6**	435,6
	Média				49,8	268,1		44,1	218,6		43,8	314,3		**42,4**	429,1

Tabela 3.1: Limites duais e primais gerados pelos algoritmos NDRC

A Tabela 3.2 compara o impacto da utilização das duas formas distintas de geração do grafo suporte de soluções, discutidas na Seção 3.3.3, para uso nos algoritmos de separação. A informação disponibilizada para a tabela é a mesma descrita para a Tabela 3.1. Tanto NDRC_GCut quanto NDRC_GCut_F dualizam as desigualdades GCut. O primeiro, no entanto, utiliza o grafo suporte reduzido para esta tarefa, enquanto o segundo faz uso do grafo suporte completo (ver detalhes na Seção 3.3.3). Na tabela, podemos verificar que o algoritmo NDRC_GCut obteve melhores limites duais em menos tempo de CPU. Em particular, isso se aplica de forma muito expressiva à instâncias SOM-b_20. Uma das hipóteses levantadas para justificar esse comportamento é de que as desigualdades contendo vértices associados ao grafo suporte reduzido têm uma certa relação de equivalência com as desigualdades contendo vértices externos àquele grafo. Ou seja, se nenhuma das desigualdades internas ao grafo suporte é violada, não existirá desigualdade violada envolvendo vértices externos ao mesmo. Dessa forma, conduzir a separação sob o grafo suporte reduzido parece possibilitar a identificação de desigualdades mais relevantes, em

menor tempo de CPU.

Instâncias	n	k	LB	NDRC_Gcut			NDRC_Gcut_F		
				UB	Gap	Time	UB	Gap	Time
SOM-b_1	100	10	333	394,78	18,6	9,0	380,85	**14,4**	58,3
SOM-b_2		20	1195	1385,90	**16,0**	24,8	1463,56	22,5	219,7
SOM-b_3		30	2457	2668,87	**8,6**	64,9	2856,07	16,2	422,6
SOM-b_4		40	4142	4398,95	**6,2**	121,7	4487,50	8,3	717,9
SOM-b_5	200	20	1247	1657,04	**32,9**	43,9	1702,04	36,5	435,5
SOM-b_6		40	4450	5464,21	**22,8**	179,2	6191,44	39,1	1249,8
SOM-b_7		60	9437	10564,99	**12,0**	583,2	12104,60	28,3	2000,3
SOM-b_8		80	16225	17385,52	**7,2**	1204,0	19130,17	17,9	2365,0
SOM-b_9	300	30	2694	3692,58	**37,1**	125,4	3905,75	45,0	1231,9
SOM-b_10		60	9689	12130,60	**25,2**	705,0	14403,64	48,7	2185,4
SOM-b_11		90	20743	23416,29	**12,9**	2441,4	29204,48	40,8	2540,5
SOM-b_12		120	35881	38630,73	**7,7**	4363,6	44981,51	25,4	3909,7
SOM-b_13	400	40	4658	6597,76	**41,6**	273,4	7012,54	50,5	2533,5
SOM-b_14		80	16956	21205,26	**25,1**	1940,7	25832,57	52,4	3735,1
SOM-b_15		120	36317	41128,97	**13,2**	5956,4	54231,91	49,3	4094,7
SOM-b_16		160	62487	67836,07	**8,6**	7202,4	87771,89	40,5	5097,2
SOM-b_17	500	50	7141	10248,40	**43,5**	505,3	11006,95	54,1	4845,7
SOM-b_18		100	26258	32588,78	**24,1**	4228,9	41706,85	58,8	4408,4
SOM-b_19		150	56572	63735,42	**12,7**	7204,0	86409,22	52,7	6364,7
SOM-b_20		200	97344	104921,40	**7,8**	7204,9	145132,29	49,1	7200,4
Média					**19,2**	2219,1		37,5	2780,8

Tabela 3.2: Algoritmos NDRC com desigualdades Gcut

Nas Tabelas 3.3, 3.4 e 3.5, para cada algoritmo testado, trinta execuções foram efetuadas para cada instância Type1_22 e quinze execuções foram efetuadas para cada instância b2500. Este tipo de procedimento é padrão para meta-heurísticas e se justifica, para nossos algoritmos NDRC, pela existência de operações de natureza randômica em suas heurísticas primais. A Tabela 3.3 está restrita aos algoritmos NDRC e as demais envolvem algoritmos da literatura e nosso melhor algoritmo NDRC. Os resultados apresentados para cada instância são relativos tanto à melhor execução obtida quanto à média de todas as execuções. A informação disponibilizada diz respeito ao valor das soluções primais obtidos. Em particular, os resultados apresentados para os algoritmos ITS, VNS, TIG, LTS-EDA e TS/MA são aqueles obtidos em [24], onde tempos limites de 17 segundos e de 256 segundos foram impostos à cada execução dos mesmos, respectivamente para as instâncias Type1_22 e b2500. Para possibilitar uma comparação, ainda que superficial, entre os tempos de CPU em [24] e nossos tempos de CPU, utilizamos o Standard Performance Evaluation Corporation (SPEC) [36]. A máquina que utilizamos, segundo o SPEC, apresenta uma pequena vantagem de desempenho a um fator de 1.7. Nos testes com nossos algoritmos NDRC, os limites de tempo foram configurados, então, para

10 segundos e 150 segundos, respectivamente, para as instâncias Type1_22 e b2500. O valor absoluto das soluções é muito grande. Para facilitar a visualização dos resultados, apresentamos nas colunas Best e Avg a diferença entre o valor da melhor solução conhecida, indicado na coluna BKS, e o valor obtido, respectivamente, na melhor execução do algoritmo e na média das execuções. Dessa forma, os melhores resultados são aqueles que mais se aproximam de zero.

Instância	BKS	NDRC_S		NDRC_3cut		NDRC_4cut		NDRC_Gcut	
		Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
Type1_22.1	114271	5	80,70	**0**	98,53	60	183,50	7	90,63
Type1_22.2	114327	**0**	91,77	**0**	83,00	**0**	113,13	**0**	149,77
Type1_22.3	114195	21	98,53	**0**	105,83	31	127,87	10	116,93
Type1_22.4	114093	6	92,43	6	111,07	6	144,87	6	85,63
Type1_22.5	114196	18	125,77	73	145,03	91	177,47	16	137,57
Type1_22.6	114265	**0**	39,23	7	55,37	24	55,50	**0**	67,70
Type1_22.7	114361	**0**	18,70	**0**	28,87	**0**	36,07	**0**	20,37
Type1_22.8	114327	**0**	105,20	**0**	147,10	**0**	139,93	**0**	111,27
Type1_22.9	114199	3	66,07	16	91,37	16	109,23	**0**	75,17
Type1_22.10	114229	**0**	75,17	7	80,63	49	115,37	67	88,10
Type1_22.11	114214	15	72,63	34	101,33	15	73,47	29	106,87
Type1_22.12	114214	**0**	50,37	15	47,87	**0**	57,17	**0**	20,50
Type1_22.13	114233	22	92,57	**0**	85,27	52	162,30	33	133,03
Type1_22.14	114216	**0**	118,53	**0**	60,77	**0**	141,90	**0**	50,10
Type1_22.15	114240	2	18,50	1	43,43	2	28,07	1	51,03
Type1_22.16	114335	9	51,77	8	45,20	43	65,50	43	51,00
Type1_22.17	114255	2	73,03	2	76,40	39	114,67	17	124,93
Type1_22.18	114408	**0**	65,10	**0**	73,70	4	133,37	2	110,90
Type1_22.19	114201	**0**	37,33	**0**	19,83	**0**	50,80	**0**	111,90
Type1_22.20	114349	**0**	79,37	6	111,13	**0**	135,83	**0**	117,70
Av.		5,2	72,64	8,8	80,59	21,6	108,30	11,6	91,06

Tabela 3.3: Algoritmos NDRC nas instâncias Type1_22 executando por 10 s

A Tabela 3.3 mostra os resultados obtidos para quatro das nossas heurísticas NDRC para o MDP. Vemos que o método que apresenta melhores resultados para o limite de tempo utilizado é o NDRC_S. Esse comportamento pode estar relacionado com o fato de que, nos algoritmos NDRC_3Cut, NDRC_4Cut e NDRC_GCut mais desigualdades são tratadas e a demanda computacional é maior. A limitação de tempo não permite uma quantidade satisfatória de iterações, e isso pode desfavorecer a convergência e consequentemente a qualidade das soluções primais encontradas.

Na Tabela 3.4 Vemos que o NDRC_S é competitivo com as melhores heurísticas propostas para o MDP, perdendo apenas para o TS/MA e para o LTS-EDA. Podemos observar que o NDRC_S apresenta a terceira melhor média de aproximação da melhor solução. Na Tabela 3.5 nosso algoritmo supera o LTS-EDA e perde apenas para o TS/MA em melhor solução obtida. Em média, no entanto, o LTS-EDA ainda supera nosso algoritmo.

Na Tabela 3.6 são mostrados resultados para quatro das nossas heurísticas NDRC

Instância	BKS	ITS(2007)		VNS (2009)		TIG(2011)		LTS-EDA(2012)		TS/MA (2014)		NDRC_S	
		Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
Type1_22.1	114271	65	209,87	48	150,60	48	101,57	5	60,73	**0**	10,37	5	80,70
Type1_22.2	114327	29	262,27	**0**	168,87	**0**	69,90	**0**	89,87	**0**	8,80	**0**	91,77
Type1_22.3	114195	69	201,40	19	110,83	5	117,77	**0**	98,97	**0**	8,53	21	98,53
Type1_22.4	114093	22	200,53	70	188,13	58	141,93	**0**	79,87	**0**	19,90	6	92,43
Type1_22.5	114196	95	273,27	87	184,10	99	194,70	51	134,47	**0**	29,50	18	125,77
Type1_22.6	114265	41	168,17	30	99,30	9	96,20	**0**	40,17	**0**	15,60	**0**	39,23
Type1_22.7	114361	12	167,47	**0**	56,30	**0**	71,27	**0**	18,20	**0**	0,00	**0**	18,70
Type1_22.8	114327	25	256,40	**0**	163,33	**0**	193,60	**0**	159,10	**0**	25,23	**0**	105,20
Type1_22.9	114199	9	139,83	16	78,47	16	80,37	**0**	70,97	**0**	7,83	3	66,07
Type1_22.10	114229	24	204,93	7	139,33	35	121,43	**0**	56,20	**0**	4,10	**0**	75,17
Type1_22.11	114214	74	237,77	42	145,13	59	139,57	3	69,87	**0**	24,30	15	72,63
Type1_22.12	114214	55	249,53	95	143,30	88	156,00	15	84,93	**0**	21,50	**0**	50,37
Type1_22.13	114233	93	279,87	22	168,07	42	167,40	6	85,30	**0**	1,23	22	92,57
Type1_22.14	114216	92	248,50	117	194,30	64	202,80	**0**	81,00	**0**	3,57	**0**	118,53
Type1_22.15	114240	11	117,50	1	62,87	6	80,53	**0**	22,03	**0**	1,73	2	18,50
Type1_22.16	114335	11	225,40	42	215,43	35	67,90	**0**	36,47	**0**	7,27	9	51,77
Type1_22.17	114255	56	217,53	**0**	170,00	18	144,53	6	57,07	**0**	11,73	2	73,03
Type1_22.18	114408	46	169,97	**0**	57,00	2	117,37	2	22,83	**0**	1,00	**0**	65,10
Type1_22.19	114201	34	243,20	**0**	124,60	**0**	144,37	**0**	35,87	**0**	4,00	**0**	37,33
Type1_22.20	114349	151	270,67	65	159,43	45	187,23	**0**	95,40	**0**	15,60	**0**	79,37
Av.		50,7	217,20	33,1	138,97	31,5	129,82	4,4	69,97	**0**	11,09	5,2	72,64

Tabela 3.4: Heurísticas MDP para as instâncias Type1_22

Instância	BKS	ITS(2007)		VNS (2009)		TIG(2011)		LTS-EDA(2012)		TS/MA (2014)		NDRC_S	
		Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
b2500-1	1153068	624	3677,33	96	1911,93	42	1960,33	**0**	369,20	**0**	0,00	42	461,60
b2500-2	1129310	128	3677,33	88	1034,33	109	1958,47	154	453,53	**0**	73,87	**0**	746,40
b2500-3	1115538	316	3281,93	332	1503,67	34	2647,87	**0**	290,40	**0**	184,73	**0**	344,13
b2500-4	1147840	870	2547,93	436	1521,07	910	1937,13	**0**	461,73	**0**	159,00	**0**	895,60
b2500-5	1144756	356	1800,27	**0**	749,40	674	1655,87	**0**	286,07	**0**	45,20	**0**	310,00
b2500-6	1133572	250	2173,47	**0**	1283,53	964	1807,60	80	218,00	**0**	54,40	112	611,87
b2500-7	1149064	306	1512,70	116	775,47	76	1338,73	44	264,60	**0**	65,00	**0**	241,33
b2500-8	1142762	**0**	247,73	96	862,47	588	1421,53	22	146,47	**0**	1,20	22	813,73
b2500-9	1138866	642	2944,67	54	837,07	658	1020,60	6	206,33	**0**	0,00	**0**	249,33
b2500-10	1153936	598	2024,60	278	1069,40	448	1808,73	94	305,27	**0**	0,00	**0**	372,40
Av.		409	2388,796	149,6	1154,834	450,3	1755,686	40	300,16	**0**	58,34	17,6	504,64

Tabela 3.5: Heurísticas MDP para as instâncias b2500

Instância	BKS	NDRC_S		NDRC_3cut		NDRC_4cut		NDRC_Gcut	
		LB	TTB	LB	TTB	LB	TTB	LB	TTB
Type1_22.1	114271	2	1047,436	5	373,598	14	246,614	**0**	48,976
Type1_22.2	114327	**0**	59,693	**0**	33,408	**0**	167,974	**0**	289,23
Type1_22.3	114195	**0**	230,715	**0**	64,445	**0**	720,743	**0**	110,427
Type1_22.4	114093	10	806,981	7	27,638	12	871,494	2	366,335
Type1_22.5	114196	**0**	626,021	16	491,392	**0**	985,888	3	405,657
Type1_22.6	114265	**0**	8,143	**0**	500,073	**0**	170,961	**0**	539,973
Type1_22.7	114361	**0**	35,734	**0**	48,735	**0**	290,72	**0**	72,742
Type1_22.8	114327	**0**	1,606	**0**	111,544	**0**	88,59	**0**	151,115
Type1_22.9	114199	**0**	152,539	2	65,62	24	500,449	2	591,268
Type1_22.10	114229	**0**	397,087	**0**	477,86	3	837,559	**0**	14,953
Type1_22.11	114214	**0**	303,558	**0**	383,931	33	930,108	**0**	229,785
Type1_22.12	114214	15	433,456	**0**	57,351	17	32,093	17	628,713
Type1_22.13	114233	**0**	204,235	13	14,84	4	63,929	**0**	516,64
Type1_22.14	114216	**0**	243,232	**0**	1,806	**0**	1,794	**0**	1,853
Type1_22.15	114240	**0**	293,558	1	261,675	1	129,128	1	44,517
Type1_22.16	114335	**0**	302,318	8	151,617	8	108,472	**0**	119,959
Type1_22.17	114255	**0**	364,628	2	151,852	**0**	654,702	14	289,866
Type1_22.18	114408	**0**	171,875	**0**	100,578	**0**	134,139	**0**	122,675
Type1_22.19	114201	**0**	66,492	**0**	11,045	**0**	13,85	**0**	160,92
Type1_22.20	114349	**0**	126,715	**0**	343,094	**0**	183,041	**0**	605,896
Av.		**1,35**	293,8011	2,7	183,6051	5,8	356,6124	1,95	265,575

Tabela 3.6: Algoritmos NDRC nas instâncias Type1_22 executando por 1200 s

rodando uma única vez para cada instância Type1_22 com limite de tempo de 1200 segundos. Assim como praticado anteriormente, o valor da melhor solução conhecida para cada instância é dado na coluna *BKS*. Os resultados apresentados nas colunas *LB* indicam a diferença entre o limite primal obtido naquela execução do algoritmo e o valor da melhor solução conhecida para a instância. A coluna *TTB* indica o tempo de CPU gasto pelo algoritmo até encontrar a melhor solução fornecida pelo mesmo. Podemos ver que, nessas condições, o desempenho das heurísticas são mais balanceados. O método que apresenta melhores soluções encontradas ainda é o NDRC_S, seguido pelo NDRC_Gcut. Podemos supor que, quando os métodos apresentam uma melhor convergência, isso leva a melhores soluções primais e aumenta as chances de se obter a melhor solução conhecida.

Capítulo 4

Algoritmos exatos para o MDP

Dada uma formulação "compacta" para um problema de Programação Inteira (PI),

$$z = \max\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\}, \tag{4.1}$$

onde c é um vetor n-dimensional, A é uma matriz $m \times n$ e b é um vetor m-dimensional, o conjunto de soluções viáveis para esse problema é definido por

$$X = \{Ax \leq b, x \in \mathbb{Z}_+^n\} \tag{4.2}$$

e a envoltória convexa dessas soluções, $Conv(X) = \{\overline{A}x \leq \overline{b}, x \geq 0\}$, também define um poliedro [1].

Dessa maneira, é possível então formular, em tese, um problema de Programação Linear (PL)

$$\overline{z} = \max\{cx : \overline{A}x \leq \overline{b}, x \geq 0\}, \tag{4.3}$$

tal que uma solução ótima para (4.3) é uma solução ótima para (4.1). Esse resultado também é válido para problemas de Programação Inteira Mista (PIM) [1], como é o caso do MDP, onde uma parte das variáveis deve assumir valores inteiros, enquanto as restantes devem assumir valores reais.

Na prática, no entanto, para problemas NP-Difíceis não existe a possibilidade de se obter uma descrição completa da envoltória convexa do problema, a menos que P = NP [1]. Assim sendo, o que se pode fazer é tentar obter a melhor aproximação possível de $Conv(X)$.

Ao longo dos anos, diversos algoritmos foram propostos para resolver o problema de PI. Dentre outros, o algoritmo Branch-and-Bound, proposto originalmente em [37], resolve o problema por divisão e conquista. Algoritmos de planos de corte [38], por sua vez, tentam se aproveitar das propriedades poliedrais de uma dada formulação para gerar desigualdades válidas adicionais e assim gerar/aproximar uma descrição completa de $Conv(X)$ no entorno do ótimo. O algoritmo Branch-and-

Cut, o mais bem sucedido para resolver problemas de PI até o momento, é uma combinação dos dois algoritmos que acabamos de descrever.

A seguir, apresentamos de forma sucinta o funcionamento geral dos três algoritmos mencionados acima, detalhando um pouco mais o algoritmo Branch-and-Cut. Na sequência, descrevemos nosso algoritmo Branch-and-Cut desenvolvido especificamente para o MDP.

4.1 Branch-and-Bound

Uma forma de se resolver um problema de PI/OC é decompondo-o em subproblemas menores e mais fáceis de se resolver, de forma a que as soluções desses subproblemas nos levem à solução do problema original. Tomando por base a formulação (4.1), seja $X = X_1 \cup \cdots \cup X_T$ uma decomposição do conjunto de soluções viáveis para o nosso problema e $z^t = max\{cx : x \in X_t\}$ para $l = 1, \ldots, T$ o conjunto de subproblemas associado a essa decomposição. Temos então que $z = max_t z^t$ [1].

A decomposição sugerida acima é normalmente representada por uma estrutura de "árvore de enumeração". A formulação do problema como um todo está associado ao nó raiz dessa árvore. Normalmente, o espaço de soluções é decomposto inicialmente em duas regiões menores, $X = X_0 \cup X_1$, e dois subproblemas são assim definidos e representados como filhos do nó raiz. O procedimento é então estendido, de tal forma que $X_0 = X_{00} \cup X_{01}$ e $X_1 = X_{10} \cup X_{11}$ resultam, dando origem a subproblemas cada vez mais restritos. Essa decomposição continua até se chegar, em um caso extremo, em regiões de viabilidade que contêm um único ponto ou são vazias. Estas, por sua vez, estão associadas a folhas da árvore de enumeração.

Os algoritmos Branch-and-Bound utilizam a decomposição descrita acima para resolver problemas de PI/OC de forma implícita. Nesse sentido, limites duais e primais são utilizados em cada nó da árvore de anumeração para abreviar o processo de enumeração. Dessa forma, tão logo se tenha informação de que um dado subproblema não leva a melhores soluções que aquelas já conhecidas, o processo de decomposição é interrompido para seus nós descendentes.

Especificamente, em cada nó da árvore de enumeração Branch-and-Bound é resolvida a relaxação linear da formulação do problema de PI/OC ali definido. Esta fornece um limite dual para o valor ótimo, inteiro, desse subproblema. No caso de um problema de maximização, isto representa um limite superior para o valor da solução ótima inteira para o mesmo. Quando o subproblema é inviável ou o valor de sua relaxação linear é dominado pelo valor de uma solução primal já disponível para nosso problema inicial, o nó é então "podado", ou seja, interrompe-se o processo de decomposição à partir do mesmo. Caso contrário, se a solução obtida é fracionária e seu valor não é dominado pelo valor da melhor solução primal disponível, o nó é "ra-

mificado", dando origem a dois novos subproblemas, mais restritos. Tais problemas são representados como filhos daquele que se está investigando.

Bons limites duais implicam em poda de nós mais próximos do nó raiz. Do mesmo modo, limites duais fracos implicam em podas mais distantes daquele nó, gerando, consequentemente, árvores de enumeração maiores. O método se beneficia então da utilização de formulações que levem a limites duais mais fortes. Em outras palavras, quanto mais próximo do valor ótimo, inteiro, forem os limites duais obtidos, mais bem sucedido será o método. Com esse intuito, utilizar uma formulação que nos aproxima de $Conv(X)$ seria, em tese, vantajoso. No entanto, isso deve se confrontado com o número de restrições envolvidas nessa formulação. Se esse número for excessivamente grande, o esforço computacional para resolver sua relaxação linear pode ser muito alto. O equilíbrio entre esses dois aspectos que acabamos de discutir é fundamental para o sucesso de um algoritmo Branch-and-Bound.

4.2 Algoritmos de Planos de Corte

Como foi dito anteriormente, para problemas de PI/OC NP-Difíceis, é improvável obter uma formulação que descreve $Conv(X)$. No entanto, ao se trabalhar com uma instância específica do mesmo, não é incomum obter uma formulação que aproxima $Conv(X)$ no entorno de uma solução ótima para a mesma. Dessa forma, eventualmente, pode-se chegar até mesmo a uma formulação cuja relaxação linear é naturalmente inteira. Além disso, tal formulação geralmente não apresenta uma quantidade excessiva de restrições, o que torna possível a resolução de sua relaxação linear, na prática.

Em um algoritmo de "planos de corte", tenta-se iterativamente chegar a uma formulação mais forte para o problema, no sentido descrito acima. A cada iteração desse procedimento, a relaxação linear da formulação corrente é resolvida. Feito isso, para uma dada família de desigualdades válidas para o problema, algumas das desigualdades que violem a solução daquela relaxação são então adicionadas a ela, obtendo-se assim uma formulação mais forte para o problema. O procedimento é repetido de forma a construir, sequencialmente, relaxações lineares cada vez mais fortes, sem a necessidade de trabalhar com relaxações lineares de muito grande porte, obtidas, por exemplo, pela adição simultânea de todas as desigualdades daquela família, violadas ou não.

Especificamente, assuma que $\mathcal{D}$ é uma família de desigualdades válidas, $\pi x \leq \pi_0$, identificadas para o problema. Normalmente essas famílias contêm exponencialmente muitas desigualdades, de modo que adicioná-las todas à formulação, a priori, seria impraticável.

A princípio, a relaxação linear da formulação inicial do problema é resolvida

e uma solução ótima $\overline{x}$ é obtida para ela. Se $\overline{x}$ for naturalmente inteira, o algoritmo é interrompido, pois esta caracteriza uma solução ótima para o problema. Do contrário, $\overline{x}$ é fracionária e um algoritmo de separação é aplicado para se encontrar uma desigualdade $(\pi^t, \pi_0^t) \in \mathcal{D}$ tal que $\pi^t \overline{x} > \pi_0^t$ se aplica, ou seja, uma desigualdade válida que é violada por $\overline{x}$. Se nenhuma dessas desigualdades for encontrada, o algoritmo se encerra. Caso contrário, uma ou mais dessas desigualdades são adicionadas a formulação e o processo é repetido para a formulação resultante.

É importante ressaltar que o problema de separação deve ser fácil de resolver, seja de forma exata ou heurística, do contrário seria impraticável resolvê-lo a cada iteração do método. Note também que o algoritmo pode terminar sem que seja encontrada uma solução inteira para o problema. No entanto, mesmo que esta situação ocorra, a formulação obtida ao final é normalmente mais forte que a formulação inicial, e pode ser utilizada com vantagem em um algoritmo Branch-and-Bound.

4.3 Branch-and-Cut

Um algoritmo Branch-and-Cut utiliza o arcabouço de um algoritmo Branch-and-Bound, reforçado pela aplicação de algoritmos de planos de corte em cada nó de sua árvore de enumeração. Assim procedendo, limites duais, geralmente mais fortes, e limites primais são gerados em cada nó da árvore de enumeração para se proceder a operações de ramificação ou poda.

Dada uma formulação "compacta" para um problema de PI

$$\max\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\}, \tag{4.4}$$

suponha que sua relaxação linear leva a limites duais de baixa qualidade.

Assuma também que existe uma família de desigualdades válidas capaz de fortalecer a formulação (4.4), digamos:

$$Dx \leq g. \tag{4.5}$$

Finalmente, assuma também que D tem dimensão $m \times n$ e que m é uma função exponencial de n. Adicionar a família de desigualdades (4.5) à formulação (4.4) fortalece os limites duais obtidos em cada nó da árvore de enumeração. Tal conduta parece, de início, atrativa, já que, como discutido anteriormente, a qualidade do limite dual implica diretamente no tamanho da árvore de enumeração. No entanto, o esforço computacional para resolver, diretamente, um problema de programação linear com exponencialmente muitas desigualdades seria impraticável. É inviável, portanto, adicionar simultaneamente todas as desigualdades à formulação original e

aplicar diretamente o método Branch-and-Bound à formulação resultante. É nesse contexto que se faz necessária a utilização de um algoritmo de planos de corte.

Como vimos, em um algoritmo de planos de corte primeiro se resolve a relaxação linear do problema ou subproblema em mãos. Em seguida, caso a solução obtido não seja naturalmente inteira, aplica-se uma rotina de separação para identificar desigualdades válidas que violem a solução obtida. Tais desigualdades são adicionadas à formulação e uma nova relaxação, reforçada, é daí obtida. A seguir o procedimento é repetido. O método se encerra nos seguintes casos: nenhuma desigualdade foi encontrada pelo algoritmo de separação; a solução obtida para a relaxação linear é naturalmente inteira; ou um limite de iterações para o algoritmo de planos de corte foi atingido.

O algoritmo de planos de corte é aplicado em todo nó da árvore de enumeração e normalmente as desigualdades válidas por ele identificadas são armazenadas em um "pool de cortes gerados". O número de iterações para um tal algoritmo depende, dentre outros fatores, do impacto que ele traz em termos de melhora do limite dual. No entanto, tipicamente, aplica-se um maior número de iterações do algoritmo de planos de corte no nó raiz da árvore de enumeração.

Normalmente, mais de uma desigualdade é separada e adicionada a uma relaxação linear à cada iteração. Idealmente, um algoritmo de planos de corte deve tentar identificar desigualdades que não se sobreponham, ou seja, que não tenham uma interseção muito grande entre elas. Entretanto, a separação de muitas desigualdades à cada rodada pode não surtir o mesmo efeito que se esperaria ao separá-las gradativamente ao longo de mais de uma iteração.

Eventualmente, algumas desigualdades adicionadas a uma relaxação linear podem se tornar inativas. Denominamos "ativa" uma desigualdade que, se retirada de uma relaxação linear, seria violada por sua solução. Desigualdades se tornam inativas pela sobreposição ou dominação imposta por novas desigualdades que são geradas posteriormente. Periodicamente, desigualdades inativas devem ser removidas de uma relaxação linear, já que são desnecessárias para ela.

Um caso especial do algoritmo Branch-and-Cut ocorre quando uma família de desigualdades ou restrições utilizadas como corte são necessárias para formular o problema. Em outras palavras, a formulação sem essas desigualdades define um problema diferente do que se quer resolver. Um bom exemplo disso é um algoritmo Branch-and-Cut para o *Problema do Caixeiro Viajante* (PCV) utilizando as Desigualdades de Eliminação de Subrotas (SECs). Nesse caso, para se resolver a relaxação linear da formulação, um algoritmo de planos de corte, separando as SECs, deve ser aplicado.

A utilização do método Branch-and-Cut permitiu a resolução exata de instâncias do PCV envolvendo mais de 90.000 cidades [39]. Atualmente, os principais resolve-

dores comerciais de problemas de otimização inteira mista (CPlex, Gurobi, Xpress) utilizam o método Branch-and-Cut como opção padrão para resolver tais problemas. Os mesmos utilizam algumas famílias de cortes válidas para o problema genérico de PI, como os cortes de Gomory, por exemplo, e outros cortes válidos para certas estruturas de uso comum e fácil identificação, dentro de uma formulação qualquer. Além disso, permitem também a utilização de cortes definidos pelo usuário.

4.4 Algoritmos Branch-and-Cut para o MDP

Em nosso trabalho implementamos alguns algoritmos Branch-and-Cut específicos para o MDP. Cada um desses algoritmos utiliza como corte diferentes combinações das desigualdades GClique e GCut, discutidas no Capítulo 2. Estas contêm um número exponencialmente grande de desigualdades, o que justifica utilizá-las como planos de corte. Fazemos isso até mesmo para os casos particulares 3Cut e 4Cut das desigualdades GCut. Isso porque, mesmo para estas, seria impraticável adicioná-las todas simultaneamente à nossa formulação do MDP. Como exemplo, para uma instância do problema definida sobre um grafo $G = (V, E)$ com $|V| = 100$, teríamos respectivamente 100^3 e 100^4 desigualdades 3Cut e 4Cut a adicionar, o que demandaria um tempo de CPU excessivo para resolver as relaxações lineares correspondentes. A Tabela 4.1 mostra como a resolução da relaxação linear das formulações utilizando todas as desigualdades 3Cut e 4Cut simultaneamente se torna impraticável, mesmo para instâncias pequenas. As instâncias utilizadas para demonstrar isso são as mesmas utilizadas para testar o algoritmo exato proposto em [3], o melhor deste tipo existente na literatura para o MDP. Tais instâncias são também aquelas em que iremos testar nossos algoritmos exatos. Para tanto, um tempo limite de CPU de 7200 segundos foi imposto. Para as instâncias com o símbolo "-" aparecendo na tabela, não foi possível resolver a relaxação linear das formulações, sob as condições impostas.

Seja $\mathcal{R}_3$ a região poliedral dada pela interseção de $\mathcal{R}_1$ com a família de desigualdades (2.11). Uma formulação para o MDP é então dada por:

$$\{\max \sum_{e \in E} c_e y_e : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_3 \cap (\mathbb{B}^{|V|}, \mathbb{R}_+^{|E|})\}, \tag{4.6}$$

e sua relaxação linear é

$$\{\max \sum_{e \in E} c_e y_e : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_3\}, \tag{4.7}$$

Nossos algoritmos reforçam a relaxação linear (4.7) com as desigualdades GClique e GCut que vamos separar aplicando os algoritmos descritos no Capítulo 2.

Inicialmente, a heurística NDRC_GCut, descrita no Capítulo 3, é utilizada para se obter uma solução viável para o MDP e um limite primal correspondente. A heurística fornece, ainda, uma ordenação das variáveis binárias do problema, como indicado no Capítulo 3. Esta ordenação define o "grau de atratividade" dessas variáveis para fazerem parte de uma solução ótima para o problema. Feito isso, aplicamos o algoritmo de geração de planos de corte à (4.7), no nó raiz da árvore de enumeração. Nessa etapa são efetuadas várias iterações de separação e adição de cortes, gerando assim uma sequência de formulações potencialmente mais fortes para o MDP. Dessa forma, à cada iteração, é resolvido um problema de separação e desigualdades válidas que violam a solução da relaxação linear em mão são identificadas e utilizadas para reforçá-la, de acordo com os algoritmos descritos no Capítulo 2. A cada r_1 iterações, os cortes inativos são removidos da formulação. É importante notar que, um corte removido pode voltar a se tornar ativo em uma iteração subsequente, desde que seja identificado pelo algoritmo de separação.

Algumas regras determinam o fim da fase de geração de planos de corte no nó raiz da árvore de enumeração. A ocorrência de r_2 iterações consecutivas sem melhora significativa no limite dual (*tailing off*, veja [1]), uma prova de otimalidade, ou a ocorrência de uma interação sem que sejam encontrados novos cortes. A prova de otimalidade ocorre quando a diferença percentual entre os valores do limite primal e do limite dual, ou seja, o "gap de dualidade", é inferior a um valor γ, predeterminado, ou quando uma solução naturalmente inteira é obtida para a relaxação linear.

Após a fase de geração de planos de corte no nó raiz da árvore de enumeração, o algoritmo parte para a fase de ramificação, quando isso se faz necessário. Dois ou mais nós, "filhos do nó raiz", são então abertos e processados, exatamente como fizemos para o nó raiz. Esse procedimento se repete, sucessivamente, até que um critério de parada para o algoritmo Branch-and-Cut seja atingido: uma prova de otimalidade para dada solução viável para o MDP ou exceder o tempo limite de CPU.

Com uma única exceção para os oito algoritmos Branch-and-Cut que implementamos, o software Gurobi, versão 7.0 [40], foi utilizado para resolver as relaxações lineares e para gerenciar a árvore de enumeração. A ordenação fornecida pela heurística NDRC_GCut é usada para definir uma ordem de prioridade para fazer a ramificação sobre as variáveis fracionárias. Essa ordenação é obtida da resolução do SL que levou ao melhor limite primal fornecido pela heurística (mais detalhes no Capítulo 3). A exceção é implementada através do software CPLEX, versão 12.3 [41] e será discutida mais adiante.

A geração de cortes em nós distintos do nó raiz foi implementada através de *callbacks* ao Gurobi, ou seja, subrotinas disponibilizadas pelo próprio sistema. Nesse caso, o software tem o controle do pool de cortes e da eliminação de cortes inativos.

As desigualdades válidas a separar, no entanto, são definidas por nós e correspondem às desigualdades GClique e GCut.

Na aplicação do algoritmo de planos de corte à relaxações lineares definidas para o nó raiz, optou-se por utilizar uma implementação própria. Nesse caso, à cada iteração do algoritmo a relaxação linear em mãos é resolvida pelo Gurobi. Em seguida, sob nosso inteiro controle, é realizada a separação de cortes para reforçá-la e o procedimento é repetido. Razões para assim proceder serão apresentadas na Subseção 4.6. Uma estrutura de lista é utilizada para armazenar as desigualdades separadas. Desigualdades inativas são periodicamente removidas da formulação e também da lista. Ao final da execução do algoritmo, as desigualdades ativas presentes na lista são adicionadas à formulação (4.6) que passa a ser o ponto de partida para as formulações definidas nos demais nós da árvore de enumeração.

4.4.1 Regra de Ramificação

Uma parte importante de um algoritmo Branch-and-Cut é determinar como o espaço de soluções é subdividido a cada nó da árvore de enumeração. Uma forma fácil de se fazer isso é escolher uma variável inteira com valor fracionário na solução correspondente à relaxação linear e particionar a região de viabilidade em função dessa variável. Se $x_i = \overline{x}_i \notin \mathbb{Z}^1$, particionamos a região de viabilidade de X em

$$X_1 = X \cap \{x : x_j \leq \lfloor \overline{x}_i \rfloor\}$$

e

$$X_2 = X \cap \{x : x_j \geq \lceil \overline{x}_i \rceil\}.$$

Na solução de uma relaxação linear, normalmente, diversas variáveis inteiras assumem valores fracionários. Deve-se definir então uma regra para escolher aquela variável sobre a qual será conduzida a ramificação. Uma regra comum é escolher a variável "mais fracionária", isto é, a que apresenta valor mais distante do valor inteiro mais próximo. Uma regra mais elaborada, o "Strong Branching", se baseia na ideia de estimar o limite obtido por cada uma das variáveis candidatas a assumir valores inteiros e escolher aquela que leva ao melhor limite. Regras como essa exigem mais esforço computacional, já que são resolvidas duas relaxações lineares para cada variável candidata.

Existem, no entanto, outras formas de se particionar o espaço de soluções. Na verdade, qualquer forma de particionar este espaço de soluções de forma sistemática pode ser aplicada. Com o intuito de usufruir da informação obtida pela heurística, em particular, pela ordenação fornecida pela mesma, implementamos uma regra de ramificação nela baseada. Essa regra é motivada pela boa qualidade das soluções

primais que obtemos. Dado um conjunto de índices $I = \{a_1, a_2, ..., a_n\}$ que determina a ordenação fornecida pela heurística NDRC, toma-se o conjunto dos primeiros $k + r$ índices, $I' = \{a_1, a_2, ..., a_{k+r}\}$, onde k é o número de vértices (itens) a escolher para o MDP e r é um inteiro de "pequeno valor". A nova regra de ramificação toma a forma que descrevemos à seguir.

Inicialmente os valores $min = 0$ e $max = k$ são associados ao nó raiz e calcula-se o valor de $mid = \lfloor (max + min)/2 \rfloor$. A ramificação é então feita tomando-se:

$$X_1 = X \cap \{x : \sum_{i \in I'} x_i \leq mid\},$$

$$X_2 = X \cap \{x : \sum_{i \in I'} x_i \geq mid + 1\}.$$

Os valores de min e max são repassados para os nós filhos. No nó filho associado a região X_1 atualiza-se o valor de max para mid. No nó associado a região X_2 atualiza-se o valor min para $mid + 1$. Para a implementação da ramificação em cada nó, atualiza-se o valor de mid de acordo com os valores correspondentes de min e max, e a região de viabilidade é novamente dividida como indicado acima. Em um determinado momento, um nó terá o valor de min igual ao valor de max. A partir desse ponto, caso a solução obtida continue fracionária, mudamos a regra de ramificação a ser aplicada aos seus descendentes. Especificamente, ramificamos sobre a variável de valor mais fracionário, descrita acima.

Implementamos essa regra de ramificação no algoritmo BCNR. O algoritmo também se utiliza da geração de cortes 4Cut no nó raiz e GCut nos demais nós da árvore de enumeração. Utilizou-se o valor de $r = 0$, por apresentar melhores resultados preliminares. O algoritmo foi implementado em C++ utilizando o software Cplex. O software Gurobi, utilizado na implementação das demais versões do nosso algoritmo Branch-and-Cut, não oferece meios necessários para a implementação de regras de ramificação particulares, como a que propomos. O Cplex, por sua vez, permite tal implementação. No Cplex, no entanto, não é possível implementar uma regra que particione a região de viabilidade em mais de duas subregiões, apenas a geração de dois nós descendentes é permitida.

4.5 Soluções viáveis para MDP

Além da solução fornecida por NDRC_GCut, soluções viáveis para o MDP foram também obtidas a partir das soluções das relaxações lineares com que trabalhamos. Seja $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ uma tal solução. Primeiramente, "pseudo-custos" $c'_i = \overline{\mathbf{x}}_i + \sum_{e \in \delta(i)} \frac{\overline{\mathbf{y}}_e}{2}$ são computados e associados à cada vértice $i \in V$. Seja $I = \{a_1, a_2, ..., a_n\}$ um conjunto de índices de vértices tal que $c'_{a_i} \geq c'_{a_{i+1}}$. Então, uma solução viável para o MDP é

obtida tomando-se a clique induzida pelos vértices indexados por $\{a_1, a_2, \ldots, a_k\}$.

4.6 Testes computacionais

Implementamos oito versões do algoritmo Branch-and-Cut: BC3Cut, BC4Cut, BCFGCut, BCGClique, BCFGCut3Cut, BCFGCut4Cut, BCFGCutGClique e BCNR. Em todos eles é aplicada a separação lenta de GCuts em cada nó da árvore de enumeração com exceção da raiz. Os cortes gerados no nó raiz, em cada caso, serão discutidos no parágrafo seguinte. A heurística NDRC_GCut é utilizada de início para se obter um limite primal para o MDP, e é executada com um limite de tempo de 60 segundos sob os seguintes parâmetros para o MS: maxiter=1000, $\epsilon = 0.125$ e maxhalf=50.

Nossos algoritmos Branch-and-Cut separam conjuntos distintos de desigualdades válidas no nó raiz da árvore de enumeração. Desigualdades GCut são separadas no nó raiz pelo algoritmo de separação rápida. A partir deste ponto do texto, iremos nos referir às desigualdades GCut obtidas dessa forma por FGCut. São utilizadas as famílias 3Cut, 4Cut, FGCut e GClique, respectivamente, para as versões BC3Cut, BC4Cut, BCFGCut e BCGClique. As versões BCFGCut3Cut, BCFGCut4Cut e BCFGCutGClique utilizam a família GCut juntamente com as famílias 3Cut, 4Cut e GClique, respectivamente. O tempo de execução gasto na fase de geração de planos de corte no nó raiz é limitado a 1/6 do tempo de execução máximo permitido para o algoritmo. À exceção da versão BCNR, que utiliza a nova regra de ramificação definida na Seção 4.4.1, todos os demais algoritmos utilizam a regra da variável mais fracionária.

Comparamos computacionalmente nossos algoritmos com o algoritmo BBmax, proposto em [3]. O BBmax implementa um algoritmo Branch-and-Bound sob uma árvore de busca não binária. As regras de ramificação e poda são detalhadas em [3]. Uma heurística simples é lá utilizada para obter limites primais e soluções viáveis. Não é utilizada relaxação linear para obter limites duais, mas sim um procedimento combinatório. Este procedimento é similar ao que utilizamos na resolução de SL, pela Abordagem II, associada a nossos algoritmos NDRC (ver [3] ou a subseção 3.3.2 para mais detalhes).

Nossos algoritmos foram implementados em C++. O BCNR é o único implementado utilizando o software Cplex, versão 12.63. Todos e os demais foram implementados utilizando Gurobi, versão 7.0, para resolver relaxações lineares e para controlar a árvore de enumeração. Optamos preferencialmente pelo Gurobi porque os resultados apresentados pela companhia que o comercializa sugerem que o mesmo supera em desempenho os demais softwares comerciais disponíveis no mercado (http://www.gurobi.com/pdfs/benchmarks.pdf). No entanto, o Gurobi não

permite a implementação de regras de ramificação mais gerais. Optou-se então por utilizar o Cplex para implementação do BCNR, por nos dar essa opção.

O Gurobi também não permite um controle externo dos critérios de parada do algoritmo de planos de corte que utiliza. Em razão disso, quando o utilizamos conduzimos uma implementação própria do algoritmo de planos de corte, especificada na Seção 4.4. Na versão implementada utilizando o Cplex, o BCNR, utilizou-se o algoritmo de planos de corte do próprio Cplex. Nesse sistema é possível controlar aquelas regras de parada. Para todas as versões implementadas, os parâmetros do algoritmo de planos de corte aplicado no nó raiz foram: $r_1 = 2$, $r_2 = 50$ e $\gamma = 10^{-3}$. Com uma única exceção para o algoritmo BCNR, no qual a definição do parâmetro r_1 ficou a cargo do Cplex. Nos demais nós da árvore de enumeração, é realizada uma única iteração do algoritmo de planos de corte e os parâmetros para o mesmo, portanto, não são utilizados.

No Cplex, o "Presolver" foi desativado para permitir a definição da regra de ramificação. Foi utilizado o valor "Tradicional" para o parâmetro MIP::Strategy::Search e o número de "Threads" foi fixado em 1. No Gurobi, o parâmetro "PreCrush" foi fixado em 1 para permitir adição de cortes. O software foi configurado para utilizar uma única "Thread".

As instâncias utilizadas nos testes são as mesmas utilizadas em [3] e podem ser obtidas em http://www.optsicom.es/mdp. Uma descrição das mesmas segue:

- Glover2 (GKD-b): Conjunto de 50 instâncias definidas em [19]. São divididas em 10 conjuntos de 5 instâncias, cada, com as seguintes especificações: n = 25 e k = 2; n = 25 e k = 7; n = 50 e k = 5; n = 50 e k = 15; n = 100 e k = 10; n = 100 e k = 30; n = 125 e k = 12; n = 125 e k = 37; n = 150 e k = 15; n = 150 e k = 45.

- Silva (SOM-a): Conjunto de 50 instâncias definidas em [35]. São divididas em 10 conjuntos de 5 instâncias, cada, com as seguintes especificações: n = 25 e k = 2; n = 25 e k = 7; n = 50 e k = 5; n = 50 e k = 15; n = 100 e k = 10; n = 100 e k = 30; n = 125 e k = 12; n = 125 e k = 37; n = 150 e k = 15; n = 150 e k = 45.

Os resultados computacionais para os nossos algoritmos BC foram obtidos em um computador equipado com um processador Intel Xeon X5675, rodando em *thread* única a 3.06GHz, e com 48 GB de memória RAM disponível. Os resultados para o algoritmo BBmax são aqueles apresentados em [3]. Naquela referência, um limite de tempo de CPU de 3600 segundos foi imposto para cada execução do algoritmo. Para determinar a diferença entre nossa máquina e aquela utilizada por [3], utilizamos o SPEC - Standard Performance Evaluation Corporation. Segundo o SPEC, nossa

máquina é 2 vezes mais rápida que aquela utilizada em [3]. Em função disso, o tempo de CPU para cada execução de qualquer um de nossos algoritmos foi limitado a 1800 segundos.

Para cada combinação de n e k, os resultados exibidos nas tabelas a seguir representam a média dos resultados obtidos para as 5 instâncias de cada conjunto. Na coluna *Gap* é indicada a média da diferença percentual entre o melhor limite inferior e o melhor limite superior. A coluna Time indica a média do tempo de CPU consumido e a coluna *#Opt* indica o número de certificados de otimalidade obtidos dentre as 5 instâncias. Na última linha, apresentamos as médias obtidas para cada coluna. Os melhores resultados em cada tabela são apresentados em negrito.

Instância	n	k	Básico		3Cut		4Cut	
			Gap	Time	Gap	Time	Gap	Time
GKD-b	25	2	0,0	0,01	0,0	0,06	0,0	0,34
	25	7	3,7	0,02	0,0	0,25	0,0	0,55
	50	5	7,0	0,15	1,5	6,11	0,4	42,00
	50	15	11,1	0,21	2,0	23,71	0,7	99,63
	100	10	9,4	5,96	2,8	261,6	1,5	4389,01
	100	30	6,4	6,23	0,5	1023,73	-	-
	125	12	14,4	5,99	6,7	730,87	-	-
	125	37	5,4	23,45	0,4	4606,49	-	-
	150	15	9,7	15,75	3,3	2281,54	-	-
	150	45	5,2	36,23	-	-	-	-
SOM-a	25	2	0,0	0,01	0,0	0,05	0,0	0,51
	25	7	14,2	0,02	0,3	0,16	0,0	0,71
	50	5	6,6	0,13	2,0	2,46	0,0	20,08
	50	15	28,4	0,28	6,3	12,27	1,1	268,32
	100	10	20,1	3,68	14,0	171,7	8,1	2080,83
	100	30	39,8	4,98	14,8	741,65	-	-
	125	12	25,4	6,62	19,2	670,24	-	-
	125	37	43,4	9,57	17,7	2855,81	-	-
	150	15	29,6	12,77	23,0	1712,98	-	-
	150	45	46,1	44,77	-	-	-	-

Tabela 4.1: Relaxação linear das formulações

A Tabela 4.1 mostra o valor da relaxação linear da formulação (4.6), expresso na coluna *"Básico"*, e suas versões reforçadas respectivamente por 3Cut e 4Cut, expressos nas colunas *"3Cut"* e *"4Cut"*. Todas as desigualdades das famílias 3Cut e 4Cut são adicionadas de início nas suas respectivas formulações. Na tabela, as entradas com o simbolo "-" indicam execuções que ultrapassaram o tempo limite de CPU imposto de 2h. Podemos notar que as duas famílias de desigualdade de fato

fortalecem a formulação. Para as instancias SOM-a com n = 25 e k = 7, o gap de 14% foi reduzido a zero com o uso das desigualdades 4Cut. No entanto, o custo computacional aumenta rapidamente com o aumento da dimensão da instância. Com o uso das desigualdades 4Cut, a relaxação linear de instâncias com apenas 100 vértices ultrapassam o tempo limite estabelecido. Tudo isso reforça a necessidade da utilização dessas famílias de desigualdades como cortes, gerados dinamicamente à medida que se fizeram necessários.

Instância	n	k	BC3Cut			BC4Cut			BCFGCut			BCGClique		
			Gap	Time	#Opt	Gap	Time	#Opt	Gap	Time	#Opt	Gap	Time	#Opt
GKD-b	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,6	5	0,0	0,5	5	0,0	0,5	5	0,0	0,6	5
	50	5	0,0	2,4	5	0,0	2,4	5	0,0	1,6	5	0,0	2,0	5
	50	15	0,0	12,8	5	0,0	16,6	5	0,0	8,4	5	0,0	25,3	5
	100	10	0,0	94,0	5	0,0	252,6	5	0,0	144,4	5	0,0	350,6	5
	100	30	0,0	312,0	5	0,0	297,0	5	0,0	146,1	5	0,0	573,0	5
	125	12	0,3	679,4	4	**0,2**	858,0	4	0,3	538,9	4	0,4	1051,8	4
	125	37	0,1	1157,6	4	**0,0**	1187,7	**5**	0,1	1042,7	4	0,7	1536,7	3
	150	15	0,1	1176,0	4	**0,0**	1202,0	**5**	**0,0**	981,1	**5**	1,1	1798,6	1
	150	45	1,0	1798,8	0	**0,2**	1798,8	**2**	0,4	1798,8	0	1,4	1798,8	1
SOM-a	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,8	5	0,0	0,7	5	0,0	1,9	5	0,0	1,4	5
	50	5	0,0	2,5	5	0,0	2,2	5	0,0	29,6	5	0,0	8,4	5
	50	15	0,0	149,1	**5**	0,0	94,3	**5**	1,4	1339,4	2	0,0	957,8	**5**
	100	10	7,0	1800,0	0	**4,4**	1800,0	0	7,9	1800,0	0	8,3	1800,0	0
	100	30	17,4	1800,0	0	**14,4**	1800,0	0	22,2	1800,0	0	26,5	1800,0	0
	125	12	14,0	1800,0	0	**12,2**	1800,0	0	14,5	1800,0	0	17,4	1800,0	0
	125	37	**23,4**	1800,0	0	24,2	1800,0	0	33,0	1800,0	0	34,3	1800,0	0
	150	15	19,1	1800,0	0	**18,2**	1800,0	0	20,4	1800,0	0	23,0	1800,0	0
	150	45	**30,7**	1800,0	0	30,9	1800,0	0	38,6	1800,0	0	37,9	1800,0	0
Méd.			5,7	809,3	3,1	**5,2**	825,6	**3,3**	6,9	841,7	3,0	7,5	945,3	3,0

Tabela 4.2: Algoritmos exatos Branch-and-Cut

A Tabela 4.2 apresenta os resultados obtidos para os quatro algoritmos Branch-and-Cut que utilizam apenas uma única família de desigualdades como reforço para a formulação. Na tabela vemos que o algoritmo BC4Cut é o que apresenta melhores resultados. O método obtém o maior número de certificados de otimalidade e a menor média de Gap, dentre os algoritmos presentes na tabela. Na Tabela 4.3 são apresentados os resultados para os três algoritmos que utilizam duas famílias de desigualdades para reforçar a formulação. Na tabela, o algoritmo BCFGCut4Cut apresenta os melhores resultados. Numa comparação entre os resultados apresentados nessas duas tabelas, podemos perceber que a utilização simultânea de duas famílias de desigualdades não parece trazer benefícios. Isso decorre do fato de que, nesses casos, dois problemas de separação devem ser resolvidos à cada iteração do algoritmo de planos de corte. Podemos ver também que as desigualdades GClique são pouco vantajosas em relação às desigualdades GCut. O algoritmo BCFGCutGClique, que utiliza as duas famílias de desigualdades simultaneamente, apresenta um desempenho inferior ao do algoritmo BCFGCut, que utiliza apenas as desigualdades GCut.

Instância	n	k	BCFGCut3Cut			BCFGCut4Cut			BCFGCutGClique		
			Gap	Time	#Opt	Gap	Time	#Opt	Gap	Time	#Opt
GKD-b	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,5	5	0,0	0,5	5	0,0	0,5	5
	50	5	0,0	1,3	5	0,0	1,6	5	0,0	1,6	5
	50	15	0,0	7,9	5	0,0	16,8	5	0,0	9,0	5
	100	10	0,0	86,8	5	0,0	145,1	5	0,0	120,3	5
	100	30	0,0	130,8	5	0,0	318,3	5	0,0	146,5	5
	125	12	**0,1**	500,4	4	0,2	537,7	4	0,2	537,1	4
	125	37	0,1	892,8	4	0,1	898,7	4	0,1	945,9	4
	150	15	**0,0**	767,2	**5**	0,1	1144,2	4	**0,0**	984,2	**5**
	150	45	**0,2**	1784,3	2	0,3	1798,5	**3**	0,3	1798,8	2
SOM-a	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,9	5	0,0	0,9	5	0,0	1,4	5
	50	5	0,0	7,6	5	0,0	5,7	5	0,0	19,2	5
	50	15	**0,0**	268,6	**5**	**0,0**	256,4	**5**	1,0	1250,2	3
	100	10	7,3	1800,0	0	**6,8**	1800,0	0	7,9	1800,0	0
	100	30	18,6	1800,0	0	**13,1**	1800,0	0	22,5	1800,0	0
	125	12	14,6	1800,0	0	**14,3**	1800,0	0	15,1	1800,0	0
	125	37	27,2	1800,0	0	**22,4**	1800,0	0	32,9	1800,0	0
	150	15	**19,2**	1800,0	0	21,4	1800,0	0	20,1	1800,0	0
	150	45	38,9	1800,0	0	**30,7**	1800,0	0	39,3	1800,0	0
Méd.			6,3	762,5	**3,3**	**5,5**	796,2	**3,3**	7,0	830,7	3,2

Tabela 4.3: Algoritmos exatos Branch-and-Cut com duas famílias de desigualdades

Instância	n	k	BBmax			BCNR			BCFGCut4Cut			BC4Cut		
			Gap	Time	#Opt	Gap	Time	#Opt	Gap	Time	#Opt	Gap	Time	#Opt
GKD-b	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,0	5	0,0	0,3	5	0,0	0,5	5	0,0	0,5	5
	50	5	0,0	0,0	5	0,0	2,5	5	0,0	1,6	5	0,0	2,4	5
	50	15	0,0	0,4	5	0,0	40,0	5	0,0	16,8	5	0,0	16,6	5
	100	10	0,0	4,4	5	0,0	741,0	5	0,0	145,1	5	0,0	252,6	5
	100	30	8,6	3576,2	1	**0,0**	392,7	**5**	**0,0**	318,3	**5**	**0,0**	297,0	**5**
	125	12	**0,0**	297,9	**5**	1,4	1324,8	2	0,2	537,7	4	0,2	858,0	4
	125	37	13,7	3600,0	0	0,7	1796,9	0	0,1	898,7	4	**0,0**	1187,7	**5**
	150	15	5,4	1834,4	3	1,0	1720,4	1	0,1	1144,2	4	**0,0**	1202,0	**5**
	150	45	10,9	3600,1	0	1,7	1800,0	0	0,3	1798,5	**3**	**0,2**	1798,8	2
SOM-a	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,0	5	0,0	0,5	5	0,0	0,9	5	0,0	0,7	5
	50	5	0,0	0,0	5	0,0	2,2	5	0,0	5,7	5	0,0	2,2	5
	50	15	0,0	22,8	5	0,0	303,4	5	0,0	256,4	5	0,0	94,3	5
	100	10	**0,0**	38,3	**5**	7,9	1800,0	0	6,8	1800,0	0	4,4	1800,0	0
	100	30	31,7	3600,0	0	19,5	1800,0	0	**13,1**	1800,0	0	14,4	1800,0	0
	125	12	**0,0**	2678,9	**5**	14,4	1800,0	0	14,3	1800,0	0	12,2	1800,0	0
	125	37	34,6	3600,0	0	33,8	1800,0	0	**22,4**	1800,0	0	24,2	1800,0	0
	150	15	26,7	3600,1	0	22,9	1800,0	0	21,4	1800,0	0	**18,2**	1800,0	0
	150	45	35,6	3600,1	0	43,2	1800,0	0	**30,7**	1800,0	0	30,9	1800,0	0
Méd.			8,4	1502,7	3,2	7,3	946,2	2,7	5,5	796,2	**3,3**	**5,2**	825,6	**3,3**

Tabela 4.4: Algoritmos exatos para o MDP

Na Tabela 4.4 são apresentados os resultados do algoritmo BBmax, em comparação àqueles obtidas pelo BCNR e pelos dois melhores algoritmos presentes nas Tabelas 4.2 e 4.3. Pela Tabela 4.4, podemos verificar que o algoritmo BC4Cut é competitivo com o BBmax. Dentre todos os métodos presentes na tabela, aquele foi o método que apresentou o maior número de certificados de otimalidade obtidos e o menor Gap médio. Ele foi capaz de resolver várias instâncias em aberto. No entanto, falhou em resolver algumas instâncias resolvidas pelo BBmax. Para o conjunto de instâncias GKD-b, BC4Cut mostrou-se superior aos demais. Para as instâncias SOM-a, no entanto, o BBmax supera nossos algoritmos em número de certificados de otimalidade. Pela Tabela 4.4, podemos verificar que BCNR não apresentou bom desempenho, perdendo para os demais algoritmos em número de certificados de otimalidade e em média de tempo de CPU.

Instância	n	m	BC4Cut			BCFGCut3Cut			BCFGCutGClique		
			Gap	Time	#Opt	Gap	Time	#Opt	Gap	Time	#Opt
GKD-b	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,3	5	0,0	0,0	5	0,0	0,1	5
	50	5	0,0	1,8	5	0,0	0,5	5	0,0	0,9	5
	50	15	0,0	11,3	5	0,0	3,6	5	0,0	6,1	5
	100	10	0,0	157,7	5	0,0	69,6	5	0,0	171,2	5
	100	30	0,0	288,6	5	0,0	56,7	5	0,0	102,0	5
	125	12	0,0	981,1	**5**	0,0	843,8	4	0,0	1597,4	**5**
	125	37	0,0	1000,9	5	0,0	354,9	5	0,0	664,8	5
	150	15	0,0	1394,8	5	0,0	413,6	5	0,0	1635,3	5
	150	45	0,0	1835,6	5	0,0	751,1	5	0,0	1499,5	5
SOM-a	25	2	0,0	0,0	5	0,0	0,0	5	0,0	0,0	5
	25	7	0,0	0,4	5	0,0	0,2	5	0,0	0,5	5
	50	5	0,0	1,9	5	0,0	1,4	5	0,0	5,7	5
	50	15	0,0	51,5	5	0,0	155,9	5	0,0	1053,0	5
	100	10	**0,0**	1984,7	**5**	5,3	3600,0	0	6,7	3600,0	0
	100	30	10,4	3600,0	0	**5,0**	3600,0	0	8,3	3600,0	0
	125	12	**9,6**	3600,0	0	**9,6**	3600,0	0	11,3	3600,0	0
	125	37	16,0	3600,0	0	**8,9**	3600,0	0	14,4	3600,0	0
	150	15	13,6	3600,0	0	**12,8**	3600,0	0	14,7	3600,0	0
	150	45	24,4	3600,0	0	**14,2**	3600,0	0	24,9	3600,0	0
Méd.			3,7	1285,5	**3,75**	**2,8**	1212,6	3,45	4,0	1416,8	3,5

Tabela 4.5: Algoritmos Branch-and-Cut rodando por 1 hora

Na Tabela 4.5 são apresentados os resultados para três de nossos algoritmos, com cada execução limitada a 3600 segundos. Podemos verificar que, sob essas condições, o algoritmo BC4Cut foi capaz de obter certificado de otimalidade para todas as instâncias da família GKD-b, bem como para cinco instâncias adicionais da família SOM-a. O algoritmo BCFGcut3Cut apresentou o menor Gap médio e também o menor gasto de tempo médio de CPU dentre os três algoritmos, dando indícios de que, permitindo-se tempos de CPU maiores, bons resultados podem

eventualmente se obtidos.

É evidente a influência do tipo de instância sobre o desempenho dos diferentes algoritmos. Apesar dos dois conjuntos de instâncias apresentarem as mesmas características em termos de dimensão, o comportamento dos algoritmos variou de um para o outro. Enquanto as instâncias GKD-b foram facilmente resolvidas ou encontrados gaps de otimalidade muito pequenos por todos os nossos algoritmos, as instâncias de SOM-a se mostraram particularmente difíceis de resolver.

Capítulo 5

Resultados e Discussões

Neste trabalho, investigamos a utilização de algumas famílias de desigualdades válidas para o MDP, em particular as desigualdades do politopo booleano quadrático, para fortalecer uma formulação linear padrão para o problema. Essas desigualdades foram dualizadas dinamicamente em heurísticas Lagrangeanas e utilizadas como planos de corte nos algoritmos Branch-and-Cut. Algumas das heurísticas aqui propostas apresentaram bons resultados computacionais e foram capazes de competir com as melhores heurísticas presentes na literatura para o problema.

A grande maioria dos trabalhos existentes na literatura para o problema apresentam abordagens heurísticas. É possível observar uma constante evolução, ao longo dos anos, na qualidade das soluções fornecidas por esses métodos. Essa evolução confirma a qualidade dos algoritmos Lagrangeanos aqui propostos, que apesar de não terem sido capazes de superar os melhores da literatura, foram capazes de competir com os mesmos. Além disso, devemos ressaltar que os algoritmos Lagrangeanos aqui propostos não são simples heurísticas primais. Eles utilizam informação dual, o que não é comum nas heurísticas existentes para o problema, e fornecem limites duais a cada iteração. Tal abordagem, no entanto, implica em um custo computacional adicional que é refletido a cada iteração do método, mas geralmente leva a soluções de boa qualidade em poucas iterações.

O bom desempenho das desigualdades estudadas nos algoritmos Lagrangeanos nos motivou a investigar o uso delas em um algoritmo exato Branch-and-Bound. Abordagens exatas para o MDP são escassas na literatura. O melhor algoritmo deste tipo presente na literatura para o problema, o BBmax [3], consegue apresentar certificado de otimalidade para instâncias com até 150 vértices. Nosso algoritmo Branch-and-Cut mais bem sucedido foi capaz de provar otimalidade para algumas instâncias em aberto, utilizadas nos testes computacionais daquele trabalho. Foi capaz, ainda, de reduzir o gap de otimalidade de várias outras. Os gaps encontrados por ele para essas instâncias são os melhores existentes na literatura.

Embora nossos algoritmos exatos tenham apresentado um bom desempenho com-

putacional, ainda é evidente a carência de métodos deste tipo para o problema. Em um trabalho futuro seria interessante a implementação de um algoritmo exato Branch-and-Cut que utilize os limites duais fornecidos pelo algoritmo NDRC_GCut. Experimentos computacionais mostraram que ela fornece limites duais de boa qualidade em baixo tempo de CPU. NDRC_GCut seria utilizado em um esquema Branch-and-Cut em substituição aos algoritmos de planos de corte.

Para tanto, são necessárias algumas modificações no algoritmo NDRC_GCut. A principal delas é que este deve ser capaz de encontrar limites duais para os subproblemas obtidos a cada nó da árvore de enumeração. Táticas de simplificação desses subproblemas, compatíveis com esse tipo de algoritmo, também devem ser estudadas para se reduzir o tempo de solução. Estratégias como essas são essenciais para o bom funcionamento de um algoritmo Branch-and-Cut.

De modo geral, foi possível observar que as desigualdades do politopo booleano quadrático são fortes para o MDP, em particular as desigualdades aqui designadas GCut. É razoável supor que o uso dessas desigualdades em problemas similares pode levar a bons resultados práticos. Também acreditamos que a utilização de outras famílias de desigualdades válidas para o problema, tais como algumas famílias de desigualdades do Politopo Booleano Quadrático que não foram estudadas nesse trabalho, pode favorecer as heurísticas Lagrangeanas aqui propostas, e sua investigação é, portanto, relevante. Os resultados aqui apresentados também ajudam a demonstrar o potencial de algoritmos NDRC, ainda pouco utilizados na literatura, e nos levam a crer que sua utilização para resolver outros problemas de OC, similares ao MDP, pode levar a bons resultados práticos.

Referências Bibliográficas

[1] WOLSEY, L. A. *Integer Programming*. Wiley-Interscience, 1998.

[2] KUO, C.-C., GLOVER, F., DHIR, K. S. "Analyzing and Modeling the Maximum Diversity Problem by Zero-One Programming*", *Decision Sciences*, v. 24, n. 6, pp. 1171–1185, 1993.

[3] MARTÍ, R., GALLEGO, M., DUARTE, A. "An exact method for the maximum diversity problem", *European Journal of Operational Research*, v. 200, n. 1, pp. 36–44, 2010.

[4] MARTÍ, R., GALLEGO, M., DUARTE, A., et al. "Heuristics and metaheuristics for the maximum diversity problem", *Journal of Heuristics*, v. 19, n. 4, pp. 591–615, 2013.

[5] SILVA, G. C., DE ANDRADE, M. R., OCHI, L. S., et al. "New heuristics for the maximum diversity problem", *Journal of Heuristics*, v. 13, n. 4, pp. 315–336, 2007.

[6] BOMZE, I. M., BUDINICH, M., PARDALOS, P. M., et al. "The maximum clique problem". In: *Handbook of combinatorial optimization*, Springer, pp. 1–74, 1999.

[7] KUBY, M. J. "Programming Models for Facility Dispersion: The p-Dispersion and Maxisum Dispersion Problems", *Geographical Analysis*, v. 19, n. 4, pp. 315–329, 1987.

[8] RAVI, S. S., ROSENKRANTZ, D. J., TAYI, G. K. "Heuristic and special case algorithms for dispersion problems", *Operations Research*, v. 42, n. 2, pp. 299–310, 1994.

[9] MACAMBIRA, E. M., DE SOUZA, C. C. "The edge-weighted clique problem: valid inequalities, facets and polyhedral computations", *European Journal of Operational Research*, v. 123, n. 2, pp. 346–371, 2000.

[10] CHANDRA, B., HALLDÓRSSON, M. M. "Approximation algorithms for dispersion problems", *Journal of algorithms*, v. 38, n. 2, pp. 438–465, 2001.

[11] MACAMBIRA, E. M. "An application of tabu search heuristic for the maximum edge-weighted subgraph problem", *Annals of Operations Research*, v. 117, n. 1, pp. 175–190, 2002.

[12] FEIGE, U., PELEG, D., KORTSARZ, G. "The dense k-subgraph problem", *Algorithmica*, v. 29, n. 3, pp. 410–421, 2001.

[13] BUNZEL, J. H., AU, J. K. "Diversity or Discrimination?-Asian Americans in College", *The Public Interest*, , n. 87, pp. 49, 1987.

[14] MCCONNELL, S. "The New Battle Over Immigration", *Fortune*, 1988.

[15] DUARTE, A., MARTÍ, R. "Tabu search and GRASP for the maximum diversity problem", *European Journal of Operational Research*, v. 178, n. 1, pp. 71–84, 2007.

[16] BRIMBERG, J., MLADENOVIĆ, N., UROŠEVIĆ, D., et al. "Variable neighborhood search for the heaviest k-subgraph", *Computers & Operations Research*, v. 36, n. 11, pp. 2885–2891, 2009.

[17] LAWLER, E. L., WOOD, D. E. "Branch-and-bound methods: A survey", *Operations research*, v. 14, n. 4, pp. 699–719, 1966.

[18] PALUBECKIS, G. "Iterated tabu search for the maximum diversity problem", *Applied Mathematics and Computation*, v. 189, n. 1, pp. 371–383, 2007.

[19] GLOVER, F., LAGUNA, M. *Tabu Search**. Springer, 2013.

[20] FEO, T. A., RESENDE, M. G. "Greedy randomized adaptive search procedures", *Journal of global optimization*, v. 6, n. 2, pp. 109–133, 1995.

[21] GALLEGO, M., DUARTE, A., LAGUNA, M., et al. "Hybrid heuristics for the maximum diversity problem", *Computational Optimization and Applications*, v. 44, n. 3, pp. 411, 2009.

[22] LAGUNA, M., MARTI, R. *Scatter search: methodology and implementations in C*, v. 24. Springer Science & Business Media, 2012.

[23] MLADENOVIĆ, N., HANSEN, P. "Variable neighborhood search", *Computers & operations research*, v. 24, n. 11, pp. 1097–1100, 1997.

[24] WANG, Y., HAO, J.-K., GLOVER, F., et al. "A tabu search based memetic algorithm for the maximum diversity problem", *Engineering Applications of Artificial Intelligence*, v. 27, pp. 103–114, 2014.

[25] NERI, F., COTTA, C., MOSCATO, P. *Handbook of memetic algorithms*, v. 379. Springer, 2012.

[26] SHERALI, H. D., ADAMS, W. P. "A hierarchy of relaxations and convex hull characterizations for mixed-integer zero—one programming problems", *Discrete Applied Mathematics*, v. 52, n. 1, pp. 83 – 106, 1994.

[27] PADBERG, M. "The boolean quadric polytope: some characteristics, facets and relatives", *Mathematical programming*, v. 45, n. 1-3, pp. 139–172, 1989.

[28] LUCENA, A. "Steiner problem in graphs: Lagrangean relaxation and cutting-planes", *Mathematical Programming Society*, v. 21, 1992.

[29] ESCUDERO, L., GUIGNARD, M., MALIK, K. "A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships", *Annals of Operations Research*, v. 50, n. 1, pp. 219–237, 1994.

[30] LUCENA, A. "Non delayed relax-and-cut algorithms", *Annals of Operations Research*, v. 140, n. 1, pp. 375–410, 2005.

[31] BEASLEY, J. E. "Lagrangian relaxation". In: *Modern heuristic techniques for combinatorial problems*, pp. 243–303. John Wiley & Sons, Inc., 1993.

[32] CUNHA, J. O., SIMONETTI, L., LUCENA, A. "Lagrangian heuristics for the Quadratic Knapsack Problem", *Computational Optimization and Applications*, v. 63, n. 1, pp. 97–120, 2016.

[33] LOZANO, M., MOLINA, D., GARCI, C., et al. "Iterated greedy for the maximum diversity problem", *European Journal of Operational Research*, v. 214, n. 1, pp. 31–38, 2011.

[34] WANG, J., ZHOU, Y., CAI, Y., et al. "Learnable tabu search guided by estimation of distribution for maximum diversity problems", *Soft Computing*, v. 16, n. 4, pp. 711–728, 2012.

[35] SILVA, G. C., OCHI, L. S., MARTINS, S. L. "Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem". In: *International Workshop on Experimental and Efficient Algorithms*, pp. 498–512. Springer, 2004.

[36] CORPORATION, S. P. E. "SPEC Standard Performance Evaluation Corporation". 1995–2017. Disponível em: <`http://www.spec.org/index.html`>.

[37] LAND, A. H., DOIG, A. G. "An automatic method of solving discrete programming problems", *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.

[38] DANTZIG, G., FULKERSON, R., JOHNSON, S. "Solution of a large-scale traveling-salesman problem", *Journal of the operations research society of America*, v. 2, n. 4, pp. 393–410, 1954.

[39] APPLEGATE, D. L., BIXBY, R. E., CHVÁTAL, V., et al. *The Traveling Salesman Problem: A Computational Study*. Princeton university press, 2011.

[40] GUROBI OPTIMIZATION, I. "Gurobi Optimizer Reference Manual". 2017. Disponível em: <http://www.gurobi.com>.

[41] IBM CORPORATION, I. "IBM ILOG CPLEX Optimization Studio V12.3 documentation". 2017. Disponível em: <https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.3.0/welcome.html>.