ALGORITMOS PARA O PROBLEMA DA MOCHILA QUADRÁTICA 0-1

Jesus Ossian da Cunha Silva

Tese de Doutorado 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 Doutor em Engenharia de Sistemas e Computação.

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Rio de Janeiro
Agosto de 2014

ALGORITMOS PARA O PROBLEMA DA MOCHILA QUADRÁTICA 0-1

Jesus Ossian da Cunha Silva

TESE 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 DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Abilio Pereira de Lucena Filho, Ph.D.

Prof. Luidi Gelabert Simonetti, D.Sc.

Prof. Luiz Satoru Ochi, D.Sc.

Profa. Marcia Helena Costa Fampa, D.Sc.

Prof. Nelson Maculan Filho, D.Sc.

Prof. Paulo Roberto Oliveira, Dr.Ing.

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

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2014

Silva, Jesus Ossian da Cunha

Algoritmos para o Problema da Mochila Quadrática 0-1/Jesus Ossian da Cunha Silva. – Rio de Janeiro: UFRJ/COPPE, 2014.

VIII, 70 p. 29, 7cm.

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 68 – 70.

1. problema da mochila quadrática. 2. relax-and-cut. 3. branch-and-cut. I. Lucena Filho, Abilio Pereira de *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Gostaria de agradecer ao CNPQ pelo auxílio financeiro durante a realização do doutorado.

Aos meus orientadores Abilio Lucena e Luidi Simonetti pela orientação deste trabalho.

Ao Professor Maculan pelas orientações e auxílios.

Ao Professor Paulo Roberto pela orientação durante mestrado no PESC.

Ao professores da UFPI, Gilvan, João Xavier e Jurandir, por me incentivarem a estudar no PESC.

A Fátima Marques e Carolina Vieira pelos auxílios.

Ao pessoal da secretaria do PESC, Solange, Guty, Mercedes, Claudia Prata e Sonia pelos diversos auxílios.

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

ALGORITMOS PARA O PROBLEMA DA MOCHILA QUADRÁTICA 0-1

Jesus Ossian da Cunha Silva

Agosto/2014

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Programa: Engenharia de Sistemas e Computação

O Problema da Mochila Quadrática 0-1 é um problema desafiador e que tem atraido considerável atenção na literatura nas últimas três décadas.

Nesta tese, apresentamos um algoritmo *Relax-and-Cut* para geração de limites inferiores e superiores para o problema. Estes limites são comparados, sobre um mesmo conjunto de instâncias teste, com outros limites disponíveis na literatura.

Adicionalmente, propomos um algoritmo *Branch-and-Cut*, que se baseia numa linearização da formulação clássica do problema, fortalecida com a utilização de algumas famílias de desigualdades válidas. A primeira família está associada com a quadratização da desigualdade da mochila que define o problema. As demais definem facetas do Politopo Booleano Quadrático e do Politopo da Mochila 0-1. Desigualdades dos dois últimos tipos são separadas dinamicamente, à medida que se fazem necessárias. Por sua vez, desigualdades da primeira família são incorporadas à reformulação desde o início. Experimentos computacionais mostram que o algoritmo é uma opção atraente para a resolução exata do problema.

Propomos ainda dois algoritmos adicionais *Branch-and-Bound*, onde nosso algoritmo *Relax-and-Cut* é primeiramente aplicado no nó raiz da árvore de enumeração junto com alguns testes de fixação de variáveis. Para um dos algoritmos, em cada nó da árvore de enumeração, desigualdades violadas são anexadas a formulação como planos de cortes, transformando assim esse em um algoritmo *Branch-and-Cut* baseado em Programação Linear. Para o outro, os limites para cada nó da árvore de enumeração são obtidos diretamente através do nosso algoritmo *Relax-and-Cut*.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

ALGORITHMS FOR THE QUADRATIC KNAPSACK PROBLEM 0-1

Jesus Ossian da Cunha Silva

August/2014

Advisors: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Department: Systems Engineering and Computer Science

The Quadratic 0-1 Knapsack Problem is a very challenging problem that has attracted considerable attention in the literature over the past three decades.

In this thesis, we present a Relax-and-Cut algorithm for generating lower and upper bounds for the problem. These bounds are compared, over a same set of test instances, with bounds available in literature.

Additionally, we also propose a Branch-and-Cut algorithm which is based on linearizing a classic formulation of the problem, strengthened with the use of some families of valid inequalities. The first family is associated with the quadratization of the knapsack inequality that defines the problem. Additional families used define facets of the Quadratic Boolean Polytope and the 0-1 Knapsack Polytope. Inequalities in the latter two families are dynamically separated, as they become necessary. In turn, inequalities from the former family are incorporated into the formulation from the start. Computational experiments show that the algorithm is an attractive option for exactly solving the problem.

Furthermore, we also propose two additional Branch-and-Bound algorithms where our Relax-and-Cut algorithm is firstly applied at the root node of the enumeration tree, together with some variable fixing tests. For one of the algorithms, at every enumeration tree node, violated inequalities are appended to the formulation as cutting planes, thus turning it a Linear Programming based Branch-and-Cut algorithm. For the other one, bounds for every enumeration tree node are directly obtained through our Relax-and-Cut algorithm.

Sumário

Capítulo 1

Introdução

Suponha que estão disponíveis um conjunto de n objetos $N = \{1, \ldots, n\}$, com pesos $W = \{w_i \in \mathbb{N} : i \in N\}$, e uma mochila com capacidade $c \in \mathbb{N}$, sendo $\max_{i \in N} w_i \leq c < \sum_{i \in N} w_i$. Colocar na mochila um objeto $i \in N$ traz um benefício $p_{ii} \in \mathbb{N}$ e deseja-se escolher um subconjunto de objetos $N' \subset N$ que, carregados na mesma, leve a um benefício total máximo. Denomina-se Problema da Mochila 0-1 (KP) [1–3] a escolha de um tal N'.

Se benefícios cruzados $p_{ij},\ i < j,\ i, j \in N$, são também obtidos quando $i, j \in N'$, a escolha de N' passa a se chamar Problema de Mochila Quadrática 0-1 (QKP) [4–11].

Introduzindo um conjunto de variáveis $\mathbf{x} = \{x_i \in \{0, 1\} : i \in N\}$, tomando $q_i = p_{ii}$ e notando que $x_i = x_i x_i$, QKP pode ser formulado como,

$$\max \sum_{i \in N} q_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} x_i x_j \tag{1.1}$$

$$\text{s.a.} \sum_{i \in N} w_i x_i \leq c \tag{1.2}$$

$$x_i \in \{0, 1\}, \quad i \in N. \tag{1.3}$$

Sob esse mesmo conjunto de variáveis, obtem-se a seguinte formulação para KP,

$$\max \sum_{i \in N} q_i x_i \tag{1.4}$$

$$\text{s.a.} \sum_{i \in N} w_i x_i \leq c \tag{1.5}$$

$$x_i \in \{0, 1\}, \ i \in N. \tag{1.6}$$

Ou seja, QKP e KP são formulados para um mesmo espaço de soluções, diferindo apenas nas funções objetivo associadas a cada um deles, quadrática para o primeiro

e linear para o segundo.

QKP também pode ser interpretado em termos de Teoria dos Grafos. Nesse sentido, seja $G = (V, E)$ um grafo completo não direcionado com um conjunto V de n vértices, onde cada um deles corresponde a um item de V, e um conjunto E de arestas. Adicionalmente, assuma que o conjunto de pesos $\{w_i : \ i \in V\}$ estão associados com os vértices de G enquanto o conjunto de benefícios $\{p_{ii} : \ i \in V\}$ e $\{p_e : \ e = (i, j), \ i, j \in N, \ e \in E\}$ são respectivamente asssociados aos vértices e arestas de G. O QKP então corresponde a encontrar um subconjunto de G com a soma máxima dos benefícios, tal que a soma dos pesos dos vértices não ultrapasse a capacidade c.

Apesar de sua fácil descrição e formulação QKP é um problema desafiador, tendo sido amplamente estudado na literatura. Ele é um problema NP-difícil pois admite KP, que pertence àquela classe de problemas, como um caso particular.

QKP possui uma grande quantidade de aplicações, podendo ser citadas, dentre outras:

- Witzgall descreve em [12] um problema de telecomunicação onde um conjunto de estações de satélites tem de ser selecionado, tal que o tráfico entre as estações é maximizado e uma restrição de orçamento tem de ser respeitada. Ou seja, um problema similar a QKP;

- Johnson, Mehrotra, Nemhauser em [13] citam um problema de design de compiladores que pode ser formulado como um QKP;

- O problema de encontrar cliques máximais de um grafo [5, 7, 14] pode ser formulado como um QKP [5, 7]. Para tanto, assuma que um grafo esparso não direcionado $G = (V, E)$ é dado. Denomina-se uma clique a um subgrafo completo de G e o problema em questão é o de encontrar a maior clique contida naquele grafo.

Ao longo das últimas três décadas diversos limites superiores tem sido propostos para o QKP, sendo que esses limites são baseados em Técnicas de Linearização, Planos Superiores, Relaxação Lagrangeana, Planos de Cortes, Decomposição Lagrangeana, Programação Semidefinida ou Reformulações. Por exemplo:

- Planos Superiores por Gallo, Hammer e Simeone em [4];

- Relaxação Lagrangeana por Chaillou, Hansen, Mahieu em [15] e por Palmeira em [7];

- Linearização e Método Planos de Corte por Johnson, Mehrotra, Nemhauser em [13] e por Rodrigues, Quadri, Michelon e Gueye [16];

- Relaxação Lagrageana e Reformulação por Caprara, Pisinger e Toth em [5];
- Decomposição Lagrangeana por Michelon and Veuilleux em [9] e por Billionnet, Faye, Soutif em [10];
- Programação Semidefinida por Helmberg, Rendl e Weismantel em [17, 18].

Para maiores detalhes sobre o QKP sugerimos as fontes [1–3].

Esta tese está organizada em 5 capítulos. O Capítulo 2 faz uma breve descrição sobre Relaxação Lagrangena, Método Subgradiente e *Branch-and-Bound*, o Capítulo 3 descreve desigualdades válidas para o KP e QKP, formulações e algoritmos para o QKP. Adicionalmente o Capítulo 4 descreve nossas propostas de algoritmos para o QKP. Por sua vez o Capítulo 5 mostra os resultados computacionais obtidos pelos nossos experimentos computacionais. Finalmente o Capítulo 6 traz nossas conclusões, contribuições e proposta de trabalhos futuros.

Capítulo 2

Programação Inteira

Suponha um Problema de Programação Inteira (PI), $\max\{p^T x : Ax \leq b,\ Dx \leq e,\ x \in \{0,1\}\ ,\ i \in N\}$, considerado muito difícil de se resolver, onde um subconjunto de restrições $Ax \leq b$ são consideradas complicadas. Uma técnica para se facilitar a resolução do mesmo é utilizar Relaxação Lagrangeana, onde restrições complicadas são adicionadas à função objetivo junto com os multiplicadores de Lagrange, tornando assim o problema fácil de se resolver. Um método muito utilizado para se resolver um problema por Relaxação Lagrangeana é o Método do Subgrandiente.

Um algoritmo *Branch-and-Bound* [3] é um algoritmo de enumeração implícita que pode enumerar todas soluções viáveis e selecionar uma com melhor valor ótimo.

Um algoritmo *Relax-and-Cut* pode ser visto como um análogo a um Lagrangeano para Programação Linear baseada em algoritmos de planos de corte.

Um algoritmo *Branch-and-Cut* é um algoritmo do tipo *Branch-and-Bound* no qual planos de corte são gerados para os diferentes problemas definidos nos nós da árvore de enumeração.

Neste capítulo definimos Relaxação Lagrangeana [19], Método Subgradiente [19, 20], algoritmos *Relax-and-Cut*, algoritmos *Branch-and-Bound* [3] e algoritmos *Branch-and-Cut* [21].

2.1 Relaxação Lagrangeana e Método Subgradiente

Considere o Problema de Programação Inteira (PI):

$$z_p = \max\ p^T x \tag{2.1}$$
$$\text{s.a } Ax \leq b \tag{2.2}$$
$$Dx \leq e \tag{2.3}$$
$$x \in \{0,1\},\ i \in N. \tag{2.4}$$

onde $p \in \mathbb{R}^n,\ A \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^m,\ D \in \mathbb{R}^{p\times n}$ e $e \in \mathbb{R}^p$.

Suponha que as restrições (2.3) sejam consideradas fáceis, ou seja, o problema (2.1), (2.3) e (2.4) é um problema fácil de se resolver. Já as restrições (2.2) são consideradas difíceis ou complicadas, neste caso quando incluimos estas no PI, o problema torna-se difícil de se resolver.

Associemos a restrição (2.2) o vetor $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, tal que $\lambda_i \geq 0,\ \forall\ i \in \{1, \ldots, m\}$, o qual definimos como vetor dos multiplicadores de Lagrange. Formulamos então o seguinte problema, o qual denominamos Problema Lagrangeano Relaxado (PI(λ)):

$$\mathcal{L}(\lambda) = \max\left\{p^T x + \lambda^T (b - Ax)\right\} \tag{2.5}$$
$$\text{s.a } Dx \leq e$$
$$x \in \{0,1\},\ i \in N.$$

Suponha que as restrições (2.2), (2.3) e (2.4) formam um conjunto não-vazio. Temos então a seguinte proposição:

Proposição 2.1.1. $\mathcal{L}(\lambda) \geq z_p$.

Demonstração. Suponha que x^* seja solução ótima para PI com valor ótimo $z_p = p^T x^*$. Considerando $\lambda \geq 0$ e $b - Ax^* \geq 0$ temos $\lambda^T(b - Ax^*) \geq 0$, segue então que $p^T x^* + \lambda^T(b - Ax^*) \geq p^T x^* = z_p$. E como $\mathcal{L}(\lambda) \geq p^T x^* + \lambda^T(b - Ax^*)$ então $\mathcal{L}(\lambda) \geq z_p$. □

De acordo com a proposição (2.1.1) temos que PI(λ) fornece um limite superior para PI. A idéia é encontrar multiplicadores de Lagrange que forneçam um melhor limite superior possível. Podemos encontrar estes multiplicadores resolvendo o

seguinte problema, o qual denominamos Problema Dual Lagrangeano do PI (LDP).

$$z_d = \min \ \mathcal{L}(\lambda) \tag{2.6}$$

$$\text{s.a} \ \ \lambda \geq 0. \tag{2.7}$$

Se λ^* é uma solução ótima de LDP, temos que $z_d = \mathcal{L}(\lambda^*)$.

Definido o LDP precisamos de um método para resolver o mesmo, ou seja, um método para determinar os valores ótimos dos multiplicares de Lagrange. Neste trabalho utilizamos o Método Subgradiente (SM) o qual foi sugerido por Held, Wolfe e Crowed em [20].

O SM [19, 20] é um método iterativo que busca a cada iteração caminhar em uma boa direção de acordo com a função objetivo considerada. A direção a ser tomada a cada passo tem um papel crucial no desempenho do algoritmo, sendo determinada com auxílio do vetor de subgradientes calculado a cada passo.

Definição 2.1.1. *Um vetor α é um subgradiente de uma função convexa $f : \mathbb{R}^n \to \mathbb{R}$ no ponto $\overline{x}$ se $f(x) \geq f(\overline{x}) + \alpha^T(x - \overline{x})$ para todo $x \in \mathbb{R}^n$. Se f for diferenciável em $\overline{x}$ o subgradiente de f será igual ao gradiente de f em $\overline{x}$.*

Definição 2.1.2. *O subdiferencial de f em $\overline{x}$, $\partial f(\overline{x})$, é o conjunto de todos os subgradientes de f em $\overline{x}$.*

O SM [19, 20] tem sido amplamente utilizado, isto se deve basicamente a facilidade com que o mesmo é implementado e a boa qualidade dos resultados fornecidos pelo mesmo.

Iterativamente o SM [19, 20] tenta resolver o Problema Dual Lagrageano. Dado um vetor inicial viável de multiplicadores de lagrange, λ^0, a cada iteração k, novos valores para o vetor de multiplicadores são gerados, determinando-se um novo vetor λ^k, e produzindo um novo Problema Lagrangeano Relaxado. Esse novo Problema Langrangeano Relaxado é resolvido gerando um novo limite dual para o problema original PI.

Considere a função $\mathcal{L}$ definida em PI(λ) e suponha que $\overline{x}$ seja uma solução viável, tal que $\mathcal{L}(\lambda^0) = p^T\overline{x} + (\lambda^0)^T(b - A\overline{x})$.

Proposição 2.1.2. *Seja $\mathcal{L} : \mathbb{R}^m \to \mathbb{R}$ uma função dual. O vetor $b - A\overline{x}$ é um vetor de subgradientes para $\mathcal{L}$ em $\overline{x}$.*

Demonstração. Usando o fato que $\lambda \geq 0$.

$$\begin{aligned}
\mathcal{L}(\lambda^0) + (\lambda - \lambda^0)^T(b - A\overline{x}) &= p^T\overline{x} + (\lambda^0)^T(b - A\overline{x}) + (\lambda - \lambda^0)^T(b - A\overline{x}) \\
&= p^T\overline{x} + \lambda^T(b - A\overline{x}) \\
&\leq \mathcal{L}(\lambda)
\end{aligned}$$

Logo $b - A\overline{x}$ é um vetor de subgradientes de $\mathcal{L}$ em λ^0. □

Considere a k-ésima iteração onde $\overline{x}^k$ é a solução ótima do Problema Lagrangeano Relaxado com valor $\overline{ub}$. Seja $\delta_i^k,\ i = 1, \ldots, m$ os subgradientes associado as m restrições relaxadas na solução $\overline{x}^k$ onde:

$$\delta_i^k = (b_i - a_i\overline{x}^k),\ i = 1, \ldots, m \tag{2.8}$$

A atualização dos multiplicadores de Lagrange é feita de acordo com os multiplicadores da iteração anterior, os valores dos subgradientes na iteração corrente e o tamanho do passo θ^k definido por:

$$\theta^k = \frac{\mu(\overline{ub} - \underline{lb})}{\sum_{i=1}^{m}(\delta_i^k)^2} \tag{2.9}$$

onde μ é um escalar definido no intervalo $(0, 2]$ e $\underline{lb}$ é um limite inferior para o problema original PI.

Assim a atualização dos multiplicadores de Lagrange na iteração $k+1$ é realizada da seguinte forma:

$$\lambda_i^{k+1} = \max\ \left\{0; \lambda_i^k - \theta^k\delta_i^k\right\},\ i = 1, \ldots, m \tag{2.10}$$

Abaixo o pseudocódigo do SM [19, 20]:

Algoritmo 1: Método do Subgradiente (k-ésima iteração)

1: Resolva PI(λ^k) e obtenha uma solução $\overline{x}^k$ cujo valor é $\overline{ub}$;
2: **for** $i = 1$ **to** m **do**
3: $\quad \delta_i^k \leftarrow b_i - a_i\overline{x}^k$;
4: **end for**
5: $\theta^k \leftarrow \dfrac{\mu(\overline{ub} - \underline{lb})}{\sum_{i=1}^{m}(\delta_i^k)^2}$, onde $\mu \in (0, 2]$ e $\underline{lb}$ é um limite inferior para (PI);
6: **for** $i = 1$ **to** m **do**
7: $\quad \lambda_i^{k+1} \leftarrow \max\ \left\{0; \lambda_i^k - \theta^k\delta_i^k\right\}$
8: **end for**
9: $k \leftarrow k + 1$

2.2 *Branch-and-Bound*

Um algoritmo *Branch-and-Bound* [3] é um algoritmo de enumeração implícita que evita, através do uso de limitantes inferiores e superiores, uma enumeração explícita de todas as soluções viáveis do problema.

O principal conceito do algoritmo *Branch-and-Bound* [3] é baseado em uma inteligente enumeração completa do espaço solução já que em muitos casos somente um pequeno subconjunto de soluções viáveis são enumeradas explicitamente. Ele portanto garante que parte do espaço solução que não foi enumerado explicitamente não contém a solução ótima. Dizemos assim que estas soluções viáveis foram enumeradas implicitamente.

O algoritmo *Branch-and-Bound* [3] é baseado em dois fundamentais princípios: *branching* e *bounding*. Suponha que queiramos resolver o seguinte problema.

$$z = \max\ f(x) \tag{2.11}$$

$$\text{s.a } Ax \leq b \tag{2.12}$$

$$x_i \in \{0,1\},\ i \in N. \tag{2.13}$$

onde $S = \{x\ :\ Ax \leq b,\ x_i \in \{0,1\},\ i \in N\}$.

Na parte do *branching* um dado subconjunto de soluções $S' \subset S$ é dividido em um número de subconjuntos menores $S_1, S_2, \ldots, S_l$. Estes subconjuntos podem se sobrepor, mas a união deles devem gerar o S', assim $S_1 \cup S_2 \cup \ldots \cup S_l = S'$. O processo é em princípio repetido até que cada conjunto contenha somente uma solução viável. A escolha da melhor de todas soluções apresentadas para a função objetivo, $f(x)$, em questão garante que a solução ótima para o problema proposto foi encontrada.

Na parte do *bounding* são encontrados limites superiores e limites inferiores, $\overline{ub}_{S'}$ e $\underline{lb}_{S'}$ respectivamente, para um dado subconjunto de S' do espaço solução. Um limite inferior tal que $\underline{lb}_{S'} < ub^*$, onde ub^* é solução ótima do problema em questão, é escolhido como a melhor solução viável encontrada na árvore de enumeração até então. Se nenhuma solução foi considerada ainda, podemos escolher $\underline{lb}_{S'}$ como o valor obtido através de uma heurística. Um limite superior para um dado espaço solução $S' \subset S$ é um número real satisfazendo,

$$\overline{ub}_{S'} \geq f(x),\ \forall\ x \in S'. \tag{2.14}$$

O limite superior é utilizado para podar, eliminar, partes do espaço de busca. Suponha que $\overline{ub}_{S'} \leq \underline{lb}_{S'}$ para um dado S'. Então por (2.14) temos que

$$f(x) \leq \overline{ub}_{S'} \leq \underline{lb}_{S'},\ \forall\ x \in S' \tag{2.15}$$

o que significa que uma solução melhor que $\underline{lb}_{S'}$ não pode ser encontrada em S' e assim não necessitamos inventigar S' mais ainda.

2.3 *Relax-and-Cut*

Um algoritmo *Relax-and-Cut* pode ser visto como um análogo a um Lagrangeano para Programação Linear baseada em algoritmos de planos de corte. Assim, desigualdades violadas, possivelmente pertecentes a famílias de desigualdades com tamanho exponencial, são dinâmicamente identificadas e dualizadas dentro de um *framework* Lagrangeano. Esta ação é tomada na tentativa de fortalecer limites Lagrangeanos.

Seguindo [22], algoritmos *Relax-and-Cut* foram propostos em duas variantes: *Delayed Relax-and-Cut* [23] e *Non Delayed Relax-and-Cut* [22, 24, 25]. Estes foram propostos independentemente ao mesmo tempo. No entanto o termo *Relax-and-Cut* foi atribuido a [23]. A diferença entre essas duas variantes se dar essencialmente ao tempo em que a identificação e dualização das desigualdades violadas ocorre. *Delayed Relax-and-Cut*, por exemplo, espera até que o Problema Dual Lagrangeano (LDP) seja resolvido, afim de fazer isto. Desigualdades que violam a solução do LDP são então, identificadas e dualizadas, dando origem a um novo LDP reforçado. O procedimento em seguida se repete. *Non Delayed Relax-and-Cut* por outro lado não espera até que o LDP seja resolvido para identificar e dualizar desigualdades violadas. Assim, as desigualdades violadas são identificadas e dualizadas para cada solução do subproblema Lagrangeano. Os experimentos computacionais realizados em [22] mostraram que a perfomace do *Non Delayed Relax-and-Cut* tem desempenho melhor que o *Delayed Relax-and-Cut*. Neste trabalho, um algoritmo *Non Delayed Relax-and-Cut* é proposto para o QKP.

2.4 *Branch-and-Cut*

Um algoritmo *Branch-and-Cut* [21] é um algoritmo do tipo *Branch-and-Bound* [3] no qual planos de corte são gerados para os diferentes problemas definidos nos nós da árvore de enumeração. Isso é feito com o intuito de fortalecer os limitantes duais associados aos mesmos e eventualmente reduzir o número de nós a enumerar na árvore.

Na prática existe um *trade-off* a ser respeitado ao se implementar um algoritmo *Branch-and-Cut*. Se muitos planos de cortes são adicionados em cada nó da árvore de enumeração, as reotimizações podem se tornar muito caras. Demandas adicionais relativas ao uso de memória RAM devem também ser consideradas, já que temos que armazenar todas as informações relativas aos problemas definidos em cada nó da árvore de enumeração. Dependendo do número de cortes gerados, essa demanda pode se tornar significativa.

Um *cut pool* é utilizada para armazenar informações referente aos cortes. Por

exemplo, limites duais e bases Simplex a eles associados devem ser guardadas. Da mesma forma, é também necessário indicar quais planos de corte são necessárias para reconstruir a formulação relativa a um dado nó da árvore de procura (os ponteiros para estas restrições devem também ser armazenados no *cut pool*).

Capítulo 3

Desigualdades Válidas, Reformulações e Algoritmos de Soluções

Nas últimas três décadas diversas reformulações, desigualdades válidas e algoritmos de solução foram propostas para o QKP. A seguir fazemos uma síntese dos mesmos. Para tanto, utilizamos as formulações descritas para KP e QKP no Capítulo 1, assim como as suas respectivas relaxações contínuas. Estas são obtidas ao substituirmos as restrições de integralidade $x_i \in \{0,1\}$, para todo $i \in N$, por $x_i \in [0,1]$, para todo $i \in N$.

3.1 Desigualdades de Quadratização da Desigualdade da Mochila

Considere a restrição da mochila,

$$\sum_{i \in N} w_i x_i \leq c. \tag{3.1}$$

Multiplicando a mesma por x_j, para cada $j \in N$, obtemos as seguintes desigualdades quadratizadas:

$$\sum_{i \in N \setminus \{j\}} w_i x_i x_j \leq (c - w_j x_j)\, x_j,\ j \in N. \tag{3.2}$$

Tais desigualdades foram propostas por Adams e Sherali em [26]. Para o caso

onde a matriz de benefícios é triangular superior temos a desigualdade,

$$\sum_{i<j,\ i\in N} w_i x_i x_j + \sum_{i>j,\ i\in N} w_i x_j x_i \leq (c - w_j x_j)\, x_j,\ j \in N. \tag{3.3}$$

3.2 Desigualdades do Politopo Booleano Quadrático

Para efeito de linearização dos termos quadráticos da função objetivo do QKP, considere o uso de variáveis $\mathbf{y} = \{y_{ij} \in \{0,1\} : i < j,\ i,j \in N\}$. Dessa forma, $x_i x_j$ deve ser então substituido por y_{ij}, para todo $i < j,\ i,j \in N$. Adicionalmente, desigualdades válidas propostas por Padberg [27] para o Politopo Booleano Quadrático (BQP), podem ser utilizadas para ajudar a aproximar o valor de y_{ij} daquele correspondente a $x_i x_j$.

Nesse sentido, as desigualdades,

$$y_{ij} \leq x_i,\ i < j,\ i,j \in N, \tag{3.4}$$

$$y_{ij} \leq x_j,\ i < j,\ i,j \in N, \tag{3.5}$$

$$x_i + x_j \leq 1 + y_{ij},\ i < j,\ i,j \in N. \tag{3.6}$$

se mostram particularmente úteis.

As desigualdades (3.3) e (3.4)-(3.5) foram utilizadas por Billionnet e Calmels [8], por Palmeira [7] e por Caprara, Pisinger e Toth [5]. Nesta última referência, para efeito de uma reformulação de QKP, variáveis adicionais $\{y_{ij} \in \{0,1\} : i > j,\ i,j \in N\}$ são também utilizadas, sob a restrição adicional de que $y_{ij} = y_{ji}$, para qualquer par $i,j \in N$, onde $i \neq j$. Ainda naquela referência, o benefício cruzado p_{ij}, relativo a um par $i,j \in N$, onde $i \neq j$, é dividido igualmente entre as duas variáveis y_{ij} e y_{ji} que lhe são correspondentes.

Outra família de desigualdades válidas para o BQP, denominadas desigualdades do triângulo, foram também propostas por Padberg [27]. Estas foram utilizadas por Palmeira em [7] e são descritas por

$$x_i + x_j + x_k - y_{ij} - y_{ik} - y_{jk} \leq 1,\ i < j < k,\ i,j,k \in N, \tag{3.7}$$

$$-x_i + y_{ij} + y_{ik} - y_{jk} \leq 0,\ i < j < k,\ i,j,k \in N, \tag{3.8}$$

$$-x_j + y_{ij} + y_{jk} - y_{ik} \leq 0,\ i < j < k,\ i,j,k \in N, \tag{3.9}$$

$$-x_k + y_{ik} + y_{jk} - y_{ij} \leq 0,\ i < j < k,\ i,j,k \in N. \tag{3.10}$$

Considerando a linearização onde $i < j,\ i,j \in N$, a desigualdade (3.2) é escrita

como,

$$\sum_{i<j,\ i\in N} w_i y_{ij} + \sum_{i>j,\ i\in N} w_i y_{ji} \leq (c - w_j x_j)\, x_j,\ j \in N \tag{3.11}$$

3.3 Desigualdades válidas para o KP e suas quadratizações

De forma a introduzir desigualdades válidas para o QKP, considere o conjunto

$$K = \left\{ x : \sum_{i\in N} w_i x_i \leq c,\ x_i \in \{0,1\}^n,\ i \in N \right\} \tag{3.12}$$

associado ao KP.

Definição 3.3.1. *Seja $C \subseteq N$. Dizemos que C é uma cover se $\sum_{i\in C} w_i > c$. Uma cover é mínima se $C \setminus \{i\}$ não é uma cover para algum $i \in C$. Veja [21].*

Considere um *vetor* de incidência (ou vetor caracteristico), $x^c \in \{0,1\},\ i \in N$, associado a C e definido como se segue: $x_i^c = 1$ se $i \in C$ e $x_i^c = 0$ se $i \notin C$. Assim, sendo C é uma *cover* se e somente se x^c inviável para K.

Proposição 3.3.1. *Seja $C \subseteq N$ uma cover para K. A desigualdade de cover,*

$$\sum_{i\in C} x_i \leq |C| - 1, \tag{3.13}$$

é válida para K. Veja [21].

Demonstração. Seja $R \subseteq N$. Suponha que $x^R \in K$ e não satisfaça a desigualdada de *cover*, ou seja, $\sum_{i\in C} x_i^R > |C| - 1$, então $|R \cap C| = |C|$ e assim $C \subseteq R$. Segue que $\sum_{i\in N} a_i x_i^R = \sum_{i\in R} a_i \geq \sum_{i\in C} a_i > c$, logo $x^R \notin K$. □

Definição 3.3.2. *Seja C uma cover de K, a extensão de C é dada pelo subconjunto*

$$E(C) = C \cup \{i \in N \setminus C\ :\ w_i \geq w_j,\ \forall\ j \in C\}$$

Proposição 3.3.2. *Seja C uma cover para K. Então a desigualdade de extended cover,*

$$\sum_{i\in E(C)} x_i \leq |C| - 1 \tag{3.14}$$

é válida para K. Veja [21].

Demonstração. Seja $R \subseteq N$. Suponha que $x^R \in K$ e não satisfaça a desigualdade de *extended cover*, ou seja, $\sum_{i \in E(C)} x_i^R > |C| - 1$, então $|R \cap E(C)| = |E(C)|$ e assim $E(C) \subseteq R$. Segue que $\sum_{i \in N} a_i x_i^R = \sum_{i \in R} a_i \geq \sum_{i \in C} a_i > c$, logo $x^R \notin K$. □

A partir das desigualdades válidas para KP podemos obter desigualdades válidas para QKP. Por exemplo, multiplicando a desigualdade de *cover*,

$$\sum_{i \in C} x_i \leq |C| - 1$$

por $x_j,\ j \neq i,\ j \in N$, obtemos

$$\sum_{i \in C} x_i x_j \leq (|C| - 1) x_j,\ j \neq i,\ j \in N. \tag{3.15}$$

Da mesma forma, multiplicado a desigualdade de *extended cover*,

$$\sum_{i \in E(C)} x_i \leq |C| - 1$$

por $x_j,\ j \neq i,\ j \in N$, obtemos

$$\sum_{i \in E(C)} x_i x_j \leq (|C| - 1) x_j,\ j \neq i,\ j \in N. \tag{3.16}$$

Note que a mesma observação feita na Seção 3.2, relativa à linearização das desigualdades (3.2), também se aplica à linearização das desigualdades que acabamos de descrever.

3.4 Identificação e *liftings* para as desigualdades de *cover*

Seja $\mathbf{x}^*$ uma solução ótima para o Problema de Relaxação Contínua do KP, formulado como

$$\max \sum_{i \in N} q_i x_i \tag{3.17}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.18}$$

$$x_i \in [0, 1],\ i \in N. \tag{3.19}$$

Associado a $\mathbf{x}^*$, o Problema de Separação (PS) das desigualdades de *cover* con-

siste em identificar uma desigualdade de *cover* violada em $\mathbf{x}^*$ ou estabelecer que nenhuma desigualdade desse tipo existe.

Um procedimento para resolver o PS definido acima, foi sugerido por Crowder, Johnson e Padberg em [28]. Para descrevê-lo, considere o KP,

$$\gamma = \min \left\{ \sum_{i \in N} (1 - x_i^*) t_i : \sum_{i \in N} w_i t_i > c,\ t_i \in \{0,1\},\ i \in N \right\} \tag{3.20}$$

e o teorema que se segue, relativo a ele.

Teorema 3.4.1. *Veja [21]*

1. *se* $\gamma \geq 1$, $\mathbf{x}^*$ *satisfaz a todas as desigualdades de cover.*
2. *se* $\gamma < 1$, *com uma solução ótima para (3.20) dada por* $\mathbf{t}^R$, *a desigualdade de cover* $\sum_{i \in R} x_i \leq |R| - 1$ *é violada em* $\mathbf{x}^*$ *por uma quantidade* $1 - \gamma$.

Demonstração. Reescrevendo a desigualdade de *cover* temos,

$$\begin{aligned} \sum_{i \in C} x_i &\leq |C| - 1 \\ &\leq \underbrace{1 + \ldots + 1}_{C} - 1 \\ 1 &\leq \underbrace{(1 - x_i) + \ldots + (1 - x_i)}_{C} \\ &\leq \sum_{i \in C} (1 - x_i) \end{aligned}$$

Observe que o PS de desigualdades de *cover* reduz-se a procurar um subconjunto $C \subseteq N$ que define uma *cover* tal que $\sum_{i \in C} (1 - x_i^*) < 1$, onde x^* é solução de (3.17). Isto pode ser feito resolvendo-se o seguinte problema

$$\gamma = \min \sum_{i \in N} (1 - x_i^*) t_i \tag{3.21}$$

$$\text{s.a} \sum_{i \in N} w_i t_i > c \tag{3.22}$$

$$t_i \in \{0,1\},\ i \in N \tag{3.23}$$

Seja t^R solução ótima do problema (3.21)-(3.23) onde t^R é um vetor caracteristico associado a R, ou seja $t_i^R = 1$ se $i \in R$, $t_i^R = 0$ se $i \notin R$.

Seja γ^* valor ótimo do problema (3.21)-(3.23). Se $\gamma^* \geq 1$ temos que x^* satisfaz todas as desigualdades de *cover*, se $\gamma^* < 1$ então R é uma *cover* e $\sum_{i \in R} (1 - x_i)$ é violada em x^* por uma quantidade $1 - \gamma^*$.

□

Seja $conv(K)$ o fecho convexo definido pelas soluções viáveis de KP. Balas em [29] e Wolsey em [30] mostraram que, dada uma desigualdade de *cover* mínima, existe no mínimo uma faceta definida pela desigualdade:

$$\sum_{i \in C} x_i + \sum_{i \in N \setminus \{C\}} \alpha_i x_i \leq |C| - 1 \tag{3.24}$$

onde $\alpha_i \geq 0$ para todo $i \in N \setminus \{C\}$. Esta desigualdade é denominada desigualdade de *lifting*.

Wolsey em [21] descreve um procedimento para encontrar desigualdades de *lifting* que definem facetas para a $conv(K)$ quando C é uma *cover* mínima e $w_i \leq c$ para todo $i \in N$.

Algoritmo 2: Procedimento para encontrar desigualdades de *lifting* [21]

$r = |N \setminus C|$;
Ordene os índices $i \in N \setminus C$ em ordem crescente de acordo com cada peso w_i, definindo uma ordem $\{i_1, i_2, \ldots, i_r\}$;
for $t = 1$ **to** r **do**
Resolva o problema,

$$\begin{aligned} \zeta_t = \max & \sum_{j=1}^{t-1} \alpha_{i_j} x_{i_j} + \sum_{i \in C} x_i \\ \text{s.a} & \sum_{j=1}^{t-1} w_{i_j} x_{i_j} + \sum_{i \in C} w_i x_i \leq c - w_{i_t} \\ & x \in \{0,1\}^{|C|+t-1}. \end{aligned}$$

$\alpha_{i_t} = |C| - 1 + \zeta_t$;
end for

3.5 Modelos para o QKP

Ao longo das últimas décadas vários modelos foram propostos para o QKP, os quais utilizam de algumas das desigualdades citadas acima. Abaixo citamos os diversos modelos propostos em trabalhos relacionados ao QKP ao longo das últimas décadas.

- Gallo, Hammer and Simeone em [4] e Chaillou, Hansen and Mahieu em [15]

utilizaram o modelo,

$$\max \sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij} x_i x_j \tag{3.25}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.26}$$

$$x_i \in \{0,1\}, \ i \in N. \tag{3.27}$$

- Michelon e Veuilleux em [9] reformularam o QKP definindo variáveis auxiliares ligadas à variáveis originais através de desigualdades de igualdade. Esta formulação foi utilizada em um algoritmo de decomposição Lagrangeana. Segue formulação,

$$\max \sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij} x_i x_j \tag{3.28}$$

$$\text{s.a} \sum_{i \in N} w_i y_i \leq c \tag{3.29}$$

$$x_i = y_i, \ i \in N \tag{3.30}$$

$$x_i \in \{0,1\}, \ i \in N \tag{3.31}$$

$$y_i \in \{0,1\}, \ i \in N. \tag{3.32}$$

- Billionnet e Calmels em [8] reformularam o QKP utilizando desigualdades de linearização, (3.4)-(3.6), desigualdade da mochila quadratizada, (3.3), e cortes de *Chvátal-Gomory*, (3.7). Segue formulação,

$$\max \sum_{i \in N} p_{ii} x_i + \sum_{i \in N} \sum_{j \in N \setminus \{i\}} p_{ij} y_{ij} \tag{3.33}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.34}$$

$$y_{ij} \leq x_i, \ i < j, \ i,j \in N, \tag{3.35}$$

$$y_{ij} \leq x_j, \ i < j, \ i,j \in N, \tag{3.36}$$

$$x_i + x_j \leq 1 + y_{ij}, \ i < j, \ i,j \in N, \tag{3.37}$$

$$\sum_{i<j, \ i \in N} w_i y_{ij} + \sum_{i>j, \ i \in N} w_i y_{ji} \leq (c - w_j) x_j, \ j \in N, \tag{3.38}$$

$$x_i + x_j + x_k - y_{ij} - y_{ik} - y_{jk} \leq 1, \ i < j < k, \ i,j,k \in N, \tag{3.39}$$

$$x_i \in \{0,1\}, \ i \in N, \tag{3.40}$$

$$y_{ij} \in \{0,1\}, \ i < j, \ i,j \in N. \tag{3.41}$$

- Palmeira em [7] reformulou o QKP utilizando desigualdades de linearização,

(3.4)-(3.6), desigualdade da mochila quadratizada, (3.3), e desigualdades do triângulo, (3.7)-(3.10). Abaixo formulação,

$$\max \sum_{i \in N} p_{ii} x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} p_{ij} y_{ij} \tag{3.42}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.43}$$

$$\sum_{i<j,\ i \in N} w_i y_{ij} + \sum_{i>j,\ i \in N} w_i y_{ij} \leq (c - w_j)\, x_j,\ j \in N \tag{3.44}$$

$$y_{ij} \leq x_i,\ i < j,\ i, j \in N, \tag{3.45}$$

$$y_{ij} \leq x_j,\ i < j,\ i, j \in N, \tag{3.46}$$

$$x_i + x_j \leq 1 + y_{ij},\ i < j,\ i, j \in N, \tag{3.47}$$

$$x_i + x_j + x_k - y_{ij} - y_{ik} - y_{jk} \leq 1,\ i < j < k,\ i, j, k \in N, \tag{3.48}$$

$$-x_i + y_{ij} + y_{ik} - y_{jk} \leq 0,\ i < j < k,\ i, j, k \in N, \tag{3.49}$$

$$-x_j + y_{ij} + y_{jk} - y_{ik} \leq 0,\ i < j < k,\ i, j, k \in N, \tag{3.50}$$

$$-x_k + y_{ik} + y_{jk} - y_{ij} \leq 0,\ i < j < k,\ i, j, k \in N, \tag{3.51}$$

$$x_i \in \{0, 1\},\ i \in N, \tag{3.52}$$

$$y_{ij} \in \{0, 1\},\ i < j,\ i, j \in N. \tag{3.53}$$

- Caprara, Pisinger e Toth em [5] reformularam o QKP utilizando desigualdades de linearização, (3.4)-(3.6), desigualdade da mochila quadratizada, (3.3). Segue formulação,

$$\max \sum_{i \in N} p_{ii} x_i + \sum_{i \in N} \sum_{j \in N \setminus \{i\}} p_{ij} y_{ij} \tag{3.54}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.55}$$

$$\sum_{i \in N \setminus \{j\}} w_i y_{ij} \leq (c - w_j)\, x_j,\ j \in N \tag{3.56}$$

$$0 \leq y_{ij} \leq x_j \leq 1,\ j \neq i,\ i, j \in N, \tag{3.57}$$

$$y_{ij} = y_{ji},\ i < j,\ i, j \in N, \tag{3.58}$$

$$x_i \in \{0, 1\},\ i \in N, \tag{3.59}$$

$$y_{ij} \in \{0, 1\},\ i \neq j,\ i, j \in N. \tag{3.60}$$

- Billionnet e Soutif em [31] propuseram duas formulação. Uma primeira for-

mulação a qual denominaram LIN1,

$$(LIN1) : \max \sum_{i \in N} p_{ii} x_i + \sum_{i=1}^{n-1} z_i \tag{3.61}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c$$

$$z_i \leq \overline{\phi}_i x_i,\ i \in \{1, \ldots, n-1\} \tag{3.62}$$

$$z_i \leq \phi_i(x),\ i \in \{1, \ldots, n-1\}, \tag{3.63}$$

$$x_i \in \{0, 1\},\ i \in N. \tag{3.64}$$

onde $\phi_i = \sum_{j=i+1}^{n} p_{ij} x_j,\ i \in \{1, \ldots, n-1\}$ e $\overline{\phi_i},\ i \in \{1, \ldots, n-1\}$ é solução do seguinte problema,

$$(AUX1) : \max\ \phi_i(x) \tag{3.65}$$

$$\text{s.a} \sum_{j=i+1}^{n} w_j x_j \leq c - w_i \tag{3.66}$$

$$x_j \in \{0, 1\},\ j \in \{i+1, \ldots, n\}. \tag{3.67}$$

E uma segunda formulação a qual denominaram LIN2,

$$(LIN2) : \max \sum_{i \in N} p_{ii} x_i + \sum_{i=1}^{n-1} z_i + \sum_{i=1}^{n-1} \underline{\phi}_i x_i \tag{3.68}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.69}$$

$$z_i \leq (\overline{\phi}_i - \underline{\phi}_i) x_i,\ i \in \{1, \ldots, n-1\} \tag{3.70}$$

$$z_i \leq \phi_i(x) - \underline{\phi}_i,\ i \in \{1, \ldots, n-1\} \tag{3.71}$$

$$x_i \in \{0, 1\},\ i \in N. \tag{3.72}$$

O cálculo de $\underline{\phi}_i$ é feito de acordo com $\overline{x}$, o qual é uma solução viável para o QKP obtida através da heurística proposta por Billionnet e Camels [8], onde

temos que $\alpha = f(\overline{x})$. Deste modo se $\overline{x}_i = 1$, $\underline{\phi}_i$ é solução do seguinte problema,

$$(AUX2) : \min \ \phi_i(x) \tag{3.73}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.74}$$

$$z_k \leq \overline{\phi}_k x_k, \ k \in \{1, \ldots, n-1\} \tag{3.75}$$

$$z_k \leq \phi_k(x), \ k \in \{1, \ldots, n-1\} \tag{3.76}$$

$$\sum_{j \in N} c_j x_j + \sum_{k=1}^{n-1} z_k \geq \alpha \tag{3.77}$$

$$x_i \in [0, 1], \ i \in N. \tag{3.78}$$

se $\overline{x}_i = 0, \ \underline{\phi}_i$ é solução do problema

$$(AUX3) : \min \ \phi_i(x) \tag{3.79}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.80}$$

$$z_k \leq \overline{\phi}_k x_k, \ k \in \{1, \ldots, n-1\} \tag{3.81}$$

$$z_k \leq \phi_k(x), \ k \in \{1, \ldots, n-1\} \tag{3.82}$$

$$\sum_{j \in N} c_j x_j + \sum_{k=1}^{n-1} z_k \geq \alpha \tag{3.83}$$

$$x_i = 0 \tag{3.84}$$

$$x_i \in [0, 1], \ i \in N. \tag{3.85}$$

temos que $\overline{\phi}_i$ é cálculado para todo $i \in \{1, \ldots, n-1\}$.

- Rodrigues, Quadri, Michelon e Gueye [16] propuseram um esquema de linearização para o QKP, ao qual denominaram t-linearização. Neste esquema uma nova variável $t \in \mathbb{R}$ é criada, e esta substitui o termo quadrático na função objetivo. Este termo quadrático é agora inserido no problema como restrição que passa a limitar t superiormente. O problema é então reescrito como,

$$(TLP(Q)): \ \max t + \sum_{i \in N} p_{ii} x_i \tag{3.86}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.87}$$

$$t \leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} p_{ij} x_i x_j \tag{3.88}$$

$$t \in \mathbb{R}, \ x_i \in \{0,1\}, \ i \in N. \tag{3.89}$$

Em seguida a restrição (3.88) é linearizada e o QKP é reescrito como,

$$(TLP): \ \max t + \sum_{i \in N} p_{ii} x_i \tag{3.90}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{3.91}$$

$$t \leq \sum_{i=2}^{n} \sum_{j=1}^{i-1} p_{\pi(j)\pi(i)} x_{\pi(j)}, \ \forall \ \pi \in S_n \tag{3.92}$$

$$t \in \mathbb{R}, \ x_i \in \{0,1\}, \ i \in N. \tag{3.93}$$

Onde S_n é o conjunto das permutações dos índices $i, j \in N$ geradas ao longo do algoritmo.

3.6 Algoritmos para o QKP

Gallo, Hammer and Simeone em [4] propuseram um algoritmo *Branch-and-Bound* [21, 32] para resolver o QKP. Neste trabalho eles obtiveram um limite superior substituindo a função objetivo $f(x) = \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j$ por uma função linear $g(x)$ tal que $g(x)$ domina $f(x)$ em todos os ponto viáveis. O plano superior para $f(x)$ foi obtido por $g(x) = \sum_{i=1}^{n} v_i x_i$, onde

$$v_i = \max \left\{ \sum_{j=1}^{n} p_{ij} x_j : \sum_{j=1}^{n} w_j x_j \leq c, \ x_j \in \{0,1\}, \ j \in N \right\}, \ i \in N$$

Depois resolveram o seguinte problema,

$$\begin{aligned} &\max\ g(x) \\ &\text{s.a} \sum_{i \in N} w_i x_i \leq c \\ &\quad x_i \in \{0,1\}\,,\ i \in N. \end{aligned}$$

Chaillou, Hansen and Mahieu em [15] propuseram um algoritmo *Branch-and-Bound* [21, 32] para resolver o QKP. O limite superior foi calculado com base em relaxação Lagrangeana [19].

Michelon e Veuilleux em [9] propuseram um algoritmo *Branch-and-Bound* [21, 32] para resolver o QKP e utilizaram decomposição Lagrangeana [33] para calcular o limite superior.

Billionnet e Calmels em [8] propuseram um algoritmo *Branch-and-Bound* para resolver o QKP e obtiveram o limite superior utilizando relaxação Lagrangeana [19].

Billionnet, Faye e Soutif em [11] propuseram um algoritmo *Branch-and-Bound* [21, 32] para resolver o QKP e obtiveram o limite superior utilizando decomposição Lagrangeana [33] e partição de conjuntos.

Palmeira em [7] propôs um algoritmo *Branch-and-Bound* [21, 32] para resolver o QKP e utilizou um algoritmo *Relax-and-Cut* [22–24].

Caprara, Pisinger e Toth em [5] propuseram um algoritmo *Branch-and-Bound*, onde reformularam o modelo (3.54)-(3.60), e utilizaram técnicas de fixação de variáveis e redução do problema.

Billionnet e Soutif em [31] propuseram reformulações, citadas na Seção 3.5, do QKP e utilizando resolvedores para MIP mostraram que é possível resolver eficientemente QKP comparado aos métodos existentes na literatura.

Rodrigues, Quadri, Michelon e Gueye [16] propuseram um algoritmo *Branch-and-Bound* baseado no esquema de t-linearização para o QKP citado na Seção 3.5.

3.7 Limites inferiores para o QKP

Limites inferiores podem ser obtidos de modo eficiente através de heurísticas. Podemos dividir as heurísticas para obter limite inferior para o QKP em duas classes: a primal, que mantém a viabilidade ao longo da construção, e a dual a qual inicia com uma solução inviável e constrói uma solução viável.

Billionnet e Calmels em [8] propuseram uma heurística dual, onde primeiro geram uma solução gulosa inicial definindo $x_j = 1$ para todo $j \in N$, e então interativamente trocam o valor das variáveis de 1 para 0, de modo a atingir o menor decréscimo na função objetivo, até que uma solução viável é atingida. No segundo passo um

procedimento ao qual denominamos *melhora* é executado, onde uma sequência de iterações são executadas com a finalidade de melhorar a solução através de busca local. Seja $S = \{j \in N : x_j = 1\}$ o conjunto que define a atual solução. Para cada $j \in N \setminus S$, se $w_j + \sum_{l \in S} w_l \leq c$ defina $I_j = \emptyset$. E seja δ_j o valor que define o crescimento da função objetivo quando x_j toma valor 1. Caso contrário, seja δ_j o maior crescimento quando definimos $x_j = 1$ e $x_i = 0$ para algum $i \in S$ tal que $w_j - w_i + \sum_{l \in S} w_l \leq c$, e seja $I_j = \{i\}$. Escolha um k tal que $\delta_k = \max_{j \in N \setminus S} \delta_j$, a heurística finaliza se $\delta_k \leq 0$, caso contrário o corrente conjunto solução é definido como $S \setminus I_k \cup \{k\}$ e outra iteração é executada.

Caprara, Pisinger e Toth em [5] utilizaram a heurística proposta por Billionnet e Calmels em [8] na primeira parte de sua heurística, antes de executar o Método do Subgradiente(SM), e a cada duas iterações do SM tomaram o piso da solução do LP relaxado do Problema (3.95)-(3.97), gerando desta forma uma solução viável para o QKP e aplicaram a esta solução viável o procedimento *melhora*.

Fomeni e Letchford em [34] propuseram uma heurística baseada em Programação Dinâmica para o QKP, uma adaptação do algoritmo de Programação Dinâmica para o KP. A seguir descrevemos a mesma.

Seja $k = 1, \ldots, n$ os estágios ao se resolver QKP. Para $r \in \{0, \ldots, c\}$, seja $f(r)$ o melhor benefício corrente colocado na mochila até então, independente do estágio k, tendo um peso total de r. Também para $r \in \{0, \ldots, c\}$ e $i = 1, \ldots, n$, seja $B(r, i)$ uma variável booleana tomando valor 1 se o item i é atualmente selecionado no conjunto cujo benefício é $f(r)$, e 0 caso contrário. $S(k, r)$ define o conjunto solução dos itens no estágio k com peso total igual a r. A heurística pode ser implementada como descrita no pseudocódigo a seguir.

Algoritmo 3: Heurística de Programação Dinâmica para QKP

for $r := 0$ **to** c **do**
  $f(r) \leftarrow 0$;
end for
for $r := 0$ **to** c **do**
  for $i := 1$ **to** n **do**
    $B(r, i) \leftarrow 0$;
  end for
end for
for $k := 1$ **to** n **do**
  for $r := c$ **to** 0 **do**
    if $r \geq w_k$ **then**
      $\beta \leftarrow f(r - w_k) + p_{kk} + 2 \times \sum_{i=1}^{k-1} B(r - w_k, i) p_{ik}$
      if $\beta > f(r)$ **then**
        $S(k, r) \leftarrow S(k - 1, r - w_k) \cup \{k\}$;
        $f(r) \leftarrow \beta$;
        $B(r, k) \leftarrow 1$;
      end if
    end if
  end for
end for
Seja $r* \leftarrow \arg\max_{0 \leq r \leq c} f(r)$.

Fomeni e Letchford em [34] aplicaram ainda algumas melhorias à heurística de Programação Dinâmica para o QKP. Eles propuseram uma reordenação da matriz de benefícios de acordo com planos superiores. Entre os planos superiores testados, o plano gerado por (3.94) gerou melhores resultados. Em seguida a matriz de benefício foi reordenada de acordo com $\frac{\pi_j}{w_j}$, $j \in N$, onde π_j é o plano superior gerado.

Outra melhoria foi a aplicação de um procedimento ao qual denomiraram *breaking ties intelligently*. Suponha que para um dado r no Algoritmo 3, encontramos que $\beta = f(r)$. Então se $|S(k-1, r-w_k)| + 1 > |S(k, r)|$ definimos $S(k, r) = S(k-1, r-w_k) \cup \{k\}$.

Uma terceira melhoria foi a aplicação da busca local através do procedimento de *fill-up-and-exchange*, semelhante ao procedimento *melhora* do modo como foi utilizado por Caprara, Pisinger e Toth em [5].

A complexidade do algoritmo de acordo com [34] é $O(n^2 c)$.

3.8 Técnicas de fixação e redução de variáveis

Ao longo do SM podemos tentar reduzir tamanho do problema através de técnicas de fixação de variáveis, desta forma podemos resolver instância maiores para o QKP.

Caprara, Pisinger e Toth em [5] utilizaram a seguinte regra para a fixação de variáveis: suponha que no final do SM obtemos uma solução ótima $\overline{x}$ com valor ótimo $\overline{ub}$ para o Problema Lagrangeano Relaxado e $\underline{lb}$ solução viável para Problema Original. Seja u_j^1 o limite superior do Problema Lagrangeano Relaxado obtido pela imposição da restrição adicional $x_j = 1$ para algum $j \in N$. Se $u_j^1 < \underline{lb}$ então podemos fixar x_j em 0. De modo semelhante, seja u_j^0 um limitante superior para o Problema Lagrangeano Relaxado quando impomos x_j igual a 0, se $u_j^0 \leq \underline{lb}$ podemos fixar x_j em 1.

Se variáveis x_j forem fixadas em 0 ou 1, a correspondente linha e coluna associada a j é removida do problema. No caso de fixação de variáveis em 1 além da linha e coluna associada a j serem removidas do problema o valor q_i associado a diagonal $i \in N, i \neq j$ é aumentado em uma quantidade $p_{ij} + p_{ji}$ e a capacidade da mochila c é decrescida em w_j.

Palmeira [7] propôs um procedimento de fixação utilizando a idéia de custo reduzido: Seja $\overline{x}$ solução obtida ao longo do SM do Problema Lagrangeano Relaxado e c_i^R o custo reduzido associado a variável $\overline{x}_i$ onde $c_i^R = q_i - \frac{q_k}{w_k} w_i$ se $\overline{x}_i = 0$, $i \in N$, ou $c_i^R = -q_i + \frac{q_k}{w_k} w_i$ se $\overline{x}_i = 1$, $i \in N$, e k é o indice da variável fracionária na solução do Problema Lagrangeano Relaxado. Seja $\overline{ub}$ e $\underline{lb}$ valor ótimo do Problema Lagrangeano Relaxado Contínuo e solução viável para Problema Original respectivamente, se $\overline{ub} + c_i^R < \underline{lb}$ então x_i pode ser fixado na solução corrente.

3.9 Dois *frameworks* para o QKP

Nesta parte do trabalho descrevemos dois *frameworks* propostos para resolver o QKP de forma exata. O primeiro foi proposto por Caprara, Pisinger e Toth em [5] e o segundo por Rodrigues, Quadri, Michelon e Gueye em [16]. Para maiores detalhes consulte as fontes.

3.9.1 Caprara, Pisinger e Toth

Caprara, Pisinger e Toth propuseram em [5] limites superiores para o QKP, esses são obtidos seguindo a abordagem de planos superiores introduzida em [4]. Esses planos superiores são incoporados em [5] seguindo um *framework* de relaxação Lagrangeana, levando a bons limites superiores para o QKP. Estes limites são obtidos com a resolução de $n + 1$ KP's auxiliares. Os primeiros n KP's estão associados a

cada linha da matriz de benefícios P, e são utilizados para gerar coeficientes a serem introduzidos na função objetivo do último KP. Por sua vez, uma solução ótima para este KP, no caso, a relaxação do LP associado ao mesmo, nos fornece um limite superior válido para o QKP. Denominamos este último KP, problema limitante para o QKP.

Os autores consideraram a formulação (3.54)-(3.60) para o QKP.

As restrições $\{y_{ji} = y_{ij},\ i < j,\ i, j \in N\}$ foram relaxadas pelo autores seguindo uma forma Lagrangeana. Feito isto, os autores implementaram uma estratégia de decomposição para resolver o LP relaxado contínuo associado ao Subproblema Lagrangeano resultante. Essa estratégia exige que $n + 1$ KP's sejam resolvidos. Estes são precisamente os $n + 1$ KP's citados acima.

Os coeficientes $\pi_j,\ j \in N$, a serem introduzidos na função objetivo do problema limitante para o QKP, são obtidos pela resolução do seguinte KP, para cada $j \in N$.

$$\pi_j = q_j + \max \left\{ \sum_{i \in N \setminus \{j\}} p_{ij} \overline{x}_i : \sum_{i \in N \setminus \{j\}} w_i \overline{x}_i \leq c - w_j,\ \overline{x}_i \in \{0, 1\},\ i \in N \setminus \{j\} \right\} \tag{3.94}$$

Para um dado $j \in N$, (3.94) tenta indentificar o melhor item a ser colocado junto com o item j na mochila, assim o maior benefício possível é atingido. Esta é claramente uma estratégia gulosa já que os benefícios cruzados entre dois itens não são levados em consideração. Portanto, (3.94) retorna um limite superior sobre uma verdadeira contribuição indivídual que o item j traria para uma solução do QKP que o colocaria na mochila. Em seguida, um limite superior válido para o QKP é obtido se resolvendo o seguinte problema KP:

$$U_{cpt} = \max \sum_{j \in N} \pi_j x_j^o \tag{3.95}$$

$$\text{s.a.} \sum_{j \in N} w_j x_j^o \leq c, \tag{3.96}$$

$$x_j^o \in \{0, 1\},\ j \in N. \tag{3.97}$$

Observe que um limite superior barato computacionalmente é obtido, se em vez de resolvemos o exato de (3.94), optamos por resolver o correspondente LP relaxado. Fazendo assim obtemos um limite superior válido para cada $\pi_j,\ j \in N$. Após obtemos esses limites, os substituimos no problema (3.95)-(3.97), onde um limite superior U_{cpt} é gerado. Esta estratégia de resolver o relaxado e não o exato pode ainda ser seguida ao se resolver (3.95)-(3.97). Como indicado em [5] este limite superior é obtido em tempo $O(n^2)$ e acabou sendo uma escolha para o algoritmo exato para o QKP introduzido em [5].

Como implementado em [5], SM executa no máximo $200 + n$ iterações. Adicionalmente o valor inicial de μ, a ser utilizado no cálculo do tamanho do passo, é igual a 1, sendo reduzido pela metade a cada 20 consecutivas iterações do SM sem melhora no limite superior. O fator de tolerância definido como $\epsilon = 10^{-6}$ é também utilizado com critério de parada.

Finalizado um ciclo de iterações do subgradiente, é executado um procedimento de fixação de variáveis e redução no tamanho do problema como citado em Seção 3.8. Caso alguma variável seja fixada, o problema é reduzido e um novo ciclo do SM é executado. A matriz de benefícios modificados a ser utilizada como entrada no processo de fixação de variáveis está associada a solução ótima no final do SM. O limite inferior utilizado é o melhor limite obtido ao longo do SM, esse baseia-se na heurística proposta por Billionnet e Calmels, a qual foi descrito em Seção 3.7.

Após ciclos do SM e procedimentos de fixação e redução de variáveis. Seja $\overline{N}$ o conjunto de variáveis não fixada e $\overline{P} = (\overline{p_{ij}})$ matriz dos benefícios Lagrangeano associado a melhor solução encontrada pelo SM, $\overline{q} = (\overline{q}_i)$ o vetor diagonal dos benefícios modificados de acordo com variáveis fixadas em 1 pelo procedimento de fixação como citado em Seção 3.8.

O *Branch-and-Bound* proposto pelos autores é baseado em busca em profundidade, e a ordem na qual as variáveis são fixadas pelo *branching* é definida por,

$$\gamma_i = \overline{q}_i + \max \left\{ \sum_{j \in \overline{N} \setminus \{i\}} \overline{p}_{ji} x_j : \sum_{j \in \overline{N} \setminus \{i\}} w_j x_j \leq c - w_i,\ x_j \in [0,1], j \in \overline{N} \setminus \{i\} \right\}. \tag{3.98}$$

Com a finalidade de acelerar a busca, os autores definaram um vetor $\underline{w} \in \mathbb{N}^{\overline{N}}$ para armazenar os minímos pesos definido por,

$$\underline{w}_i = \min_{j \geq i} w_j,\ i \in \overline{N}. \tag{3.99}$$

Sempre que uma variável x_j é fixada em $\overline{x}_j,\ j = 1, \ldots, i-1$, tal que $\sum_{j=1}^{i-1} w_j x_j + \underline{w}_i > c$, não há necessidade de executar *branching* naquele nó, já que nenhuma variável que está em um nível abaixo daquele nó pode ser fixada em 1.

3.9.2 Rodrigues, Quadri, Michelon e Gueye

Rodrigues, Quadri, Michelon e Gueye [16] propuseram um algoritmo exato baseado em um *framework* de linearização para o QKP, o qual denominaram t-linearização. Esse *framework* fornece um limite superior, a ser utilizado em um esquema *Branch-and-Bound*.

O *framework* de t-linearização foi proposto em 2 passos. Em um primeiro passo o termo quadrático da função objetivo do QKP é substituido por um $t \in \mathbb{R}$ e é adicionado uma restrição quadrática ao problema, de acordo com (3.86)-(3.89). O segundo passo é a linerização desta adicional restrição quadrática. Essa nova restrição é derivada da caracterização da envoltória convexa quadrática.

Considere o problema (3.86)-(3.89), o qual denominamos $TLP(Q)$. Para o *framework* de linearização considere em um primeiro momento o seguinte conjunto:

$$X = \left\{ (x,t) \in \mathbb{R}^{n+1} : t \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} x_i x_j,\ x \in \{0,1\} \right\}. \tag{3.100}$$

Assumindo que é possível descrever completamente a envoltória convexa de X, $conv(X)$, temos o seguinte problema:

$$\max\ t + \sum_{i \in N} q_i x_i, \tag{3.101}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c, \tag{3.102}$$

$$(x,t) \in conv(X), \tag{3.103}$$

$$t \in \mathbb{R},\ x \in \{0,1\}^n. \tag{3.104}$$

Dado agora o conjunto:

$$QP_n = \left\{ (x,t) \in \mathbb{R}^{n+1}\ :\ t \leq \sum_{i=2}^{n} \sum_{j+1}^{n-1} p_{\pi(j)\pi(i)} x_{\pi(i)},\ x \in [0,1],\ \forall x \in S_n \right\} \tag{3.105}$$

onde S_n é o conjunto de todas as permutações de $\{1, \ldots, n\}$. Os autores mostraram que $QP_n = conv(X)$.

Substituindo a desigualdade (3.103) de acordo com QP_n, obtemos um Problema de Programação Linear 0-1, TLP, com uma variável adicional t e $n!$ restrições adicionais, o qual é equivalente ao QKP.

Os autores observaram não haver necessidade de gerar todas as $!n$ restrições para o problema. De fato, somente um subconjunto destas restrições necessitam ser geradas e adicionadas ao problema, e esse processo de gerar e adicionar restrições se dar através de um algoritmo. Em tal algoritmo, inicialmente somente um subconjunto de restrições são consideradas no TLP, o qual é resolvido fornecendo uma solução $(\overline{x}, \overline{t})$. Uma nova restrição, se existir, violada por $(\overline{x}, \overline{t})$ é adicionada ao *pool* de restrições. Esse processo continua até que nenhuma restrição gerada seja violada pela corrente solução encontrada, desta forma é fornecido uma solução ótima para o QKP.

A mesma ideia pode ser considerada para o TLP relaxado contínuo, $\overline{TLP}$, onde no final do procedimento é fornecido um limite superior para o QKP.

Os autores consideraram o conjunto dos pontos (x,t) que inclui a restrição pertencente a X e a restrição da mochila, ou seja,

$$Y = \left\{ (x,t) \in \mathbb{R}^{n+1} : t \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} x_i x_j,\ \sum_{i \in N} w_i x_i \leq c,\ x \in \{0,1\} \right\} \tag{3.106}$$

Os autores mostraram que Y é o conjunto viável do problema TLP. Mostraram ainda que $conv(Y)$ é a envoltória convexa do conjunto viável de TLP e estabeleceram os seguintes resultados: $opt(QKP) = opt(TLP) = opt(TLP(X)) = opt(TLP(Y))$, onde $opt()$ define a solução ótima do problema.

Diante dos resultados e observações acima aos autores propuseram um esquema para gerar desigualdades válidas para $conv(Y)$ baseado na aproximação relativa a $conv(X)$. Para isto consideraram a seguinte desigualdade que descreve $conv(X)$:

$$t \leq \sum_{i \in N} \alpha_{\pi(i)} x_{\pi(j)}, \forall \pi \in S_n, \tag{3.107}$$

onde o coeficiente $\alpha_{\pi(i)}$ pode ser definido como,

$$\alpha_{\pi(i)} = \max \left\{ \sum_{j=1}^{i-1} p_{\pi(j)\pi(i)} x_{\pi(j)} \ :\ x_{\pi(j)} \in \{0,1\},\ j < i,\ x_{\pi(i)} = 1 \right\} \tag{3.108}$$

De modo similar, desigualdades foram definidas para aproximar $conv(Y)$,

$$t \leq \sum_{i=1}^{n} \beta_{\pi(i)} x_{\pi(i)}, \forall \pi \in S_n, \tag{3.109}$$

onde o coeficiente $\beta_{\pi(i)}$ pode ser definido como,

$$\beta_{\pi(i)} = \max \left\{ \sum_{j=1}^{i-1} p_{\pi(j)\pi(i)} x_{\pi(j)} : \sum_{j=1}^{i-1} w_{\pi(j)} x_{\pi(j)} \leq c - w_{\pi(i)},\ x_{\pi(j)} \in \{0,1\},\ j < i \right\} \tag{3.110}$$

Os autores mostraram que desigualdades do tipo (3.109) são válidas para Y.

O cálculo dos coeficientes de β requerem a resolução de KP's, um problema NP-difícil. Então consideraram a relaxação contínua, $\overline{\beta}$. Se (3.109) é válida para $conv(Y)$, então temos que a desigualdade

$$t \leq \sum_{i=1}^{n} \overline{\beta}_{\pi(i)} x_{\pi(i)},\ \forall \pi \in S_n \tag{3.111}$$

também é válida para $conv(Y)$.

De acordo com (3.107), (3.109) e (3.111) foram definidos os problemas $TLP(\alpha)$, $TLP(\beta)$ e $TLP(\overline{\beta})$ que podem ser reescritos como TLP onde restrições associadas à t são substituidas por (3.107), (3.109) e (3.111), respectivamente.

Supondo X_α, X_β e $X_{\overline{\beta}}$ conjuntos viáveis de $TLP(\alpha), TLP(\beta)$ e $TLP(\overline{\beta})$ respectivamente, os autores mostratam que $X_\alpha \subseteq X_\beta \subseteq X_{\overline{\beta}}$, $opt(TLP(\alpha)) = opt(TLP(\beta)) = opt(TLP(\overline{\beta})) = opt(QKP)$ e $opt(\overline{TLP}(\alpha)) \geq opt(\overline{TLP}(\overline{\beta})) \geq opt(\overline{TLP}(\beta)) \geq opt(QKP)$.

Os autores obtiveram diferentes aproximações para resolver o QKP através de t-linearização, reescrevendo o problema TLP nas formas $TLP(\alpha), TLP(\beta)$ e $TLP(\overline{\beta})$.

Diante do que foi exposto acima os autores formularam um algoritmo para resolver as formulações $TLP(\alpha), TLP(\beta)$ e $TLP(\overline{\beta})$.

Algoritmo 4: Algoritmo para geração de restrição

1: $x_H \leftarrow billionnet()$;
2: $\pi \leftarrow permutacao(x_H)$;
3: adiciona $t \leq \alpha_\pi x$ a $TLP(\alpha)$;
4: $x_\alpha \leftarrow resolve(TLP(\alpha))$;
5: $\pi \leftarrow permutacao(x_\alpha)$;
6: **if** $t > \alpha_\pi x$ **then**
7: adiciona $t \leq \alpha_\pi x$ a $TLP(\alpha)$;
8: $x_\alpha \leftarrow resolve(TLP(\alpha))$;
9: $\pi \leftarrow permutacao(x_\alpha)$;
10: **else**
11: stop;
12: **end if**

O Algoritno 4 fornece um limite superior para o QKP, o qual foi incorporado a um algoritmo *Branch-and-Bound* proposto pelos autores. O procedimento *billionnet()* fornece uma solução viável para o QKP de acordo com a heurística proposta por Billionnet e Calmels [8]. Já o procedimento *permutacao*() gera uma permutação dos elementos pertecentes a solução fornecida por *resolve()*.

Os autores notaram que as restrições vindas de $TLP(\beta)$ e $TLP(\overline{\beta})$, não são suficientes para descrever o correspondente conjunto viável.

Suponha que a restrição abaixo seja gerada através do procedimento de t-linearização,

$$t \leq \sum_{i=1}^{n} \overline{\beta}_{\pi(i)} x_{\pi(i)}, \ \forall \pi \in S_n. \tag{3.112}$$

No intuito de fortalecer a restrição (3.112) os autores propuseram o cálculo de novos coeficientes β^* tal que $\beta^*_{\pi(i)} \leq \overline{\beta}_{\pi(i)}$, para todo $i \in N$. Obtendo-se então um novo

problema, $TLP(\beta^*)$, com as seguintes propriedades $opt(TLP(\beta^*)) = opt(QKP)$ e $opt(\overline{TLP}(\beta^*)) \leq opt(\overline{TLP}(\overline{\beta}))$.

Sem perda de generalidade, considere a permutação identidade para o algoritmo no cálculo de β^*. Seja (x^*, t^*) solução ótima para o TLP, então $t^* = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} x_i^* x_j^*$ o qual pode ser reescrita como,

$$t^* = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (p_{ij} - \delta_{ij}) x_i^* x_j^* + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta_{ij} x_i^* x_j^*, \ \delta_{ij} \in \mathbb{R}, \ 1 \leq i < j \leq n.$$

Temos que,

$$t^* = \sum_{i=1}^{n} \left[\sum_{j=1}^{i-1} (p_{ji} - \delta_{ij}) x_j^* x_j^* + \sum_{j=i+1}^{n} \delta_{ij} x_j^* \right] x_i^* \tag{3.113}$$

Para cálculo dos coeficientes de β^*, os quais fornecem uma melhor aproximação para os coeficientes de x_i^* em (3.113), foi definido o problema

$$P_i(\delta): \ \max \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (p_{ij} - \delta_{ij}) x_i x_j + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta_{ij} x_i x_j \tag{3.114}$$

$$s.a \sum_{j \in N} w_j x_j \leq c \tag{3.115}$$

$$x_i = 1 \tag{3.116}$$

$$x_j \in \{0, 1\}, i \neq j, \ j = 1, \ldots, n. \tag{3.117}$$

Consequentemente se $\beta_i(\delta) = opt(P_i(\delta))$, então

$$t \leq \sum_{i=1}^{n} \beta_i(\delta) x_i, \ \forall \delta \in \mathbb{R}^{\frac{n(n-1)}{2}} \tag{3.118}$$

são desigualdades válidas para $conv(Y)$. Para $\delta = 0$ temos que $\beta(\delta) = \beta$ e $opt(\overline{P}_i(\delta)) = \overline{\beta}(\delta) = \overline{\beta}$

O problema $\overline{P}_i(\delta)$ é um Problema de Programação Linear, pela teoria de Programação Linear se for adicionado aos coeficientes da função objetivo de cada variável não básica, em sua base ótima, uma quantidade que é menor ou igual ao custo reduzido de cada variável, então a base ótima não se altera.

De acordo com o exposto acima foi definido uma série de vetores $\delta^k = (\delta_{ij}^k)_{1 \leq i < j \leq n} \in \mathbb{R}^{\frac{n(n-1}{2}}, \ k = 1, \ldots, n-1$, como segue

$$\delta_{ij}^k = \begin{cases} 0, & \text{se } k = 0, \\ -\overline{c}_j^i(\delta^{k-1}), & \text{se } i = k,\ k = 1, \ldots, n-1, \\ \delta_{ij}^{k-1}, & \text{se } i \neq k,\ k = 1, \ldots, n-1. \end{cases}$$

onde $\overline{c}_j^i(\delta),\ i, j = 1, 2, \ldots, n$ é o custo reduzido de x_j em $\overline{P}_i(\delta)$ na base ótima. Tem-se que $\overline{\beta}_i(\delta^{n-1}) \leq \overline{\beta}_i(0),\ i = 1, \ldots, n$.

Os autores estabeleceram um procedimento para cálculo do $\overline{\beta}(\delta^{n-1})$ ou β^*, o qual mostramos abaixo.

Algoritmo 5: Cálculo de β^*

Entrada: $P, w, c, \overline{c}$;

Saida: $\delta,\ \beta_i^*,\ i = 2, 3, \ldots, n$;

1: $\delta = \delta^{pred} = 0$;
2: **for** $k := 1$ **to** $n - 1$ **do**
3: $\quad \beta_{k+1}^* \leftarrow \text{resolve}(\overline{P_k}(\delta^{pred}))$;
4: $\quad \delta_{kj} \leftarrow -\overline{c}_j^k(\delta^{pred}),\ j = k + 1, \ldots, n$;
5: $\quad \delta^{pred} \leftarrow \delta$;
6: **end for**

Os autores propuseram um algoritmo *Branch-and-Bound* para resolver o problema de linearização $TLP(\beta^*)$ e portanto o QKP. Inicialmente foi calculado uma solução viável x_H através da heurística proposta por Billionnet e Calmels [8]. Esta solução x_H nos permite gerar uma permutação π_H dado pela ordem não crescente de x_H.

Em cada nó da árvore de enumeração, um limite superior ub é calculado de acordo com o algoritmo de geração de restrições, Algoritmo 4. Após resolução do último $\overline{P}_{\beta^*}$ temos um vetor solução $\overline{x}$, e vetor de custo reduzido $\overline{c}$. Em seguida um procedimento de fixação de variáveis utilizando variáveis de custo reduzido foi executado. Dado o limite inferior lb, obtido pela heurística proposta em [8], temos que

$$\text{se } |\overline{c}_i| > ub - lb, \text{ então fixe } x_i \text{ em } \overline{x}_i$$

Durante o procedimento de geração de retrições para o problema, muitas restrições podem ser geradas em cada nó da árvore de enumeração. Os autores proporam uma agregração, *surrogate*, das restrições geradas em cada nó, onde essa agregração é passada como restrição única para cada nó filho.

A agregação gerada em um nó problema para um dado conjunto de restrições indexadas por K, consiste da adição ao problema de uma nova restrição $t \leq \beta_K^* x$ tal que

$$\beta_K^*(j) = \sum_{i \in K} \frac{\overline{\mu}_i \beta_{\pi(i)}^*}{\sum_{k \in K} \overline{\mu}_k}, \tag{3.119}$$

onde $\overline{\mu}_i$ é o valor dual resultante da solução relaxação contínua relacionada a restrição $t \leq \beta_\pi^* x$.

A regra de *branching* adotada pelos autores foi aquela na qual a variável da solução linear é a variável mais fracionária, ou seja, a variável mais próxima de 0.5.

Capítulo 4

Algoritmos proposto para o QKP

Neste capítulo propomos algoritmos para resolver o QKP, um algoritmo *Relax-and-Cut*, dois algoritmos *Branch-and-Bound* e um algoritmo *Branch-and-Cut*.

O uso de desigualdades válidas para o QKP, com um número elevado desigualdades se fazem interessante em um algoritmo tipo *Relax-and-Cut* para o QKP, onde as mesmas são dualizadas de forma de dinâmica. Desigualdades válidas para o KP e quadratizadas também se mostram interessantes em um algoritmo *Relax-and-Cut*. O uso destas desigualdades, combinado a outras ideias geram limites válidos para o QKP, de boa qualidade e baratos computacionalmente. Estes limites parecem interessantes para serem implementados em um algoritmo *Branch-and-Bound*.

Desigualdades válidas para o KP se mostram interessantes para serem implementadas em um algoritmo de planos de cortes estilo *Branch-and-Cut* para o QKP. A quadratização de desigualdades válidas para o KP parece ser outra ideia interessante a ser implementada em um algoritmo *Branch-and-Cut* para o QKP.

4.1 *Relax-and-Cut*

Nesta seção propomos um algoritmo *Relax-and-Cut* para o QKP. Investigaremos

nesta parte do trabalho o seguinte modelo para o QKP.

$$\max \sum_{i \in N} q_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} y_{ij} \tag{4.1}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{4.2}$$

$$\sum_{i<j,\ i \in N} w_i y_{ij} + \sum_{i>j,\ i \in N} w_i y_{ji} \leq (c - w_j)\, x_j,\ j \in N \tag{4.3}$$

$$y_{ij} \leq x_i,\ i < j,\ i, j \in N, \tag{4.4}$$

$$y_{ij} \leq x_j,\ i < j,\ i, j \in N, \tag{4.5}$$

$$x_i + x_j \leq y_{ij} + 1,\ i < j,\ i, j \in N, \tag{4.6}$$

$$x_i \in \{0, 1\},\ i \in N, \tag{4.7}$$

$$y_{ij} \in \{0, 1\},\ i < j,\ i, j \in N. \tag{4.8}$$

Substituindo no Problema (4.1)-(4.8) as restrições (4.7) e (4.8) por $x_i \in [0, 1],\ i \in N$ e $y_{ij} \in [0, 1],\ i < j,\ i, j \in N$, respectivamente, obtemos o Problema Relaxado de Programação Linear (LPRP).

$$\max \sum_{i \in N} q_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} y_{ij} \tag{4.9}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{4.10}$$

$$\sum_{i<j,\ i \in N} w_i y_{ij} + \sum_{i>j,\ i \in N} w_i y_{ji} \leq (c - w_j)\, x_j,\ j \in N \tag{4.11}$$

$$y_{ij} \leq x_i,\ i < j,\ i, j \in N, \tag{4.12}$$

$$y_{ij} \leq x_j,\ i < j,\ i, j \in N, \tag{4.13}$$

$$x_i + x_j \leq y_{ij} + 1,\ i < j,\ i, j \in N, \tag{4.14}$$

$$x_i \in [0, 1],\ i \in N \tag{4.15}$$

$$y_{ij} \in [0, 1],\ i < j,\ i, j \in N. \tag{4.16}$$

Com a finalidade de resolver o LPRP, Relaxação Lagrageana [19], por exemplo, pode ser utilizada. Observe que o número de desigualdades do LPRP cresce em função de n, dizemos que para n maior que 100, esse valor é elevado. Cada uma das famílias de desigualdades (4.12), (4.13) e (4.14) contribuem com C_2^n desigualdades para a formulação do problema. Em situações como esta um tradicional algoritmo de Relaxação Lagrangeana pode ter lenta convergência e limites duais pobres. Algoritmos *Relax-and-Cut* parecem não sofrerem com estas desvantagens.

Seja (4.11) e (4.12)-(4.14) dois conjuntos de restrições tratadas como difíceis,

as quais deverão ser dualizadas a moda Lagrangeana tradicional [19] e a moda dinâmica *Relax-and-Cut* respectivamente. Dualizando as mesmas em um modo ou outro, obtemos o seguinte Subproblema Lagrangeano (LS),

$$\max \sum_{i\in N} q_i x_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} p_{ij} y_{ij} \tag{4.17}$$
$$\sum_{j\in N} \lambda_j^{MQ} \left((c - w_j)\, x_j - \sum_{i<j,\ i\in N} w_i y_{ij} - \sum_{i>j,\ i\in N} w_i y_{ji} \right)$$
$$+ \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \lambda_{ij}^{LINI}(x_i - y_{ij}) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \lambda_{ij}^{LINJ}(x_j - y_{ij})$$
$$+ \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \lambda_{ij}^{LINIJ}(y_{ij} + 1 - x_i - x_j)$$
$$\text{s.a} \sum_{i\in N} w_i x_i \leq c \tag{4.18}$$
$$x_i \in [0,1],\ i \in N \tag{4.19}$$
$$y_{ij} \in [0,1],\ i < j,\ i,j \in N. \tag{4.20}$$

onde $\lambda^{MQ}, \lambda^{LINI}, \lambda^{LINJ}$ e λ^{LINIJ} definem os multiplicadores associados as desigualdades (4.11), (4.12), (4.13) e (4.14) respectivamente. Definimos Λ como o conjunto de componentes dos multiplicadores.

LS pode alternativamente ser escrito em uma forma compacta e simples.

$$\max \sum_{i\in N} q_i(\Lambda) x_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} p_{ij}(\Lambda) y_{ij} + Const(\Lambda) \tag{4.21}$$
$$\text{s.a} \sum_{i\in N} w_i x_i \leq c \tag{4.22}$$
$$x_i \in [0,1],\ i \in N \tag{4.23}$$
$$y_{ij} \in [0,1],\ i < j,\ i,j \in N. \tag{4.24}$$

onde $Const(\Lambda)$ é a parte constante associada a Λ. Consequentemente, o Problema Dual Lagrangeano correspondente a LS e denotado por DQKP, é dado por:

$$min_{\Lambda\geq 0} = \begin{cases} \max & \sum_{i\in N} q_i(\Lambda) x_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} p_{ij}(\Lambda) y_{ij} + Const(\Lambda) \\ \text{s.a} & \sum_{i\in N} w_i x_i \leq c \\ & x_i \in [0,1],\ i \in N \\ & y_{ij} \in [0,1],\ i < j,\ i,j \in N. \end{cases}$$

Em nossa aplicação, utilizamos o SM para resolver o DQKP e seguimos a implementação sugerida em [22]. Esta, deveria ser salientada, são voltadas para o algoritmo *Non Delayed Relax-and-Cut*. Para dar uma descrição do mesmo, seja $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ solução ótima para o LS em uma corrente iteração do SM. As desigualdades (4.11) - (4.14) deverão ser definidas como ativas ou inativas. Dois tipos de desigualdades são qualificadas com ativas. O primeiro tipo corresponde as desigualdades que são dualizadas a moda Lagrangeana tradicional, no caso de nossa aplicação desigualdades (4.11). A segunda é definida por um particular conjunto das desigualdades (4.12)-(4.14), as quais são candidatas a serem dualizadas seguindo a moda *Relax-and-Cut*. As desigualdades (4.12)-(4.14) são marcadas como ativas se elas satisfazem no mínimo uma das duas condições que definiremos agora. A primeira é que elas sejam violadas por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$. A segunda é que mesmo não violadas elas possuam o corrente multiplicador de Lagrange associado a elas diferente de zero. A segunda condição implica que a desigualdade foi violada por uma solução do LS em iteração anterior do SM e foi explicitamente dualizada. Adicionalmente uma desigualdade violada permanecerá dualizada desde então, até que seu multiplicador de Lagrange se torne igual a zero.

Durante as iterações do SM denotamos por A o conjunto que define as restrições ativas e por I o conjunto que define as restrições inativas. Desigualdades pertecentes à A são utilizadas para atualização dos parâmetros do SM. Contudo, em nossa implementação do SM permitimos que as desigualdades (4.12)-(4.13) permaneçam mais tempo pertecentes no conjunto de ativas. Isto é, mesmo que as mesmas deixem de ser violadas e seus multiplicadores de Lagrange caiam a zero, permitimos que as mesmas continuem no conjunto A por mais algumas iterações do SM. Se após este número de iterações as mesma continuem não violadas pela dada solução do LS e seus multiplicadores iguais a zero, eliminamos as mesmas do conjunto A e as inserimos no conjunto I.

O pseudocódigo Algoritmo 6 mostra como definimos os conjuntos A e I. O vetor v define para cada restrição t o número de vezes que a mesma permanece em A após deixar de ser violada e seu multiplicador de Lagrange caiu a zero. Definimos o valor 2 como o número máximo de iterações nestas condições.

Algoritmo 6: Definir conjunto A e I

1: **for** $i := 1$ **to** $n-1$ **do**
2: **for** $j := i+1$ **to** n **do**
3: **if** $t_{ij}^{LINI} \in I$ **and** $\overline{y}_{ij} - \overline{x}_i > 0$ **then**
4: $A \leftarrow A \cup \{t_{ij}^{LINI}\}$; $I \leftarrow I \setminus \{t_{ij}^{LINI}\}$; $v_{ij}^{LINI} \leftarrow 0$;
5: **else if** $t_{ij}^{LINI} \in A$ **and** $\overline{y}_{ij} - \overline{x}_i \leq 0$ **and** $\lambda_{ij}^{LINI} = 0$ **then**
6: **if** $v_{ij}^{LINI} = 2$ **then**
7: $I \leftarrow I \cup \{t_{ij}^{LINI}\}$; $A \leftarrow A \setminus \{t_{ij}^{LINI}\}$; $v_{ij}^{LINI} \leftarrow 0$;
8: **else**
9: $v_{ij}^{LINI} \leftarrow v_{ij}^{LINI} + 1$;
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **for** $i := 1$ **to** $n-1$ **do**
15: **for** $j := i+1$ **to** n **do**
16: **if** $t_{ij}^{LINJ} \in I$ **and** $\overline{y}_{ij} - \overline{x}_j > 0$ **then**
17: $A \leftarrow A \cup \{t_{ij}^{LINJ}\}$; $I \leftarrow I \setminus \{t_{ij}^{LINJ}\}$; $v_{ij}^{LINJ} \leftarrow 0$;
18: **else if** $t_{ij}^{LINJ} \in A$ **and** $\overline{y}_{ij} - \overline{x}_j \leq 0$ **and** $\lambda_{ij}^{LINJ} = 0$ **then**
19: **if** $v_{ij}^{LINJ} = 2$ **then**
20: $I \leftarrow I \cup \{t_{ij}^{LINJ}\}$; $A \leftarrow A \setminus \{t_{ij}^{LINJ}\}$; $v_{ij}^{LINJ} \leftarrow 0$;
21: **else**
22: $v_{ij}^{LINJ} \leftarrow v_{ij}^{LINj} + 1$;
23: **end if**
24: **end if**
25: **end for**
26: **end for**
27: **for** $i := 1$ **to** $n-1$ **do**
28: **for** $j := i+1$ **to** n **do**
29: **if** $t_{ij}^{LINIJ} \in I$ **and** $\overline{x}_i + \overline{x}_j - \overline{y}_{ij} - 1 > 0$ **then**
30: $A \leftarrow A \cup \{t_{ij}^{LINIJ}\}$; $I \leftarrow I \setminus \{t_{ij}^{LINIJ}\}$; $v_{ij}^{LINIJ} \leftarrow 0$;
31: **else if** $t_{ij}^{LINIJ} \in A$ **and** $\overline{x}_i + \overline{x}_j - \overline{y}_{ij} - 1 \leq 0$ **and** $\lambda_{ij}^{LINIJ} = 0$ **then**
32: **if** $v_{ij}^{LINIJ} = 2$ **then**
33: $I \leftarrow I \cup \{t_{ij}^{LINIJ}\}$; $A \leftarrow A \setminus \{t_{ij}^{LINIJ}\}$; $v_{ij}^{LINIJ} \leftarrow 0$;
34: **else**
35: $v_{ij}^{LINIJ} \leftarrow v_{ij}^{LINIJ} + 1$;
36: **end if**
37: **end if**
38: **end for**
39: **end for**

Seja $\Lambda^0, \Lambda^1, \Lambda^2, \ldots$ as matrizes dos multiplicadores de Lagrange associadas as restrições (4.11)-(4.14) e geradas ao longo das iterações do SM, onde temos que $\Lambda^0 := 0$ e Λ^{k+1} sendo atualizada de acordo com Λ^k. Suponha que $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ seja a solução do LPRP. Os componentes de Λ pertencentes a A são calculados da seguinte forma:

Algoritmo 7: Cálculo dos subgradientes

1: **for** $j := 1$ **to** n **do**
2: $\quad \delta_j^{MQ} \leftarrow \left((c - w_j)\,\overline{x}_j - \sum_{i<j,\ i \in N} w_i \overline{y}_{ij} - \sum_{j<i,\ i \in N} w_i \overline{y}_{ji} \right);$
3: **end for**
4: **for** $i := 1$ **to** $n - 1$ **do**
5: $\quad$ **for** $j := i + 1$ **to** n **do**
6: $\quad\quad$ **if** $t_{ij}^{LINI} \in A$ **then**
7: $\quad\quad\quad \delta_{ij}^{LINI} \leftarrow \overline{x}_i - \overline{y}_{ij};$
8: $\quad\quad$ **else**
9: $\quad\quad\quad \delta_{ij}^{LINI} \leftarrow 0;$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: **end for**
13: **for** $i := 1$ **to** $n - 1$ **do**
14: $\quad$ **for** $j := i + 1$ **to** n **do**
15: $\quad\quad$ **if** $t_{ij}^{LINJ} \in A$ **then**
16: $\quad\quad\quad \delta_{ij}^{LINJ} \leftarrow \overline{x}_j - \overline{y}_{ij};$
17: $\quad\quad$ **else**
18: $\quad\quad\quad \delta_{ij}^{LINJ} \leftarrow 0;$
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**
22: **for** $i := 1$ **to** $n - 1$ **do**
23: $\quad$ **for** $j := i + 1$ **to** n **do**
24: $\quad\quad$ **if** $t_{ij}^{LINIJ} \in A$ **then**
25: $\quad\quad\quad \delta_{ij}^{LINIJ} \leftarrow \overline{y}_{ij} + 1 - \overline{x}_i - \overline{x}_j;$
26: $\quad\quad$ **else**
27: $\quad\quad\quad \delta_{ij}^{linij} \leftarrow 0;$
28: $\quad\quad$ **end if**
29: $\quad$ **end for**
30: **end for**

Em seguida o cálculo do quadrado do subgrandiente, *qgrad*, deve ser executado. Seguimos para este procedimento [22], onde subgradientes associados as desigualda-

des pertecentes a I não são considerados.

Algoritmo 8: Cálculo do quadrado dos subgradientes

1: $qgrad \leftarrow 0$;
2: **for** $j := 1$ **to** n **do**
3: $\quad qgrad \leftarrow qgrad + \left(\delta_j^{MQ}\right)^2$;
4: **end for**
5: **for** $i := 1$ **to** $n-1$ **do**
6: $\quad$ **for** $j := i+1$ **to** n **do**
7: $\quad\quad$ **if** $t_{ij}^{LINI} \in A$ **then**
8: $\quad\quad\quad qgrad \leftarrow qgrad + \left(\delta_{ij}^{LINI}\right)^2$;
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end for**
12: **for** $i := 1$ **to** $n-1$ **do**
13: $\quad$ **for** $j := i+1$ **to** n **do**
14: $\quad\quad$ **if** $t_{ij}^{LINJ} \in A$ **then**
15: $\quad\quad\quad qgrad \leftarrow qgrad + \left(\delta_{ij}^{LINJ}\right)^2$;
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end for**
19: **for** $i := 1$ **to** $n-1$ **do**
20: $\quad$ **for** $j := i+1$ **to** n **do**
21: $\quad\quad$ **if** $t_{ij}^{LINIJ} \in A$ **then**
22: $\quad\quad\quad qgrad \leftarrow qgrad + \left(\delta_{ij}^{LINIJ}\right)^2$;
23: $\quad\quad$ **end if**
24: $\quad$ **end for**
25: **end for**

O tamanho do passo, θ^k, é o próximo a ser cálculado, onde k é a iteração do SM. Para o cálculo do mesmo utilizamos um fator, μ, definido como 2 no início do SM e reduzido por 0.6μ a cada 90 iterações sem melhora do limite superior, $\overline{ub}$. Utilizamos ainda para cálculo de θ^k o limite inferior, $\underline{lb}$, obtido através de uma heurística. Temos que o limite superior do QKP em uma determinada iteração do SM pode está muito próximo do limite inferior, assim cálculamos o tamanho do passo, θ^k, da seguinte forma:

Algoritmo 9: Cálculo do tamanho do passo

if $\dfrac{(\overline{ub} - \underline{lb})}{\underline{lb}} < 0.01$ **then**
  $ubp \leftarrow \overline{ub} \times 1.01$;
else
  $ubp \leftarrow \overline{ub}$;
end if
$\theta^k \leftarrow \mu \times \dfrac{(ubp - \underline{lb})}{qgrad}$;

A atualização dos multiplicadores de Lagrange em uma dada iteração k é dada da seguinte forma,

Algoritmo 10: Atualização dos multiplicadores de Lagrange

for $j := 1$ **to** n **do**
  $\lambda_j^{MQ,k+1} \leftarrow \max\left\{\lambda_j^{MQ,k} - \theta^k \delta_j^{MQ,k}; 0\right\}$;
end for
for $i := 1$ **to** $n-1$ **do**
  for $j := i+1$ **to** n **do**
    $\lambda_{ij}^{LINI,k+1} \leftarrow \max\left\{\lambda_{ij}^{LINI,k} - \theta^k \delta_{ij}^{LINI,k}; 0\right\}$;
  end for
end for
for $i := 1$ **to** $n-1$ **do**
  for $j := i+1$ **to** n **do**
    $\lambda_{ij}^{LINJ,k+1} \leftarrow \max\left\{\lambda_{ij}^{LINJ,k} - \theta^k \delta_{ij}^{LINJ,k}; 0\right\}$;
  end for
end for
for $i := 1$ **to** $n-1$ **do**
  for $j := i+1$ **to** n **do**
    $\lambda_{ij}^{LINIJ,k+1} \leftarrow \max\left\{\lambda_{ij}^{LINIJ,k} - \theta^k \delta_{ij}^{LINIJ,k}; 0\right\}$;
  end for
end for

Palmeira [7] observou que as desigualdades (4.11) tendem a ter subgradientes associados a elas com valor tipicamente maiores que os subgradientes associados as outras desigualdades dualizadas. Palmeira também observou que se um subconjunto de subgradientes possuem valores muitos maiores em relação aos demais, a influência dos subgradientes com valores pequenos será insignificante no cálculo do tamanho do passo, e o efeito das restrições dualizadas não será sentido em um algoritmo *Relax-and-Cut*. A não multiplicação dos subgradientes associados as restrições (4.11) por um fator pode ser suficiente para uma diferença de 30% no limite superior fornecido pelo algoritmo *Relax-and-Cut*. Observamos em nossos experimentos computacionais mesmo comportamento, assim cálculamos um fator de amortecimento composto pela

soma dos $w_i,\ i \in N$ e a capacidade da mochila, c.

Algoritmo 11: Cálculo do fator de amortecimento para restrição (4.3)

$fator \leftarrow c$;
for $i := 1$ **to** n **do**
  $fator \leftarrow fator + w_i$;
end for

Este fator deve ser multiplicado aos subgradientes associados a restrição (4.11).

Algoritmo 12: Aplicação do fator de amortecimento

for $j := 1$ **to** n **do**
  $\delta_j^{MQ} \leftarrow \delta_j^{MQ} \times fator$;
end for

A atualização dos benefícios $q_i(\Lambda),\ i \in N$ e $p_{ij}(\Lambda),\ i < j,\ i, j \in N$, a serem utilizados na resolução do Problema Lagrangeano é realizada do seguinte modo:

Algoritmo 13: Atualização do benefícios Lagrangeano

for $i := 1$ **to** n **do**
  $q_i(\Lambda) \leftarrow q_i + \lambda_i^{MQ} \times (c - w_i)$;
  for $j := i + 1$ **to** n **do**
    $p_{ij}(\Lambda) \leftarrow p_{ij} - \lambda_j^{MQ} \times w_j$;
  end for
end for
for $i := 1$ **to** $n - 1$ **do**
  for $j := i + 1$ **to** n **do**
    $q_i(\Lambda) \leftarrow q_i(\Lambda) - \lambda_{ij}^{LINI}$;
    $p_{ij}(\Lambda) \leftarrow p_{ij}(\Lambda) + \lambda_{ij}^{LINI}$;
  end for
end for
for $i := 1$ **to** $n - 1$ **do**
  for $j := i + 1$ **to** n **do**
    $q_j(\Lambda) \leftarrow q_j(\Lambda) - \lambda_{ij}^{LINJ}$;
    $p_{ij}(\Lambda) \leftarrow p_{ij}(\Lambda) + \lambda_{ij}^{LINJ}$;
  end for
end for
for $i := 1$ **to** $n - 1$ **do**
  for $j := i + 1$ **to** n **do**
    $q_i(\Lambda) \leftarrow q_i(\Lambda) + \lambda_{ij}^{LINIJ}$;
    $q_j(\Lambda) \leftarrow q_j(\Lambda) + \lambda_{ij}^{LINIJ}$;
    $p_{ij}(\Lambda) \leftarrow p_{ij}(\Lambda) - \lambda_{ij}^{LINIJ}$;
  end for
end for

Para resolver LS utilizamos o algoritmo de Dantzig [35], o qual definimos abaixo. Considere o seguinte Problema da Mochila (KP),

$$\begin{aligned} \max & \sum_{i \in N} p_i x_i \\ \text{s.a} & \sum_{i \in N} w_i x_i \leq c \\ & x_i \in [0,1],\ i \in N. \end{aligned}$$

Algoritmo 4.1.1 (Dantzig)**.** *Ordenando em ordem não-crescente os itens de acordo com a relação* $\frac{p_i}{w_i}$, $i \in N$, *inserimos na mochila os itens* $i \in K$, *tal que* $K \subseteq N$ *e* $\sum_{i \in K} w_i \leq c$. *Seja* $h \in N$ *o item tal que* $\sum_{i \in K} w_i + w_h > c$, *definimos* x_h *como item de parada. A solução do KP é dada por* $x_i = 1$ *para* $i \in K$, $x_h = \frac{\left(c - \sum_{i \in K} w_i\right)}{w_h}$, *onde* $h = |K| + 1$ *e* $x_i = 0,\ i \in N \setminus K \cup \{h\}$.

A implementação do Algoritmo (4.1.1) é executado em $O(n \log n)$ tempo de acordo com [35]. O gargalo está na ordenação do itens. Um KP contínuo, de fato, pode ser resolvido em tempo $O(n)$ através de uma técnica que utiliza mediana desenvolvida em [36].

Palmeira [7] obteve o limite superior para o Problema Lagrageano associado ao QKP em dois momentos, em um primeiro momento resolvendo um KP associado a parte linear, $\mathbf{x}$, onde foi utilizado o algoritmo de Dantzig, em um segundo momento relacionado a parte não-linear, $\mathbf{y}$, de acordo com o valor de $p_{ij}(\Lambda)$, se $p_{ij}(\Lambda) \geq 0$ então $y_{ij} = 1$ e $p_{ij}(\Lambda)$ é acumulado ao limite superior, caso contrário $y_{ij} = 0$. Lembrado que é acumulado ao limite superior a parte constante associada aos multiplicadores de Lagrange.

Observe que somando as restrições (3.4) e (3.5), obtemos a seguinte desigualdade,

$$y_{ij} \leq \frac{x_i + x_j}{2},\ i < j,\ i, j \in N. \tag{4.25}$$

Temos que $p_{ij}(\Lambda) \in \mathbb{R}$ e $\frac{x_i + x_j}{2} \geq 0$, segue que $p_{ij}(\Lambda)\frac{x_i + x_j}{2}$ pode assumir valores positivos, negativos ou nulos. Nosso interesse é que $p_{ij}(\Lambda)\frac{x_i + x_j}{2} \geq 0$, e assim obtemos um limite superior válido para o QKP. Buscando isto, reformulamos o Problema (4.21)-(4.24), se $p_{ij}(\Lambda) > 0$ definimos y_{ij} como $\frac{x_i + x_j}{2}$ e acumulamos $\frac{p_{ij}(\Lambda)}{2}$ a $q_i(\Lambda)$ e $\frac{p_{ij}(\Lambda)}{2}$ a $q_j(\Lambda)$, se $p_{ij}(\Lambda) \leq 0$, sumimos com y_{ij} da formulação do problema e definimos $y_{ij} = 0$.

Algoritmo 14: Acumulando benefícios cruzados positivos na diagonal

for $i := 1$ **to** n **do**
  $\overline{q}_i \leftarrow q_i(\Lambda)$;
  for $j := i + 1$ **to** n **do**
    if $p_{ij}(\Lambda) > 0$ **then**
      $\overline{q}_i \leftarrow \overline{q}_i + \dfrac{p_{ij}(\Lambda)}{2}$;
      $\overline{q}_j \leftarrow \overline{q}_j + \dfrac{p_{ij}(\Lambda)}{2}$;
    end if
  end for
end for

Estamos agora com o seguinte problema,

$$\max \sum_{i \in N} \overline{q}_i x_i \tag{4.26}$$

$$\text{s.a} \sum_{i \in N} w_i x_i \leq c \tag{4.27}$$

$$x_i \in [0,1],\ i \in N. \tag{4.28}$$

o qual definimos como limitante KP.

Para resolver o Problema (4.26)-(4.28) implementamos o algoritmo de Dantzig [35], com devidas adaptações. Observe que $\overline{q}_i \in \mathbb{R}$.

Algoritmo 4.1.2 (Dantzig modificado)**.** *Ordenando em ordem não-crescente os itens de acordo com a relação* $\dfrac{\overline{q}_i}{w_i}$, $i \in N$, *inserimos na mochila os itens* $i \in K$, *tal que* $K \subseteq N$, $\sum_{i \in K} w_i \leq c$ *e* $\overline{q}_i \geq 0$, $i \in K$. *Seja* $h \in N$ *o item tal que* $\sum_{i \in K} w_i + w_h > c$ *e* $\overline{q}_h \geq 0$, *definimos* x_h *como item de parada. Seja* $j \in N$ *o item tal que* $\overline{q}_j < 0$ *abortamos o procedimento. A solução do Problema (4.26)-(4.28) é dada por* $x_i = 1$ *para* $i \in K$, $x_h = \dfrac{\left(c - \sum_{i \in K} w_i\right)}{w_h}$ *onde* $h = |K| + 1$ *e* $x_i = 0$ *para* $i \in N \setminus K \cup \{h\}$.

Seja $\overline{\mathbf{x}}$ solução e ub o valor solução do Problema (4.26)-(4.28) em uma dada iteração do SM, para obtermos a solução associada a $\overline{\mathbf{y}}$ executamos o seguinte pro-

cedimento,

Algoritmo 15: Obtendo solução $\overline{\mathbf{y}}$

```
for i := 1 to n − 1 do
  for j := i + 1 to n do
    if (p_ij(Λ) > 0) then
      ȳ_ij ← (x̄_i + x̄_j)/2;
    else
      ȳ_ij ← 0
    end if
  end for
end for
```

Somamos a ub, limite superior, a parte constante referente as restrições (4.14),

Algoritmo 16: Somando parte constante ao ub

```
for i := 1 to n − 1 do
  for j := i + 1 to n do
    ub ← ub + λ_ij^{LINIJ};
  end for
end for
```

Com isto obtemos o limite superior, ub, para o QKP em uma dada iteração do SM.

4.1.1 Soluções viáveis para o QKP

Claramente, uma solução ótima para o Problema (4.26)-(4.28) inteiro implica em uma solução viável para KP. Além disso, esta solução é construida sobre informações duais fornecidas pelo nosso algoritmo *Relax-and-Cut*. Essa iteração entre informações primais e duais, parece ser uma das melhores características para uma heurística baseada no Lagrangeano.

De acordo com Seção 3.6, limites superiores válidos para o QKP também são geradas em [5] através de uma solução para um KP. Como tal, embora nosso problema limitante KP e o definido em [5] sejam construidos sobre diferentes informações duais, soluções viáveis para QKP são obtidas de forma similar nos dois algoritmos. Tal solução pode ser melhorada aplicando-se um procedimento de busca local sobre esta solução. Para o nosso algoritmo *Relax-and-Cut*, semelhante ao utilizado no algoritmo em [5], o procedimento de busca local sugerido em [8] é utilizado. Tal procedimento foi descrito em detalhes na Seção 3.7. Em nosso algoritmo a solução viável passada para a busca local é a solução inteira fornecida pelo Algoritmo de Dantzig modificado.

Foi observado em nossa implementação do *Relax-and-Cut*, ao longo do SM, que soluções viáveis geradas podem ser repetidas. Com isso, nos parece mais eficiente

aplicar o procedimento de busca local apenas em soluções que não tenham ocorrido em iterações anteriores do SM. Observamos em nossos experimentos computacionais que as melhores soluções viáveis são obtidas nas primeiras $1500 + n$ iterações do SM, desta forma aplicamos em nosso algoritmo o procedimento da busca local nas primeiras $1500 + n$ iterações do SM geradas pelo limitante KP inteiro tal que esta solução não se repete, após as $1500 + n$ iterações aplicamos busca local apenas em caso de melhora no limite superior obtido pelo LS.

Em nossa proposta de algoritmo o número máximo de iterações do SM em um determinado ciclo foi definido em $3000 + n$, com adicionais critérios de paradas sendo imposto. Eles são especificamente: μ reduzindo-se a um valor inferior a 10^{-6}, a diferença entre o melhor limite superior e o melhor limite inferior para o QKP reduzindo-se a um valor inferior a 1, e $qgrad$, a soma do quadrado dos subgradientes, reduzindo-se a um valor inferior a 10^{-6}.

Abaixo pseudo código do SM,

Entrada: n, P, w, c;
Saida: $\overline{x}, \overline{ub}, \underline{lb}$;

1: $step \leftarrow 0$;
2: $\overline{ub} \leftarrow \infty$;
3: $ub \leftarrow 0$;
4: $\underline{lb} \leftarrow 0$;
5: $\mu \leftarrow 2.0$;
6: $count \leftarrow 0$;
7: Algoritmo 11; /* *cálculo do fator moderador* */
8: $\underline{lb} \leftarrow heuristica()$; /* *heurística Billionnet e Calmels* */
9: **for** it:=0 **to** 3000+n **do**
10: Algoritmo 13; /* *atualiza benefícios modificados* */
11: Algoritmo 14; /* *acúmulo dos benefícios cruzados positivos*/
12: Algoritmo 4.1.1 e 16; /* *resolve problema Lagrangeano* */
13: Algoritmo 15; /* *cálculo de y* */;
14: Algoritmo 6; /* *atualiza conjuntos A e I* */
15: Algoritmo 7 e 12; /* *cálculo dos subgradientes associados a A* */
16: $\underline{x} \leftarrow (int)x$;
17: **if** $(ub < \overline{ub})$ **then**
18: $\overline{ub} \leftarrow ub$; $\overline{x} \leftarrow x$;
19: $count \leftarrow 0$;
20: **if** $it > 1500 + n$ **then**
21: $\underline{lb} \leftarrow buscalocal(\underline{x})$; /* *cálculo do limite inferior* */
22: **end if**
23: **else**

24: $count \leftarrow count + 1$;
25: **end if**
26: **if** $it \leq 1500 + n$ **and** *novo* $\underline{x}$ **then**
27: $\underline{lb} \leftarrow buscalocal(\underline{x})$; /* *cálculo do limite inferior* */
28: **end if**
29: **if** $(count = 90)$ **then**
30: $\mu \leftarrow \mu 0.6$;
31: $count \leftarrow 0$;
32: **end if**
33: **if** $\mu < 10^{-6}$ **then**
34: $break$;
35: **end if**
36: **if** $\overline{ub} - \underline{lb} < 1$ **then**
37: $break$;
38: **end if**
39: **if** $qgrad < 10^{-6}$ **then**
40: $break$;
41: **end if**
42: Algoritmo 8; /* *cálculo do quadrado dos subgradientes* */
43: Algoritmo 9; /* *cálculo do tamanho do passo* */
44: Algoritmo 10; /* *atualiza multiplicadores de Lagrange* */
45: **end for**

Palmeira em [7] propós um algoritmo *Relax-and-Cut* para resolver o QKP. Ele reformulou o QKP, Problema 1.1, acrecentando ao mesmo as desigualdades (3.3)-(3.10) obtendo o modelo (3.42)-(3.53). Os autores utilizaram a relaxação contínua para resolve o problema. O SM foi utilizado para resolver o Problema Dual Lagrangeano. A desigualdade (3.3) foi dualizada ao estilo tradicional, já as desigualdades (3.4)-(3.10) foram dualizadas em um estilo *Relax-and-Cut*. Foi definido um número limite igual a 5000 iterações para o SM, além dos critérios de paradas que propomos em nosso algoritmo. O cálculo do limite superior depende dos termos, para o termo linear, $x_i,\ i \in N$, Palmeira utilizou o algoritmo de Dantzig, já para o termo não-linear, $y_{ij},\ i < j,\ i, j \in N$, quando o benefício Lagrangeano modificado $p_{ij}(\Lambda)$ é não-negativo, y_{ij} é colocado na solução, ou seja, $y_{ij} = 1$, caso contrário $y_{ij} = 0$.

4.1.2 Fixação de variáveis

Implementamos em nossa proposta de algoritmo *Relax-and-Cut* técnicas de fixação e redução de variáveis que foram propostas por Caprara, Pisinger e Toth em [5]. As mesmas foram citadas na Seção 3.8.

Finalizado um ciclo do SM aplicamos um procedimento de fixação de variáveis seguido do procedimento de redução do tamanho do problema. Fixando no mínimo uma variável é aplicado o procedimento de redução do problema e então um novo ciclo do SM é executado. Caso nenhuma variável seja fixada, damos por encerrado os ciclos do SM.

4.1.3 *Branch-and-Bound*

Finalizado os ciclos do SM, combinados com os procedimentos fixação variáveis e redução do tamanho do problema no algoritmo *Relax-and-Cut* seguimos para o *Branch-and-Bound*.

Como estratégia de busca implementamos uma busca em profundidade no algoritmo *Branch-and-Bound*.

Seja $\overline{N} \subset N$ o conjunto que representa as variáveis não fixadas durante o processo de fixação de variáveis, onde $\overline{N} = \{1, 2, \ldots, \overline{n}\}$.

A seguir definimos nossa ordem de busca proposta. Seja $ordb \in \mathbb{R}^{\overline{n}}$ determinado por:

Algoritmo 17: Definindo ordem de busca no *Branch-and-Bound*

1: **for** $i := 1$ **to** $\overline{n}$ **do**
2: $\quad ordb_i \leftarrow q_i$;
3: **end for**
4: **for** $i := 1$ **to** $\overline{n}$ **do**
5: $\quad$ **for** $j := i + 1$ **to** $\overline{n}$ **do**
6: $\quad\quad ordb_i \leftarrow ordb_i + \frac{p_{ij}}{2}$;
7: $\quad\quad ordb_j \leftarrow ordb_j + \frac{p_{ij}}{2}$;
8: $\quad$ **end for**
9: **end for**
10: **for** $i := 1$ **to** $\overline{n}$ **do**
11: $\quad ordb_i \leftarrow \frac{ordb_i}{w_i}$;
12: **end for**

O vetor $ordb$ é ordenado de forma não-crescente e define a ordem de busca a ser seguida na árvore de enumeração do *Branch-and-Bound*. Observe que estamos acumulando os q_i em $ordb_i$ e metade dos benefícios originais p_{ij}, os quais são não negativos, em $ordb_i$ e a outra metade em $ordb_j$, em seguida divimos este pelo seu correspodente w_i, o coeficiente gerado define a ordem a ser seguida na árvore de enumeração do *Branch-and-Bound*.

Definido a ordem de *branching* a ser percorrida na árvore de enumeração do *Branch-and-Bound*, implementamos melhorias ao *Branch-and-Bound* com objetivo de acelerar a busca ao longo da árvore de enumeração, essas ideias foram propostas

por Caprara, Pisinger e Toth [5], de acordo com (3.99), e citadas na Subseção 3.9.1.

Em nossos experimentos computacionais tal ideia reduz o tempo de execução na árvore de enumeração do *Branch-and-Bound*.

Abaixo o pseudocódigo do algoritmo *Branch-and-Bound* implementado, onde $\overline{P}$ é a matriz de benefícios reduzida, $fixw$ é a parte acumulada referente aos w_i dos itens fixados em 1 e $fixp$ é a parte acumulada referente aos benefícios dos itens fixados em 1.

Entrada: $\overline{N}, \overline{P}, \underline{lb}, fixp, fixw$;

Saida: z^*, x^*;

1: $lb \leftarrow \underline{lb}$;
2: $ps \leftarrow 0$;
3: $tps \leftarrow 0$;
4: $ws \leftarrow 0$;
5: $tws \leftarrow 0$;
6: $l \leftarrow 0$;
7: $push(pl, x_l = 0, ps, ws)$;
8: **if** $ws + w_l \leq c$ **then**
9: $\quad ps \leftarrow ps + q_l$;
10: $\quad ws \leftarrow ws + w_l$;
11: $\quad$ **if** $ps + fixp > lb$ **then**
12: $\quad\quad tps \leftarrow ps$;
13: $\quad\quad tws \leftarrow ws$;
14: $\quad\quad lb \leftarrow fixp + ps$;
15: $\quad$ **end if**
16: $\quad push(pl, x_l = 1, ps, ws)$;
17: **end if**
18: **while** $pl \neq empty$ **do**
19: $\quad no \leftarrow pop(pl, l, ps, ws)$;
20: $\quad$ **if** $x_l = 1$ **then**
21: $\quad\quad$ **for** $i := l + 1$ **to** n **do**
22: $\quad\quad\quad q_i \leftarrow q_i + p_{li}$;
23: $\quad\quad$ **end for**
24: $\quad$ **end if**
25: $\quad u \leftarrow solve(no)$;
26: $\quad$ **if** $u > \underline{lb}$ **then**
27: $\quad\quad l \leftarrow l + 1$;
28: $\quad\quad$ **if** $ws + \underline{w}_l \leq c$ **then**
29: $\quad\quad\quad push(pl, x_l = 0, ps, ws)$;
30: $\quad\quad\quad$ **if** $ws + w_l \leq c$ **then**

```
31:             ps ← ps + q_l;
32:             ws ← ws + w_l;
33:             if ps + fixp > lb then
34:               tps ← ps;
35:               tws ← ws;
36:               lb ← fixp + ps;
37:             end if
38:             push(pl, x_l = 1, ps, ws);
39:           else
40:             poda por inviabilidade;
41:           end if
42:         end if
43:       end if
44: end while
45: z* ← lb;
46: x* ← x;
```

Para obtermos um limite superior ao longo da árvore de *Branch-and-Bound* associado a cada nó problema a ser resolvido, utilizamos o algoritmo *Relax-and-Cut* definido na Seção 4.1. Nesse o número de iterações do SM foi definido em $1000 + n$ iterações, e os outros paramêtros e critérios de paradas são os mesmos definidos na Seção 4.1.

4.2 *Branch-and-Cut*

O algoritmo *Branch-and-Cut* que implementamos para o QKP se baseia na separação de desigualdades válidas para o KP. Nossa implementação se utilizou do resolvedor de Programação Linear Inteira Mista, *Xpress Optimizer*. A escolha da variável a ser ramificada na árvore de enumeração foi determinada automaticamente pelo *Xpress Optimizer*, e para percorrer a árvore de enumeração utilizamos a busca em profundidade. Desabilitamos o preprocessamento, estratégias de cortes e estratégias de heurísticas do *Xpress Optimizer*. No mais, todas as funções de gerenciamento da árvore de enumeração foram deixados a cargo do resolvedor.

Em nossa implementação, utilizamos a seguinte reformulação linear do QKP:

$$\max \sum_{i \in N} q_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} y_{ij} \tag{4.29}$$

$$\text{s.a.} \sum_{i \in N} w_i x_i \leq c \tag{4.30}$$

$$\sum_{i<j,\ i \in N} w_i y_{ij} + \sum_{i>j,\ i \in N} w_i y_{ji} \leq (c - w_j)\, x_j,\ j \in N \tag{4.31}$$

$$y_{ij} \leq x_i,\ i < j,\ i, j \in N, \tag{4.32}$$

$$y_{ij} \leq x_j,\ i < j,\ i, j \in N, \tag{4.33}$$

$$x_i + x_j \leq 1 + y_{ij},\ i < j,\ i, j \in N, \tag{4.34}$$

$$y_{ij} \in \{0, 1\},\ i < j,\ i, j \in N, \tag{4.35}$$

$$x_i \in \{0, 1\},\ i \in N. \tag{4.36}$$

Especificamente, trabalhamos sobre a relaxação contínua de (4.29)-(4.36). Neste caso, substituimos as restrições $x_i, y_{ij} \in \{0, 1\}, i < j,\ i, j \in N$, pelas desigualdades $x_i \in [0, 1],\ i \in N$ e $y_{ij} \in [0, 1], i < j,\ i, j \in N$.

$$\max \sum_{i \in N} q_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} y_{ij} \tag{4.37}$$

$$\text{s.a.} \sum_{i \in N} w_i x_i \leq c \tag{4.38}$$

$$\sum_{i<j,\ i \in N} w_i y_{ij} + \sum_{j<i,\ i \in N} w_i y_{ji} \leq (c - w_j)\, x_j,\ j \in N \tag{4.39}$$

$$y_{ij} \leq x_i,\ i < j,\ i, j \in N, \tag{4.40}$$

$$y_{ij} \leq x_j,\ i < j,\ i, j \in N, \tag{4.41}$$

$$x_i + x_j \leq 1 + y_{ij},\ i < j,\ i, j \in N, \tag{4.42}$$

$$y_{ij} \in [0, 1],\ i < j,\ i, j \in N, \tag{4.43}$$

$$x_i \in [0, 1],\ i \in N. \tag{4.44}$$

Além das desigualdades que utilizamos para reforçar a relaxação contínua de (4.37)-(4.44) e, por conseguinte, as relaxações contínuas dos problemas definidos nos demais nós de nossa árvore de enumeração, poderíamos também ter utilizado outras famílias de desigualdades válidas. No entanto, nossos experimentos computacionais indicaram que o preço computacional que teríamos que pagar para fazê-lo, não se justificaria. Optamos então por reforçar nossas formulações, em cada nó da árvore de enumeração, utilizando apenas as desigualdades (3.7), as desigualdades de *cover* (3.13), as desigualdades *extended* (3.14) e as desigualdades de *lifting* (3.24).

Desigualdades violadas pertencentes a tais famílias são inseridas ao problema apenas quando violadas pela solução corrente.

Como visto no Capítulo 3, desigualdades de *cover* violadas são identificadas através da resolução de um KP auxiliar. Podemos resolver este KP utilizando, dentre outras alternativas, Programação Dinâmica [21] ou o próprio resolvedor *Xpress Optimizer* [37], por exemplo. O procedimento utilizado para identificar desigualdades (3.13), (3.14) e (3.24) violadas é aquele descrito no Capítulo 3. De acordo com nossos testes computacionais, utilizar o resolvedor *Xpress Optimizer* [37] para resolver tais problemas de separação se mostrou uma alternativa satisfatória e é a que aqui empregamos.

No nó raiz da árvore de enumeração, tentamos indentificar variáveis que devem ser fixadas em zero na solução ótima do QKP. Suponha $(\overline{x}, \overline{y})$ solução do linear QKP (4.37)-(4.44), ub valor solução do LP e lb uma limite inferior obtido através de alguma heurística, em nosso algoritmo implementamos a heurística proposta por Billionnet e Calmels em [8] citada no Capítulo 3. Associado a solução $(\overline{x}, \overline{y})$ temos o custo reduzido (cx, cy). Pela teoria de dualidade de Programação Linear [38], uma variável x_i deve ser fixada em zero se $ub + cx_i < lb$ ocorre, de modo semelhante podemos fixar y_{ij} em zero se $ub + cy_{ij} < lb$ ocorre.

4.2.1 Um exemplo numérico

Considere a seguinte instância para o QKP 0-1.

$$\begin{aligned}
\max\ & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + && (4.45)\\
& 14y_{12} + 14y_{13} + 49y_{14} + 37y_{15} + 33y_{16} + 3y_{17} +\\
& 13y_{23} + 36y_{24} + 6y_{25} + 2y_{26} + y_{27} +\\
& 26y_{34} + 21y_{35} + 11y_{36} + 2y_{37} +\\
& 12y_{45} + 46y_{46} + 10y_{47} +\\
& 24y_{56} + 46y_{57} +\\
& 19y_{67}\\
\text{s.a. } & 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19 && (4.46)\\
& x_i \in \{0, 1\}\,,\ i \in \{1, \ldots, 7\}\,. && (4.47)
\end{aligned}$$

Utilizando o resolvedor *Xpress Optimizer*, obtemos a seguinte solução para Problema de Relaxação Contínua associado ao exemplo (4.45)-(4.47):

$$\overline{\mathbf{x}} = \left[\begin{array}{ccccccc} 0,00 & 0,12 & 0,58 & 0,92 & 0,92 & 0,92 & 0,92 \end{array}\right] \text{e}$$

$$\overline{\mathbf{y}} = \begin{bmatrix} 0,00 & 0,00 & 0,00 & 0,00 & 0,00 & 0,00 \\ & 0,00 & 0,12 & 0,12 & 0,06 & 0,12 \\ & & 0,50 & 0,50 & 0,50 & 0,50 \\ & & & 0,92 & 0,92 & 0,92 \\ & & & & 0,92 & 0,92 \\ & & & & & 0,92 \end{bmatrix},$$

para uma função objetivo de valor igual a $363,46$.

Resolvendo o Problema de Separação associado às desigualdade de *cover*, obtemos a solução

$$z = \begin{bmatrix} 0,00 & 0,00 & 1,00 & 1,00 & 1,00 & 1,00 & 0,00 \end{bmatrix},$$

com valor ótimo igual a $0,65$. Dessa maneira, temos que $C = \{3,4,5,6\}$ e

$$x_3 + x_4 + x_5 + x_6 \leq 3$$

é a desigualdade de *cover* encontrada.

Esta, de acordo com (3.20) e o Teorema (3.4.1), é viola por $\overline{\mathbf{x}}$.

Quadratizando a desigualdade de *cover* obtida acima, obtemos as seguintes desigualdades:

$$y_{13} + y_{14} + y_{15} + y_{16} \leq 3x_1 \tag{4.48}$$
$$y_{23} + y_{24} + y_{25} + y_{26} \leq 3x_2 \tag{4.49}$$
$$x_3 + y_{34} + y_{35} + y_{36} \leq 3x_3 \tag{4.50}$$
$$y_{34} + x_4 + y_{45} + y_{46} \leq x_4 \tag{4.51}$$
$$y_{35} + y_{45} + x_5 + y_{56} \leq x_5 \tag{4.52}$$
$$y_{36} + y_{46} + y_{56} + x_6 \leq 3x_6 \tag{4.53}$$
$$y_{37} + y_{47} + y_{57} + y_{67} \leq 3x_7. \tag{4.54}$$

Em particular, para a solução $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, as desigualdades (4.48) e (4.49) são não-violadas, enquanto (4.50), (4.51), (4.52), (4.53) e (4.54) são violadas.

Observe que, dada uma desigualdade de *cover* violada, é possível obter uma desigualdade de *cover* quadratizada não-violada onde o termo referente a quadratização está definido no intervalo $(0,1)$ (na verdade, aquele termo pode ser definido no intervalo $(0,1]$). Para tanto, observe que $y_{ij} \leq x_i$ e $y_{ij} \leq x_j$ se aplicam e, dessa forma, $0 < x_j \leq 1$ e $y_{ij} \neq x_i x_j$ resultam. Isto, por sua vez, implica em um decréscimo do lado esquerdo da desigualdade de *cover* quadratizada. Já para o caso onde $x_i = 0$, temos que a desigualdade quadratizada obtida será sempre não-violada.

Dada uma desigualdade de *cover* não-violada, seria possível obter uma desigualdade de *cover* quadratizada, a ela associada, e que seja violada? Note que, nesse caso, a desigualdade de *cover* quadratizada seria obtida pela multiplicação da desigualdade de *cover* por um termo pertencente ao conjunto que a define. A resposta a pergunta aqui formulada será dada através da construção de um exemplo.

Suponha que temos a seguinte solução para relaxação contínua do QKP previamente definido:

$$\overline{\mathbf{x}} = \left[\begin{array}{cccccc} 0,00 & 0,50 & 0,00 & 0,50 & 0,00 & 0,50 \end{array} \right].$$

Considere uma *cover* definida pelo conjunto $C = \{2, 4, 6\}$ e observe que a mesma é não-violada por $\overline{\mathbf{x}}$. Quadratizando a desigualdade de *cover* correspondente a C, para $j \notin C$, obtemos

$$y_{12} + y_{14} + y_{16} \leq 2x_1 \tag{4.55}$$

$$y_{23} + y_{34} + y_{36} \leq 2x_3 \tag{4.56}$$

$$y_{25} + y_{45} + y_{56} \leq 2x_5. \tag{4.57}$$

Estas, são não-violadas, pois o máximo valor que y_{ij} podem assumir é 0, de acordo com as desigualdades $y_{ij} \leq x_i,\ i < j,\ i, j \in N$ e $y_{ij} \leq x_j,\ i < j,\ i, j \in N$. Efetuando o mesmo procedimento para $j \in C$, obtemos

$$x_2 + y_{24} + y_{26} \leq 2x_2 \tag{4.58}$$

$$y_{24} + x_4 + y_{46} \leq 2x_4 \tag{4.59}$$

$$y_{26} + y_{46} + x_6 \leq 2x_6. \tag{4.60}$$

Observe então que temos $y_{24} \leq x_2,\ y_{24} \leq x_4,\ y_{26} \leq x_2,\ y_{26} \leq x_6,\ x_2 + x_4 \leq 1 + y_{24}$ e $x_2 + x_6 \leq 1 + y_{26}$, onde o maior valor que y_{24} e y_{26} podem assumir é $0,50$. Dessa forma, temos que

$$y_{24} + y_{26} + x_2 = 1,5$$

e

$$(|C| - 1)x_2 = 2 \times 0,50 = 1,00$$

resultam. Logo,

$$y_{24} + y_{26} + x_2 > (|C| - 1)x_2$$

se aplica e temos uma desigualdade quadratizada violada.

Vamos agora encontrar uma desigualdade de *lifting*. Utilizando o Algoritmo 2,

obtemos a seguinte desigualdade de *lifting*,

$$2x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 4,$$

definida para $L = \{1, 2, 3, 4, 5, 6\}$. Esta desigualdade é violada por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$.

Observe que se a desigualdade de *cover* obtida é violada, a desigualdade de *lifting* também o será.

Quadratizando a desigualdade de *lifting*, obtemos as seguintes desigualdades

$$2x_1 + y_{12} + y_{13} + y_{14} + y_{15} + y_{16} \leq 3x_1 \tag{4.61}$$
$$2y_{12} + x_2 + y_{23} + y_{24} + y_{25} + y_{26} \leq 3x_2 \tag{4.62}$$
$$2y_{13} + y_{23} + x_3 + y_{34} + y_{35} + y_{36} \leq 3x_3 \tag{4.63}$$
$$2y_{14} + y_{24} + y_{34} + x_4 + y_{45} + y_{46} \leq 3x_4 \tag{4.64}$$
$$2y_{15} + y_{25} + y_{35} + y_{45} + x_5 + y_{56} \leq 3x_5 \tag{4.65}$$
$$2y_{16} + y_{26} + y_{36} + y_{46} + y_{56} + x_6 \leq 3x_6 \tag{4.66}$$
$$2y_{17} + y_{27} + y_{37} + y_{47} + y_{57} + y_{67} \leq 3x_7. \tag{4.67}$$

Observe que (4.61) não é violada por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, enquanto (4.62), (4.63), (4.64), (4.65), (4.66) e (4.67) são.

De acordo com o exemplo acima, temos que, dados os elementos pertencentes ao conjunto das *lifting*, L, ao quadratizamos uma desigualdade de *lifting* em relação a tais elementos, as desigualdades resultantes podem ou não serem violadas pela solução contínua em mãos.

No Capítulo 5 descrevemos os resultados obtidos de acordo com nossos experimentos computacionais para o algoritmo *Branch-and-Cut*.

Capítulo 5

Resultados Computacionais

Neste capítulo descrevemos instâncias, infraestrutura e resultados computacionais obtidos em nossos experimentos computacionais.

5.1 Instâncias

As instâncias utilizadas em nossos experimentos computacionais foram geradas aleatoriamente de acordo com [4, 5, 8, 9, 15]. Estas instâncias obedecem as seguintes regras: % define a densidade da instância, ou seja, o percentual de benefícios $p_{ij},\ i, j \in N$ não-nulos. Cada peso w_j é distribuído aleatoriamente entre $[1, 50]$ para cada $j \in N$, enquanto os benefícios $p_{ij},\ i, j \in N$ com densidade % são distribuidos aleatoriamente entre $[1, 100]$ e $p_{ij} = p_{ji}$. A capacidade c é distribuída aleatoriamente em $[50, \sum_{j=1}^{n} w_j]$.

No endereço `http://cos.ufrj.br/~jossian/instance/` disponibilizamos as instâncias geradas aleatoriamente de acordo com gerador que implementamos. Essas foram utilizadas em nossos diversos experimentos computacionais.

Não foi possível obter as instâncias geradas de outros trabalhos citados, as mesmas seguem as regras utilizadas em nossas instâncias geradas. No algoritmo proposto por Caprara, Pisinger e Toth em [5] o gerador de instância está embutido no código do algoritmo. Billionnet e Soutif disponibilizaram algumas das instâncias utilizadas em seus trabalhos no endereço `http://cedric.cnam.fr/~soutif/QKP/`.

5.2 Infraestrutura

Em nossos experimentos computacionais utilizamos máquinas com as seguintes configurações:

1. Intel(R) Core(TM)2 CPU E7500@2.93GHz, 4GB, 64 bits, Ubuntu, GNU gcc/g++ 4.4.3, denominada **máquina 1**;

2. Intel(R) Xeon(R) CPU X5472 @ 3.00GHz, 8 GB, 64 bits, Ubuntu, GNU gcc/g++ 4.6.3, denominada **máquina 2**;

3. Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz, 2 GB, 32 bits, Ubuntu, GNU gcc/g++ 4.6.3, denominada **máquina 3**.

Utilizamos ainda o resolvedor *Xpress Optimizer*, *release* 23.

5.3 Resultados

Abaixo mostramos através de tabelas os resultados obtidos em nossos experimentos computacionais de acordo com as implementações dos algoritmos descritos em capítulos anteriores.

Nas tabelas abaixo, a coluna n indica o número de itens das instâncias testes e % a sua densidade. Para cada diferente %, 10 instâncias foram geradas. Adicionalmente, a coluna *lb* indica a média dos limites inferiores, a coluna *gap* indica o percentual médio entre limites superiores e limites inferiores, onde o cálculo do *gap* para uma instância r é determinado por $gap_r = \frac{ub_r - lb_r}{lb_r}$. A coluna *nf* indica o número médio de variáveis fixadas em 0 ou em 1. A coluna *no* indica o número de nós percorridos ao longo da árvore de enumeração do *Branch-and-Bound*, enquanto *cut* indica o número de cortes adicionais inseridos durante a árvore de enumeração do *Branch-and-Bound*. A coluna *t(s)* indica o tempo médio em segundos. A coluna *solved* indica o número de instâncias resolvidas em um conjunto de 10 instâncias. Finalmente, a entrada *média* nos mostra a média de cada coluna em uma determinada tabela.

Nas tabelas abaixo mostramos os resultados referente aos algoritmos citados nas seções anteriores, a identificação é feita pelas entradas das colunas: algoritmo proposto por Billionnet e Calmels em [8], coluna *billionnet*, algoritmo proposto por Fomeni e Letchford em [34], coluna *dynamic*, algoritmos proposto por Caprara, Pisinger e Toth em [5] e seu modificado, coluna *cpt* e coluna *cpt-mod*, respectivamente, algoritmo proposto por Rodrigues, Quadri, Michelon e Gueye em [16], coluna *t-linear*, o modelo proposto por Billionnet e Soutif em [31] de acordo com formulação LIN2(3.68), coluna *lin2* e nossas propostas de algoritmos, colunas *jal*, *jal-bc* e *jal-lag*.

O código do algoritmo proposto por Caprara, Pisinger e Toth em [5] foi obtido no endereço `http://www.diku.dk/~pisinger/codes.html`, neste endereço David Pisinger disponibiliza vários códigos referente a seus diversos trabalhos. O código do algoritmo proposto por Billionnet e Calmels em [8] corresponde a uma implementação de David Pisinger e foi obtido no endereço `http://www.diku.dk/~pisinger/codes.html`. O código do algoritmo proposto por Fomeni e Letchford em [34] foi obtido através de *e-mail*, assim como o código do algoritmo proposto

por Rodrigues, Quadri, Michelon e Gueye em [16]. Modelamos a formulação proposta por Billionnet e Soutif em [31] e utilizamos o *Xpress Optimizer* para resolver o problema.

Definimos um tempo limite igual a 43200 segundos para execução de cada instância.

Em nossas instâncias a matriz de benefícios P é gerada como $\{p_{ij} \in \mathbb{N} : i \leq j,\ i, j \in N\}$. Observe que alguns algoritmos trabalham com a matriz de benefícios cheia e simétrica, e outros trabalham apenas com a parte diagonal superior. Para que dado uma instância com matriz de benefícios P os dois algoritmos obtenham o mesmo valor solução devemos adequar a matriz P a cada algoritmo. Desta forma para os algoritmos que utilizam a matriz de benefícios simétrica temos $\{p_{ij} \in \mathbb{N} : p_{ji} = p_{ij},\ i < j,\ i, j \in N\}$, já para os que utilizam apenas a diagonal superior $\left\{p_{ij} \in \mathbb{N} : p'_{ii} = p_{ii},\ p'_{ij} = 2p_{ij},\ i < j,\ i, j \in N\right\}$.

As Tabelas 5.1 e 5.2 mostram os resultados obtidos pelas heurísticas propostas para se obter solução viável para o QKP: a herística proposta por Billionnet e Calmels em [8], a heurística proposta por Fomeni e Letchford em [34], a heurística proposta por Caprara, Pisinger e Toth em [5] e a heurística que estamos propondo baseada em um algoritmo guloso, um procedimento construtivo para gerar solução viável e procedimento de busca local.

Tabela 5.1: Limites inferiores para $n \in \{100, 200, \ldots, 500\}$

		billionnet		*dynamic*		*jal*		*cpt*		*cpt-mod*	
n	*%*	*lb*	*t(s)*	*lb*	*t(s)*	*lb*	*t(s)*	*lb*	*t(s)*	*lb*	*t(s)*
100	25	32009	0,0	32019	0,4	**32025**	3,3	32024	0,2	**32025**	1,9
	50	51088	0,0	**51100**	0,3	51093	4,0	51093	0,3	51093	3,1
	75	99782	0,0	**100079**	0,3	**100079**	2,0	100046	0,3	**100079**	2,2
	100	135201	0,0	135529	0,3	**135533**	1,6	135532	0,2	**135533**	1,4
200	25	168960	0,0	169013	4,1	**169041**	26,0	169025	2,0	169020	18,3
	50	155818	0,0	**155877**	3,7	155833	24,5	155833	2,3	155833	22,3
	75	386208	0,0	386392	3,7	**386412**	10,9	386297	1,5	386369	12,2
	100	536360	0,1	**536900**	4,1	536731	7,9	536727	1,0	536731	7,5
300	25	273078	0,0	273223	19,1	**273232**	83,0	**273232**	8,3	**273232**	61,2
	50	601561	0,1	**601695**	18,2	601629	79,6	601608	9,5	601608	78,1
	75	889353	0,0	890045	17,7	**890050**	19,8	889721	6,0	889916	42,5
	100	1375434	0,0	1375847	16,1	**1375850**	17,6	1375711	2,9	1375847	19,8
400	25	635844	0,1	635930	56,0	**635932**	195,8	**635932**	19,2	**635932**	140,2
	50	1489291	0,1	**1489748**	50,4	1489717	87,5	1489639	13,7	1489639	100,7
	75	1585633	0,1	1586347	52,5	**1586350**	48,0	1585721	16,7	1585775	109,0
	100	1491173	0,1	1492529	54,3	**1492545**	39,2	1491866	7,7	1491990	50,4
500	25	766058	0,2	**766180**	124,5	766102	363,0	766088	53,1	766088	342,2
	50	1643827	0,2	**1644164**	131,9	1644089	310,0	1643995	59,4	1643995	399,4
	75	3107009	0,2	3108622	130,5	**3108635**	99,5	3107009	43,5	3107009	265,1
	100	4115356	0,1	4117476	130,0	**4117515**	68,9	4116787	18,0	4117414	97,8

De acordo com as Tabelas 5.1 e 5.2 obtemos na média melhores limites inferiores em 22 dos 40 conjuntos de instâncias testadas. Nossos tempos computacionais são melhores para instâncias de alta densidade, mas perdemos para instâncias de baixa densidade.

A Tabela 5.3 mostra os resultados obtidos de acordo com *gaps* gerados pelos al-

Tabela 5.2: Limites inferiores para $n \in \{600, 700, \ldots, 1000\}$

		billionnet		*dynamic*		*jal*	
n	*%*	*lb*	*t(s)*	*lb*	*t(s)*	*lb*	*t(s)*
600	25	1308827	0,2	**1309556**	277,6	1309439	875,6
	50	2470744	0,3	**2471953**	222,2	2471935	361,6
	75	4079592	0,3	**4082108**	244,3	4081732	144,3
	100	3635726	0,4	3640788	267,0	**3640795**	105,5
700	25	1566908	0,6	**1567179**	429,4	1567012	992,7
	50	3403025	0,5	**3405522**	467,6	3405178	841,9
	75	3598813	0,8	3599652	418,9	**3599675**	237,1
	100	4951411	0,7	4955279	506,0	**4955300**	177,6
800	25	2494139	0,6	**2494287**	778,7	2494232	1920,3
	50	3825535	1,0	**3826974**	779,1	3826056	1549,0
	75	6539477	0,9	**6540490**	793,3	6540345	403,0
	100	7144316	1,0	7148070	737,6	**7148098**	210,4
900	25	2907631	1,1	**2908719**	1245,5	2908428	3384,5
	50	5918829	1,1	**5922138**	1222,9	5921434	3023,2
	75	7777714	1,3	7779437	1202,1	**7779450**	458,9
	100	9853480	1,4	9857789	1320,8	**9857837**	404,6
1000	25	3763411	1,5	**3764266**	1761,4	**3764266**	4920,4
	50	4749226	2,1	**4752256**	2106,1	4751488	4230,2
	75	10292321	1,7	10293273	1633,9	**10293333**	511,1
	100	12875207	1,7	13826435	1754,8	**13826466**	481,5

goritmos proposto por Caprara, Pisinger e Toth em [5], esse modificado, o algoritmo proposto por Rodrigues, Quadri, Michelon e Gueye em [16] e o algoritmo *Relax-and-Cut* que estamos propondo para $n \in \{100, \ldots, 500\}$. A Tabela 5.4 mostra os resultados obtidos de acordo com a média das densidades da Tabela 5.3. Esses *gaps* foram gerados no nó raiz e não foi executado o procedimento de fixação de variáveis.

Tabela 5.3: Gaps Primal-Dual para $n \in \{100, 200, \ldots, 500\}$

		jal		*cpt*		*cpt-mod*		*t-linear*	
n	*%*	*gap*	*t(s)*	*gap*	*t(s)*	*gap*	*t(s)*	*gap*	*t(s)*
100	25	1,56	3,3	2,08	0,2	**1,54**	1,9	3,61	0,7
	50	**1,01**	4,0	2,06	0,3	1,81	3,1	3,13	1,4
	75	**0,19**	2,0	1,52	0,3	1,24	2,2	1,89	0,6
	100	**0,14**	1,6	0,61	0,2	0,37	1,4	0,56	0,5
200	25	2,17	26,0	2,49	2,0	**2,12**	18,3	2,33	4,9
	50	**3,47**	24,5	4,01	2,3	3,76	22,3	5,37	5,7
	75	**0,25**	10,9	1,29	1,5	1,14	12,2	1,83	5,9
	100	**0,16**	7,9	0,40	1,0	0,24	7,5	0,52	4,2
300	25	1,76	83,0	1,98	8,3	**1,65**	61,2	2,54	19,9
	50	**1,16**	79,6	1,71	9,5	1,57	78,1	2,18	21,4
	75	**0,06**	19,8	0,89	6,0	0,77	42,5	1,35	28,1
	100	**0,17**	17,6	0,41	2,9	0,28	19,8	0,34	16,0
400	25	0,93	195,8	1,13	19,2	**0,75**	140,2	1,25	44,5
	50	**0,36**	87,5	0,67	13,7	0,48	100,7	0,70	36,3
	75	**0,06**	48,0	1,25	16,7	1,09	109,0	1,76	36,4
	100	**0,12**	39,2	0,45	7,7	0,24	50,4	0,31	26,6
500	25	1,95	363,0	2,07	53,1	**1,62**	342,2	7,28	108,0
	50	2,21	310,0	2,60	59,4	2,50	399,4	**0,79**	19,8
	75	**0,04**	99,5	0,91	43,5	0,86	265,1	2,04	53,4
	100	**0,06**	68,9	0,20	18,0	0,09	97,8	0,74	28,2
média		**1,19**	24,3	1,82	2,6	1,54	20,8	2,03	23,1

Com relação a *gaps*, as Tabelas 5.3 e 5.4 mostram que nossos *gaps* são melhores para os conjuntos de instâncias com densidade 50%, 75% e 100% comparado aos outros algoritmos, exceto o conjunto onde n é igual a 500 e possui densidade 50%.

Tabela 5.4: Média por densidade das instâncias da Tabela 5.3

	jal		*cpt*		*cpt-mod*		*t-linear*	
%	*gap*	*t(s)*	*gap*	*t(s)*	*gap*	*t(s)*	*gap*	*t(s)*
25	1,67	134,2	1,95	16,6	**1,53**	112,8	3,40	35,6
50	**1,64**	101,1	2,21	17,0	2,02	120,7	2,43	16,9
75	**0,12**	36,0	1,17	13,6	1,02	86,2	1,78	24,9
100	**0,13**	27,0	0,42	6,0	0,24	35,4	0,49	15,1

A Tabela 5.5 mostra os resultados referente ao número de variáveis fixadas pelos algoritmos proposto por Caprara, Pisinger e Toth em [5], esse modificado e o algoritmo *Relax-and-Cut* que estamos propondo.

Tabela 5.5: Fixação de variáveis

		jal		*cpt*		*cpt-mod*	
n	%	*nf*	*t(s)*	*nf*	*t(s)*	*nf*	*t(s)*
100	25	32	6,4	**51**	0,5	**51**	2,9
	50	18	6,7	**23**	0,4	**23**	3,8
	75	**78**	2,4	52	0,4	60	3,2
	100	86	1,5	**89**	0,2	**89**	1,7
200	25	53	27,2	56	2,4	**85**	19,6
	50	37	30,7	**39**	2,5	**39**	23,2
	75	**127**	13,8	79	2,4	79	18,4
	100	156	13,1	162	1,8	**166**	12,2
300	25	83	90,2	84	9,7	**134**	71,2
	50	**60**	86,7	43	10,2	43	78,6
	75	**245**	24,3	136	15,1	136	80,6
	100	191	24,9	**203**	7,4	**203**	34,5
400	25	32	185,2	35	19,9	**111**	148,7
	50	**255**	150,6	205	19,8	228	111,3
	75	**183**	57,5	124	19,3	124	113,3
	100	327	54,9	325	14,1	**340**	64,5
500	25	105	389,3	39	56,0	**142**	559,7
	50	**86**	371,9	0	62,0	0	402,9
	75	**387**	125,8	171	66,6	171	325,2
	100	382	90,0	399	43,0	**400**	137,2
média		**146**	87,7	116	17,7	131	110,6

Pela Tabela 5.5 temos que nossa proposta de algoritmo *Relax-and-Cut* ganha quanto ao número de variáveis fixadas em todos os conjuntos de instâncias com densidade 75%. O algoritmo *cpt-mod* gerou melhores resultados, apesar de que na média nossa proposta de algoritmo fixou mais variáveis.

A Tabela 5.6 mostra os resultados obtidos por algoritmos *Branch-and-Bound*: o algoritmo proposto por Caprara, Pisinger e Toth em [5], o algoritmo *Branch-and-Bound* proposto por Rodrigues, Quadri, Michelon e Gueye em [16] e nossas propostas de algoritmos. Incluimos ainda os resultados obtidos pela formulação LIN2 (3.68) proposta por Billionnet e Soutif em [31] onde utilizamos o resolvedor de MIP *Xpress Optimizer*.

Em nossa proposta de algoritmo *Branch-and-Bound*, colunas *jal-bc* e *jal-lag*, implementamos um algoritmo *Relax-and-Cut* no nó raiz combinado aos procedimentos de fixação de variáveis e redução do tamanho do problema, já a implementação da árvore de enumeração do *Branch-and-Bound* foi feita de duas formas. Na primeira forma, coluna *jal-bc*, implementamos um algoritmo algoritmo *Branch-and-*

Cut, onde inserimos desigualdades de *extended*, quando violadas, como planos de corte na árvore de enumeração do *Branch-and-Bound*, essas ideias foram descritas no Capítulo 4. Já a segunda forma, coluna *jal-lag*, a implementação foi feita de acordo com nossa proposta de algoritmo *Branch-and-Bound* citada na Seção 4.1.3, onde implementamos em cada nó problema um algoritmo *Relax-and-Cut*.

Tabela 5.6: *Branch-and-Bound QKP*

		lin2			*jal-bc*				*cpt*			*t-linear*			*jal-lag*		
n	*%*	*no*	*t(s)*	*solved*	*cut*	*no*	*t(s)*	*solved*	*no*	*t(s)*	*solved*	*no*	*t(s)*	*solved*	*no*	*t(s)*	*solved*
100	25	352	1,7	10	17	15	22,3	10	6442	0,5	10	125	0,3	10	634	155,0	10
	50	799	3,4	10	20	46	90,7	10	35763	0,6	10	494	0,7	10	3056	1031,1	10
	75	662	3,5	10	21	24	20,8	10	1123	0,3	10	1354	0,7	10	477	25,1	10
	100	486	3,8	10	9	6	11,0	10	303	0,3	10	499	0,2	10	1258	24,2	10
200	25	6546	53,0	9	153	602	4051,2	9	44211536	340,2	8	462193	3051,9	10	6992	10141,8	4
	50	26412	289,4	10	21	189	2334,7	10	636587	12,3	10	5473	20,5	10	7566	14488,1	7
	75	48265	446,0	10	189	410	5839,8	9	128634	2,9	10	76027	24,8	10	1177	150,6	7
	100	264899	1005,7	10	43	55	1322,6	10	391469	2,6	10	1380237	184,0	10	2323	97,9	7
300	25	229539	2568,7	10	21	234	5367,0	9	162321973	2995,3	7	27283	479,6	10	3386	6140,3	4
	50	149327	4308,6	10	160	438	16199,9	7	88891395	1335,9	10	48645	106,0	10	1117	77,0	1
	75	22666	575,2	9	45	91	7855,0	9	472467	23,7	10	641655	857,7	10	204358	1218,0	8
	100	378546	5285,9	8	3	6	2631,3	6	499496596	152,5	10	8567932	2286,8	8	45748	2707,4	5
400	25	120225	3139,9	8	19	36	7735,8	5	207586606	6542,4	7	24177	631,1	10	4003	396,0	1
	50	127356	4677,1	9	14	27	2073,8	6	309939	22,4	9	40172	340,6	10	34972	6502,1	7
	75	32611	1444,6	5	10	12	527,5	4	46337835	277,7	10	1087149	1739,6	9	56321	5238,2	5
	100	378076	10057,0	6	7	8	1198,0	7	908969	16,4	9	1755686	414,0	8	23452	786,0	6

Como mostrado na Tabela 5.6 os algoritmos *cpt* e *t-linear* obtiveram melhores resultados quanto ao número de instâncias resolvidas. O algoritmo *cpt* obteve melhor performace para instâncias com densidades 75% e 100%, enquanto *t-linear* melhor performace para instâncias com densidade 25% e 50%. Em nossas proposta de algoritmo, *jal-bc* e *jal-lag*, um número inferior de nós são percorridos ao longo da árvore de enumeração do *Branch-and-Bound*, esse número de nós é bem inferior para o algoritmo *jal-bc*. Quanto ao tempo de CPU, *cpt* tem melhor performace comparado aos outros algoritmos.

As Tabelas 5.7, 5.8 e 5.9 trazem os resultados referente aos algoritmos *Branch-and-Cut* descrito no Capítulo 4, onde comparamos entre si as desigualdades (3.7), *cover*, *extended* e *lifting*, estas foram inseridas ao modelo quando violadas como planos de corte, seguindo as ideias de um algoritmo *Branch-and-Cut*. As Tabelas 5.7 e 5.8 trazem ainda resultados de um algoritmo *Branch-and-Bound*, coluna *xpress*, onde passamos a formulação (4.29)-(4.36) para o resolvedor *Xpress Optimizer*, e deixamos a resolução a cargo do mesmo, sem alteração de parâmetros.

Tabela 5.7: *Branch-and-Cut* para o QKP para $n \in \{20, \ldots, 80\}$

		xpress		*triângulo*			*cover*			*extended*			*lifting*		
n	*%*	*no*	*t(s)*	*cut*	*no*	*t(s)*	*cut*	*no*	*t(s)*	*cut*	*no*	*t(s)*	*cut*	*no*	*t(s)*
20	25	9	0,4	204	7	0,8	7	7	0,3	4	6	0,3	5	6	0,5
	50	14	0,5	164	10	0,8	10	11	0,7	6	8	0,5	6	8	0,7
	75	21	0,6	117	15	0,7	20	14	0,9	6	11	0,7	4	8	0,9
	100	25	0,5	136	20	0,8	16	20	1,0	4	8	0,6	4	8	0,9
40	25	33	3,6	1614	35	10,5	20	24	3,6	14	24	3,3	13	19	3,7
	50	35	4,8	646	27	8,2	18	22	4,7	12	21	4,6	5	17	4,1
	75	64	6,3	779	54	8,6	24	25	5,7	5	13	4,0	7	13	4,6
	100	254	11,9	1968	205	15,5	116	181	14,3	13	40	9,2	14	31	8,1
60	25	29	10,9	4947	69	135,0	26	35	12,3	13	29	10,0	23	38	14,0
	50	158	37,3	3751	120	133,6	67	104	37,0	30	85	29,6	26	82	35,6
	75	312	62,4	4632	269	123,3	129	219	65,1	34	117	47,5	53	121	51,2
	100	2366	233,2	18836	1796	370,6	799	1477	234,0	68	184	70,1	19	63	40,0
80	25	95	53,4	9000	103	881,6	51	47	39,5	56	74	53,2	27	56	45,6
	50	115	86,3	8328	150	1844,9	66	89	88,3	25	75	78,2	23	57	77,8
	75	2003	616,5	51688	1500	1266,9	671	1091	632,6	99	204	209,9	58	135	163,8
	100	6513	1592,4	49653	7914	2236,0	1462	2746	1605,2	20	89	128,0	22	76	149,3

Tabela 5.8: *Branch-and-Cut* para o QKP para $n \in \{100, 120\}$

		xpress		*cover*			*extended*			*lifting*		
n	*%*	*no*	*t(s)*	*cut*	*no*	*t(s)*	*cut*	*no*	*t(s)*	*cut*	*no*	*t(s)*
100	25	109	132,7	35	38	75,2	31	46	83,6	20	36	73,7
	50	110	195,3	85	114	260,9	24	60	173,6	39	122	277,7
	75	310	367,2	98	170	286,4	97	213	410,4	28	62	184,6
	100	600	530,8	234	508	534,7	15	54	186,7	54	134	337,2
120	25	274	333,6	56	152	252,4	17	99	129,9	36	132	198,1
	50	482	686,9	72	105	558,3	25	62	329,7	30	83	378,4
	75	1138	2453,0	1193	2351	3891,6	288	742	2099,4	57	123	828,4
	100	8656	7855,1	1510	2779	4158,6	13	61	457,7	25	82	550,6

Pela Tabela 5.7 temos que a desigualdade (3.7), coluna *triângulo*, não é uma boa opção a ser utilizada como planos de corte em um algoritmo estilo *Branch-and-Cut*, uma das razões é o elevado tempo de CPU. É possível observar ainda um elevado

Tabela 5.9: *Branch-and-Cut* para o QKP para $n \in \{140, 200\}$

		extended				*lifting*			
n	%	*cut*	*no*	*t(s)*	*solved*	*cut*	*no*	*t(s)*	*solved*
140	25	77	261	694,5	10	60	157	615,5	10
	50	269	651	4408,6	10	128	414	4099,3	10
	75	358	876	4598,4	10	140	458	5083,5	10
	100	17	259	2623,8	10	112	686	3692,9	9
200	25	245	968	12386,0	9	176	895	10003,2	9
	50	62	407	7115,2	9	90	454	9170,0	10
	75	64	177	7305,3	7	153	424	9684,5	6
	100	8	83	6076,9	8	162	419	6898,0	7

número de cortes inseridos ao longo da árvore de enumeração, além de um elevado número de nós percorridos.

De acordo com as Tabelas 5.7, 5.8 e 5.9 entre as desigualdades testadas verificamos que as desigualdades de *extended* e *lifting* demonstram ser melhores opções a serem utilizadas como planos de corte em um algoritmo estilo *Branch-and-Cut*. A principal razão é o tempo de CPU combinado ao número de nós percorridos e cortes inseridos ao longo da árvore de enumeração.

A Tabela 5.10 mostra os resultados referente a experimentos computacionais de dois algoritmos *Branch-and-Cut*. A coluna *quadratic extended* refere-se a um algoritmo onde desigualdades de *extended* quadratizadas, quando violadas, foram inseridas como planos de corte ao longo da árvore de enumeração do *Branch-and-Bound*. A coluna *fixation extended* indica os resultados referentes a um algoritmo onde desigualdades de *extended*, quando violadas, foram inseridas como planos de corte ao longo da árvore de enumeração do *Branch-and-Bound* e variáveis x, coluna $\#x$, e y, coluna $\#y$, foram fixadas no nó raiz utilizando-se ideias de custos reduzidos.

Quanto ao uso das desigualdades *extended* quadratizadas como planos de corte em um algoritmo estilo *Branch-and-Cut*, não observamos melhora de performace computacional comparado ao uso apenas das desigualades de *extended*, conforme Tabela 5.10. Já a implementação do procedimento de fixação de variáveis no nó raiz utilizando-se as desigualdades de *extended* como planos de corte ao longo da árvore de enumeração do *Branch-and-Bound*, também não nos forneceu ganho de performace comparado ao uso apenas das desigualdades de *extended* sem o procedimento de fixação de variáveis, conforme Tabela 5.10.

As implementações cujos resultados constam na Tabela 5.6 foram executadas na **máquina 1**, enquanto as implementações cujos resultados constam nas Tabelas 5.1, 5.2, 5.3, 5.4 e 5.5 foram executadas na **máquina 2** e finalmente as implementações cujas implementações constam nas Tabelas 5.7, 5.8, 5.9 e 5.10 foram executadas na **máquina 3**.

Tabela 5.10: *Branch-and-Cut QKP - Extended*

		quadratic extended			*fixation extended*				
n	*%*	*cut*	*no*	*t(s)*	*#x*	*#y*	*cut*	*no*	*t(s)*
20	25	58	6	0,4	1	17	4	5	0,3
	50	75	6	0,6	0	12	5	7	0,5
	75	100	6	0,7	0	20	6	9	0,8
	100	77	2	0,5	0	20	4	7	0,5
40	25	321	18	3,5	1	77	15	20	3,0
	50	227	13	3,2	1	59	10	17	4,2
	75	181	11	3,9	3	128	6	14	4,2
	100	392	16	6,0	3	118	14	39	8,9
60	25	602	39	13,4	4	263	14	30	9,7
	50	1477	97	41,8	3	188	28	83	29,0
	75	1151	65	35,3	1	75	33	111	45,0
	100	2145	69	37,7	4	199	68	185	69,8
80	25	2204	65	77,4	5	475	40	56	44,9
	50	1824	99	110,2	4	320	25	76	78,3
	75	7999	229	261,4	4	408	99	206	212,5
	100	1148	40	77,6	2	219	20	82	122,5
100	25	2368	66	129,2	3	287	30	61	86,1
	50	2382	90	306,8	6	629	26	64	183,7
	75	4116	98	346,3	9	851	98	214	415,9
	100	1079	11	46,2	13	1046	15	53	183,8
120	25	1292	128	215,8	4	625	18	98	125,8
	50	3046	88	659,4	11	1155	31	92	409,4
	75	28773	638	2911,7	4	807	297	778	2728,0
	100	1624	49	269,7	7	1405	17	77	560,7

Capítulo 6

Conclusões

Neste trabalho propomos uma heurística Lagrangeana baseada em um algoritmo *Relax-and-Cut*. Essa heurística é voltada para geração de soluções viáveis de boa qualidade para o QKP, embora o tempo de CPU seja não desprezível.

Nossos experimentos computacionais confirmaram que nossa heurística e a heuristíca baseada em Programação Dinâmica [34] possuem uma boa performace. Este fato é certificado pela qualidade dos limites inferiores gerados pelas heurísticas de acordo com as Tabelas 5.1 e 5.2.

De acordo com nossos resultados computacionais obtidos nossos limites superiores são competitivos com aqueles gerados pelos algoritmos propostos em [5] e [16]. De fato, nossos *gaps* para instâncias onde $n \in \{100, \ldots, 500\}$ acabaram sendo melhores que aqueles gerados pelo algoritmo proposto em [5] para instâncias com densidades 50%, 75% e 100%. Em comparação com o algoritmo proposto em [16], nosso algoritmo gera melhores *gaps* para todos os grupos de instâncias teste, exceto para instâncias de tamanho 500 e densidade 50%. É bom frisar que o limite inferior utilizado em [16] é o limite gerado pela heurística proposta por Billionnet e Calmels em [8].

Quanto ao número de variáveis fixadas, nosso algoritmo fixou um considerável número de variáveis comparados aos fixados pelo algoritmo proposto por [5] e seu modificado de acordo com Tabela 5.5.

Como indicado em nossa investigação, desigualdades válidas para o KP podem ser utilizadas em um algoritmo *Branch-and-Cut* para o QKP. Isto é feito com o intuito de reduzir o número de nós percorridos na árvore de enumeração do algoritmo *Branch-and-Bound* e, eventualmente, o tempo total de CPU.

Em nossos testes computacionais observamos que o uso das desigualdades (3.7) se torna computacionalmente caro com o aumento da dimensão das instâncias. Observamos ainda um elevado número de nós são visitados nas árvore de enumeração, quando utilizamos tais desigualdades. Por sua vez, o uso das desigualdades de *cover*, *extended* e *lifting* se mostraram particularmente interessantes.

Realizamos ainda testes computacionais utilizando as desigualdades de *cover*, *extended* e *lifting*, quadratizando-as e realizando fixação de variáveis. Entretanto, os resultados computacionais obtidos não se mostraram atraentes.

Quanto ao algoritmo *Branch-and-Bound*, observamos que temos ganho quando comparado aos outros algoritmos no número de nós percorridos na árvore de enumeração na média para todas instâncias teste quanto a densidade.

Uma contribuição relevante neste trabalho, além dos algoritmos *Relax-and-Cut*, *Branch-and-Cut* e *Branch-and-Bound*, é o fato de utilizamos os mesmos conjuntos de instâncias e o fato da comparação entre algoritmos de acordo com tabelas terem sido realizados na mesma máquina.

Como proposta de trabalho futuro temos o investimento na implementação, bem como busca de novas ideias que possam contribuir para melhora na árvore de enumeração de nosso algoritmo *Branch-and-Bound* proposto, de forma que possamos obter melhores resultados para o algoritmo e consegui resolver QKP de tamanhos maiores.

Referências Bibliográficas

[1] MARTELLO, S., TOTH, P. *Knapsack Problems: algorithms and computer implementations*. New York, USA, John Wiley & Sons, Inc., 1990.

[2] PISINGER, D., TOTH, P. *Knapsack Problems, in: D. Du, P. Pardalos (Eds.), Handbook of Combinatorial Optimization*. Kluwer Academic, 1998.

[3] KELLERER, H., PFERSCHY, U., PISINGER, D. *Knapsack Problems*. Berlim, Germany, Springer, 2004.

[4] GALLO, G., HAMMER, P. L., SIMEONE, B. "Quadratic knapsack problems", *Mathematical Programming*, v. 12, pp. 132–149, 1980.

[5] CAPRARA, A., PISINGER, D., TOTH, P. "Exact Solution of the Quadratic Knapsack Problem", *Journal on Computing*, v. 11, pp. 125–137, 1999.

[6] PISINGER, D. "The quadratic knapsack problem - a survey", *Discrete Applied Mathematics*, pp. 623–648, 2007.

[7] PALMEIRA, M. M. *Um Algoritmo Relax-and-Cut para o Problema Quadrático da Mochila Binária*. Tese de Mestrado, PUC, Rio de Janeiro, RJ, Brasil, 1999.

[8] BILLIONNET, A., CALMELS, F. "Linear programming for the 0-1 quadratic knapsack problem", *European J. Oper. Res.*, v. 92, pp. 310–325, 1996.

[9] MICHELON, P., VEILLEUX, L. "Lagrangean methods for the 0-1 quadratic knapsack problem", *European J. Oper. Res.*, v. 92, pp. 326–341, 1996.

[10] BILLIONNET, A., FAYE, A., SOUTIF, E. "A new upper bound for the 0-1 quadratic knapsack problem", *European J. of Oper. Res.*, v. 112, n. 3, pp. 664–672, 1999.

[11] BILLIONNET, A., SOUTIF, E. "An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem", *European J. Oper. Res.*, v. 157, n. 3, pp. 565–575, 2004.

[12] WITZGALL, C. "Mathematical methods of site selection for electronic message system (EMS)", *Technical Report, NBS Internal Report*, 1975.

[13] JOHNSON, E. J. L., MEHROTRA, A., NEMHAUSER, G. L. "Min-cut clustering", *Mathematical Programming*, v. 62, pp. 133–151, 1993.

[14] JOHNSON, D., TRICK, M. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. American Mathematical Society, 1996.

[15] CHAILLOU, P., HANSEN, P., MAHIEU, Y. "Best network flow bound for the quadratic knapsack problem", *Combinatorial Optimization, Lecture Notes in Mathematics*, v. 1403, pp. 225–235, 1989.

[16] RODRIGUES, C. D., QUADRI, Q., MICHELON, P., et al. "0-1 Quadratic Knapsack Problems: An exact approach based on t-linearization", *Journal on Optimization*, v. 22, pp. 1449–1468, 2012.

[17] HELMBERG, C., RENDL, F., WEISMANTEL, R. "Quadratic knapsack relaxations using cutting planes and semidefinite programming", *Proceedings of the Fifth IPCO Conference, Lecture Notes in Computer Science*, v. 1084, pp. 175–189, 1996.

[18] HELMBERG, C., RENDL, F., WEISMANTEL, R. "A semidefinite programming approach to the quadratic knapsack problem", *J. Combin. Optim.*, v. 4, pp. 197–215, 1996.

[19] BEASLEY, J. *Lagrangian Reaxation*. London, The Manegement School - Imperial College, 1992.

[20] HELD, M., WOLFE, P., CROWDER, H. "Validation of subgradiente optimization", *Mathematical Programming*, v. 6, pp. 62–88, 1974.

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

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

[23] ESCUDERO, L., GUIGNARD, M., MALIK, K. "A Lagrangean relax and cut approach forthe sequential ordering with precedence constraints." *Annals of Operations Research*, v. 50, pp. 219–237, 1994.

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

[25] LUCENA, A. "Tight bounds for the steiner problem in graphs." *In Proceedings of NET-FLOW93*, pp. 147–154, 1993.

[26] ADAMS, W. P., SHERALI, H. D. "A tight linearization and an algorithm for zero-one quadratic programming problems", *Manage. Sci.*, v. 32, pp. 1274–1290, 1986.

[27] PADBERG, M. "The Boolean Quadratic Polytope: Some Characteristics, Facets and Relatives", *Mathematical Programming*, v. 45, pp. 139–172, 1989.

[28] CROWDER, H., JOHNSON, E., PADBERG, M. "Solving large-scale 0-1 linear programming programs", *Oper. Res.*, v. 31, pp. 803–834, 1983.

[29] BALAS, E. "The prize collecting traveling salesman problem", *Networks*, v. 19, pp. 621–636, 1989.

[30] WOLSEY, L. A. "Faces for linear inequalities in 0-1 variables", *Mathematical Programming*, v. 8, pp. 165–178, 1975.

[31] BILLIONNET, A., SOUTIF, E. "Using a Mixed Integer Programming Tool for Solving the 0-1 Quadratic Knapsack Problem", *J. on Computing*, v. 16, pp. 188–197, 2004.

[32] MACULAN, N., FAMPA, M. *Otimização linear*. Brasília, Brasil, Brasília: Editora da Universidade de Brasília, 2006.

[33] ELLOUMI, S., FAYE, A., SOUTIF, E. "Decomposition and Linearization for 0-1 Quadratic Programming", *Annals of Operations Research*, v. 99, pp. 79–93, 2000.

[34] FOMENI, F. D., LETCHFORD, A. N. "A Dynamic Programming Heuristic for the Quadratic Knapsack Problem", *J. on Computing*, v. 26, pp. 173–182, 2013.

[35] DANTZIG, G. B. "Discrete Variables Extremum Problems", *Operations Research*, v. 5, pp. 266–277, 1957.

[36] BALAS, E., ZEMEL, E. "An Algorithm for Large Zero-One Knapsack Problems", *Operations Research*, v. 28, pp. 511–538, 1980.

[37] CORPORATION, F. I. "FICO Xpress Optimization Suite". 2013.

[38] SIMONETTI, L., CUNHA, A. S., LUCENA, A. "Polyhedral results and a Branch-and-cut algorithm for the k-cardinality tree problem", *Mathematic Programming*, v. 142, pp. 511–538, 2013.