UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

INSTITUTO DE CIÊNCIAS EXATAS

CURSO DE GRADUAÇÃO EM MATEMÁTICA

Isabela de Aquino Souza

O Problema do Caixeiro Viajante e o Limitante de Held-Karp: Uma aplicação à Elaboração de Roteiros Turísticos

SEROPÉDICA

2017

Isabela de Aquino Souza

O Problema do Caixeiro Viajante e o Limitante de Held-Karp: Uma aplicação à Elaboração de Roteiros Turísticos

Monografia apresentada à Banca Examinadora da Universidade Federal Rural do Rio de Janeiro, como requisito parcial para obtenção do título de Bacharel em Matemática, sob a orientação do Prof. Dr. Vinicius Leal do Forte

SEROPÉDICA

2017

UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE MATEMÁTICA

COORDENAÇÃO DO CURSO DE GRADUAÇÃO EM MATEMÁTICA.

A monografia "O PROBLEMA DO CAIXEIRO VIAJANTE E O LIMITANTE DE HELD-KARP: UMA APLICAÇÃO À ELABORAÇÃO DE ROTEIROS TURÍSTICOS", apresentada e defendida por ISABELA DE AQUINO SOUZA matrícula 201319023-7 foi aprovada pela Banca Examinadora, com conceito "S" recebendo o número 686.

Seropédica, 14 de dezembro de 2017.

BANCA EXAMINADORA

Prof. Dr. Vinícius Leal do Forte
Orientador

Douglas Monsôres de Melo

Prof. Dr. Douglas Monsôres de Melo Santos

Renan Vicente Pinto

Prof. Dr. Renan Vicente Pinto

Uns confiam em carros e outros em cavalos, mas nós confiamos no nome do Senhor nosso Deus.

Salmos 20:7

Agradecimentos

A Deus, por ter me sustentado sempre, por ter me dado saúde e força para superar as dificuldades, Ele foi o meu auxílio em todos os momentos.

Aos meus pais que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida. Amo vocês.

Às minhas irmãs, Isadora e Iasmin que, com muito senso de humor, amenizaram meus momentos de preocupação.

Ao meu noivo, Leonardo, que de forma especial e carinhosa me deu força e coragem, me apoiando nos momentos de dificuldade.

Ao professor Vinicius Leal do Forte pela paciência na orientação e incentivo que tornaram possível a conclusão desta monografia.

Agradeço também a todos os professores do DEMAT que me acompanharam durante a graduação e foram tão importantes na minha vida acadêmica e no desenvolvimento desta monografia.

Por fim, agradeço aos amigos e colegas pelo incentivo e apoio constante, sem vocês essa jornada não teria tantos momentos divertidos.

Resumo

Este trabalho relata um exemplo prático e bastante atual de aplicação do Problema do Caixeiro Viajante (PCV). Neste problema um vendedor deve visitar n cidades exatamente uma vez e retornar ao seu ponto de partida. O custo necessário para ir da cidade i até a cidade j é c_{ij}. Esse custo pode ser oriundo do tempo de viagem, da distância entre as cidades ou da dificuldade de realizar o percurso. O objetivo é localizar a melhor ordem (rota) em que esse vendedor deve fazer a viagem de modo a minimizar o custo total da viagem. Analisando este problema como um grafo, cada cidade é vista como um vértice e o caminho que liga diretamente a cidade i à cidade j corresponde a uma aresta. Portanto, o caminho que desejamos encontrar, o qual apresenta solução ideal para o PCV, é o ciclo hamiltoniano de menor custo. Esse é um problema muito relevante devido à sua grande aplicação prática, enorme relação com outros problemas e à dificuldade de determinar uma solução exata. Em virtude dessa complexidade, existem várias maneiras de determinar soluções aproximadas para o PCV, entre elas as Relaxações e as Heurísticas. O exemplo efetivamente abordado no trabalho é o de conectar 7 pontos turísticos a fim de formar uma rota que minimize o tempo de viagem entre cada um dos pontos. Isto é interessante, por exemplo, para uma agência de viagens que deseja formar um roteiro turístico gastando o menor tempo possível no translado.

Palavras-Chave: Roteiros Turísticos; Caixeiro Viajante; Relaxação; Heurística

Sumário

Introdução **1**

1 Árvore Geradora **4**

2 Programação Inteira **11**

2.1 Exemplos: Formulações de alguns PPI importantes 13

2.1.1 Problema de Atribuição - $Assignment$ 13

2.1.2 Problema de Cobertura de Conjuntos 14

2.2 Notações e Resultados Básicos 15

2.3 Otimalidade e Limitantes 16

3 Problema do Caixeiro Viajante (PCV) **20**

3.1 Uma formulação para o PCV 21

4 Relaxação Lagrangeana **25**

4.1 Conceitos Básicos 25

4.2 Relaxação Lagrangeana Avançada 30

4.3 Otimização Subgradiente 31

5 Limitante de Held e Karp **36**

5.1 A *1-tree* 37

5.2 A proposta de Held e Karp 38

5.3 Exemplo Prático 42

5.3.1 Resultados 47

6 Considerações Finais **53**

7 Apêndice **59**

7.1 APÊNDICE A - Código em Python 59

8 Anexos **66**

8.1 ANEXO A - Algoritmo de Prim 66

8.2 ANEXO B - Algoritmo de Kruskal 67

Introdução

O Problema do Caixeiro Viajante (PCV) é um dos mais tradicionais e conhecidos Problemas de Programação Inteira (PPI). Este problema foi inspirado em uma profissão muito antiga, chamada Caixeiro Viajante, de uma pessoa que vendia produtos em cidades onde eles não eram produzidos. Nos tempos em que não havia facilidade de transporte, estes profissionais representavam uma importante forma de levar produtos para regiões distantes das grandes cidades.

Basicamente o PCV consiste em um vendedor (caixeiro viajante) que visitará n cidades e precisa traçar uma rota entre elas de tal forma a sair de um determinado ponto, visitar todos os outros uma única vez e retornar para o ponto de partida. O objetivo central deste problema é propor uma rota que tenha o menor custo possível, seja ele, dirimindo as distância, o tempo de viagem ou gasto de combustível, por exemplo.

Embora o Problema do Caixeiro Viajante seja dificil de solucionar, uma vez que é NP-difícil ([1], [2]) ou seja, não existe um algoritmo eficiente para resolvê-lo, ele tem sido muito utilizado no experimento de diversos métodos de otimização por ser, principalmente, um problema de fácil descrição e compreensão. Além disso, existem várias aplicações para o PCV, muitas delas estão listadas em [3], um exemplo clássico é a perfuração de Placas de Circuito Impresso. Numa placa dessas existem diversos furos, de diâmetros diferentes, que são feitos para a soldagem de componentes eletrônicos. Como a troca da ferramenta que efetua as perfurações é um processo lento, deve-se perfurar de uma vez todos os furos de mesmo diâmetro, otimizando assim o procedimento. As perfurações de furos de mesmo diâmetro podem ser vistas como um PCV. O objetivo é percorrer o melhor caminho possível, economizando tempo e

aumentando a produtividade.

Neste trabalho, apresenta-se como aplicação para o PCV a elaboração de Roteiros Turísticos. Dados n pontos turísticos a serem visitados, este exemplo tem como objetivo traçar uma rota passando por todos estes n destinos propostos e retornar para o ponto inicial no menor tempo possível. Como o intuito é apenas exemplificar, fez-se uso de uma instância menor com apenas 6 pontos turísticos a serem visitados, tendo como ponto de partida e chegada o hotel em que os turistas estão hospedados. Este exemplo encaixa-se perfeitamente no projeto de uma agência de turismo que deseja elaborar um roteiro para seus clientes, proporcionando o máximo aproveitamento no passeio e o mínimo de tempo no translado.

Tendo em vista que o PCV é NP-difícil, existe método analítico para solucioná-lo, por exemplo o Método *Branch-and-Bound* [2], que demanda um grande custo computacional para chegar à solução e, na maioria das vezes não consegue alcançá-la. Porém existem vários métodos que aproximam soluções de forma eficiente. Entre eles a Relaxação Lagrangeana [4], que retira algumas restrições do PCV adicionando-as à função objetivo com um fator de penalização, chamado Multiplicador de Lagrange. Esse método retorna bons limitantes inferiores, porém Held e Karp [5] propuseram uma forma ainda melhor, de aprimorar esta relaxação retornando limitantes melhores [6]. A eles dá-se o nome de Limitante de Held-Karp. Para os limitantes superiores, existem os métodos denominados Heurísticas que nos casos de minimização, como o PCV, retornam bons resultados. Muitos estudiosos fazem uso dos limitantes calculados nessas duas técnicas para aprimorar ainda mais a solução final que será dada para o PCV. Na maioria dos casos, a solução exata não é encontrada, porém combinando essas duas técnicas consegue-se determinar limitantes ainda melhores para o PCV, isto é, bem próximos da solução ideal.

Este trabalho está organizado da seguinte forma. No Capítulo 1 é apresentada uma parte da Teoria de Grafos, que será importante ao longo do texto. Em seguida, neste mesmo capítulo, são propostos dois diferentes algoritmos que determinam uma Árvore Geradora Mínima. No Capítulo 2 são introduzidas algumas notações de Progra-

mação Inteira usadas no texto e, para exemplificar, algumas formulações de problemas importantes são apresentadas. Ainda neste capítulo será definido o que são limitantes para um PPI, como eles podem ser determinados e como sua qualidade pode ser avaliada. A formulação do PCV é descrita com detalhes no Capítulo 3. No Capítulo 4, a Relaxação Lagrangeana será exposta, bem como o Método Subgradiente, segundo [4], e exemplificada para o Problema de Cobertura de Conjuntos. O Capítulo 5 define o Limitante de Held e Karp explicando sua proposta para melhorar ainda mais o limitante inferior dado apenas pela Relaxação Lagrangeana e aplicando-o ao PCV. Além disso, neste mesmo capítulo, é apresentado o exemplo de Roteiros Turísticos e são expostas as soluções obtidas para ele com o auxílio de um código em Python [7]. Finalmente, o último capítulo apresenta algumas conclusões e sugestões para trabalhos futuros.

Capítulo 1

Árvore Geradora

A fim de iniciar o estudo sobre árvore geradora mínima, algumas definições básicas são apresentadas a seguir.

Definição 1.0.1. Um *grafo* $G = (V, E)$ é um conjunto não-vazio V, cujos elementos são chamados vértices, e um conjunto E de arestas. Uma aresta é um par não-ordenado $\{v_i, v_j\}$ onde v_i e v_j são elementos de V.

Dados dois vértices v_i e v_j, a $a_k = \{v_i, v_j\}$ é dita incidente a ambos os vértices v_i e v_j. Duas arestas não-paralelas que são incidentes a um mesmo vértice são ditas adjacentes. Dois vértices ligados por uma mesma aresta são ditos vizinhos ou adjacentes. O número de arestas incidentes a um vértice v_i é chamado o grau deste vértice.

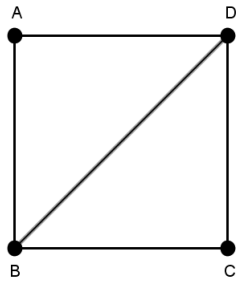

Figura 1.1: Representação de um grafo que possui conjunto de vértices $V = \{A, B, C, D\}$ e o conjunto de arestas $E = \{\{A, B\}, \{A, D\}, \{B, C\}, \{B, D\}\{C, D\}\}$.

Observe que, na figura (1.1), as arestas $\{A, B\}$ e $\{A, D\}$ são adjacentes, pois

incidem no mesmo vértice, a saber o vértice A. Os vértices A e B são adjacentes, pois são ligados pela aresta AB. O grau do vértice B (denotado por $grau(B)$) é igual a 3, pois três arestas são incidentes a ele, são elas: $\{A, B\}$, $\{B, D\}$, e $\{B, C\}$.

Definição 1.0.2. Um *subgrafo* de $G = (V, E)$ é um grafo $H = (V', E')$ com as seguintes propriedades:

- $V' \subseteq V$ e
- $E' \subseteq E$
- E' conecta os elementos de V'

Definição 1.0.3. Um $passeio$ em um grafo é qualquer sequência finita de vértices e arestas.

Definição 1.0.4. Um $caminho$ em um grafo $G = (V, E)$ é uma sequência de vértices $v_1, v_2, ..., v_n$, tal que $\{v_i, v_{i+1}\}$ é um elemento de E, para $1 \leq i \leq (n-1)$ e $v_i \neq v_j$, $\forall i \neq j$.

Na figura (1.1), um exemplo de caminho é ABCD.

Definição 1.0.5. Um caminho onde o primeiro vértice da sequência é igual ao último é um $ciclo$.

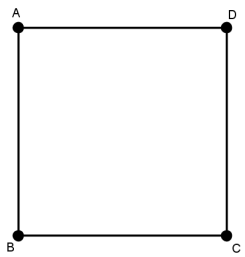

Figura 1.2: Um ciclo

Dessa forma, um ciclo consiste em um caminho onde há repetição de apenas um vértice. Na figura (1.2), uma possível sequência de vértices para o ciclo seria (A, B, C, D, A).

Definição 1.0.6. Um grafo $G = (V, E)$ é *conexo* quando existe um caminho entre cada par de elementos de V.

A figura (1.1) é um exemplo de grafo conexo. Na figura (1.3) temos um exemplo de grafo desconexo.

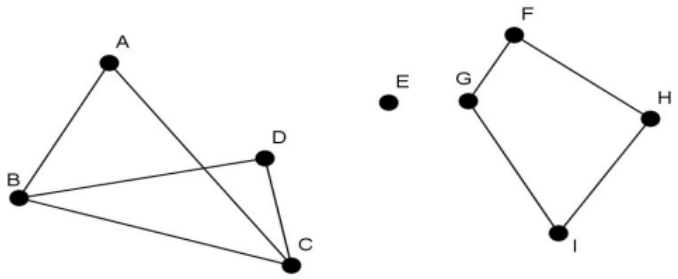

Figura 1.3: Grafo desconexo

Definição 1.0.7. Dado um grafo G, um *ciclo Hamiltoniano* para G é um caminho simples que inclui todos os vértices de G, ou seja, uma sequência de vértices adjacentes e arestas distintas tal que cada vértice de G aparece exatamente uma única vez, exceto o vértice final que é igual ao inicial.

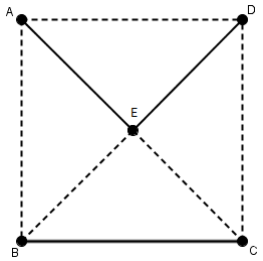

Figura 1.4: Um ciclo Hamiltoniano. Basta, por exemplo, seguir o caminho ABECDA.

Os ciclos hamiltonianos são soluções de muitos problemas complexos da atualidade, inclusive para o Problema do Caixeiro Viajante, que será abordado neste texto. Seu nome é devido a Willian Rowan Hamilton que, em 1857, propôs um jogo chamado Around the World. O jogo consistia em um dodecaedro e cada um de seus vértices representava uma cidade importante na época. O objetivo do jogo proposto por Hamilton era encontrar uma rota através dos vértices do dodecaedro que iniciasse e terminasse

numa mesma cidade sem nunca repetir uma mesma visita. Em homenagem ao seu criador, a solução para este jogo passou a ser denominada um ciclo hamiltoniano ([8], [9]).

Definição 1.0.8. Um *grafo direcionado* é um par $G = (V, E)$ que segue a definição inicial de grafos e possui arestas direcionadas (ou arcos).

Um grafo direcionado difere de um não-direcionado comum, este último é definido em termos de pares não ordenados de vértices, que são normalmente chamados arestas.

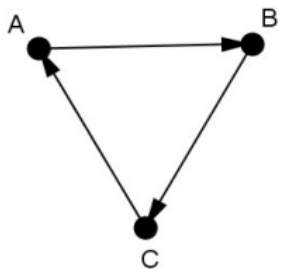

Figura 1.5: Um grafo direcionado.

Definição 1.0.9. Uma *árvore* é um grafo não-direcionado, conexo e sem ciclos.

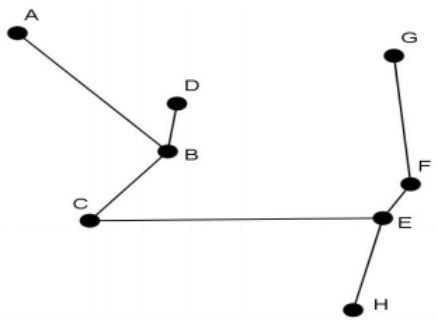

Figura 1.6: Um exemplo de árvore.

Definição 1.0.10. Uma *árvore geradora* de um grafo G é um subgrafo de G que é uma árvore e contém todos os vértice de G .

Observe que, se o grafo for conexo, então ele possui pelo menos um subgrafo que será uma árvore geradora. E ainda, duas árvores geradoras de um mesmo grafo possuem o mesmo número de arestas [8].

Para modelar problemas que envolvem distâncias ou pesos, muitas vezes é conveniente identificar cada aresta do grafo estudado com um valor. Um grafo onde cada aresta possui um peso (representado por um número real) é chamado grafo valorado. A soma dos pesos de todas as arestas é o peso total do grafo.

Se G é um grafo valorado e a_k é uma aresta de G, então definimos:

- $w(a_k)$ é o peso da aresta a_k, e
- $w(G)$ é o peso total do grafo G.

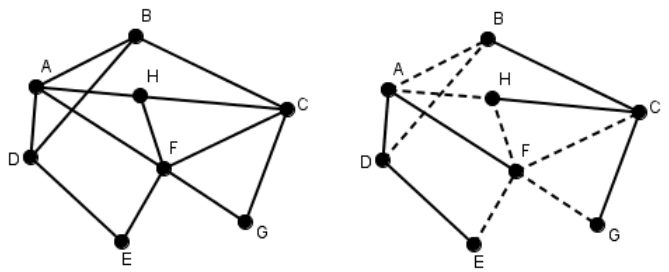

Figura 1.7: Para o grafo da esquerda, é dada uma possível árvore geradora nas linhas pontilhadas do grafo à direita.

Definição 1.0.11. Uma *Árvore Geradora Mínima* (AGM) para um grafo com peso é uma árvore geradora que tem o menor peso total possível dentre todas as possíveis árvores geradoras do grafo.

O problema de encontrar a AGM de um grafo possui solução somente para grafos conexos. Observe também que se todas as arestas do grafo considerado possuírem o mesmo peso, então toda árvore geradora terá custo mínimo. Os algoritmos mais usados para solucionar este problema são os de Prim e Kruskal. Estes algoritmos possuem caráter guloso e atuam retornando soluções exatas de forma rápida e eficiente.

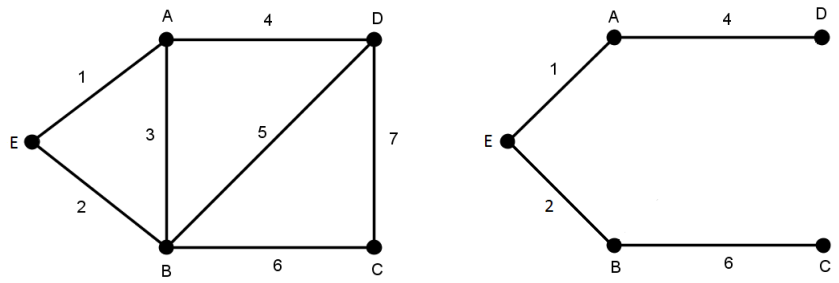

Figura 1.8: O grafo da direita representa a AGM do grafo representado a esquerda.

O Algoritmo de Prim (publicado por Robert Prim, em [10], no ano de 1957) constrói a AGM a partir de um vértice escolhido aleatoriamente, este vértice é o primeiro da árvore. A cada iteração o algoritmo seleciona uma aresta de menor custo que conecta um vértice da árvore a outro que ainda não esteja na árvore. O algoritmo termina quando todos os vértices fazem parte da árvore ou quando não se pode encontrar uma aresta que satisfaça a condição. Este algoritmo foi descrito em [9] e está apresentado neste texto em anexo. A figura (1.9), mostra a aplicação do Algoritmo de Prim ao grafo apresentado na figura (1.8). Sem perda de generalidade, o vértice A foi escolhido para inicializar o processo.

O Algoritmo de Kruskal (segundo [9], foi publicado em 1956, por Joseph Kruskal), inicialmente ordena as arestas do grafo de forma crescente e, a cada iteração, escolhe a aresta de menor custo. As iterações ocorrem até que existam arestas incidentes a todos os vértices do grafo e com a condição de que não haja ciclos no grafo resultante. Este algoritmo, assim como em [9], pode ser formalizado como no Anexo 1. A tabela (1.2) apresenta a evolução do algoritmo de Kruskal para o grafo da figura (1.8). Inicialmente as arestas do grafo são organizadas em ordem crescente, de acordo com a tabela (1.1).

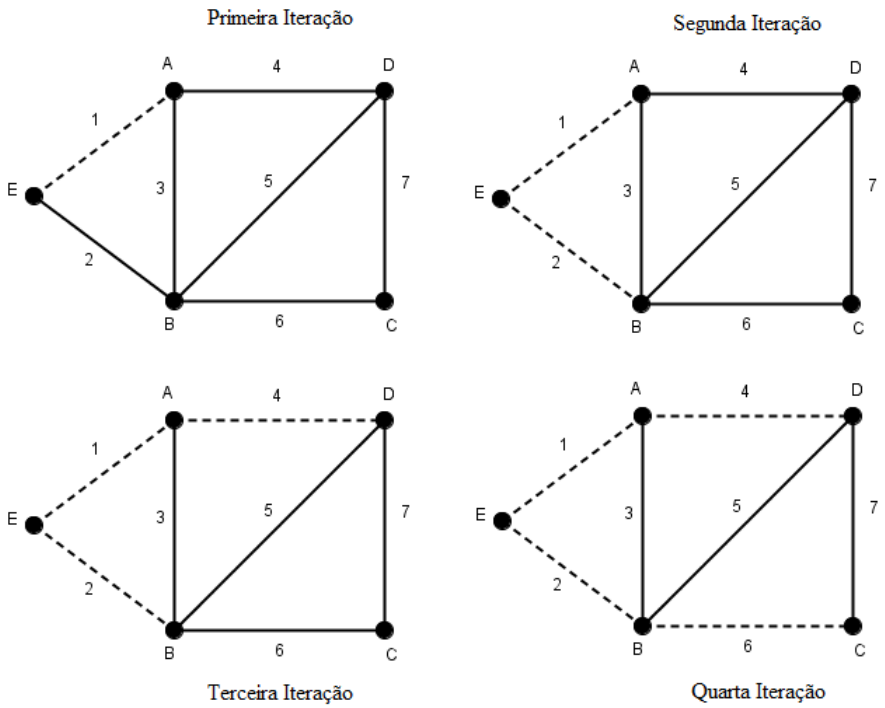

Figura 1.9: Exemplo da evolução do algoritmo de Prim

Vetor H	h_1	h_2	h_3	h_4	h_5	h_6	h_7
Aresta	$\{A, E\}$	$\{B, E\}$	$\{A, B\}$	$\{A, D\}$	$\{B, D\}$	$\{B, C\}$	$\{C, D\}$
Custo	1	2	3	4	5	6	7

Tabela 1.1: Arestas do Grafo da figura 1.8 em ordem crescente

Iterações		
$i = 1$	$T \longleftarrow h_1$	$T = \{\{A, E\}\}$
$i = 2$	$T \longleftarrow T \cup h_2$	$T = \{\{A, E\}, \{B, E\}\}$
$i = 3$	$\{A, B\} \notin T$, pois forma ciclo	$T = \{\{A, E\}, \{B, E\}\}$
$i = 4$	$T \longleftarrow T \cup h_4$	$T = \{\{A, E\}, \{B, E\}, \{A, D\}\}$
$i = 5$	$\{B, D\} \notin T$, pois forma ciclo	$T = \{\{A, E\}, \{B, E\}, \{A, D\}\}$
$i = 6$	$T \longleftarrow T \cup h_6$	$T = \{\{A, E\}, \{B, E\}, \{A, D\}, \{B, C\}\}$
$i = 7 = n$	$\{C, D\} \notin T$, pois forma ciclo	$T = \{\{A, E\}, \{B, E\}, \{A, D\}, \{B, C\}\}$

Tabela 1.2: Iterações do algoritmo de Kruskal

Capítulo 2

Programação Inteira

Problemas importantes podem ser modelados matematicamente usando a teoria de Programação Inteira. Como exemplo de problemas inteiros, podemos citar os Problemas de Corte [11], Telecomunicações [12] e Planejamentos de Produção [2]. Problemas de Corte estão relacionados, por exemplo, à linhas de produção de roupas que desejam recortar tecidos seguindo regras precisas, satisfazendo a demanda e minimizando desperdícios. Um problema na área de telecomunicações pode ser minimizar o custo de planejamento de novas redes considerando as diferentes localidades e os custos de instalações. Por fim, algumas empresas fazem uso de Programação Inteira para otimizar seus Planejamentos de Produção. Nesses problemas, há uma quantidade mínima de produção e visa-se maximizar os lucros. Em cada um desses casos, pode-se determinar o objetivo comum que é minimizar/maximizar uma certa função custo/lucro obedecendo a algumas restrições. Entre elas está a restrição de que as variáveis do problema devem ser positivas.

Um Problema de Programção Linear (PPL) possui a seguinte forma:

$$\max c^T x$$

$$Ax \leq b$$

$$x \geq 0$$

onde $A \in \mathbb{R}^{mxn}$, $x \in \mathbb{R}^n$ é o vetor que representa as variáveis de decisão do problema, $c \in \mathbb{R}^n$ e $b \in \mathbb{R}^m$.

É possível obter outros problemas a partir do anterior. Suponha que uma empresa deseja otimizar a quantidade de itens colocados dentro de uma caixa. Ou seja, a quantidade de itens dentro desse recipiente deve ser máxima, obedecendo a restrições impostas pela empresa. Observe que não existe a possibilidade de colocar metade de um item dentro da caixa, dessa forma a solução do problema deve ser um número inteiro e positivo (pois não podem haver, por exemplo, -10 itens dentro da caixa). Para este exemplo, a formulação do PPL anterior não funcionaria, visto que dentre as possíveis soluções algumas poderiam ser fracionárias. Isto quer dizer que alguns problemas somente podem admitir soluções inteiras e positivas. Para modelar esses problemas de forma adequada, o PPL precisa ser reformulado recebendo, então, uma restrição mais exigente do que a de não-negatividade ($x \geq 0$). Essa nova restrição recebe o nome de Restrição de Integralidade. O problema elaborado dessa forma é chamado Problema de Programação Inteira (PPI).

$$\max c^T x$$
$$Ax \leq b$$
$$x \geq 0 \; e \; x \in \mathbb{Z}^n$$

Porém, se apenas algumas variáveis possuem a Restrição de Integralidade, tem-se um Problema de Programação Inteira Mista (PPIM).

$$\max c^T x + h^T y$$
$$Ax + Gy \leq b$$
$$x \geq 0 \; e \; x \in \mathbb{Z}^n, \; y \geq 0$$

onde $G \in \mathbb{R}^{mxp}$, $h \in \mathbb{R}^p$, $y \in \mathbb{R}^p$.

Assumindo que todas as variáveis do problema são binárias, define-se o Problema de Programação Binária (PPB):

$$\max c^T x$$
$$Ax \leq b$$
$$x \in \{0, 1\}^n$$

Se o problema consiste em encontrar a melhor combinação (S) dentre um conjunto de variáveis (Γ) para minimizar (ou maximizar) uma função custo ($c^T x$),

trata-se de um Problema de Otimização Combinatória (POC):

$$\min \{\sum_{j \in S} c_j : S \in \Gamma,\ S \subseteq N\}$$

Um exemplo de POC é o Problema do Caixeiro Viajante que será abordado neste trabalho.

2.1 Exemplos: Formulações de alguns PPI importantes

Nesta sessão serão apresentados dois exemplos de Problemas de Programação Inteira muito conhecidos e que serão utilizados posteriormente.

2.1.1 Problema de Atribuição - $Assignment$

Suponha que existam n pessoas disponíveis para cumprir n tarefas. Para que a pessoa i cumpra a tarefa j, existe um custo c_{ij} pré-definido. Soma-se a isto o fato de que algumas pessoas são mais qualificadas do que outras para desenvolver uma tarefa. O Problema de Atribuição consiste em designar uma única tarefa a cada pessoa de modo que o custo operacional seja mínimo e pode ser formulado, segundo [9], como:

$$min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, ..., n \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, ..., n \tag{2.3}$$

$$x_{ij} \in 0, 1 \qquad i,\ j\ = 1,\ ...,\ n \tag{2.4}$$

As restrições (2.2) e (2.3) indicam que cada pessoa realiza um trabalho e que cada trabalho é realizado por apenas uma pessoa, respectivamente. Se a pessoa i foi designada a cumpir a função j, então $x_{ij} = 1$, caso contrário $x_{ij} = 0$, a condição (2.4) retrata este fato. Por fim, a função objetivo (2.1) minimiza o custo da atribuição ij.

2.1.2 Problema de Cobertura de Conjuntos

Consiste em cobrir as linhas de uma matriz binária $(a_{ij})_{mxn}$ por um subconjunto das colunas, a um custo mínimo.

Considere as seguintes definições:

x_j = 1, se a coluna j está na solução

x_j = 0, caso contrário

Sendo assim, uma formulação para o PCC consiste em:

$$min \sum_{j=1}^{n} c_j x_j \tag{2.5}$$

$$sujeito\ a \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad i = 1, ..., m \tag{2.6}$$

$$x_j \in \{0, 1\} \quad j = 1, ..., n \tag{2.7}$$

A primeira restrição assegura que cada linha é coberta por, pelo menos, uma coluna e a segunda restrição é a Restrição de Integralidade.

Uma aplicação prática para este problema pode ser vista em [2]. Dado um certo número de bairros, faz-se necessário decidir onde instalar uma unidade do Corpo de Bombeiros. Para cada unidade existe um custo de instalação e uma quantidade de bairros que podem ser atendidos. Uma região é coberta se é garantido que o atendimento chegue a ela em um determinado tempo, por exemplo 8 minutos. Seja $M = \{1, 2, ..., m\}$ o conjunto de bairros e $N = \{1, 2, ..., n\}$ o conjunto de possíveis unidades dos bombeiros. Seja $S_j \subseteq M$ o conjunto de bairros que podem ser atendidos pela unidade $j \in N$ e c_j o custo de instalação dessa unidade.

Observe que $x_j = 1$ se a unidade j é escolhida, caso contrário $x_j = 0$. Dessa forma, $a_{ij} = 1$ se $i \in S_j$, ou seja se o bairro i é atendido pela unidade j, se isso não acontece $a_{ij} = 0$. Como consequência disso, a restrição (2.6) informa que cada bairro é atendido por, no mínimo, uma unidade do Corpo de Bombeiros.

2.2 Notações e Resultados Básicos

Definição 2.2.1. Um subconjunto de $\mathbb{R}^n$ descrito por um conjunto finito de restrições $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ é uma *região poliedral*.

Portanto, uma região poliedral é a interseção de um número finito de semiespaços [2].

Definição 2.2.2. Uma região poliedral $P \subseteq \mathbb{R}^{n+p}$ é uma formulação para o conjunto $\mathbb{Z}^n \times \mathbb{R}^p$ se, e somente se, X =$P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$, sendo X um conjunto finito e $X \subseteq \mathbb{Z}^n$.

Definição 2.2.3. Dado um conjunto $X \subseteq \mathbb{R}^n$, a envoltória convexa de X, denotada $conv(X)$, é definida como:

$$conv(X) = \{x : x = \sum_{i=1}^{t} \lambda_i x^i, \ \sum_{i=1}^{t} \lambda_i = 1, \ \lambda_i \leq 0 \, para \, i = 1, ..., t\}$$

para todo subconjunto finito $\{x^1, ..., x^t\}$ de X.

Observe que a envoltória convexa de um conjunto $X \subseteq \mathbb{R}^n$ é a menor região poliedral que o contém.

Proposição 2.2.4. $conv(X)$ *é uma região poliedral.*

Proposição 2.2.5. *Os vértices da envoltória convexa pertencem a X.*

As proposições (2.2.4) e (2.2.5) estão demonstradas em [2]. Com esses resultados, pode-se escrever o problema inteiro $\{\max c^T x : x \in X\}$ como o problema linear equivalente $\{\max c^T x : x \in conv(X)\}$. Esta redução para um problema linear também é válida para conjuntos inteiros limitados $X = \{x : Ax \leq b, x \geq 0 \; e \; x \in \mathbb{Z}\}$ e conjuntos inteiros mistos $X = \{x : Ax + Gy \leq b, x \geq 0, y \geq 0 \; e \; x, \; y \in \mathbb{Z}\}$ com A, G, b racionais. No entanto, se X é limitado ou não, essa é apenas uma formulação geral, porque na maioria dos casos, existe um número exponencial de desigualdades necessário para descrever $conv(X)$ e não há nenhuma formulação simples para elas.

Definição 2.2.6. Dado um conjunto $X \subseteq \mathbb{R}^n$, e duas formulações P_1 e P_2 para X, P_1 é uma formulação melhor do que P_2 se $P_1 \subset P_2$.

Definição 2.2.7. Dada uma região poliedral $Q \subseteq (\mathbb{R}^n \times \mathbb{R}^p)$ a projeção de Q no subespaço $\mathbb{R}^n$, denotada por $proj_x Q$, é definida como:

$$proj_x Q = \{x \in \mathbb{R}^n : (x, w) \in Q \; para \; algum \; w \in \mathbb{R}^p\}$$

Assim $proj_x Q \subset \mathbb{R}^n$ é a formulação obtida a partir de Q no espaço $\mathbb{R}^n$ de variáveis originais x, isso permite comparar Q com outras formulações $P \subseteq \mathbb{R}^n$.

2.3 Otimalidade e Limitantes

Seja $z = \{\text{max } c^T x : x \in X \subseteq \mathbb{Z}^n\}$ um PPI ou POC. Buscaremos fundamentação teórica para provar que x^* é a solução ótima deste problema. Ou seja, nesta seção temos o intuito de determinar condições de otimalidade que fornecerão critérios de parada a um Algoritmo de Programação Inteira. Neste contexto, buscaremos encontrar valores $\overline{z}$ e $\underline{z}$, os quais chamaremos limitantes superiores e inferiores, respectivamente, e tais que $\underline{z} \leq z$ e $\overline{z} \geq z$, assim teremos $\underline{z} \leqslant z \leqslant \overline{z}$. Na prática, o algoritmo que soluciona um PPI ou POC busca encontrar uma sequência crescente de limitantes inferiores $\underline{z}_1 > \underline{z}_2 > ... > \underline{z}_s \geq z$ e uma sequência decrescente de limitantes superiores $\overline{z}_1 < \overline{z}_2 < ... < \overline{z}_t \leq z$ e parar quando $\overline{z}^t_{-}\underline{z}_s \leq \epsilon$, onde $\epsilon > 0$ é um número muito pequeno convenientemente escolhido. Ou seja, esse algoritmo visa determinar limitantes superiores tão pequenos quanto possível e limitantes inferiores tão grandes quanto possível.

Um limitante primal para um dado problema de programação é qualquer solução viável deste problema. Como o Problema do Cixeiro Viajante é um problema de minimização, todas as soluções viáveis $x^* \in X$ fornecem um limitante superior $\overline{z} = c(x^*) \leq z$, também chamado limitante primal. Para certos PPI, encontrar soluções viáveis é uma tarefa fácil, a grande dificuldade está em determinar boas soluções. Para o PCV, um passeio viável é aquele em que o vendedor se desloca de uma cidade até a outra, passando por todas as cidades apenas uma vez e voltando ao ponto de partida. Isso é o suficiente para avaliar o custo do percurso e para obter um limitante primal $\overline{z}$.

Determinar o limitante inferior, também chamado limitante dual, para o PCV apresenta um desafio diferente. A forma mais interessante, e a que será abordada neste trabalho, de calcular limitantes duais é por relaxação. Esta técnica consiste em substituir o problema "difícil" por um problema mais fácil de ser resolvido e que informe um bom limitante (próximo da solução ideal) para o problema em questão, embora, na maioria das vezes, não retorne a solução exata. A fim de que o problema relaxado tenha esta propriedade, é apresentada a seguinte definição:

Definição 2.3.1. O problema $z^R = min\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ é a relaxação de $z = min\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ se:

i) X $\subseteq$ T

ii) f(x) $\leq$ c(x), $\forall$x $\in$ X.

De forma equivalente, essas condições informam que existem duas possibilidades para que a solução da relaxação do PCV seja um limitante inferior para a solução ideal, são elas: expandir o conjunto de soluções viáveis a fim de otimizar através de um conjunto maior ou substituir a função objetivo z por uma outra função z^R que tenha o mesmo valor (ou um valor menor) em todos os pontos ($\forall x \in X \subseteq \mathbb{R}^n$).

Proposição 2.3.2. *Seja* $z^R = min\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ *uma relaxação para* $z = min\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ *, então* $z^R \leq z$.

Demonstração: Seja x^* a solução ótima de z, $x^* \in X \subseteq T$ e $z = c(x^*) \geq f(x^*)$. Como $x^* \in T$, $f(x^*)$ é um limitante inferior para z^R e então $z \geq f(x^*) \geq z^R$. ■

Entre as relaxações mais utilizadas estão a Relaxação Linear, a Relaxação Combinatória e a Relaxação Lagrangeana. Para tratar desta última é dedicado um capítulo inteiro neste trabalho. Nesta seção aborda-se de forma simplificada os outros dois métodos de relaxação.

Definição 2.3.3. Seja $z = min\{c^T x : x \in P \cap \mathbb{Z}^n\}$ um PPI com a seguinte formulação $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$. A relaxação linear de z será $z^{LP} = min\{c^T x : x \in P\}$.

Para obter a relaxação linear de um problema basta desconsiderar a restrição de integralidade da variável inteira, ou seja, para z^{LP}, a variável de decisão deixa de pertencer a $\mathbb{Z}^n$ e passa a pertencer a $\mathbb{R}^n_+$. Como $P \cap \mathbb{Z}^n \subseteq P$ e a função objetivo continua inalterada, z^{LP} é uma relaxação para z. O problema obtido a partir dessa retirada é um PPL. O PPL pode ser resolvido usando o Algoritmo Simplex [9]. A solução do problema relaxado é sempre menor do que solução do problema original, pois estamos tratando de um problema de minimização, isto se deve ao fato que z^{LP} é composto de todas as soluções reais do problema, inclusive as inteiras, logo é formado por um conjunto de soluções viáveis mais abrangente.

Proposição 2.3.4. *[2] Suponha que P_1 e P_2 são duas formulações para o PPI $z = min\{c^T x : x \in X \subseteq \mathbb{Z}^n\}$. Suponha também que $P_1 \subset P_2$, ou seja, P_1 é uma formulação melhor do que P_2 para z. Se $z_i^{LP} = min\{c^T x : x \in P_i\}$ com $i = 1, 2$ são os valores associados às respectivas relaxações lineares de z, então $z_1^{LP} \geq z_2^{LP}$, para todo c.*

Proposição 2.3.5. *[2] (i) se a relaxação z^R é um problema inviável, então o problema original z também é inviável.*

(ii) Seja x^* a solução ótima da relaxação z^R. Se $x^* \in X$ e $f(x^*) = c(x^*)$, então x^* é a solução ótima do PPI, z.

Essa última proposição reforça a ideia de que relaxações não informam somente limitantes duais, elas algumas vezes permitem provar otimalidade. Em muitos casos, no entanto, resolver a relaxação linear de um problema é muito difícil, normalmente porque o problema envolve um número grande de variáveis e/ou restrições. Faz-se necessário a utilização de técnicas alternativas para a geração de limitantes inferiores.

Sempre que a relaxação é um problema de otimização combinatória, tem-se uma Relaxação Combinatória. Por exemplo, para o PCV a eliminação das restrições (3.3), resulta em um problema de Atribuição. Na prática, a retirada dessas restrições indica que passam admitidos como solução todos os ciclos possíveis que envolvam

todos os vértices do grafo. Ou seja, o problema de Atribuição é uma relaxação combinatória para o PCV, visto que nesse conjunto de soluções viáveis está a solução do PCV. Seja z^{PCV} um problema do caixeiro viajante e z^{ATR} um problema de atribuição, então:

$$z^{PCV} = min_{T \subseteq A} \{ \sum_{(i,j) \in T} c_{ij} : \text{ T forma um caminho} \} \geq \tag{2.8}$$

$$z^{ATR} = min_{T \subseteq A} \{ \sum_{(i,j) \in T} c_{ij} : \text{ T forma uma atribuição} \} \tag{2.9}$$

Existem algoritmos que resolvem z^{ATR} de forma exata e em tempo polinomial, ou seja, a relaxação gerou, neste caso, um problema mais fácil de ser resolvido se comparado com o problema original.

Capítulo 3

Problema do Caixeiro Viajante (PCV)

Neste problema, um vendedor deve visitar n cidades exatamente uma vez e retornar ao seu ponto de partida. O custo necessário para ir da cidade i até a cidade j é c_{ij}. Esse custo pode ser o tempo de viagem ou a distância entre uma cidade e outra. O objetivo é localizar a ordem (rota) em que esse vendedor deve fazer a viagem de modo a minimizar o somatório de custos c_{ij} computados em função da rota escolhida pelo vendedor. Analisando este problema como um grafo, cada cidade é vista como um vértice e o caminho que liga diretamente a cidade i à cidade j corresponde a uma aresta. Portanto, o caminho que desejamos encontrar, o qual apresenta solução ideal para o PCV, é o ciclo hamiltoniano de menor custo. Neste trabalho, trabalhamos com o PCV Simétrico em qu os custos de ida (da cidade i até a cidade j) e volta (da cidade j até a cidade i) são iguais, ou seja, $c_{ij} = c_{ji}$. O caso em que esses custos são considerados diferentes $c_{ij} \neq c_{ji}$ é denominado PCV Assimétrico.

Após entender o PCV, é fundamental observar sua importância para problemas atuais. Esse problema é muito relevante devido à sua grande aplicação prática, enorme relação com outros problemas e à dificuldade de determinar a solução exata. Entre as aplicações práticas do PCV, pode-se destacar o Problema de Roteamento de Veículos [13] e de Navios ([14], [15]), Cortes em Chapas de Aço e Vidro [16] e Planejamento da Produção [17] .

É importante observar que este é um problema combinatório no sentido que

uma solução viável é um subconjunto de um conjunto finito de todos o ciclos hamiltonianos possíveis para um grafo. Assim, em princípio, suponha que esse problema possa ser resolvido pela simples enumeração dos ciclos hamiltonianos. Dessa forma, partindo da primeira cidade, o vendedor possui $n-1$ escolhas para a segunda cidade; para a terceira cidade, o vendedor possui $n-2$ escolhas, e assim por diante. Assim existem $(n-1)!$ possíveis soluções. Observe que para 3 ou 4 cidades, a enumeração não é um método tão difícil. Porém se o número de cidades considerado for razoavelmente grande, a função que enumera as possíveis soluções cresce exponencialmente. Isso torna a enumeração das rotas possíveis uma alternativa inviável. Esse fato explica a dificuldade de determinar soluções exatas para o PCV. Uma solução inteligente consiste em desenvolver algoritmos que encontrem soluções heurísticas para tais problemas, ou seja, por métodos de aproximações sucessivas da solução.

3.1 Uma formulação para o PCV

Neste texto, o PCV será formulado como um PPB sobre o grafo não direcionado $G = (V, E)$. As variáveis do problema são: x_{ij} a variável binária que determinará se o vendedor foi ou não da cidade i até a cidade j. Assim, $x_{ij} = 1$, se o vendedor escolhe ir diretamente da cidade i até a cidade j, ou seja, a aresta $\{i, j\} \in E$ foi escolhida para compor o ciclo hamiltoniano, e $x_{ij} = 0$, se essa ligação não for escolhida. Em seguida, as restrições do problema serão definidas; para isso, observe que a cada vértice do grafo que compõe a solução desse problema estão associadas apenas duas arestas:

$$\sum_{i:\, i \neq j} x_{ij} = 1, \; onde \; j = 1, 2, ..., n \tag{3.1}$$

$$\sum_{j:\, j \neq i} x_{ij} = 1, \; onde \; i = 1, 2, ..., n \tag{3.2}$$

Estas restrições, semelhantes às do Problema de Atribuição (sessão 2.1.1), modelam o fato que o vendedor pode deixar a cidade apenas uma vez, da mesma forma

ele só pode chegar a determinada cidade apenas uma vez. Para ilustrar, considere o grafo da figura (1.8). Enumerando os vértices da seguinte forma: $A = 1, B = 2, C = 3, D = 4, E = 5$, temos que $i, j \in V = \{1, 2, 3, 4, 5\}$.

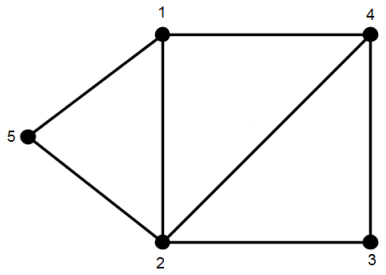

Figura 3.1: Figura (1.8) modificada.

Sendo assim, se $j = 1$ podemos reescrever a restrição (3.1) como se segue:

$$\sum_{i \neq j} x_{i1} = 1, \ onde \ i = 2, ..., 5$$

ou seja,

$$x_{21} + x_{41} + x_{51} = 1$$

E se $i = 1$, da restrição (3.2) segue que:

$$\sum_{i \neq j} x_{1j} = 1, \ onde \ j = 2, ..., 5$$

ou seja,

$$x_{12} + x_{14} + x_{15} = 1$$

Observe que a aresta $\{i, j\} = \{1, 1\}$ não existe nesta formulação. Sendo assim, se o vendedor faz o caminho $\{i, j\} = \{1, 2\}$ ele só poderá fazer um caminho de volta, a saber $\{i, j\} = \{1, 4\}$ ou $\{i, j\} = \{1, 5\}$. As restrições (3.1) e (3.2) excluem a possibilidade de haver ambos os caminhos na solução.

Note também que se o problema tiver apenas estas restrições, caminhos desconexos poderiam ser admitidos como solução para o PCV. Ou seja, o caminho per-

corrido pelo vendedor poderia não formar um único ciclo. Para desconsiderar soluções desse tipo, Restrições de Eliminação de Sub-Rotas serão incluídas na formulação. Para entender melhor essas restrições, define-se S, um subconjunto de vértices do grafo G, em que $|S|$ representa o número de vértices desse subconjunto. Dessa forma, uma possível solução para o Problema de Atribuição seriam os subciclos formados por $\{(1,5),(5,2),(2,1)\}$ e $\{(3,4),(4,3)\}$, representados na figura (3.2):

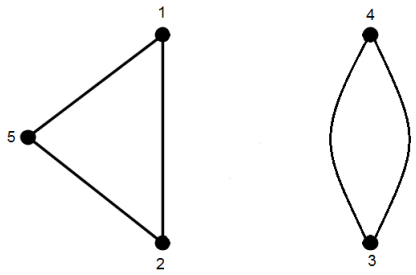

Figura 3.2: Exemplo de uma solução inviável para o PCV.

Esta, apesar de ser uma solução viável para o Problema de Atribuição, é inviável para o PCV, pois não foi formado um ciclo hamiltoniano entre todos os vértices. Para eliminar soluções desse tipo, as restrições abaixo são inseridas na formulação do PCV:

$$\sum_{i \in S,\ j \in S} x_{ij} \leq |S| - 1 \text{ para } S \subset V,\ 2 \leq |S| \leq n-1 \tag{3.3}$$

Escrevendo as Restrições de Eliminação de Sub-Rotas para o exemplo comentado anteriormente, pode-se observar que são obtidas as seguintes inequações:

$$x_{15} + x_{51} + x_{52} + x_{25} + x_{12} + x_{21} \leq |S_1| - 1 = 3 - 1 = 2$$

e

$$x_{34} + x_{43} \leq |S_2| - 1 = 2 - 1 = 1$$

onde $S_1 = \{1,2,5\} \subset V$ e $S_2 = \{3,4\} \subset V$.

Se $x_{15} = x_{52} = x_{21} = 1$, o somatório do lado esquerdo seria igual a $3 > 2$. Para $x_{34} = x_{43} = 1$ e $x_{34} + x_{43} = 2 > 1$, esse exemplo retorna desigualdades inválidas

para o PCV. Portanto as restrições (3.3) eliminam a hipótese de formação dos sub-ciclos $\{\{1,5\},\{5,2\},\{2,1\}\}$ e $\{\{3,4\},\{4,3\}\}$ e de qualquer outro sub-ciclo que não seja formado por todos os vértices do problema . Assim, o único subconjunto $S \subseteq V$ que não seria excluído pelas Restrições de Eliminação de Sub-Rota é o proprio $S = V$.

Para cada subciclo $S \subseteq V$ é possível e necessário uma restrição do tipo (3.3). Portanto, identifica-se um número exponencial dessas restrições. Isso justifica a natureza combinatória do PCV.

Por último definiremos a função objetivo. Como nosso intuito é minimizar o custo de viagem, temos:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.4}$$

Portanto, o PCV simétrico possui a seguinte formulação:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j;\ j \neq i} x_{ij} = 1,\ onde\ i = 1, 2, ..., n$$

$$\sum_{i;\ i \neq j} x_{ij} = 1,\ onde\ j = 1, 2, ..., n$$

$$\sum_{i \in S,\ j \in S} x_{ij} \leq |S| - 1\ para\ S \subset V,\ 2 \leq |S| \leq n - 1$$

$$x_{ij} \in \{0, 1\}$$

Capítulo 4

Relaxação Lagrangeana

4.1 Conceitos Básicos

Uma alternativa bastante eficiente para determinar limitantes duais é a Relaxação Lagrangeana. Este método consiste em identificar, dentre as restrições do problema, um conjunto de restrições complicadas, no sentido de que solucioná-lo seria muito mais simples na ausência dessas restrições. A proposta desse método é atribuir Multiplicadores de Lagrange às restrições difíceis e dualizá-las, ou seja, adicioná-las à função objetivo obtendo assim um novo problema, que espera-se resolver mais facilmente se comparado ao problema original.

Observe que um caso extremo de Relaxação Lagrangeana consiste em retirar todas as desigualdades da formulação e dualizá-las na função objetivo. Essa não deixa de ser uma relaxação, porém o limitante dual estará muito longe do ótimo. Com essa solução, um algoritmo do tipo *Branch-and-Bound* ([2], [9]), por exemplo, demoraria a aproximar (ou nem chegaria a encontrar) a solução ideal. Sendo assim, a escolha das restrições que serão dualizadas é feita visando chegar a um problema mais fácil de ser resolvido, e que retorne limitantes duais de qualidade.

Considere o seguinte problema binário geral e suponha que o conjunto de restrições complicadas seja $Ax \geq b$:

$$z = min\ cx$$

$$Ax \geq b \tag{4.1}$$

$$Bx \geq d$$

$$x \in \{0, 1\}$$

A Relaxação Lagrangeana para este problema é:

$$z(u)_{u \geq 0} = min\ cx + u(b - Ax) = (c - uA)x + ub$$

$$Bx \geq d \tag{4.2}$$

$$x \in \{0, 1\}$$

Este problema recebe o nome de Sub-Problema Lagrangeano e, segundo a definição (2.3.1), é uma relaxação para o problema (4.1), ou seja, fornece um limitante inferior para z. Isto pode ser visto considerando um x^* viável e dado um vetor $u \geq 0$, então $b - Ax^* \leqslant 0$, pois $Ax \geqslant b$ para todo x viável. Dessa forma, $u(b - Ax^*) \leq 0$, pois $u \geq 0$. Portanto $z(u) = c * x + u(b - Ax^*) \leq cx^*$, para todo x^* viável. Ou seja, adiciona-se à função objetivo apenas um termo menor ou igual a zero. Este fato é uma consequência da seguinte proposição:

Proposição 4.1.1. *Seja* $z(u) = min\{cx + u(b - Ax) : x \in X\}$. *Então* $z(u) \leq z$, *para todo* $u \geq 0$.

Note que o enunciado desta proposição não inclui as restrições $Bx \geq d$ no problema, contudo o resultado também abrange essas restrições. Isto pode ser visto da segunte forma, na formulação genérica (4.1) a remoção do conjunto de restrições $Ax \geq b$ apenas reduzirá o valor da função objetivo, pois quanto menos restrições, maior é o domínimo, o que dá margem para a função objetivo assumir valores menores.

A escolha do conjunto de restrições $Ax \geq b$, ao invés de $Bx \geq d$, foi feita de forma arbitrária. Em problemas específicos, essa escolha é feita de forma estratégica [4], ou seja, as restrições que tornam o problema mais difícil são escolhidas para serem dualizadas. Isto se deve ao fato de que pode ser mais fácil (em alguns casos) resolver um problema "eliminando" as restrições que o tornam difícil.

Uma outra consideração a ser feita é que ao inserir o vetor u, chamado vetor de Multiplicadores de Lagrange, no problema, o objetivo passa a ser determinar o u que torna $z(u)$ máximo, o que torna $z(u)$ mais próximo da solução z do problema original, visto que trata-se de um problema de minimização. Para que este objetivo seja alcançado da melhor forma, deve-se refinar u até que $z(u)$ esteja o mais próximo possível da solução ótima. Decorre disto o seguinte problema, chamado Dual Lagrangeano:

$$z_{DL} = max\{min\ cx + u(b - Ax), \text{sujeito a} Bx \geq d,\ x \in \{0, 1\}\}$$

$$u \geq 0 \tag{4.3}$$

Exemplo: Problema de Cobertura de Conjunto

Tendo em vista ser um POC fácil de ser formulado, o Problema de Cobertura de Conjuntos (PCC) será usado para exemplificar o método de Relaxação Lagrangeana.

Para iniciar a aplicação do método a este problema, o primeiro passo é escolher o conjunto de restrições a serem dualizadas. Como mencionado anteriormente, essa escolha é feita de forma estratégica. Porém, o PCC possui apenas um conjunto de restrições, a saber as restrições (2.6), portanto esse será o conjunto escolhido para receber os multiplicadores de Lagrange e ser anexado à função objetivo. A relaxação lagrangeana então é dada por:

$$min \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} u_i \left(1 - \sum_{j=1}^{n} a_{ij} x_j\right) \tag{4.4}$$

isto é:

$$min \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m} u_i a_{ij}\right) x_j + \sum_{i=1}^{m} u_i$$

$$x_j \in \{0,1\} \quad j = 1, ..., n$$

Note que para cada i = 1, 2, ..., m existe um multiplicador de Lagrange, u_i multiplicando as restrições (2.6), ou seja, $u \in \mathbb{R}^m_+$.

Definindo:

$$C_j := c_j - \sum_{i=1}^{m} u_i a_{ij} \quad j = 1, ..., n$$

isto é, C_j passa a ser o coeficiente de x_j na função objetivo do Problema Lagrangeano (4.4), que será dado por:

$$min \sum_{j=1}^{n} C_j x_j + \sum_{i=1}^{m} u_i \tag{4.5}$$

Agora a solução x_j para o problema (4.4) pode ser determinada, segundo as novas definições:

$$x_j = 1, \text{ se } C_j \leq 0$$

$$x_j = 0, \text{ caso contrário}$$

com valor de solução z_{RL} dado por (4.5). Sendo assim, z_{RL} é um limitante inferior para a solução ótima do PCC original.

Considere o exemplo numérico (com 3 linhas e 4 colunas) a seguir:

$$c_j = (2, 3, 4, 5) \qquad a_{ij} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$$\text{min } 2x_1 + 3x_2 + 4x_3 + 5x_4$$

$$\text{sujeito a} \quad x_1 + x_3 \geq 1$$

$$x_1 + x_4 \geq 1$$

$$x_2 + x_3 + x_4 \geq 1$$

$$x_j \in \{0,1\} \quad j = 1, 2, 3, 4$$

Neste caso, a coluna 1 tem custo 2 e cobre as linhas 1 e 2; a coluna 2 possui custo 3 e cobre apenas a linha 3; a coluna 3 tem custo 4 e cobre as linhas 1 e 3; a coluna 4, de custo 5, cobre as linhas 2 e 3. Note que este problema possui como solução ótima z = 5, com $x_1 = x_2$ = 1 e $x_3 = x_4$ = 0.

Para definir a Relaxação Lagrangeana do PCC, falta escolher o conjunto de restrições para relaxar e, após isso, atribuir multiplicadores de Lagrange às restrições a fim de dualizá-las na função objetivo. Como estratégia, todas as restrições serão dualizadas. Sendo assim, a cada restrição será atribuído um u_i, i = 1, 2, 3:

$$min\ 2x_1 + 3x_2 + 4x_3 + 5x_4 + u_1(1 - x_1 - x_3) + u_2(1 - x_1 - x_4) + u_3(1 - x_2 - x_3 - x_4)$$

$$s.a. \qquad x_j \in \{0, 1\},\ j = 1, 2, 3, 4$$

isto é, o problema (4.4) fica:

$$(2 - u_1 - u_2)x_1 + (3 - u_3)x_2 + (4 - u_1 - u_3)x_3 + (5 - u_2 - u_3)x_4 + u_1 + u_2 + u_3$$

$$s.a. \qquad x_j \in \{0, 1\},\ j = 1, 2, 3, 4$$

Portanto:

$$C_1 = 2 - u_1 - u_2$$

$$C_2 = 3 - u_3$$

$$C_3 = 4 - u_1 - u_3$$

$$C_4 = 5 - u_2 - u_3$$

Assim, o Problema Lagrangeano associado a esse PCC fica:

$$C_1x_1 + C_2x_2 + C_3x_3 + C_4x_4 + u_1 + u_2 + u_3$$

$$s.a. \qquad x_j \in \{0, 1\},\ j = 1, 2, 3, 4$$

novamente, os valores de x_j são dados por:

x_j = 1, se $C_j \leq 0$

x_j = 0, caso contrário

com a solução para o Problema Lagrangeano dada por:

$$z_{RL} = C_1x_1 + C_2x_2 + C_3x_3 + C_4x_4 + u_1 + u_2 + u_3$$

A determinação dos Multiplicadores de Lagrange é feita de forma estratégica, neste trabalho esses valores serão calculados através de Otimização Subgradiente, método que será discutido mais adiante. Para o momento, considere o conjunto de valores abaixo. Estes valores foram escolhidos de forma arbitrária.

u_1 = 1,5 $\quad u_2$ = 1,6 $\quad u_3$ = 2,2

sendo assim, tem-se:

$C_1 = 2 - u_1 - u_2$ = 2 - 1, 5 - 1,6 = -1,1

$$C_2 = 3 - u_3 = 3 - 2{,}2 = 0{,}8$$

$$C_3 = 4 - u_1 - u_3 = 4 - 1{,}5 - 2{,}2 = 0{,}3$$

$$C_4 = 5 - u_2 - u_3 = 5 - 1{,}6 - 2{,}2 = 1{,}2$$

e a solução do Problema Lagrangeano associado fica:

$$x_1 = 1 \qquad x_2 = x_3 = x_4 = 0$$

e

$$z_{RL} = C_1x_1 + C_2x_2 + C_3x_3 + C_4x_4 + u_1 + u_2 + u_3$$

$$= -1,1 + 0 + 0 + 0 + 1,5 + 1,6 + 2{,}2 = 4{,}2$$

Assim, a solução da Relaxação Lagrangeana para esse PCC é $z_{RL} = 4,2 < 5 = z$, ou seja, esse é de fato um limitante inferior para a solução ótima.

4.2 Relaxação Lagrangeana Avançada

Após o estudo básico sobre Relaxação Lagrangena, feito no início desta seção, um questionamento cabível seria em quais situações uma solução que é ótima para o problema lagrangeano (4.2) e viável para o problema original (4.1), é também a solução ótima do problema original. Observe que nem sempre uma solução ótima para (4.2) é viável para o problema original. Antes de responder à pergunta, considere o seguinte exemplo: no PCC da sessão anterior, defina $u_1 = u_2 = u_3 = 10$. Resolvendo o problema (4.4) para esses valores, a solução obtida será $x_1 = x_2 = x_3 = x_4 = 1$. Claramente, essa é uma solução viável para (4.1), pois satisfaz todas as restrições, no entanto o valor da função objetivo para essa solução é $z = 14$, esse valor não é o ótimo, pois sabe-se que a solução ideal é $z = 5$.

Para responder à pergunta, no problema (4.2) suponha que os multiplicadores de Lagrange $u \geq 0$ são tais que a solução ótima do problema lagrangeano, x^*, é viável para o problema original. O valor do limitante inferior de (4.1) para x^* é $[cx^* + u(b - Ax^*)]$ e o valor da função objetivo é cx^*. Logo, se esses valores coincidirem, x^* é ótimo. Supondo também que as restrições dualizadas sejam restrições de igualdade, o vetor de multiplicadores de Lagrange será irrestrito no sinal. Isto é, u poderá assumir

valores positivos ou negativos.

Teorema 4.2.1. *([4], [18]) Sejam os problemas (4.1) e (4.2) como definidos anteriormente. Se x é solução ótima de (4.2), para algum $u \geq 0$ e também $u(b - Ax) = 0$, então x é solução ótima para (4.1).*

Uma característica importante do problema lagrangeano é a Propriedade de Integralidade. Isto é, se a solução do problema (4.2) não se altera pela inserção de restrição de integralidade, então o problema lagrangeano possui a propriedade de integralidade. Fazer essa substituição significa trocar a restrição $x \in \{0, 1\}$ pela respectiva relaxação linear ($0 \leq x \leq 1$).

Teorema 4.2.2. *([4], [18]) Sejam z_{DL} o valor da função objetivo de (4.6) e z_L o valor da função objetivo referente à relaxação linear de (4.1). Se a relaxação lagrangeana possui a propriedade de integralidade, então $z_{DL} = z_L$.*

Esse teorema quer dizer que, se a relaxação lagrangeana tem a propriedade de integralidade, então o limitante inferior máximo (z_{DL}) alcançado pelo problema lagrangeano é igual ao valor da relaxação linear do problema original. No entanto, se a relaxação lagrangeana não possui a propriedade de integralidade, o limitante inferior máximo alcançado pelo problema lagrangeano é maior, ou igual, ao valor da relaxação linear do problema original, ou seja, $z_{DL} \geq z_L$.

4.3 Otimização Subgradiente

Este método tem por objetivo buscar o melhor valor para o vetor de multiplicadores, $u \geq 0$, do problema Dual Lagrangeano:

$$z_{DL} = max\{min\ cx + u(b - Ax), \text{sujeito a} Bx \geq d,\ x \in \{0, 1\}\}$$

$$u \geq 0 \tag{4.6}$$

Existem outras maneiras de aproximar os valores de u, como os métodos de Ajuste Multiplicador e Subida Dual [4], porém, a que mais se destaca é a utilização o

Método Subgradiente [4]. Este é um proceso de baixo custo computacional, iterativo e adaptado do Método do Gradiente. Seu funcionamento é descrito simplificadamente como: a partir de um conjunto inicial de multiplicadores de Lagrange, gera novos multiplicadores visando maximizar o valor do limitante inferior obtido a partir do Problema Lagrangeano (4.2), ou seja, resolvendo o problema dual Lagrangeano (4.6). Para isso, dado um vetor u^k, o método determinará um subgradiente G^k que indica a direção na qual os multiplicadores devem ser ajustados adequadamente para resolver o problema (4.6) sendo este de maximização ou minimização. O Método do Gradiente precisa ser adaptado, pois a função que define (4.2) é linear por partes. Sendo assim, sua diferenciabilidade não pode ser sempre garantida. Consequentemente, os métodos clássicos de Programação não Linear (método do gradiente e gradiente conjugado) não podem ser usados. O Algoritmo 1 é apresentado como algoritmo básico para a Otimização Subgradiente.

Algoritmo 1: Otimização Subgradiente

INÍCIO

$u = u_0$.

$k = 1$

Enquanto critério de parada não for satisfeito **faça**

Resolver o Problema Lagrangeano (4.2) com solução ótima $x(u^k)$

Determinar o subgradiente $G^k = b - Ax^k$

Se $G^k = 0$ **então** a solução ótima para (4.6) é u^k

$u^{k+1} = max(0, u^k + \tau G^k)$

$k \leftarrow k + 1$

Fim enquanto

FIM

Na primeira iteração, o algoritmo resolve o problema lagrangeano (4.2) com um conjunto de multiplicadores u^k inseridos como dados de entrada e obtém uma solução x_j com valor $\underline{z}$ para a função objetivo. Este é o primeiro limitante inferior calculado através deste método. Em seguida, calcula G_i, os subgradientes para as

restrições relaxadas, avaliados na solução atual:

$$G_i = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, ..., m \tag{4.7}$$

e um tamanho de passo τ:

$$\tau = \frac{\pi(\overline{z} - \underline{z}^k)}{\sum_{i=1}^{m} (G_i)^2} \tag{4.8}$$

Observe que $\overline{z}$ e $\underline{z}$ são o melhor limitante superior e o melhor limitante inferior, respectivamente. O limitante superior $\overline{z}$ é qualquer soluçao viável (para o problema de minimização) e pode ser calculado através de uma Heurística. Este limitante também pode ser melhorado fazendo uso, por exemplo, de uma Busca Tabu [9] ou de um Método *k-opt* [6]. Já o limitante inferior $\underline{z}$ é calculado a cada iteração k da Relaxação Lagrangeana e atualizado, caso haja melhora, ou seja, se $\underline{z}^{k+1} > \underline{z}^k$, para um problema de minimização.

Para as próximas iterações, os multiplicadores u^k são atualizados de acordo com $u^{k+1} = max(0, u^k + \tau G^k)$. Novamente o problema lagrangeano é resolvido fazendo uso deste novo conjunto de multiplicadores.

O critério de parada mencionado no algoritmo precisa ser pré-definido, caso contrário o procedimento poderia não terminar. Dessa forma, o usuário pode determinar um número específico de iterações a serem executadas ou reduzir o valor de π no decorrer das iterações, estipulando um valor para o parâmetro de maneira que o procedimento encerre quando este valor for atingido.

Exemplo: Problema de Cobertura de Conjunto

Para exemplificar uma iteração do procedimento descrito acima, considere novamente o exemplo do PCC descrito anteriormente. Sejam $\pi = 2$ e $u_1 = 1.5$, $u_2 = 1.6$ e $u_3 = 2.2$, por exemplo, valores escolhidos aleatoriamente. Além disso, suponha que $x_1 = x_3 = 1$; $x_2 = x_4 = 0$ tenham sido encontrados através de alguma heurística e retornem o limitante $\overline{z} = 6$. Resolvendo o problema Lagrangeano, a

solução $x_1 = 1$; $x_2 = x_3 = x_4 = 0$ será encontrada e retornará $\underline{z} = 4.2$ como limitante inferior.

Lembrando que as restrições para este problema são: [4]

$$x_1 + x_3 \geq 1$$

$$x_1 + x_4 \geq 1$$

$$x_2 + x_3 + x_4 \geq 1$$

$$x_j \in \{0, 1\} \qquad j = 1, 2, 3, 4$$

A seguir são calculados valores para os subgradientes:

$$G_1 = 1 - x_1 - x_3 = 1 - 1 - 0 = 0$$

$$G_2 = 1 - x_1 - x_4 = 1 - 1 - 0 = 0$$

$$G_3 = 1 - x_2 - x_3 - x_4 = 1 - 0 - 0 - 0 = 1$$

E o tamanho de passo é calculado de acordo com a equação (4.8):

$$\tau = \frac{2(6 - 4,2)}{0^2 + 0^2 + 1^2} = 3,6$$

O próximo passo é atualizar os multiplicadores de Lagrange, para isso o critério $u_{i+1} = max(0, u_i + \tau G_i)$ é utilizado. Assim:

$$u_1 = max(0; 1.5 + 3,6 \cdot 0) = 1.5$$

$$u_2 = max(0; 1.6 + 3,6 \cdot 0) = 1.6$$

$$u_3 = max(0; 2.2 + 3,6 \cdot 0) = 5.8$$

Utilizando este novo conjunto de multiplicadores de Lagrange, resolve-se o problema (4.2) e a solução encontrada é $x_1 = x_2 = x_3 = x_4 = 1$, que retorna um novo limitante inferior, a saber $\underline{z}^2 = -0.7$.

Observe que a qualidade do novo limitante inferior ($\underline{z}^2 = -0.7$) é bem pior do que a do antigo $\underline{z}^1 = 4.2$. Tendo em vista que a solução ótima para este PCC é $z = 5$ (como informado anteriormente), o primeiro limitante está muito mais próximo do que o segundo. Este comportamento é comum em otimização subgradiente, ou seja, não se pode garantir uma melhora contínua no limitante em cada iteração. Além disso, o limitante encontrado pelo Método Subgradiente pode vir a ser negativo, como o encontrado neste exemplo.

Em virtude do que foi mencionado com base em [4], o Método Subgradiente possui convergência imprevisível. Para alguns casos, chega-se a soluções muito boas e confiáveis, por outro lado soluções ruins podem ser obtidas, nestes casos o método apresenta um comportamento incomum tanto para os valores de subgradientes quanto para os valores dos limitantes. Graficamente, as soluções possuem comportamento oscilatório tanto para cima quanto para baixo, em ambos os casos. A grande diferença é que quando o método converge, essas oscilações têm comportamento assintótico em torno do que se espera ser a solução para o Sub-Problema Lagrangeano, ou seja, do limitante de qualidade. Por outro lado, quando o método não converge, as oscilações pioram a cada iteração [19].

Entretanto, é comum que o limitante $\underline{z}^k$ se aproxime (ou mesmo atinja) o máximo limitante inferior possível, solução do problema Dual-Lagrangeano [4]. Seja τ^1, τ^2, ..., τ^k uma sequência de tamanhos de passo que converge para zero, mas não tão rápido, ou seja, $\lim\limits_{k\to\infty} t^k = 0$ e $\sum\limits_{k=0}^{\infty} t^k = \infty$. Nessas condições, é possível mostrar [20] que a sequência de limitantes inferiores produzidos pelo Método Subgradiente converge para o valor ótimo do Dual Lagrangeano.

Capítulo 5

Limitante de Held e Karp

Considere o Problema do Caixeiro Viajante Simétrico como definido por (3.1), (3.2), (3.3) e (3.4). Suponha que o vendedor possua um número, n, de cidades a serem visitadas. Como mencionado anteriormente, o PCV possui (n-1)! possíveis soluções. Assim, se o vendedor precisa, por exemplo, visitar 9 cidades, existem 8! = 40.320 possíveis rotas e algumas dessas rotas poderão ser solução ótima para o problema. Com isso, pode-se ver que, mesmo para um número de cidades n relativamente pequeno, o problema possui uma quantidade exponencial de restrições.

Admitindo que as retrições a serem dualizadas são as de sub-rotas, imagine quantas sub-rotas terá o problema mencionado acima (com apenas 9 vértices). E para cada uma dessas restrições, havéra um Multiplicador de Lagrange $u \geq 0$ a ser determinado pelo Método Subgradiente. Para calcular o tamanho de passo nesse método, o denominador da fração é a norma do subgradiente G ao quadrado e para cada $i = 1, ..., m$ há uma restrição do tipo (3.3). Dessa forma quanto maior o valor de m, maior será o número de restrições, com isso o denominador da fração (4.8) tende a crescer e, consequentemente, τ (o tamanho de passo do método) tende a decrescer. Então, dado um número muito grande de restrições de sub-rotas o tamanho de passo τ tende a zero, ou seja, a diferença entre o multiplicador u^{k-1} e u^k estará muito próxima de zero cada iteração do algoritmo, não havendo melhora nem piora nas soluções seguintes. Por conta disso, o problema torna-se muito mais difícil do que o esperado, considerando que ao

aplicar a Relaxação Lagrangeana busca-se um problema mais fácil de ser resolvido do que o original.

Tendo em vista que o PCV dispõe de um número exponencial de restrições a serem dualizadas, Held e Karp [5] propuseram uma outra forma de determinar o limitante inferior. Nesse método algumas restrições do tipo (3.1) são dualizadas ao invés das restrições de sub-rotas. O problema obtido dessa forma é uma relaxação para o PCV.

5.1 A *1-tree*

Definição 5.1.1. Dado um grafo $G = (V, E)$ e um vértice $u \in V$, uma $1 - tree$ é um subgrafo $G' \subset G$ que consiste em uma árvore ligando todos os vértices de G e duas arestas adjacentes ao vértice u.

Seja $G = (V, E)$ o grafo não-direcionado que modela o PCV, c_e o custo de cada aresta $e \in E$ e um caminho $T \subseteq E$. Considere $u \in T$ e duas arestas e e f em T incidentes a u, seja também P um subconjunto de E, onde os elementos de P formam um caminho obtido com a retirada das arestas e e f de T. O custo do caminho T pode ser obtido da seguinte forma:

$$c(T) = c_e + c_f + c(P)$$

Encontrando números A e B tais que $A \leq c_e + c_f$ e $B \leq c(P)$, então $A + B$ será um limitante inferior para o custo do caminho T, pois dessa forma tem-se:

$$A + B \leq c_e + c_f + c(P) = c(T)$$

Este limitante inferior é válido para todo o percurso dado. Uma forma simples de se determinar o limitante é definir A como a soma dos custos das arestas mais baratas, que sejam incidentes ao vértice u. Para estimar o número B, observe que P é uma árvore geradora para o grafo $G \setminus u$ (que pode ser obtido a partir da retirada u e das arestas de G incidentes a este vértice). Assim B será o custo de uma Árvore

Geradora Mínima em $G \setminus u$ (que pode ser calculada fazendo-se uso de um dos métodos descritos no Capítulo 1). Ou seja, P terá custo, no mínimo, B. O limitante $A+B$ assim determinado é a *1-tree* de custo mínimo para o grafo $G = (V, E)$, de acordo com [6].

5.2 A proposta de Held e Karp

Conforme apresentado no Capítulo 3, o PCV simétrico possui a seguite formulação:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j;\ j \neq i} x_{ij} = 1,\ onde\ i = 1, 2, ..., n$$

$$\sum_{i;\ i \neq j} x_{ij} = 1,\ onde\ j = 1, 2, ..., n$$

$$\sum_{\substack{i \in S \\ j \in S}} x_{ij} \leq |S| - 1\ para\ S \subset V,\ 2 \leq |S| \leq n - 1$$

$$x_{ij} \in \{0, 1\}$$

Qualquer solução viável para o problema apresentado acima possui um número n de variáveis x_{ij} iguais a 1, ou seja, n deslocamentos os quais o vendedor deverá percorrer até retornar ao ponto de partida. Tendo em vista que ele faz o caminho entre cada cidade apenas uma vez, o grafo que modela este problema possui $n - 1$ arestas, ou seja até aqui tem-se $n - 1$ variáveis x_{ij} no problema. Porém este vendedor precisa fazer o caminho de volta para a cidade inicial e, dessa forma, completar o ciclo que é a solução viável do PCV. Assim, adiciona-se mais uma variável às $n - 1$ que já tinham sido contabilizadas, formando n variáveis (arestas) ao todo. Caso contrário, o vendedor poderia deixar de cumprir alguma rota entre uma das cidades ij.

Contudo, a formulação apresentada para o PCV trata todas as considerações feitas acima. Incluir mais uma restrição apenas para informar que em qualquer solução

o número de variáveis x_{ij} iguais a 1 deve ser n, seria redundante. Porém a adição dessa restrição não conduziria a uma solução errada. Partindo desse fato, a restrição abaixo será incluída na formulação do PCV:

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} x_{ij} = n \tag{5.1}$$

onde $V = \{1, ..., n\}$ é o conjunto de vértices do grafo que modela o PCV.

Somando as restrições (3.1) e (3.2):

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} x_{ij} \; + \; \sum_{j=1}^{n}\sum_{\substack{i=1\\ i\neq j}}^{n} x_{ji} \; - \; 2$$

Escolhendo um entre os n vértices de V e nomeando-o v_1, constrói-se um subconjunto de V que será chamado $V' = V \setminus \{v_1\}$. Os conjuntos de restrições (3.1) e (3.2) serão dualizados na função objetivo, para todo $v_i \in V'$, ou seja, será atribuído um Multiplicador de Lagrange u_i, $i = 2, ...n$ para cada somatório de restrições (3.1) e (3.2), como apresentado abaixo:

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \; + \; \sum_{i=2}^{n} u_i \left(\sum_{\substack{j=1\\ j\neq i}}^{n} (x_{ij} + x_{ji}) \; - \; 2 \right) \; =$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \; + \; \sum_{i=2}^{n} \left(u_i \left(\sum_{\substack{j=1\\ j\neq i}}^{n} (x_{ij} \; + \; x_{ji}) \right) \; - \; 2u_i \right) \; =$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \; + \; \sum_{i=2}^{n} \left(\sum_{\substack{j=1\\ j\neq i}}^{n} (u_i x_{ij} \; + \; u_i x_{ji}) \right) \; - \; \sum_{i=2}^{n} 2u_i \; =$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \; + \; \sum_{i=2}^{n} \left(\sum_{\substack{j=1\\ j\neq i}}^{n} u_i x_{ij} \; + \; \sum_{\substack{j=1\\ j\neq i}}^{n} u_i x_{ji} \right) \; - \; \sum_{i=2}^{n} 2u_i \; =$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \;+\; \sum_{i=2}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} u_i x_{ij} \;+\; \sum_{i=2}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} u_i x_{ji} \;-\; \sum_{i=2}^{n} 2u_i \;=$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \;+\; \sum_{i=2}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} u_i x_{ij} \;+\; \sum_{j=2}^{n}\sum_{\substack{i=1\\ i\neq j}}^{n} u_j x_{ij} \;-\; \sum_{i=2}^{n} 2u_i \;=$$

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij} \;+\; \sum_{i=2}^{n} u_i x_{i1} \;+\; \sum_{i=2}^{n}\sum_{\substack{j=2\\ j\neq i}}^{n} u_i x_{ij} \;+\; \sum_{j=2}^{n} u_j x_{1j} \;+\; \sum_{j=2}^{n}\sum_{\substack{i=2\\ i\neq j}}^{n} u_j x_{ij} \;-\; \sum_{i=2}^{n} 2u_i \;=$$

O termo

$$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij}$$

será reescrito como:

$$\sum_{j=2}^{n} c_{1j}x_{1j} \;+\; \sum_{i=2}^{n} c_{i1}x_{i1} \;+\; \sum_{i=2}^{n}\sum_{\substack{j=2\\ j\neq i}}^{n} c_{ij}x_{ij}$$

Em seguida os termos com mesmo somatório e variável x com índices iguais serão agrupados. O resultado obtido pode ser escrito como:

$$\sum_{j=2}^{n}(c_{1j} \;+\; u_j)x_{1j} \;+\; \sum_{i=2}^{n}(c_{i1} \;+\; u_i)x_{i1} \;+\; \sum_{i=2}^{n}\sum_{\substack{j=2\\ j\neq i}}^{n}(c_{ij} \;+\; u_i \;+\; u_j)x_{ij} \;-\; \sum_{i=2}^{n} 2u_i$$

onde $u \in \mathbb{R}^n$. Essa será a função objetivo da relaxação que está sendo construída para o PCV, cujas restrições formam uma *1 - tree*.

Vale ressaltar que as restrições (3.1) e (3.2) referentes ao vértice v_1 permanecerão na formulação do problema e serão escritas da seguinte forma:

$$\sum_{j=1}^{n} x_{1j} \;+\; \sum_{i=1}^{n} x_{i1} \;-\; 2 \;=\; 0 \tag{5.2}$$

Reescrevendo o PCV com as alterações discutidas anteriormente, obtém-se o problema a seguir:

$$\underset{1-tree}{z(u)} = \min \sum_{j=2}^{n}(c_{1j} + u_j)x_{1j} + \sum_{i=2}^{n}(c_{i1} + u_i)x_{i1} + \sum_{i=2}^{n}\sum_{\substack{j=2 \\ j\neq i}}^{n}(c_{ij} + u_i + u_j)x_{ij} - \sum_{i=2}^{n} 2u_i$$

$$\sum_{\substack{i\in S \\ j\in S}} x_{ij} \leq |S| - 1 \; para \; S \subset V', \; 2 \leq |S| \leq n-1$$

$$\sum_{i\in V'}\sum_{j\in V'} x_{ij} = n - 2$$

$$\sum_{j=1}^{n} x_{1j} + \sum_{i=1}^{n} x_{i1} = 2$$

$$x_{ij} \in \{0, 1\}$$

Por construção, essa é claramente uma relaxação para o PCV. Além disso essa formulação é o Sub-Problema Lagrangeano referente ao PCV. Observe que uma solução para esse novo problema obedece à definição vista na seção (5.1), ou seja, uma solução viável é uma *1-tree*. As restrições de eliminação de sub-rotas garantem um grafo conexo, ou seja, haverá um caminho entre cada par de vértices. Por outro lado, a restrição (5.2) garante que o vértice v_1 terá grau 2, ou seja, duas arestas serão adjacentes a v_1.

Outra observação importante a ser feita é que um ciclo hamiltoniano é um caso particular de *1-tree*. Basta, no ciclo hamiltoniano, eliminar um vértice e as duas arestas adjacentes a ele, o grafo obtido fazendo essas retiradas será uma árvore. Assim, se o ciclo considerado for de custo mínimo, a árvore será uma AGM.

Decorre dos fatos anteriores que o conjunto de possíveis soluções para $\underset{1-tree}{z(u)}$ contém o conjunto de possíveis soluções para o PCV. Isto é, a *1-tree* representa um limitante inferior para o PCV. O Dual Lagrangeano pode ser escrito como segue:

$$\underset{1-tree}{z_D} = \max \underset{1-tree}{z(u)} \tag{5.3}$$

$$u \in \mathbb{R}$$

O limitante inferior dado por $\underset{1-tree}{z_D(u)}$ é chamado de Limitante de Held-Karp [6].

Portanto, a proposta de Held e Karp [5] consiste em escolher um vértice do grafo que modela o PVC para o qual a restrição do Problema de Atribuição referente a esse vértice não será removida do Sub-Problema Lagrangeano. As demais restrições, referentes a todos os outros vértices, serão dualizadas na função objetivo e receberão seus respectivos Multiplicadores de Lagrange, $u^k > 0$. A solução ótima para este novo Sub-Problema Lagrangeano é a *1-tree* de custo mínimo. Esta pode não ser uma solução ótima para o problema original, mas é um limitante inferior.

Uma das vantagens de obter um limitante inferior usando o método apresentado, é a simplicidade computacional, visto que encontrar a *1-tree* mínima envolve, no máximo, complexidade polinomial [9]. Basta escolher um vértice que será chamado v_1 e eliminar as arestas incidentes a ele e o próprio v_1. Entre os vértices e arestas restantes, gerar a Árvore Geradora Mínima, fazendo uso de um dos algoritmos (Prim ou Kruskal) apresentados no Capítulo 1. Após esse procedimento, inclui-se na AGM o vértice v_1 e as duas arestas de menor custo incidentes a ele. É importante observar que o Algoritmo de Prim possui complexidade $O(n^2)$ e o algoritmo de Kruskal possui $O(a$ $\log(n))$, nos piores casos [21]. Note que n é a quantidade de vértices e a é o número de arestas do grafo.

Outra vantagem de usar este método é que ele retorna bons limitantes. Em [22], a solução obtida, utilizando somente a $1 - tree$, em um conjunto de 140 problemas é em média 0,63 do valor ótimo do PCV. Todavia, a solução pode ser melhorada dualizando as restrições de *Assignment*, como visto acima.

5.3 Exemplo Prático

Com o objetivo de exemplificar a teoria exposta no decorrer deste trabalho, considere a seguinte situação. Um casal de turistas deseja passar alguns dias de férias na cidade do Rio de Jeneiro. O hotel escolhido para ficarem hospedados foi o Novotel

RJ Santos Dumont, localizado na Avenida Marechal Câmara, 300. Como proposta de roteiro turístico, a agência de viagens indicou as seguintes atrações, marcadas no mapa da figura (5.1): Cristo Redentor, o estádio Maracanã, Bondinho do Pão de Açúcar, Jardim Botânico, Lapa e o aquário AquaRio. Cada vértice dessa figura foi nomeado por v_i, com $i = 0, ..., 6$ de acordo com a tabela (5.1a)

Figura 5.1: Hotel e destinos propostos

O problema apresentado pela agência de viagens é traçar uma rota que saia do hotel, passe por todos esses pontos turísticos, gastando o menor tempo possível no translado de um ponto até o outro e retorne para o ponto de partida (hotel). Otimizar o tempo no translado é importante a fim de que os turistas gastem o menor tempo possível no trânsito, podendo assim aproveitar ao máximo os passeios.

Este claramente é um Problema do Caixeiro Viajante. O limitante inferior para este problema é calculado fazendo uso da teoria estudada por Held e Karp [5] e apresentada no início deste capítulo. Para compor o modelo, cada ponto visitado será considerado como vértice de um grafo e nomeado de acordo com a tabela (5.1a).

O custo a ser minimizado será a soma do tempo total de translado saindo do hotel, passando por todos os pontos turísticos e retornando para o hotel. A figura (5.2) apresenta uma rota viável para este problema. Na tabela (5.1b) foram contabilizados os tempos de ida e volta em todas as possíveis combinações de caminhos. Para determinar

Local	Vértice
Hotel	v_0
Cristo Redentor	v_1
Maracanã	v_2
Pão de Açúcar	v_3
Jardim Botânico	v_4
Lapa	v_5
Aquário	v_6

(a) Lugares e Vértices

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
v_0	0	29	23	21	33	10	9
v_1	31	0	36	44	23	26	31
v_2	20	32	0	36	21	18	17
v_3	16	41	34	0	34	25	25
v_4	34	29	29	37	0	31	35
v_5	10	28	19	26	31	0	18
v_6	21	32	18	36	30	16	0

(b) Tempo de Viagem

Tabela 5.1: Pontos turísticos e os respectivos tempos de translado

os tempos de viagem entre cada ponto foi usado o site de busca *Google Maps* [23]. Os referidos intervalos de tempo estão representados na tabela (5.1b) em unidades de minuto.

Figura 5.2: Exemplo de ciclo viável para este problema [23]

Na estruturação desse problema, desconsiderou-se os horários de pico do trânsito na cidade tendo em vista que esta é uma variável estatística. Neste caso, o intervalo de tempo de ida e volta entre cada ponto deveria ser apurado em uma quantidade determinada de dias. Em seguida, uma variável estatística seria usada para normalizar esses

valores adequadamente. Como este não é o objetivo do trabalho, os cálculos foram simplificados desconsiderando esta variável.

Figura 5.3: Exemplo de custos diferentes para ida e volta [24]

A partir da tabela (5.1b), constrói-se a matriz $\overline{c}$, de custos:

$$\overline{c} = \begin{pmatrix} 0 & 29 & 23 & 21 & 33 & 10 & 9 \\ 31 & 0 & 36 & 44 & 23 & 26 & 31 \\ 20 & 32 & 0 & 36 & 21 & 18 & 17 \\ 16 & 41 & 34 & 0 & 34 & 25 & 25 \\ 34 & 29 & 29 & 37 & 0 & 31 & 35 \\ 10 & 28 & 19 & 26 & 31 & 0 & 18 \\ 21 & 32 & 18 & 36 & 30 & 16 & 0 \end{pmatrix}$$

É possível observar que essa matriz não é simétrica, pois os tempos de ida de um ponto a outro, em sua maioria, são diferentes dos de volta, isto é, $\overline{c}_{ij} \neq \overline{c}_{ji}$. Este fato é exemplificado na figura (5.3), onde a ida do Hotel (v_0) até o Cristo Redentor (v_1) possui um custo de 29 minutos e a volta (de v_1 até v_0) tem custo 31 minutos. Consequentemente, a utilização desses valores como custos retornará um PCV assimétrico. Com o objetivo de formular um PCV simétrico, propõe-se aplicar a seguinte normalização:

$$c_{ij} = \frac{\overline{c}_{ij} + \overline{c}_{ji}}{2} = c_{ji}$$

Além de normalizados, os custos foram multiplicados por 10, apenas para facilitar os cálculos trabalhando somente com números inteiros. E assim, chega-se à seguinte matriz:

$$c = \begin{pmatrix} 0 & 300 & 215 & 185 & 335 & 100 & 150 \\ 300 & 0 & 340 & 425 & 260 & 270 & 315 \\ 215 & 340 & 0 & 350 & 250 & 185 & 175 \\ 185 & 425 & 350 & 0 & 355 & 255 & 305 \\ 335 & 260 & 250 & 355 & 0 & 310 & 325 \\ 100 & 270 & 185 & 255 & 310 & 0 & 170 \\ 150 & 315 & 175 & 305 & 325 & 170 & 0 \end{pmatrix}$$

A fim de iniciar as iterações do Método Subgradiente, é preciso determinar um limitante inferior e um limitante superior para o PCV. Como visto no Capítulo 2, o limitante superior é qualquer solução viável para o problema. Já o limitante inferior é determinado resolvendo-se a relaxação do PCV, que neste caso é o problema de calcular o custo da *1-tree* mínima. O Python foi usado como linguagem de programação para desenvolver um código que solucionasse cada etapa do PCV mencionada acima.

O limitante superior foi calculado a partir de uma Heurística. Segundo [9], as Heurísticas são técnicas (algoritmos) que buscam alcançar uma boa solução utilizando um esforço computacional considerado razoável, sendo capazes de garantir a viabilidade da solução encontrada. Existem muitos tipos de Heurísticas que poderiam ser aplicadas ao PCV, como, por exemplo os algoritmos *k-opt* [6], de Inserção[9] e o Guloso[9]. Além disso as soluções obtidas com estas Heurísticas poderiam ser aprimoradas por outros algoritmos, como *Simulated Annealing* [9], Busca Tabu [9] e Algoritmos Genéticos [9].

Dentre as várias possibilidades de escolha, usou-se neste trabalho uma Heurística bem simples, seu nome é *Nearest Neighbor* [9], ou seja, a Heurística do Vizinho Mais Próximo. Este é um método construtivo que cria passo-a-passo uma solução viável (limitante superior) para o PCV, de acordo com um conjunto de regras. Essas regras dizem respeito ao método de escolher o próximo elemento que será adicionado à solu-

ção, isto é, o vértice adjacente que possui a menor distância do vértice considerado. O algoritmo que resume essa Heurística é mostrado a seguir:

Algoritmo 2: Heurística do Vizinho Mais Próximo

INÍCIO

1 - Comece com um caminho parcia que possui apenas um vértice i, escolhido aleatoriamente.

2 - Seja (1, ..., k) o caminho parcial atual, com ($k < n$). Encontre o vértice $k + 1$ que ainda não está no caminho e que esteja mais próximo de k.

3 - Inserir $k + 1$ no final do caminho parcial.

4 - **Se** todos os vértices estiverem na solução, então PARE.

Senão volte para 2.

FIM

5.3.1 Resultados

Todos os métodos para determinar limitantes inferiores para o PCV, descritos neste capítulo foram implementados através da linguagem de programação *Python* [7]. Através dela, funções foram executadas em diversas iterações a fim de encontrar a *1-tree* mínima, resolver a Otimização Subgradiente e uma Heurística Construtiva para calcular limitantes superiores. Esta última foi implementada de acordo com o Algoritmo 2. A seguir são apresentados os resultados obtidos através deste código.

A figura (5.4) mostra o primeiro limitante superior determinado pela Heurística. Observe que esta é de fato uma solução viável para o PCV e possui custo $\overline{z}^0 = 1615$

Para calcular o limitante inferior, basta encontrar a $1 - tree$ mínima para o grafo do problema. Este procedimento foi feito da seguinte forma: primeiramente o vértice v_0 foi escolhido para ser retirado do grafo juntamente com todas as arestas incidentes a ele. Em seguida, calcula-se a AGM para o grafo formado pelos vértices e arestas restantes após essa retirada. A AGM foi calculada pelo Algoritmo de Kruskal.

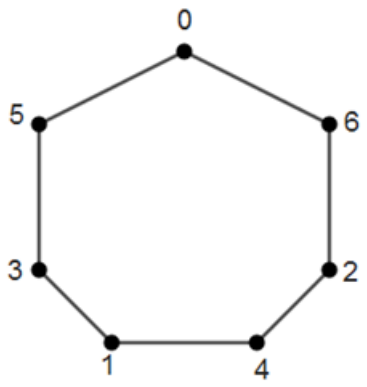

Figura 5.4: Primeiro Limitante Superior obtido através do código

Pode-se ver que a AGM possui custo 1110 e arestas v_1v_4, v_4v_2, v_2v_6, v_6v_5 e v_5v_3. Unindo novamente ao grafo o vértice v_0 e as duas arestas de menor custo incidentes a ele (a saber, v_0v_5 e v_5v_0), obtém-se o limitante inferior para o problema, $\underline{z}^0 = 1310$.

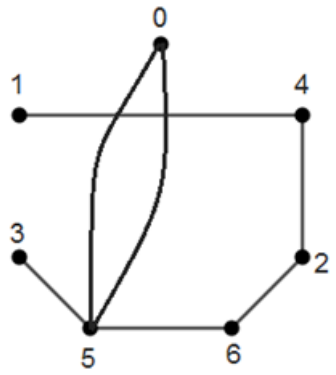

Figura 5.5: Primeiro limitante inferior obtido através do código

Os graus dos vértices da $1-tree$ na iteração $k = 0$ são representados no vetor $d^0 \in \mathbb{N}^7$ e possuem os seguintes valores $d^0 = (2, 1, 2, 1, 1, 3, 2)$, onde d_i^0 é o grau do vértice v_i, $i = 1, 2, ..., 7$. Considere como vetor de Multiplicadores de Lagrange inicial $u_i^0 = 0$, $i = 1, 2, ..., 7$ e o parâmetro $\pi = 2$.

Os subgrafos de G obtidos pelas soluções dual e primal na segunda iteração são apresentadosa figura (5.6). Seus respectivos custos são $\overline{z}^1 = 1600$ e $\underline{z}^1 = 1273,33333$. Como mencionado no Capítulo 2, a busca por uma solução ótima para o PCV se resume em diminuir os limitantes superiores e aumentar os limitantes inferiores, fazendo-os convergir para um mesmo limitante $\overline{z} = \underline{z}$. Na maioria dos problemas,

essa última igualdade não é alcançada, porém os métodos utilizados retornam bons limitantes. Aplicando esse mesmo racicínio para os quatro limitantes obtidos até agora, vê-se que $\overline{z}^0 > \overline{z}^1$, ou seja, a $2^{\underline{a}}$ iteração retornou um limitante superior melhor e $\underline{z}^0 > \underline{z}^1$, isto é, a $2^{\underline{a}}$ iteração retornou um limitante inferior pior. Portanto os valores serão atualizados da seguinte forma para a $3^{\underline{a}}$ iteração: o novo limitante superior será $\overline{z}^1$ e o limitante inferior permanecerá $\underline{z}^0$.

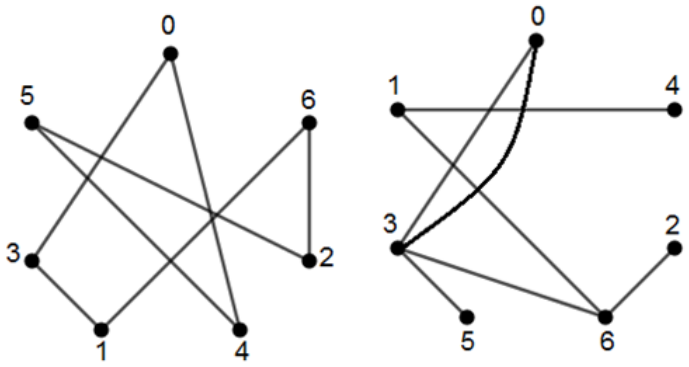

Figura 5.6: Limitante Superior e Inferior, respectivamente, na $2^{\underline{a}}$ iteração

Os subgrafos de G obtidos pelas soluções dual e primal na terceira iteração são apresentados na figura (5.7) e seus respectivos custos são $\overline{z}^3 = 1765$ e $\underline{z}^3 = 1215$. Observe que, apesar de haver uma melhora no limitante inferior, essa melhora não foi suficiente para que este limitante pudesse ser atualizado, com isso $\underline{z}^0$ continua sendo o melhor limitante inferior. Outro ponto importante a ser observado é que o limitante primal $\overline{z}^2$ se distanciou consideravelmente do atual $\overline{z}^1$, essas oscilações foram previstas por [19].

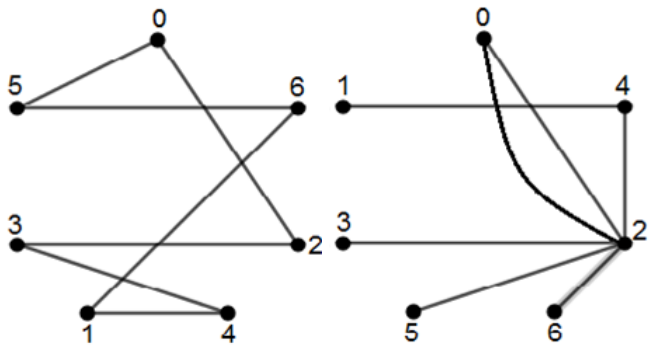

Figura 5.7: Limitante Superior e Inferior, respectivamente, na $3^{\underline{a}}$ iteração

Outros valores consideráveis foram obtidos na $6^{\underline{a}}$ iteração e estão representados na figura (5.8). Os limitantes possuem os seguintes valores: $\overline{z}^5 = 1545$ e $\underline{z}^5 = 1443,9881$. É possível observar uma redução significativa em $\overline{z}^5$, em relação aos valores apresentados anteriormente. Além disso, o valor de $\underline{z}^5$ aumenta, ou seja, melhora em relação aos valores obtidos nas iterações anteriores.

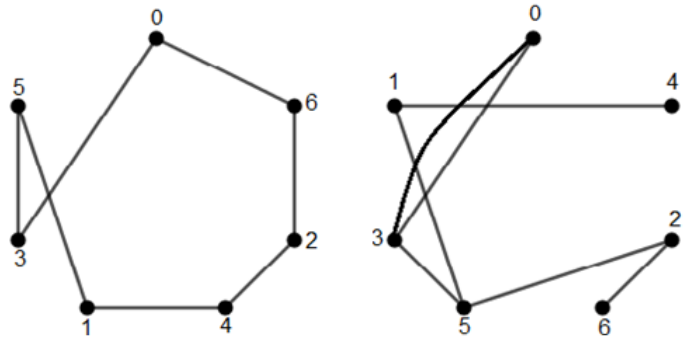

Figura 5.8: Limitante Superior e Inferior, respectivamente, na $6^{\underline{a}}$ iteração

A $7^{\underline{a}}$ iteração retorna os grafos exibidos na figura (5.9) e os seguintes limitantes: $\overline{z}^6 = 1575$ e $\underline{z}^6 = 1503,4702$. Nesta iteração, $\overline{z}^6$ aumenta, por isso não será atualizado, permanecendo $\overline{z}^5$. Porém, observe que $\overline{z}^5 = 1545$ e $\underline{z}^6 = 1503,4702$ estão bem próximos.

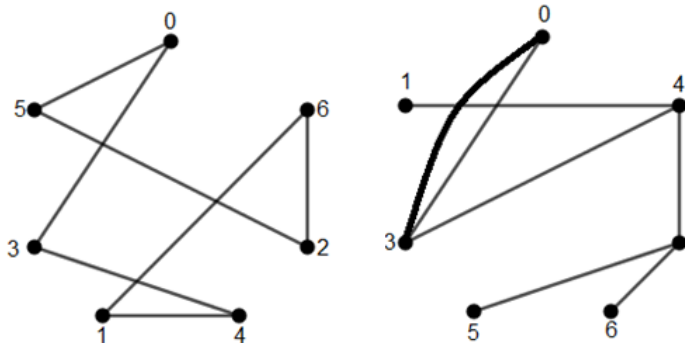

Figura 5.9: Limitante Superior e Inferior, respectivamente, na $7^{\underline{a}}$ iteração

Abaixo são mostrados os resultados obtidos na $16^{\underline{a}}$ iteração, cujos valores para os limitantes são $\overline{z}^{15} = 1545$ e $\underline{z}^{15} = 1541,326$. Destaca-se essa iteração dentre as demais, pois nela observa-se uma proximidade muito grande entre os limitantes.

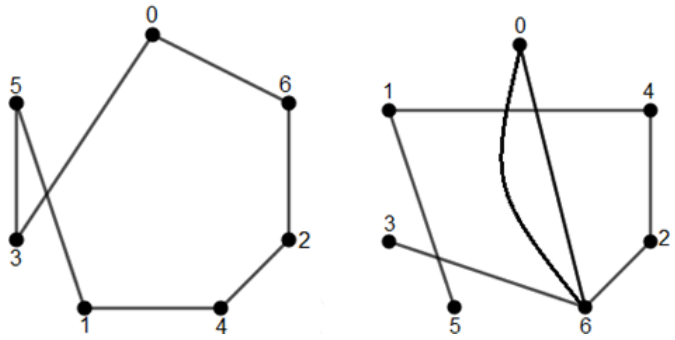

Figura 5.10: Limitante Superior e Inferior, respectivamente, na $16^{\underline{a}}$ iteração

A última iteração do código é apresentada na figura (5.11). Ao todo foram desenvolvidas 20 iterações e o valor ótimo, dado por $\overline{z}^{19} = \underline{z}^{19} = 1545$, foi alcançado nesta última. Vale ressaltar que, na maioria dos casos, o métodos não conseguem determinar a solução ótima devido à complexidade do PCV e à grande quantidade de destinos a serem visitados. Apesar disso, o Limitante de Held-Karp retorna resultados muito bons, como os apresentados em [19]. Muitos pesquisadores aliam métodos de aproximação de soluções, como Heurísticas e Relaxações, à métodos exatos como *Branch-and-Bound* [2] com o objetivo de determinar solução exata para o problema.

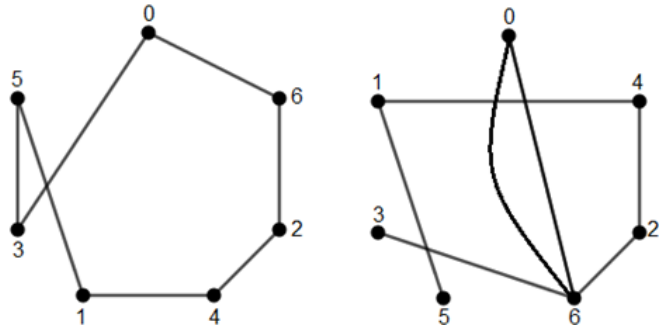

Figura 5.11: Limitante Superior e Inferior, respectivamente, na $20^{\underline{a}}$ iteração

A $20^{\underline{a}}$ iteração retorna como ciclo ótimo $v_0v_3v_5v_1v_4v_2v_6v_0$. Isso significa que o melhor caminho consiste em sair do Hotel e visitar os pontos turísticos na segunte ordem: Pão de Açúcar, Lapa, Cristo Redentor, Jardim Botânico, Maracanã, AquaRio. O tempo de translado estimado para este trajeto é 2 horas e 35 minutos. Na figura (5.12) pode-se ver o percurso traçado no mapa, feito com o auxílio do aplicativo *My Maps* [24].

Figura 5.12: Ciclo ótimo traçado no mapa [24]

Para melhor visualização de todas as iterações desenvolvidas pelo código, está abaixo o gráfico na figura (5.13) que expõe claramente as grandes oscilações dos limitantes superior e inferior em relação à solução ótima, descritas em [19].

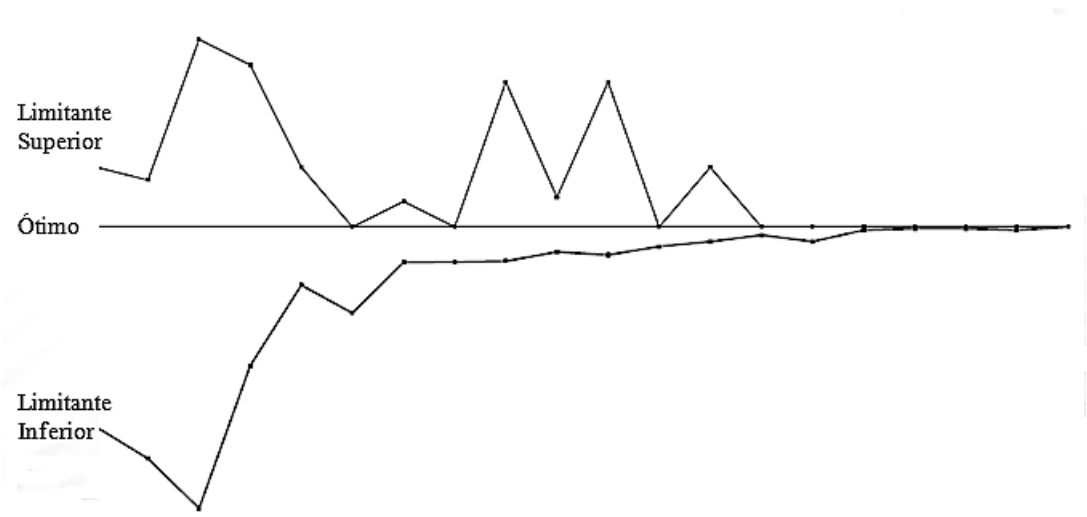

Figura 5.13: Gráfico comparativo

Capítulo 6

Considerações Finais

Dado um grafo G = (V, E) com custos c_{ij}, tais que $c_{ij} = c_{ji}$ para cada aresta $ij \in$ E, $i, j = 1, ..., n$, investigamos neste trabalho o problema de determinar um ciclo de custo mínimo que passe por todos os pontos uma única vez e retorne ao ponto de partida. Tal problema é denominado Problema do Caixeiro Viajante Simétrico.

Com o objetivo de expor o que é e como calcular o Limitante de Held-Karp para o PCV Simétrico, iniciamos este trabalho explicando conceitos iniciais de Grafos e de Programação Inteira necessários ao entendimento de toda a teoria exposta no decorrer do texto. Em seguida, definimos o PCV explicando sua importância e analisando cuidadosamente cada uma de suas restrições. Além disso, introduzimos com a definição de Relaxação Lagrangeana, exemplificando a mesma por um Problema de Cobertura de Conjuntos, de acordo com [4]. O Limitante de Held-Karp foi apresentado posteriormente como um aprimoramento da Relaxação Lagrangeana e tendo como base uma *1-tree*.

Para exemplificar toda a teoria exposta, utilizamos um exemplo muito atual e de grande aplicação prática. Este foi elaborado com dados reais e solucionado através da aproximação progressiva de limitantes superiores e inferiores. Os limitantes superiores foram determinados através da Heurística do Vizinho mais Próximo. A teoria desenvolvida para este cálculo não é o principal objetivo deste trabalho, porém foi utilizada para que pudéssemos explicar com clareza a progressiva aproximação de

limitantes até a obtenção de um erro ínfimo (pré-determinado) ou até que os limitantes sejam iguais. Os limitantes inferiores foram determinados de acordo com a teoria estudada por Held-Karp.

Todos os cálculos foram feitos através do código apresentado em anexo. As iterações mais relevantes foram apresentadas como resultados na seção (5.3.1) e ilustradas através dos grafos referentes a cada uma das iterações e cujas arestas são as variáveis x_{ij} do PCV às quais foram atribuídas o valor 1.

Duas contribuições relevantes deste trabalho podem ser destacadas. A primeira é o código que calcula e avalia os limitantes inferiores e superiores para o PCV fazendo uso do limitante de Held-Karp em paralelo com o Método Subgradiente e Relaxação Lagrangeana e ainda fazendo uso da Heurística do Vizinho mais Próximo para determinar o limitante superior. A segunda contribuição a ser destacada é a forma simplificada com a qual os conceitos foram explicados tornando-os de fácil entendimento para alunos dos primeiros anos da graduação.

Apesar de termos dissertado a respeito do PCV Simétrico, este exemplo seria melhor modelado por um PCV Assimétrico, ou seja, apresentaria um modelo mais adequado para o exemplo, visto que consideraríamos muito mais detalhes reais a respeito do problema. Outro ponto a ser melhorado nesta abordagem seria inserir variáveis estatísticas para modelar, por exemplo, os horários de pico do local considerado. Isso levaria em conta o mehor horário para realizar o passeio, desconsiderando assim os horários em que o trânsito estivesse congestionado.

A seguir são listadas algumas propostas para trabalhos futuros que poderiam ser efetuadas a partir deste trabalho:

- Usar instâncias com número maior de lugares a serem visitados. Visto que, para um número maior de lugares a serem visitados, o algoritmo possivelmente não retornaria uma solução ideal, mas um resultado aproximado e de qualidade para o problema.

- Incluir na heurística uma busca local. Isto melhoraria o limitante superior calculado e o tempo de cálculo, pois buscaria examinar um número cada vez maior

de soluções a cada iteração.

- Elaboração de um algoritmo *Relax-and-Cut* para o PCV, isso possibilitaria o uso de desigualdades fortes e uma consequente melhora do limitante inferior.

Referências Bibliográficas

[1] KARP, R. M. On the computational complexity of combinatorial problems. *Networks*, v. 5, p. 45– 68, 1975.

[2] WOLSEY, L. A. *Integer programming*. JOHN WILEY & SONS INC, 1998.

[3] REINELT, G. *The traveling salesman: Computational solutions for tsp applications*. Springer-Verlag, 1994.

[4] BEASLELY, J. E. Modern heuristic techniques for combinatorial problems. New York, NY, USA: Wiley, 1993. Cap. Lagrangian Relaxation, p. 243–303.

[5] M., H.; M., K. R. The traveling-salesman problem and minimum spanning trees. *Operations Research*, v. 18, p. 1138–1162, 1970.

[6] COOK, W. J.; CUNNINGHAM, W. H.; PULLEYBLANK, W. R.; SCHRIJVER, A. *Combinatorial optimization*. JOHN WILEY & SONS INC, 1997.

[7] Anaconda. https://www.anaconda.com/download/. acessado em 23/11/2017.

[8] BONDY, J. A.; MURTY, U. *Graph theory with applications*. Elsevier Science Ltd/North-Holland, 1976.

[9] GOLDBARG, M. C.; LUNA, H. P. L. *Otimização combinatória e programação linear: Modelos e algoritmos*. Rio de Janeiro: Elsevier, 2005.

[10] PRIM, R. C. Shortest connection networks and some generalizations. *Bell System Technical Journal*, v. 36, n. 6, p. 1389–1401, 1957.

[11] GOMORY, P. C. G. . R. E. Multistage cutting stock problems of two and more dimensions. *Operations Research*, v. 13, n. 1, p. 94–120, 1965.

[12] WITZIGAL, C. Mathematical methods of site selection for electronic message systems (ems). *NASA STI/Recon Technical Report N*, v. 76, June 1975.

[13] BODIN, L.; GOLDEN, B.; ASSAD, A.; BALL, M. Routing and scheduling of vehicles and crews : the state of the art. *Pergamon Press*, v. 10, n. 2, 1983.

[14] CHRISTIANSEN, M. Decomposition of a combined inventory and time constrained ship routing problem. *Transportation Science*, v. 33, n. 1, p. 1–14, 1999.

[15] FAGERHOLT, K. Evaluating the trad-off between the level of customer service and transportation cost in a ship scheduling problem. *Maritime Policy Management*, v. 27, n. 2, p. 145–153, 2000.

[16] HOEFT, J.; PALEKAR, U. S. Heuristics for the plate-cutting traveling salesman problem. *IIE Transactions (Institute of Industrial Engineers)*, v. 29, n. 9, p. 719–731, 1997.

[17] BEN-ARIEH, D.; QIAN, L. Activity-based cost management for design and development stage. *International Journal of Production Economics*, v. 83, n. 2, p. 169–183, 2003.

[18] GEOFFRION, A. Lagrangian relaxation for integer programming. *Mathematical Programming Study*, v. 2, p. 82–114, 1974.

[19] GUIGNARD, M. Lagrangean relaxation. In: *Top*. Madrid, Spain: Sociedad de Estadística e Investigación Operativa, 2003. v. 11, p. 151 – 228.

[20] POLYAK, B. T. A general method of solving extremum problems. *Doklady Akademmi Nauk SSSR (in Russian)*, v. 174, n. 1, p. 33 – 36, 1967.

[21] BRASSARD, G.; BRATLEY, P. *Algorithmics, theory & practice*. Prentice-Hall, 1988.

[22] LENSTRA, J. K.; LAWLER, E. L.; RINNOOY KAN, A. H. G.; SHMOYS, D. D. *The traveling salesman problem*. New York, USA: John Wiley & Sons, 1985.

[23] Google maps. https://www.google.com.br/maps/. acessado em 20/12/2017.

[24] My maps. https://play.google.com/store/apps/details?id=com.google.android.apps.m4b. acessado em 04/12/2017.

Capítulo 7

Apêndice

7.1 APÊNDICE A - Código em Python

Função para determinar a 1-tree mínima:

```
def onetree():
    global node,Cprime,G,mu,C
    Cprime=np.zeros((n,n))
    for u in range(n):
        for v in range(n):
            if u!=v:
                Cprime[u,v]=C[u,v]+mu[u]+mu[v]
    Gonetree=nx.Graph()
    for u,v,d in G.edges(data=True):
        Gonetree.add_edge(u,v,weight=Cprime[u,v])
    Gonetree.remove_edges_from([e for e in Gonetree.edges(node
        )])
    oneTree=nx.DiGraph()
    MST=nx.minimum_spanning_tree(Gonetree)
    length=0
    for i in range(n):
        length=length-(2*mu[i])
```

```
    mini=np.inf
    e1=e2=-1
    for u in range(n):
        if u!=node:
            if Cprime[node,u]<mini:
                mini=Cprime[node,u]
                e1=u
    oneTree.add_edge(node,e1,weight=mini)
    length=length+mini
    mini=np.inf
    for v in range(n):
        if v!=node:
            if Cprime[v,node]<mini:
                mini=Cprime[v,node]
                e2=v
    oneTree.add_edge(e2,node,weight=mini)
    length=length+mini
    for u,v,d in MST.edges(data=True):
        oneTree.add_edge(u,v,weight=d['weight'])
        length=length+d['weight']
    Degree=[]
    for i in oneTree.nodes():
        Degree.append(oneTree.in_degree(i)+oneTree.out_degree(
            i))
    return length,Degree,oneTree
```

Função para Método Subgradiente:

```
def subgradient_update(Degree,zub,zlb):
    global mu,alfa
    norma_quadrado=0
    subgradiente=[]
```

```
    for i in range(len(Degree)):
        subgradiente.append(2-Degree[i])
        norma_quadrado=norma_quadrado+((2-Degree[i])*(2-Degree
            [i]))
    if norma_quadrado>0.00001:
        tau=alfa*(zub-zlb)/norma_quadrado
        for i in range(len(mu)):
            mu[i]=mu[i]-(tau*(subgradiente[i]))
        return True
    else:
        return False
```

```
def custo_heuristico(sol_dual):
    global C
    Ch=C.copy()
    for u,v in sol_dual.in_edges()+sol_dual.out_edges():
        Ch[u,v]=0
        Ch[v,u]=0
    return Ch
```

Heurística do vizinho mais próximo, para calcular o limitante superior. Esta função retorna o custo da solução dual e as arestas do ciclo formado.

```
def nearest_neighbor_heuristic(Ch,start):
  global C,n
    A=[]
    B=[]
    for i in range(n):
        B.append(i)
        A.append(start)
    B.remove(start)
    folha1=folha2=start
    ciclo=nx.Graph()
```

```
11      while len(B)>0:
12          cand=[]
13          lista=[(Ch[folha1,i],i) for i in B]
14          candindex=lista[np.argmin([lista[i][0] for i in range(
               len(lista))])][1]
15          cand.append((Ch[folha1,candindex],candindex))
16          lista=[(Ch[i,folha1],i) for i in B]
17          candindex=lista[np.argmin([lista[i][0] for i in range(
               len(lista))])][1]
18          cand.append((Ch[candindex,folha1],candindex))
19          lista=[(Ch[folha2,i],i) for i in B]
20          candindex=lista[np.argmin([lista[i][0] for i in range(
               len(lista))])][1]
21          cand.append((Ch[folha2,candindex],candindex))
22          lista=[(Ch[i,folha2],i) for i in B]
23          candindex=lista[np.argmin([lista[i][0] for i in range(
               len(lista))])][1]
24          cand.append((Ch[candindex,folha2],candindex))
25          escolhido=np.argmin([cand[i][0] for i in range(len(
               cand))])
26          if escolhido<=1:
27              ciclo.add_edge(folha1,cand[escolhido][1],weight=C[
                   folha1,cand[escolhido][1]])
28              folha1=cand[escolhido][1]
29          if escolhido>=2:
30              ciclo.add_edge(folha2,cand[escolhido][1],weight=C[
                   folha2,cand[escolhido][1]])
31              folha2=cand[escolhido][1]
32          A.append(cand[escolhido][1])
33          B.remove(cand[escolhido][1])
34      ciclo.add_edge(folha1,folha2,weight=C[folha1,folha2])
```

```
    valor=0
    for u,v,d in ciclo.edges(data=True):
        valor=valor+d['weight']
    return valor,ciclo
```

Inicializando os valores do exemplo:

```
n=7
C=np.matrix('0 300 215 185 335 100 150; 300 0 340 425 260 270
   315;
215 340 0 350 250 185 175; 185 425 350 0 355 255 305;
335 260 250 355 0 310 325; 100 270 185 255 310 0 170;
150 315 175 305 325 170 0')
Cprime=np.zeros((n,n))
CH=C.copy()
Zlb=-np.inf
mu = np.zeros(n)
```

Função para construir o grafo completo com os dados do exemplo:

```
G=nx.complete_graph(n)
for u,v,d in G.edges(data=True):
    d['weight']=C[u,v]$
```

Utilizando os valores inicializados e as funções anteriores para obter os limitantes e, possivelmente, a solução ideal:

```
node=0
alfa=2
Zub,primal=nearest_neighbor_heuristic(C,node)
sub=True
NiterMAX=1000
Niter=0
NiterSM=0
ZMU=[]
```

```
ZUB=[]
ZLB=[]
ZPRIMAL=[]
saida_iter=open("iteracoes.txt","w")
while (sub==True) \& (Niter<NiterMAX) \& ((Zub-Zlb)>0.000001):
    Zmu,degree,OT=onetree()$ \textbf{determinar a 1-tree}
    $ZMU.append(Zmu)
    OT.edges(data=True)$ \textbf{arestas e pesos referentes a
        cada aresta}
    $saida_iter.write("Iteracao "+str(Niter)+" :\n")
    saida_iter.write("\n")
    saida_iter.write("\n")
    saida_iter.write("1-tree:\n")
    saida_iter.write("\n")
    saida_iter.write("\n")
    saida_iter.write("Z(lambda)="+str(Zmu)+"\n")
    saida_iter.write("\n")
    for u,v,w in OT.edges(data=True):
        saida_iter.write(" ("+str(u)+","+str(v)+") : "+str(w['
            weight']))
    saida_iter.write("\n")
    CH=custo_heuristico(OT)
    Zprimal,primal= nearest_neighbor_heuristic(CH, node)
    ZPRIMAL.append(Zprimal)
    saida_iter.write("ciclo heuristico\n")
    saida_iter.write("\n")
    saida_iter.write("Zprimal="+str(Zprimal)+"\n")
    saida_iter.write("\n")
    for u,v,w in primal.edges(data=True):
        saida_iter.write(" ("+str(u)+","+str(v)+") : "+str(w['
            weight']))
```

```
    saida_iter.write("\n")
    saida_iter.write("\n")
    if Zlb<Zmu:
        Zlb=Zmu
        ZLB.append(Zlb)
        NiterSM=0
    else:
        NiterSM=NiterSM+1
    if Zprimal<Zub:
        Zub=Zprimal
        ZUB.append(Zub)
    saida_iter.write("Zlb="+str(Zlb)+" Zub= "+str(Zub)+"\n")
    saida_iter.write("\n")
    saida_iter.write("\n")
    subgradient_update(degree,Zub,Zlb)
    Niter=Niter+1
    if NiterSM>10:
        NiterSM=0
        alfa=alfa/2
saida_iter.close()
```

Capítulo 8

Anexos

8.1 ANEXO A - Algoritmo de Prim

Seja, G = (V, E) um grafo e os seguintes subconjuntos: $S \subseteq E$, $V_T \subseteq V$ e $N \subseteq V$.

Algoritmo 3: Algoritmo de Prim

INÍCIO

Ler $G = (V, E)$ e $D = [d_{ij}]$ a matriz distância entre os nós vizinhos de G.

Escolha qualquer vértice $i \in V$

$V_T \leftarrow i$

$N \leftarrow V \setminus \{i\}$

Enquanto $V_T \neq V$, $\forall j \in V_T$ **faça:**

Início

Encontrar a menor aresta $\{j, k\} \in E$ tal que $j \in V_T, k \in N$

$V_T \leftarrow V_T \cup \{k\}$

$N \leftarrow N \setminus \{k\}$

$S \leftarrow S \cup \{j, k\}$

Fim

Escrever S {arestas da AGM}

FIM

8.2 ANEXO B - Algoritmo de Kruskal

Algoritmo 4: Algoritmo de Kruskal

INÍCIO

Ler $G = (V, E)$ e $D = [d_{ij}]$ a matriz distância entre os nós vizinhos de G.

Ordene as arestas de G (conjunto E) em ordem crescente das

distâncias d_{ij} no vetor $H = [h_i], i = 1, 2, ..., m$

$T \leftarrow h_1$

$i \leftarrow 2$

Enquanto $|T| < n$ **tome** $h_i \in H$ *e* **faça**

  Início

    Se $T \cup h_i$ é um grafo acíclico (árvore) **então**

      Início

        $T \leftarrow T \cup h_i$

        $i \leftarrow i + 1$

    Caso contrário

      $i \leftarrow i + 1$

  Fim

Escrever T $\{$ arestas da AGM$\}$

FIM