UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE APOIO À PESQUISA
PROGRAMA DE INICIAÇÃO CIENTÍFICA

FUNDAMENTOS MATEMÁTICOS DA OTIMIZAÇÃO COMBINATÓRIA E APLICAÇÕES

Bolsista: Thiago Parente da Silveira, FAPEAM

Manaus - Amazonas
2014

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE APOIO À PESQUISA
PROGRAMA DE INICIAÇÃO CIENTÍFICA

RELATÓRIO FINAL
PIB - E / 0048 / 2013-2014

FUNDAMENTOS MATEMÁTICOS DA OTIMIZAÇÃO COMBINATÓRIA E APLICAÇÕES

Bolsista: Thiago Parente da Silveira, FAPEAM
Orientador: Profº Dr. Sandro Dimy Barbosa Bitar

Manaus - Amazonas
2014

Resumo

Neste relatório são apresentados resultados relacionados à Otimização Combinatória com ênfase na Programação Linear Inteira e Teoria dos Grafos. Para o desenvolvimento teórico são explorados e apresentados as definições necessárias para a compreensão de resultados sobre Programação Linear.

As definições indispensáveis à compreensão dos temas principais são reproduzidas neste relatório para um perfeito entendimento dos modelos matemáticos. Os principais Teoremas e Corolários relacionados à teoria matemática necessária ao desenvolvimento do projeto são demonstrados.

Abstract

In this report we present the results related to Combinatorial Optimization, which **ênfase** in the Integer Linear Programming and Graph Theory. For the theoretical development are explored and presented the definitions necessary for the understanding of results on Linear Programming.

The settings necessary for the understanding of main subjects are reproduced in this report for a perfect understanding of the mathematical models. The main theorems and corollaries related to the mathematical theory necessary to project development are demonstrated.

Conteúdo

Lista de Figuras

Introdução

Este relatório expõe assuntos relacionados aos problemas de Otimização Combinatória. O projeto teve como principal objetivo uma investigação científica aos métodos e modelos clássicos relativos à teoria. Apresentam-se uma abordagem da Programação Linear Inteira e da Teoria dos Grafos, demonstrando Teoremas e Corolários relativos ao assunto. Como aplicação dos estudos desenvolvidos, buscar-se-à o entendimento de problemas clássicos da Otimização Combinatória, como por exemplo, o Problema do Caminho Mínimo.

O projeto foi dividido em três etapas. A primeira delas consiste no amadurecimento teórico necessário para o desenvolvimento da teoria. A segunda buscaram-se aplicações que foram beneficiadas pelos resultados obtidos na primeira etapa, em particular, os problemas clássicos da área. A terceira etapa foi reservada às simulações, implementação de algoritmo onde a confirmação dos principais resultados obtidos foram analisados. Neste relatório consolida-se o conhecimento adquirido na primeira etapa.

Capítulo 1

Introdução à Programação Linear Inteira

Neste capítulo será feita uma abordagem sobre os principais resultados de Programação Linear Inteira, mas antes será mostrado algumas definições e propriedades de um Problema de Programação Linear.

1.1 Definições e Propriedades dos Problemas de Programação Linear

Um problema de programação linear (PPL) pode ser definido sob a seguinte forma:

$$maximizar z = \sum_{j=1}^{n} c_j x_j \tag{1.1}$$

sujeito a:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1, 2, \ldots, m \tag{1.2}$$

$$x_j \geq 0, \tag{1.3}$$

onde c_j, a_{ij} e b_i são dados (números reais) e x_j representa para $j = 1, 2, \ldots, n$, as variáveis de decisão. A função linear a ser maximizada em (1.1) é denominada função objetivo ou função critério. As restrições de não negatividade (1.3) são conhecidas como triviais.

Cada restrição i de (1.2) pode ser substituída com o acréscimo de uma variável $x_{n+1} \geq 0$, denominada variável de folga, por uma restrição de igualdade e uma restrição trivial:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \Leftrightarrow \left\{ \begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j + x_{n+1} = b_i \\ x_{n+1} \geq 0 \end{array} \right.$$

ou

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \Leftrightarrow \left\{ \begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j - x_{n+1} = b_i \\ x_{n+1} \geq 0 \end{array} \right.$$

Uma restrição de igualdade poderá também ser substituída por duas desigualdades:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \Leftrightarrow \left\{ \begin{array}{l} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \\ \sum_{j=1}^{n} a_{ij} x_j \geq b_i \end{array} \right.$$

Sendo dado um problema de programação linear com restrições de igualdade e desigualdades, poderemos acrescentar variáveis de folga às desigualdades não triviais, passando dessa maneira a trabalhar com restrições de igualdades e desigualdades triviais.

Assim sendo, um problema de programação linear poderá sempre ser escrito da seguinte maneira:

$$(PPL) : maximizar z = \sum_{j=1}^{n} c_j x_j$$

sujeito a:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, i = 1, 2, \ldots, m$$

$$x_j \geq 0, j = 1, 2, \ldots, n$$

que poderá ser ainda apresentado sob a forma a seguir:

$$(PPL) : maximizar z = c^T x \tag{1.4}$$

sujeito a:

$$Ax = b \tag{1.5}$$

$$x \geq 0 \tag{1.6}$$

onde $c \in \mathbb{R}^n, x \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{mxn}$

A desigualdade (1.6) indica que cada componente do vetor x é não negativa.

Definição 1.1.1 *Seja $X = \{x \in \mathbb{R}^n / Ax = b, x \geq 0\}$. O conjunto X é denominado conjunto factível ou conjunto viável do (PPL) e se $x \in X$, então x é uma solução viável do mesmo problema. Dado $x^* \in X$, diz-se que x^* é solução ótima do (PPL) se $c^T x^* \geq c^T x$, para todo $x \in X$.*

Suponha, sem perda de generalidade, que a matriz A tenha posto m, isto é, existem m colunas linearmente independentes.

Como observação, pode-se dizer que a presença de uma variável x_j irrestrita em sinal será expressa como sendo: $x_j = x_j^+ - x_j^-$, com $x_j^+ \geq 0$ e $x_j^- \geq 0$, deixando sempre o problema na forma (PPL).

A matriz A será particionada da seguinte maneira: $A = (B, N)$, onde B é uma matriz quadrada de ordem m e inversível, pois é uma matriz quadrada de posto completo. Analogamente serão particionados os vetores x e c: $x = (x_B, x_N)$ e $c = (c_B, c_N)$, x_B e c_B terão m componentes associadas à matriz B. Dessa forma o (PPL) será escrito:

$$(PPL) : maximizar z = c_B^T x_B + c_N^T x_N \tag{1.7}$$

sujeito a:

$$Bx_B + Nx_N = b \tag{1.8}$$

$$x_B, x_N \geq 0 \tag{1.9}$$

Explicitando x_B em função de x_N em (1.8)

$$x_B = B^{-1}b - B^{-1}Nx_N \tag{1.10}$$

e fazendo $x_N = 0$ tem-se $x_B = B^{-1}b$.

Definição 1.1.2 *Diremos que $\overline{x}$ é uma solução básica de (1.5) se $\overline{x} = (x_B, 0)$. As variáveis associadas às componentes de $\overline{x}_B$ são denominadas básicas e as demais não básicas. Quando $\overline{x}_B$ possuir ao menos uma componente nula diremos que $\overline{x}$ é uma solução básica degenerada.*

No caso em que $\overline{x}_B$ for não negativo, isto é, $\overline{x}_B \geq 0$, então $\overline{x}$ satisfará a restrição (1.6). Diremos então que $\overline{x}$ é uma solução básica viável.

Sejam I_B o conjunto dos índices das colunas de A pertencendo à matriz B e I_N o conjunto dos demais índices de A. Note que $I_B \cap I_N = \emptyset$ e $I_B \cup I_N = \{1, 2, \ldots, n\}$.

Substituindo a expressão de x_B obtida em (1.10) na função objetivo (1.7), tem-se uma nova forma do (PPL):

$$(PPL) : maximizar z = c_B B^{-1}b - (c_B B^{-1}N - c_N)x_N \tag{1.11}$$

sujeito a:

$$x_B = B^{-1}b - B^{-1}Nx_N \tag{1.12}$$

$$x_B, x_N \geq 0 \tag{1.13}$$

Por comodidade, serão definidos novos parâmetros para o último (PPL):

$$\begin{gathered} u = c_B B^{-1}, u \in \mathbb{R}^m \\ \overline{x}_B = B^{-1}b, \overline{x}_B \in \mathbb{R}^m \\ z_j = u^T a_j (j \in I_B \cup I_N), z_j \in \mathbb{R} \\ y_j = B^{-1}a_j (j \in I_B \cup I_N), y_j \in \mathbb{R}^m \\ \overline{z} = c_B^T B^{-1}b = u^T b = c_B^T \overline{x}, \overline{z} \in \mathbb{R} \end{gathered}$$

Assim pode-se escrever $(c_B^T B^{-1}N - c_N)x_N = \sum_{j \in I_N}(z_j - c_j)x_j$ e o (PPL) se tornará:

$$(PPL) : maximizar z = \overline{z} - \sum_{j \in I_N}(z_j - c_j)x_j \tag{1.14}$$

sujeito a:

$$x_B = \overline{x}_B - \sum_{j \in I_N} y_j x_j \tag{1.15}$$

$$x_B \geq 0, x_j \geq 0, j \in I_N \tag{1.16}$$

Se $y_j^T = (y_{1j}, y_{2j}, \ldots, y_{mj})$, $x_j^T = (x_{B(1)}, x_{B(2)}, \ldots, x_{B(m)})$ e $\overline{x}_j^T = (\overline{x}_{B(1)}, \overline{x}_{B(2)}, \ldots, \overline{x}_{B(m)})$ então (1.15) pode ser reescrito como sendo:

$$x_{B(i)} = \overline{x}_{B(i)} - \sum_{j \in I_N} y_{ij} x_j, i = 1, 2, \ldots, m \tag{1.17}$$

Proposição 1.1.1 *Se $\overline{x}_B \geq 0$ e $z_j - c_j \geq 0$, $\forall j \in I_N$ então o vetor $x^* \in \mathbb{R}^n$, onde $x^*_{B(i)} = \overline{x}_{B(i)}$, $i = 1, 2, \ldots, m$ e $x_j^* = 0$, $j \in I_N$, será uma solução ótima do (PPL).*

Demonstração: Como $z_j - c_j \geq 0$ e $x_{j\geq 0}$, $\forall j \in I_N$, então de (1.14) tem-se $z \leq \overline{z} = c^T x^*$. O máximo de z não ultrapassará $\overline{z} = c^T x^*$ e como x^* é uma solução viável do (PPL), logo x^*é uma solução ótima do (PPL).

■

No caso da Proposição 1.1.1, x^* é uma solução básica de (1.5). Suponha agora que $\widehat{x} \in \mathbb{R}^n$ seja uma solução viável de (1.5) e (1.6), logo o será também de (1.15) e (1.16), isto é,

$$\widehat{x}_{B(i)} = \overline{x}_{B(i)} - \sum_{j \in I_N} y_{ij}\widehat{x}_j, i = 1, 2, \ldots, m \tag{1.18}$$

e $\widehat{x} \geq 0$, $j \in I_B \cup I_N$, fornecendo um valor $\widehat{z}$ à função objetivo:

$$\widehat{z} = \overline{z} - \sum_{j \in I_N} (z_j - c_j)\widehat{x}_j = c^T\widehat{x}$$

Suponha também que $\widehat{x} \in \mathbb{R}^n$ não seja uma solução básica de (1.5), isto quer dizer que haverá ao menos uma componente $\widehat{x}_j > 0$, $j \in I_N$.

Será possível passar da solução $\widehat{x}$ para uma solução x^* do (PPL) tal que $c^T x^* \geq c^T\widehat{x}$?

Para que esta pergunta seja respondida, será feito uma variação do valor de uma variável x_k, $k \in I_N$ enquanto o valor das outras variáveis cujos índices permanecem em I_N não se modificam, isto é, $x_j = \widehat{x}_j$ para $j \in I_N - \{k\}$.

De (1.18)

$$x_{B(i)} = \overline{x}_{B(i)} - \sum_{j \in I_N - \{k\}} y_{ij}\widehat{x}_j - y_{ik}x_k, i = 1, 2, \ldots, m \tag{1.19}$$

onde x_k poderá variar.

Sabe-se que $x_k \geq 0$, $x_{B(i)} \geq 0$, $i = 1, 2, \ldots, m$ e que os outros valores associados a x_j, $j \in I_N - \{k\}$ não serão modificados. Assim sendo, $x_{B(i)} \geq 0$ implica que

$$\overline{x}_{B(i)} - \sum_{j \in I_N - \{k\}} y_{ij}\widehat{x}_j - y_{ik}x_k \geq 0, i = 1, 2, \ldots, m \tag{1.20}$$

Considere L_0, L_1, L_2 uma partição de $\{1, 2, \ldots, m\}$, tal que

$$L_0 = \{i | y_{ik} = 0\},\ L_1 = \{i | y_{ik} > 0\},\ L_2 = \{i | y_{ik} < 0\}$$

Busca-se então os limites de variação para x_k pois sabe-se que de (1.20)

$$y_{ik}x_k \leq \overline{x}_{B(i)} - \sum_{j \in I_N - \{k\}} y_{ij}\widehat{x}_j, i = 1, 2, \ldots, m \tag{1.21}$$

Para $i \in L_0$, basta verificar que o valor de x_k seja não negativo.

Para $i \in L_1$:

$$x_k \leq \frac{1}{y_{ik}}(\overline{x}_{B(i)} - \textstyle\sum_{j \in I_N - \{k\}} y_{ij}\widehat{x}_j)$$

Para $i \in L_2$:

$$x_k \geq \frac{1}{y_{ik}}(\overline{x}_{B(i)} - \textstyle\sum_{j \in I_N - \{k\}} y_{ij}\widehat{x}_j)$$

Sejam

$$\alpha_k = \frac{1}{y_{sk}}(\overline{x}_{B(s)} - \sum_{j \in I_N - \{k\}} y_{sj}\widehat{x}_j) = \min_{i \in L_1}\{\frac{1}{y_{ik}}(\overline{x}_{B(i)} - \sum_{j \in I_N - \{k\}} y_{ij}\widehat{x}_j)\},$$

$$\beta k = \frac{1}{y_{lk}}(\overline{x}_{B(l)} - \sum_{j \in I_N - \{k\}} y_{lj}\widehat{x}_j) = \max_{i \in L_2}\{\frac{1}{y_{ik}}(\overline{x}_{B(i)} - \sum_{j \in I_N - \{k\}} y_{ij}\widehat{x}_j)\},$$

e $\gamma_k = \max\{0, \beta_k\}$.

Logo $\gamma_k \leq x_k \leq \alpha_k$. Quando $L_1 = \emptyset \Rightarrow \alpha_k = \infty$ e quando $L_2 = \emptyset \Rightarrow \beta_k = -\infty$

A matriz $B = (a_{B(1)}), a_{B(2),\ldots,a_{B(m)}}$ extraída de A foi utilizada para que se chegasse ao sistema (1.15) a partir de (1.5). Os vetores $a_{B(i)}$, $i = 1, 2, \ldots, m$, formam uma base do $\mathbb{R}^m$, logo existem $\lambda_i \in \mathbb{R}$, $i = 1, 2, \ldots, m$ para os quais $a_k = \sum_{i=1}^{m} \lambda_i a_{B(i)}$. Seja $s \in \{1, 2, \ldots, m\} = M$ tal que $\lambda_s \neq 0$, então $a_{B(s)} = \frac{1}{\lambda_s}(a_k - \sum_{i \in M - \{s\}} \lambda_i a_{B(i)})$, como $a_{B(s)} \neq 0$, a coluna a_k não pode ser escrita como combinação linear das colunas $a_{B(i)}$, $i \in M - \{s\}$, isto quer dizer que $\{a_{B(1)}, a_{B(2)}, \ldots, a_{B(s-1)}, a_k, a_{B(s+1)}, \ldots, a_{B(m)}\}$ também forma uma base do $\mathbb{R}^m$.

Seja $v = (\lambda_1, \lambda_2, \ldots, \lambda_m)$, assim pode-se escrever $a_k = Bv$, logo $v = B^{-1}a_k$, isto é, $v = y_k$. Basta que $y_{sk} \neq 0$ para que se possa substituir a base formada pelas colunas de B por uma outra base em que o vetor $a_{B(s)}$ é substituído por a_k. Portanto, já é possível responder à pergunta feita.

Procedimento 1

Tome x_k tal que $x_k = \widehat{x}_k > 0$ e $k \in I_N$.

1º caso: $z_k - c_k > 0$, decrescer o valor de x_k até alcançar γ_k;

se $\gamma_k = 0$, faça $x_k = 0$ e utilize (1.19) para atualizar os valores de $x_{B(i)}$, $i = 1, 2, \ldots, m$;

se $\gamma_k = \beta_k$, faça $x_k = \beta_k$ que ocasionará $x_{B(l)} = 0$ em (1.19), como $y_{lk} \neq 0$ então pode-se fazer

$$I_B := (I_B - \{B(l)\}) \cup \{k\}$$
$$I_N := (I_N - \{k\}) \cup \{B(l)\}$$

isto é, tem-se uma nova matriz B inversível, extraída de A, onde a coluna $a_{B(l)}$ será substituída por a_k;

2º caso: $z_k - c_k < 0$, será aumentado o valor de x_k até alcançar α_k;

se $\alpha_k = +\infty$, a solução do (PPL) será ilimitada, pois $x_k \longrightarrow +\infty \Rightarrow z \longrightarrow +\infty$;

se $\alpha_k < +\infty$ que ocasionará $x_{B(s)} = 0$ em (1.19), como $y_{sk} \neq 0$ então pode-se fazer

$$I_B := (I_B - \{B(s)\}) \cup \{k\}$$
$$I_N := (I_N - \{k\}) \cup \{B(s)\}$$

isto é, tem-se uma nova matriz B inversível, extraída de A, onde a coluna $a_{B(s)}$ será substituída por a_k;

3º caso: $z_k - c_k = 0$, aplica-se o que foi realizado no primeiro caso.

Fim do Procedimento 1

Para cada $j \in I_N$ tal que $\widehat{x}_j > 0$, o procedimento 1 feito para o índice k será repetido até que os valores atribuídos às variáveis x_j, $j \in I_N$ sejam nulos, ou que a solução máxima (ótima) do (PPL) seja ilimitada (2º, $\alpha_k = +\infty$).

O procedimento 1 será aplicado r vezes onde r é o número de elementos do conjunto $\{j \in I_N | \widehat{x}_j > 0\}$, ou seja, $r = |\{j \in I_N | \widehat{x}_j > 0\}|$.

Proposição 1.1.2 *Se* (1.5), (1.6) *admitirem uma solução viável, então haverá ao menos uma solução básica de* (1.5) *satisfazendo* (1.6).

Demonstração: De fato, basta utilizar o primeiro caso do procedimento 1 para qualquer $z_k - c_k \in \mathbb{R}$, $k \in I_N$, $\widehat{x}_j > 0$.

■

Proposição 1.1.3 *Se o (PPL) possuir ótimo finito, ao menos uma solução ótima será básica viável.*

Demonstração: De fato, basta aplicar o procedimento 1 r vezes, onde r é o número de elementos do conjunto $\{j \in I_N | \widehat{x}_j > 0\}$, ou seja, $r = |\{j \in I_N | \widehat{x}_j > 0\}|$.

■

1.2 Método Simplex

A ideia do método é a partir de uma solução básica de (1.5) satisfazendo (1.6), isto é, uma solução básica primal viável, passar para outra solução básica primal viável sem que o valor da função objetivo diminua (no caso de maximização). Como o número de soluções básicas é finito, o algoritmo, sob algumas condições irá convergir.

Dada a matriz B quadra e inversível, extraída de A, tal que $\overline{x}_B \geq 0$, será colocado o problema de programação linear (1.4), (1.5) e (1.6) sob a forma (1.14), (1.15) e (1.16). Utilizando a Proposição 1.1.1 será testado se esta solução é ótima, caso não o seja tentar-se-à aumentar o valor de uma variável x_k, $k \in I_N$ tal que $z_k - c_k < 0$, como já foi explicado no segundo caso do procedimento 1. Se $\alpha_k = +\infty$ então não haverá ótimo finito.

No caso em que se irá trabalhar com soluções básicas viáveis, o cálculo de α_k é mais simplificado:

$$\alpha_k = \frac{\overline{x}_{B(s)}}{y_{sk}} = \min_{i \in L_1} \left\{ \frac{\overline{x}_{B(i)}}{y_{ik}} \right\}$$

e caso $L_1 = \emptyset$ faz-se $\alpha_k = +\infty$

A seguir será apresentado um procedimento que resume o método simplex.

Procedimento 2 (maximização)

Dada uma solução básica primal viável do (PPL).

Se $z_j - c_j \geq 0$, $\forall j \in I_N$, a solução dada é uma solução ótima. PARE.

Caso contrário, escolhe-se um $k \in I_N$ para o qual $z_k - c_k < 0$;

se $\alpha_k = +\infty$, a solução do (PPL) é ilimitada. PARE.

se $\alpha_k < +\infty$, faça $x_k = \alpha_k$, acarretando $x_{B(s)} = 0$, a coluna α_k ocupará o lugar da coluna $\alpha_{B(s)}$ em B. MUDANÇA DE BASE.

Fim do Procedimento 2.

Para cada nova base o procedimento 2 é repetido até que uma regra de parada seja verificada. Este procedimento possui duas saídas: solução ótima encontrada ou solução ilimitada.

1.3 Programação Linear Inteira

Quando nos problemas de programação linear for obrigado que algumas ou todas as variáveis de decisão só admitam valores inteiros, se tratará de um problema de *programação linear inteira.*

Exemplo 1:

$$(P_0)\text{: maximizar } z = x_1 - 3_x 2 - 4x_3$$

sujeito a:

$$2x_1 + x_2 - x_3 \leq 4$$
$$4x_1 - 3x_2 \leq 2$$
$$3x_1 + 2x_2 + x_3 \leq 3$$
$$x_1, x_2, x_3 \geq 0$$
$$x_2 \text{ e } x_3 \text{ inteiros}$$

O problema *(P_0)* restringe x_2 e x_3 a valores inteiros e não negativos, enquanto que x_1 é um real qualquer não negativo. Pode-se dizer também que *(P_0)* é um problema de *programação linear inteira mista*, pois nem todas as variáveis são restritas a valores inteiros.

Poderia-se imaginar a solução de *(P_0)* ignorando-se as restrições que tornam as variáveis inteiras, fazendo isto o problema seria considerado como sendo de programação linear, visto na seção anterior. Caso a solução obtida fornecesse valores inteiros para as variáveis x_2 e x_3 teria sido também resolvido o problema original *(P_0)*. No entanto, se pelo menos uma dessas variáveis tivesse assumido valor não inteiro na solução do problema de programação linear, quando deveriam ser inteiras, a primeira ideia é tentar arredondar esse (s) valor (es) para o inteiro mais próximo de maneira que as soluções inteiras obtidas sejam viáveis. Infelizmente este procedimento poderá fornecer soluções inteiras que não são o ótimo do problema.

A formulação geral do problema é dada por:

$$\text{(PPLI): maximizar } z = c^T x$$

sujeito a:

$$Ax = b$$
$$x \geq 0$$
$$x_i \in \mathbb{Z},\ i = 1, 2, \ldots, n$$

onde $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{mxn}$

1.3.1 Branch and Bound

O método denominado de *Branch and Bound* baseia-se na ideia de desenvolver uma enumeração inteligente dos pontos candidatos à solução ótima de um (PPLI). O termo *branch* refere-se ao fato de que o método efetua partições no espaço das soluções. O termo *bound* ressalta que a prova da otimalidade da solução utiliza-se de limites calculados ao longo da enumeração. Definindo:

$$(P) = \text{Maximizar } \{c^T x | Ax = b, x \geq 0, x \in \mathbb{Z}\}$$
$$(\overline{P}) = \text{Maximizar } \{c^T x | Ax = b, x \geq 0, x \in \mathbb{R}\}$$

Definindo ainda x_P^* e $x_{\overline{P}}^*$ como sendo as soluções de (P) e $(\overline{P})$ respectivamente, tem-se que:

$$c^T x_P^* \leq c^T x_{\overline{P}}^*$$

Considerando ainda que qualquer solução viável y de (P), então:

$$c^T y \leq c^T x_P^*$$

e dessa forma $c^T x_{\overline{P}}^*$ é um limite superior para (P) e qualquer de suas soluções viáveis. Se $\overline{x}$ é a solução ótima de $(\overline{P})$ tal que $\overline{x}_j$ não é inteiro, definindo $\lfloor \overline{x}_j \rfloor$ o maior inteiro menor do que $\overline{x}_j$ serão formados dois novos problemas, acrescentando uma restrição em cada problema.

$$(P_1)\text{: maximizar } z = c^T x$$

sujeito a:

$$\begin{gathered} Ax = b \\ x_j \leq \lfloor \overline{x}_j \rfloor \\ x \geq 0 \\ x_i \in \mathbb{R},\ i = 1, 2, \ldots, n \end{gathered}$$

e

$$(P_2)\text{: maximizar } z = c^T x$$

sujeito a:

$$\begin{gathered} Ax = b \\ x_j \geq \lfloor \overline{x}_j \rfloor + 1 \\ x \geq 0 \\ x_i \in \mathbb{R},\ i = 1, 2, \ldots, n \end{gathered}$$

A estratégia de separação cria novos e mais restritos problemas que, normalmente, serão de mais fácil solução. Por exemplo, $(\overline{P})$ (que é a relaxação de (P)) foi separado em dois problemas (P_1) e (P_2). Para se obter uma solução onde todas as variáveis são inteiras, bastar repetir o procedimento até que todas as variáveis na solução ótima sejam inteiras e o problema ainda seja viável.

Exemplo 2:

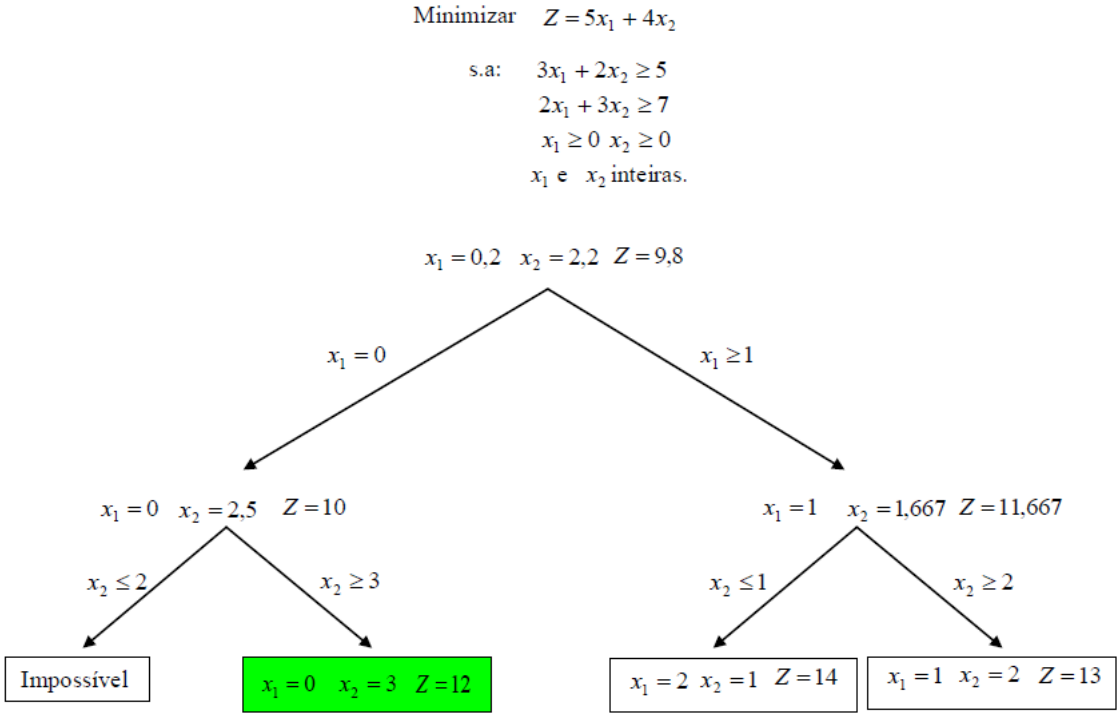

Figura 1.1: Solução do Exemplo 2 utilizando Branch and Bound

Solução: $x_1 = 2$, $x_2 = 0$. Valor ótimo: $Z = 6$.

Exemplo 3:

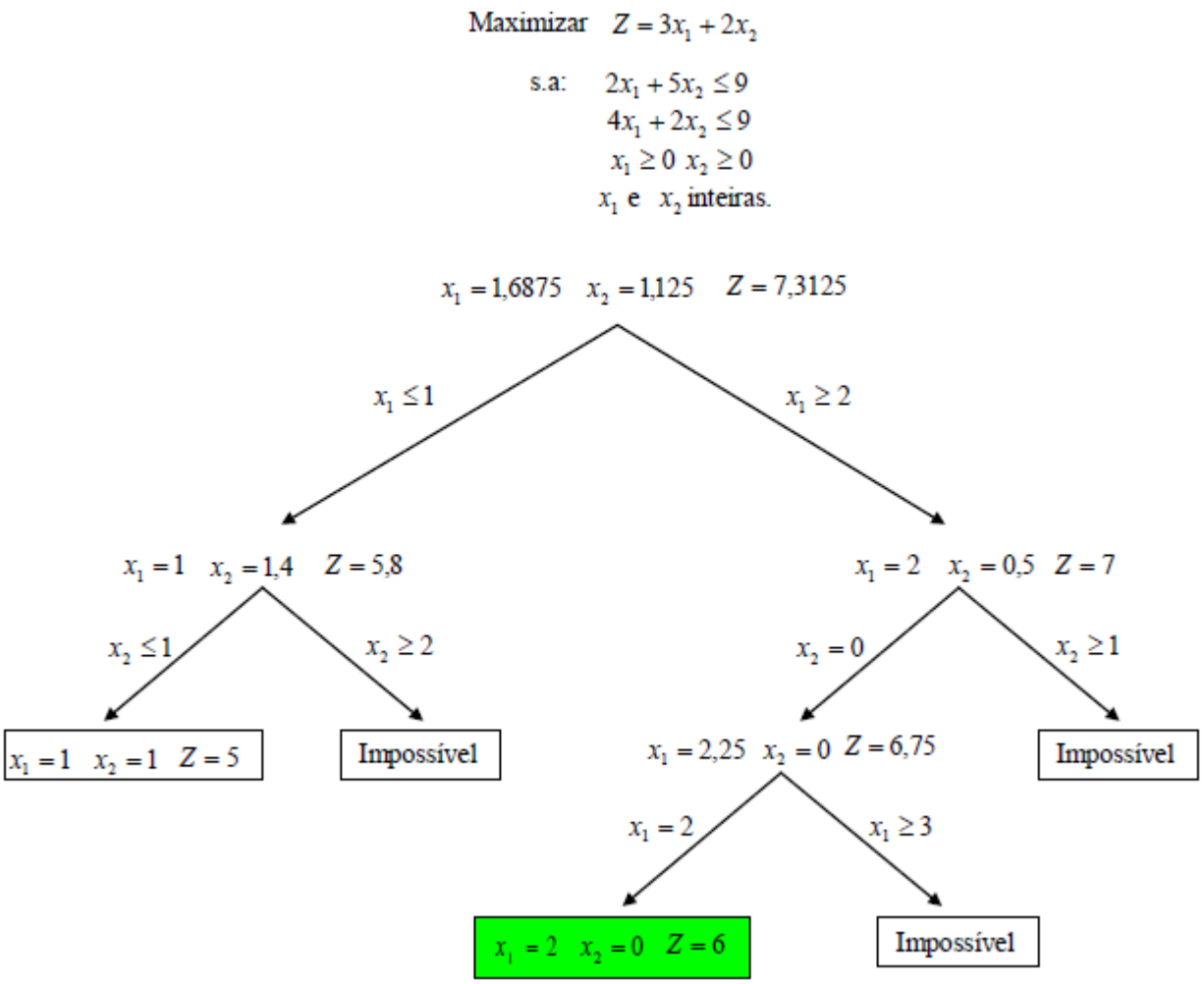

Figura 1.2: Solução do Exemplo 3 utilizando Branch and Bound

Solução: $x_1 = 2$, $x_2 = 0$. Valor ótimo: $Z = 6$.

Capítulo 2

Grafos

Neste capítulo, será feito embasamento teórico sobre a *Teoria dos Grafos*, com o intuito de ver que vários problemas de Otimização Combinatorial podem ser modelados como problemas investigados pela Teoria dos Grafos.

2.1 Definições e Propriedades da Teoria dos Grafos

Definição 2.1.1 *Um grafo G consiste de um conjunto V de elementos, que são denominados vértices, um conjunto A de elementos denominados por arestas, e uma função de incidência ψ que associa a cada aresta α de G um par não ordenado de vértices (não necessariamente distintos) de G, chamados de extremidades de α.*

O número de vértices de um grafo será simbolizado por $|V|$ ou por n, e o número de arestas será denotado por $|A|$ ou por m. Quando não houver risco de confusão, será denotado por $V(G)$ o conjunto dos vértices do grafo G e por $A(G)$ o conjunto de arestas de G.

Grafos podem ser representados por diagramas, onde cada vértice é representado por um ponto e cada aresta por uma linha ligando os pontos que representam as suas extremidades. As extremidades de uma aresta são *incidentes* à aresta, e vice-versa. Os extremos de uma aresta são *adjacentes* (mesmo que coincidam). Arestas com pelo menos um extremo em comum também são ditas *adjacentes*.

Definição 2.1.2 *Uma aresta é chamada de laço quando incide duas vezes no mesmo vértice, caso contrário diz-se que se trata de uma ligação.*

Definição 2.1.3 *Um multigrafo é um grafo onde existem pelo menos dois vértices que são ligados por mais de uma aresta.*

Com base nas definições (2.1.2) e (2.1.3) será definido um tipo de grafo na qual será baseado a maior parte desse capítulo.

Definição 2.1.4 *Um grafo simples é um grafo que não tem laços e nem é um multigrafo.*

Definição 2.1.5 *O grau de um vértice v é o número de arestas que incidem em v e será denotado por $d(v)$.*

Teorema 2.1.1 *Para todo grafo G, a soma dos graus de seus vértices é o dobro do número de suas arestas, ou seja,*

$$\sum_{v \in V(G)} d(v) = 2|A|$$

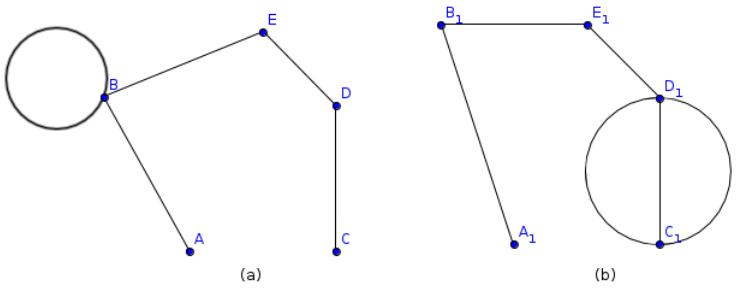

Figura 2.1: (a) Grafo com um laço no vértice B
b) Multigrafo com arestas mútiplas entre os vértices C_1 e D_1

Demonstração: Quando soma-se os graus dos vértices, contam-se as extremidades das arestas uma vez. Como cada aresta tem duas extremidades, cada aresta é contada duas vezes. Note que este fato é independente do grafo possui laços ou não. De fato, se G possui um laço em x_0, então a aresta incide duas vezes no vértice, acrescentando o grau do vértice em dois.

■

Corolário 2.1.1 *Todo grafo G possui um número par de vértices com grau ímpar.*

Demonstração: Se o número de vértices com grau ímpar fosse ímpar a soma dos graus de todos os vértices seria ímpar. Mas a soma dos graus é o dobro do número de arestas e, portanto é um número par.

■

Definição 2.1.6 *Um grafo é dito completo quando qualquer par de vértices é ligado por uma aresta. Um grafo completo com n vértices será denotado por K_n.*

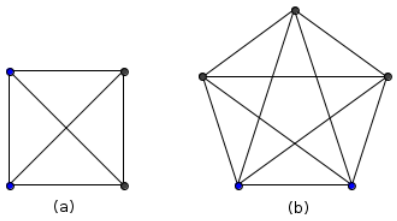

Figura 2.2: (a) Grafo completo com 4 vértices, ou seja, um K_4
(b) Grafo completo com 5 vértices, ou seja, um K_5

Proposição 2.1.1 *Um K_n possui $\binom{n}{2}$ arestas.*

Demonstração: De fato, sendo G um K_n, o grau de cada vértice é $n-1$. Sejam, $v_1, v_2, \ldots, v_n$ os vértices de G Pelo Teorema 2.1.1 tem-se que

$$\sum_{i=1}^{n} d(v_i) = 2|A|$$

ou seja,

$$n(n-1) = 2|A| \Rightarrow |A| = \frac{n(n-1)}{2} = \binom{n}{2}$$

■

Definição 2.1.7 *Um passeio em um grafo G é uma sequência de vértices $v_1, v_2, \ldots, v_k$ tal que o vértice v_i é adjacente ao vértice v_{i+1}, ou seja, quaisquer dois vértices consecutivos na sequência tem que serem ligados por uma aresta. Em um passeio pode-se repetir um ou mais vértices. Se o primeiro vértice do passeio coincide com o último diz-se que se trata de um passeio fechado, caso contrário trata-se de um passeio aberto.*

Definição 2.1.8 *Um caminho é um passeio onde não há vértices repetidos. O primeiro e o último vértice do caminho são chamados extremidades do caminho. Se estes forem o mesmo, diz-se que se trata de um ciclo. O número de arestas em um caminho é chamado comprimento. Um ciclo de comprimento k é denotado por k-ciclo. Um caminho também pode ser definido como sendo um ciclo sem uma das arestas.*

Definição 2.1.9 *Um grafo $H = (U, B)$ é dito subgrafo de $G = (V, A)$ se $U \subseteq V$ e $B \subseteq A$*

Uma noção importante na Teoria dos Grafos, é a de *grafo conexo.*

Definição 2.1.10 *Um grafo G é dito conexo se dados dois vértices quaisquer u e v, existe um caminho com extremidades u e v, ou seja, se existe um caminho ligando estes dois vértices. Caso contrário se trata de um grafo desconexo.*

Se um grafo G for desconexo, então G terá subgrafos conexos; por exemplo, o subgrafo consistindo de um único vértice v_0 é conexo. Um *componente conexo* H é um subgrafo maximal que é conexo, em outras palavras, H é um componente conexo se ele é conexo mas qualquer outro subgrafo de G que contem H é desconexo.

Talvez o resultado mais antigo em teoria dos grafos tenha sido descoberto por Euler, considerado o maior matemático do século $XVIII$.

Tudo começou com um desafio recreativo que os cidadãos de Konisberg (hoje: Kaliningrado) propuseram. A cidade era dividida em quatro distritos por braços do rio Pregel, que eram conectados por sete pontes. Era agradável caminhar por ali, atravessando as pontes, e daí surgiu a questão: é possível um passeio de modo que se atravessasse todas as pontes exatamente uma vez?

Euler publicou um artigo em 1736, no qual ele demonstrava que tal passeio era impossível. Euler concluiu que se tal passeio existisse, ele tinha que começar em um distrito e necessariamente teria que terminar em outro, mas no total haviam quatro distritos, o que não tornava possível tal passeio. Este resultado é geralmente considerado como o primeiro teorema da teoria dos grafos.

Definição 2.1.11 *Um grafo G conexo com m arestas é dito euleriano se existe um passeio fechado com comprimento m, em outras palavras, um grafo é euleriano se saindo de um vértice, passa-se por todas as arestas exatamente uma vez e ao mesmo vértice retorna-se. Se o grafo não é euleriano, mas tem um passeio aberto que passe por todas as arestas exatamente uma vez, ele é dito semi-euleriano.*

Proposição 2.1.2 *Se todo vértice de um grafo (não necessariamente simples) possui grau maior do que ou igual a 2, então o grafo contem um ciclo.*

Demonstração: Se o grafo contem um laço ou arestas múltiplas, não há o que demonstrar. Suponha que se trata de um grafo simples. Seja v_0 o vértice na qual se inicia um passeio. Ao sair de v_0 chegamos em v_1. Como v_1 tem grau maior do que ou igual a 2, tem-se que sair de v_1. Repetindo este procedimento, ao chegar em um vértice v_k, tem-se duas opções: ou se está retornando a v_k, criando assim um ciclo, ou então, está se chegando pela primeira vez em v_k, mas como $d(v_k) \geq 2$ tem-se que sair de v_k. Como o número de vértices é finito, repetindo este procedimento mais algumas vezes, uma hora terá que se retornar a algum vértice na qual já se passou, criando assim um ciclo.

■

Teorema 2.1.2 *Um grafo é euleriano se e somente se todo vértice tem grau par.*

Demonstração: Seja G um grafo euleriano com m arestas. Cada vez que o passeio passa por um vértice, ele utiliza duas arestas, uma para entrar e outra para sair. Logo o grau de cada vértice deve ser obrigatoriamente par.

Suponha que todos os vértices de G tenham grau par. Sendo G conexo todos os vértices tem que ter grau maior do que ou igual a 2. Pela Proposição 2.1.2, G contem um ciclo (que é um passeio fechado). Dentre todos os passeios fechados em G, escolhe-se o um passeio T que tenha comprimento máximo. Se T tem comprimento m, o Teorema está demonstrado. Suponha que isso não aconteça. Considere agora o grafo H resultante da retirada das arestas de T. Como retira-se um número par de arestas de cada vértice de T, e todos os vértices do grafo tem grau par, pelo menos uma das componentes de H tem um vértice em comum com T e todos os seus vértices tem grau par, assim sendo H tem uma trilha fechada que passa por todos os seus vértices, e pode-se então formar uma trilha maior concatenando T com a trilha H. Mas isto contraria a maximalidade de T.

■

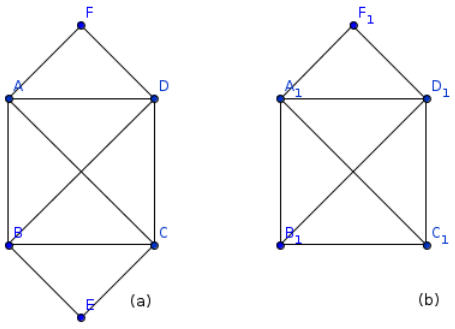

Figura 2.3: (a) Grafo euleriano
(b) Grafo semi-euleriano

Uma questão semelhante ao problema das Pontes de Konisberg foi levantada por outro famoso matemático, o irlandês William R. Hamilton em 1856. Um *ciclo hamiltoniano* é um ciclo que contem todos os nós de um grafo. O Problema do Ciclo Hamiltoniano é o problema de se decidir se dado um grafo tem ou não um ciclo hamiltoniano.

Ciclos hamiltonianos parecem bem semelhantes a passeios eulerianos: ao invés de exigir que toda aresta seja usada uma vez, deseja-se que todo nó seja usado exatamente uma vez. Mas muito menos se conhece sobre eles do que grafos eulerianos. Euler mostrou como decidir se um grafo é ou não euleriano, mas nenhuma maneira eficiente é conhecida para se verificar se um grafo tem ou não um ciclo hamiltoniano, e nenhuma condição necessária e suficiente para existência de um ciclo hamiltoniano é conhecida.

2.2 Árvores

Definição 2.2.1 *Um grafo $G = (V, A)$ é chamado de árvore se ele é conexo e não contém qualquer ciclo como subgrafo.*

A árvore mais simples tem um vértice e nenhuma aresta. A segunda árvore mais simples contém dois vértices e uma aresta ligando eles.

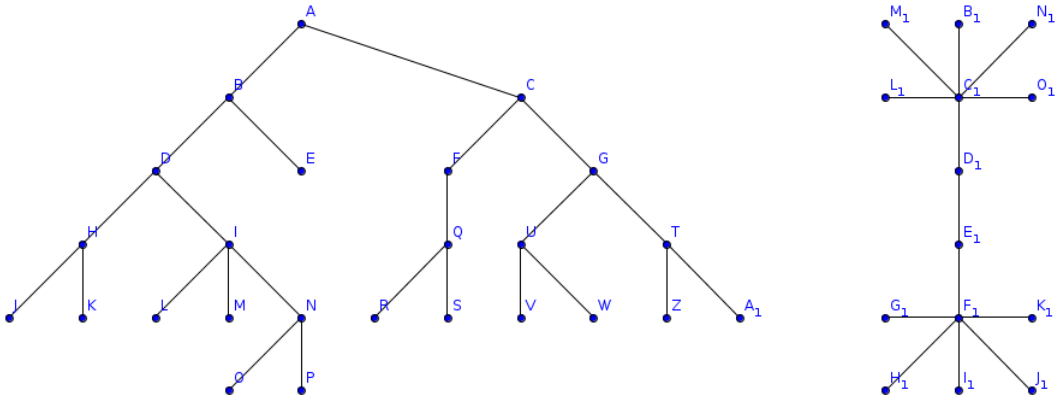

Figura 2.4: Árvores

Note que as duas propriedades que definem árvores funcionam em direções opostas: conectividade significa que o grafo não pode ter "arestas de menos"e não ter ciclos significa que o grafo não pode ter "arestas demais". Para ser mais preciso, se um grafo é conexo, então ao adicioná-lo uma nova aresta ele permanece conexo, entretanto ao remover uma de suas arestas, o grafo pode permanecer ou não conexo. Se um grafo não contém qualquer ciclo, então ao remover qualquer aresta o grafo permanecerá sem ter ciclos, mas ao adicionar um nova aresta ao grafo, este pode ter ou não um ciclo.

Teorema 2.2.1 *Um grafo G é uma árvore se e somente se ele é conexo, mas a remoção de qualquer de suas arestas resulta em um grafo desconexo.*

Demonstração : Suponha que G é uma árvore. Então pela definição de árvore G é conexo. Deseja-se provar então que após a remoção de qualquer aresta, ele não permanece conexo. Assuma que ao remover uma aresta uv do grafo G ele permaneça conexo. Então o grafo resultante terá um caminho P entre os vértices u e v, mas se for colocada a aresta uv de volta, o caminho P e a aresta uv formará um ciclo em G, o que contradiz a definição de árvores.

Suponha agora que G é um grafo conexo tal que a remoção de qualquer aresta resulta em um grafo desconexo. Suponha que G não é uma árvore. Suponha também que G contenha um ciclo C. Então ao remover qualquer aresta de C, o grafo resultante ainda é conexo, uma contradição. Portanto G é uma árvore.

■

Teorema 2.2.2 *Um grafo G é uma árvore se e somente se ele não contém nenhum ciclo, mas a adição de qualquer nova aresta cria um ciclo.*

Demonstração : Suponha que G é uma árvore. Considere dois vértices u e v. Como G é conexo, então existe um caminho P entre u e v. Ao adicionar uma nova aresta ligando u e v então o grafo G terá um ciclo formado por P e por uv. Como u e v foram tomados arbitrariamente, a adição de qualquer aresta forma um ciclo.

Agora suponha que G é um grafo sem ciclos, mas que a adição de qualquer aresta cria um ciclo. Sejam u e v dois vértices quaisquer. Se for adicionada uma aresta ligando u e v, por hipótese G terá um ciclo. Assim sendo, sebe-se que um caminho é um ciclo sem uma das arestas. Então, ao retirar a aresta uv tem-se que existe um caminho entre u e v, logo G é conexo, pois u e v foram tomados arbitrariamente. Como G não contém ciclos e é conexo, G é uma árvore.

■

Considere um grafo conexo G sobre n vértices, e uma aresta α desse grafo. Ao remover a aresta α deste grafo ele pode permanecer ou não conexo. Se o grafo ficar desconexo, então diz-se que a aresta α é uma *aresta de corte* ou então *ponte*. Pelo Teorema 2.2.1 percebe-se que toda aresta de uma árvore é uma ponte.

Definição 2.2.2 *Seja G um grafo. Um subgrafo de G com o mesmo conjunto de vértices que é também uma árvore é dito árvore geradora de G.*

Um dos processos para se encontrar uma árvore geradora de um grafo conexo é o seguinte: se existe uma aresta que não é uma ponte, então a remova. Repita este procedimento até que a remoção de qualquer outra aresta resulte em um grafo desconexo.

Frequentemente usam-se árvores que tem um nó especial, que se chama *raiz*. Pode-se tomar qualquer árvore, selecionar qualquer um de seus vértices, e chamá-lo de raiz. Uma árvore com uma raiz determinada é chamada *árvore enraizada*.

Teorema 2.2.3 *Toda árvore com pelo menos dois vértices tem pelo menos dois vértices de grau 1.*

Demonstração: Seja G uma árvore com pelo menos dois vértices. Começa-se um passeio de um vértice v_0 de grau 1 da árvore, que existe, pois caso não existisse, todos os vértices teriam grau maior do que ou igual a dois, visto que G é conexo, pela Proposição 2.1.2 G teria um ciclo, o que não pode pois G é uma árvore. Esse passeio é feito de tal maneira que não deseja-se voltar de um vértice sobre a aresta na qual se entrou nesse vértice; isso é possível, a menos que se tome um vértice de grau 1, e nesse caso a demonstração estaria concluída. Suponha que isso não aconteça, então em algum momento, tem-se que retornar a um vértice que já foi visitado, pois o número de vértices é finito. Ora, então os vértices e arestas percorridos entre as duas visitas formam um ciclo, uma contradição, pois G é uma árvore e não contém ciclos.

■

O seguinte procedimento é conhecido como *Procedimento de Crescimento de Árvore*, ou simplesmente P.C.A.

- *Comece com apenas um vértice.*
- *Repita o que se segue um número qualquer de vezes: dada uma etapa k do processo, na próxima etapa crie um novo vértice e ligue-o por uma nova aresta a qualquer vértice de G.*

Teorema 2.2.4 *Todo grafo obtido pelo Procedimento de Crescimento de Árvore é uma árvore, e toda árvore pode ser obtida dessa maneira.*

Demonstração: Seja G um grafo obtido pelo P.C.A. Para apenas um vértice, tem-se uma árvore. O que será mostrado é que nunca será criado um grafo que não seja uma árvore, em outras palavras, se G é uma árvore, e G' for obtido de G criando-se um novo vértice v e conectando-o a um vértice u de G, então G' é uma árvore. De fato, pois G' é conexo, já que quaisquer dois vértices de G podem ser conectados por um caminho em G (G é uma árvore), enquanto que v pode ser conectado a qualquer outro vértice w primeiro indo a u e depois conectando u a v tem-se então um caminho entre w e v. Além do mais, G não tem ciclo: v tem grau 1 e portanto nenhum ciclo pode passar por v, mas um ciclo que não passa por v seria um ciclo em G, o que não pode acontecer pois G é uma árvore. Logo G' também é uma árvore.

Seja G uma árvore. Se o número de vértices é 1, então a árvore surge por construção, pois essa é a maneira na qual se inicia o P.C.A. Assuma que G é uma árvore com pelo menos dois vértices. Assim pelo Teorema 2.2.3, G tem um vértice de grau 1 (pelo menos dois na verdade). Seja v um vértice com grau 1 de G, juntamente com a aresta com extremidade v, para obter o grafo G'. Afirma-se que G' é uma árvore. De fato, G' é conexo: quaisquer dois vértices de G' podem ser conectados por um caminho de G, e esse caminho não pode passar por v, pois v tem grau 1. Portanto esse caminho também é um caminho em G'. Além do mais, G' não contém nenhum ciclo pois G não contém. Assuma que toda árvore com menos vértices que G surge pela construção, em particular G' assim o faz. Mas então G surge de G', por uma iteração a mais do segundo passo, completando a prova do teorema.

■

O Procedimento de Crescimento de Árvore pode ser usado para estabelecer propriedades das árvores. Talvez a mais importante delas concerne o número de arestas. O número de arestas de uma árvore depende somente do número de vértices.

Proposição 2.2.1 *Toda árvore que tem n vértices possui $n-1$ arestas.*

Demonstração: De fato, começa-se com um vértice e nenhuma aresta, e a cada passo, um novo vértice e uma nova aresta são adicionados, e essa diferença de 1 é mantida.

■

Uma questão bastante natural com respeito ao estudo de árvores é a seguinte: quantas árvores existem sobre n vértices? Antes de responder a essa questão, outra deve ser respondida: quando que duas árvores são diferentes?

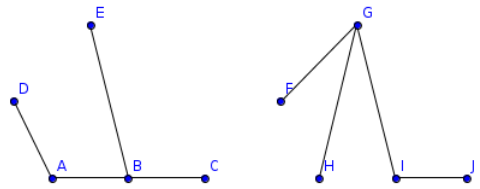

Figura 2.5: Árvores iguais

Definição 2.2.3 *Sejam $G = (V, A)$ e $H = (U, B)$ dois grafos. Um isomorfismo entre os grafos G e H é uma aplicação $\varphi : V \to U$, tal que $\{c, d\} \in A \Rightarrow \{\varphi(c), \varphi(d)\} \in U$.*

Consideram-se duas árvores iguais se pode-se rearranjar os vértices de uma árvore de modo que se obtém a outra árvore. Mais exatamente, consideram-se duas árvores iguais quando há um *isomorfismo* entre as duas árvores, ou seja, uma correspondência um pra um entre os vértice de uma árvore com os vértices da outra árvore que preserve as adjacências. Árvores *rotuladas* são aquelas em que se fixam os vértices, dando nomes aos mesmos. Se estiver trabalhando com árvores *não rotuladas*, árvores isomorfas não possuem distinção alguma.

2.2.1 Armazenamento de árvores

Suponha que por algum motivo apareça a necessidade de armazenar uma árvore rotulada, por exemplo a árvore da Figura 2.6. A maneira na qual se realizará este processo depende de quais dados deseja-se recuperar e com que frequência. No momento, a única preocupação será com a quantidade de memória necessária para tal. Deseja-se então armazenar uma árvore de tal maneira que ela ocupe o mínimo de espaço possível no computador.

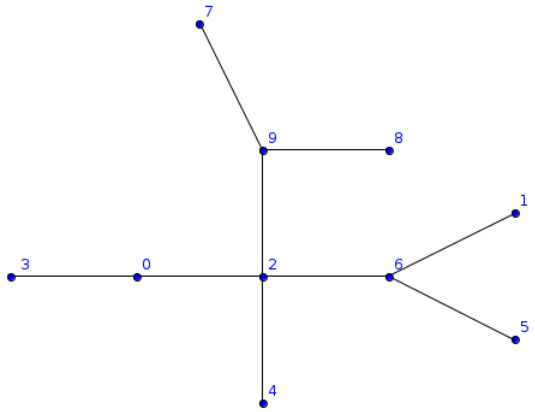

Figura 2.6: Árvore rotulada

Suponha que se tem uma árvore G sobre n vértices. Uma maneira de se armazenar esta árvore é usando uma matriz binária, conhecida como *matriz de adjacência*, definida da seguinte maneira: é uma matriz quadrada de ordem n onde a i-ésima linha e a i-ésima coluna representam o mesmo vértice, e a entrada $A_{ij} = 0$ se o vértice i não está ligado ao vértice j e $A_{ij} = 1$ se existe uma aresta entre os vértices i e j. Claramente esta é uma matriz simétrica com 0 na diagonal principal pois G é uma árvore e não possui laços. Precisa-se de um bit para armazenar cada entrada desta matriz, portanto isso necessita de n^2 bits. Pode-se economizar um pouco usando apenas a parte abaixo da diagonal principal (pois a matriz é simétrica e possui 0 na diagonal principal), no entanto isto ainda necessita de $\frac{n^2-n}{2}$ bits.

Uma maneira melhor é especificar cada árvore listando todas as suas arestas. Pode-se especificar cada aresta por suas extremidades. Será conveniente arranjar essa lista em um vetor cujas colunas representam essas arestas. Por exemplo, a árvore da Figura 2.6 pode ser codificada por:

$$\begin{matrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{matrix}$$

Ao invés de uma matriz com n linhas, obtém-se uma matriz com duas linhas. Paga-se um pouco por isso, pois passa-se a trabalhar com uma matriz cujas entradas são inteiros entre 0 e $n-1$ em vez de uma matriz binária. Mas isto certamente vale a pena: mesmo se forem contados os bits, escrever o rótulo de um vértice toma $log_2 n$ bits, portanto a matriz inteira ocupa apenas $2nlog_2 n$ bits, o que é muito menor do que $\frac{n^2-n}{2}$ se n é grande (e geralmente é).

Código de pai. De agora em diante, o vértice com rótulo 0 terá um papel especial. Será considerado esse vértice como o vértice "raiz"da árvore. Então pode-se listar os dois vértices extremos de uma aresta listando a extremidade mais distante da raiz primeiro e depois a extremidade mais próxima da raiz. Portanto para toda aresta, o vértice escrito abaixo é o pai do vértice escrito acima. Para a ordem na qual listam-se as arestas, tomar-se-à a ordem de seus primeiros vértices. Para a árvore na Figura 2.6, obtém-se a matriz

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{matrix}$$

Algo especial nessa matriz é o fato de que na primeira linha aparecem os inteiros de 1 até $n-1$ exatamente ordenados. Por conseguinte, sabe-se de antemão que se tiver uma árvore sobre n vértices, e esta for armazenada usando o código de pai, então na primeira linha da matriz consistirá de $1, 2, \ldots, n-1$. Com isso, pode-se suprimir a primeira linha sem perder qualquer informação e armazenar apenas a segunda linha. Portanto pode-se especificar a árvore por uma lista de $n-1$ números, cada um entre 0 e $n-1$. Isso toma $(n-1)log_2 n$ bits. Essa codificação não é ótima, no sentido de que nem todo código retorna uma árvore, mas esse método é quase ótimo.

Agora será descrito um método parecido com o código de pai, este é chamado *código de Prüfer*, que atribuirá a qualquer árvore rotulada de n vértices uma lista de comprimento $n-2$ e não $n-1$, consistindo dos números $0, 1, \ldots, n-1$.

O código de Prüfer pode ser considerado um refinamento do código de pai. Ainda será considerado 0 como raiz, e ordena-se as duas extremidades de uma aresta tal que filho vem primeiro, mas ordenam-se as arestas (as colunas da matriz) não pela magnitude de sua primeira extremidade porém um pouco diferentemente, mais diretamente relacionado à própria árvore.

Portanto, novamente será construída uma matriz com duas linhas cujas colunas correspondem as arestas, e cada aresta é listada de modo que o vértice mais distante da raiz esteja no topo e seu pai embaixo. A questão é a ordem na qual listam-se as arestas.

A regra é a seguinte: procura-se por um vértice de grau 1 (esses vértices são conhecido como folhas, exceto a raiz quando a mesma possui grau 1), com o menor rótulo, e escreve-se essa aresta com essa extremidade. Com base na Figura 2.6, a primeira coluna será formada pelos vértices 1 e 6 respectivamente. Então, remove-se este vértice e a aresta da árvore e repete-se o argumento até que todas as arestas sejam listadas. A matriz que obtém-se é chamada de *código estendido de Prüfer* da árvore (chama-se estendido porque, como será mostrado, precisa-se apenas de uma parte desse código como Prüfer "real"). O código de Prüfer da árvore da Figura 2.6 é

$$\begin{matrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{matrix}$$

Algo que ainda não está claro é o fato de que o código de Prüfer é melhor que o código de pai. Mas uma primeira observação é que a última entrada da segunda linha da matriz é sempre 0, pois este vértice é a raiz e é sempre o último a ser atingido. Mas aparentemente não há vantagens, pois na primeira linha guarda-se os números $1, 2, \ldots, n-1$ também, mas de maneira não ordenada. Apesar de não ser visível, a primeira linha é um tanto supérflua, e pode ser determinada pela segunda linha, como diz a Proposição 2.2.2.

Proposição 2.2.2 *A segunda linha de um código de Prüfer determina a primeira.*

Demonstração: De fato, pois cada entrada na primeira linha do código estendido de Prüfer é o menor inteiro que não ocorre na primeira linha antes dele, nem na segunda linha abaixo ou depois dele, pois quando essa entrada (por exemplo a k-ésima entrada da primeira linha)

foi registrada, então os vértices antes dela na primeira linha foram removidos (juntamente com as arestas correspondentes às $k-1$ colunas). As entradas remanescentes na segunda coluna são exatamente aqueles vértices que são pais ao mesmo tempo, o que significa que eles não são folhas. Isto descrever como a primeira linha pode ser reconstruída a partir da segunda.

■

Portanto, não é preciso armazenar toda a matriz do código estendido de Prüfer para armazenar a árvore, basta armazenar a segunda linha. Na verdade, como se sabe que a última entrada é 0, não é necessário o armazenamento desta entrada. A lista consistindo das primeiras $n-2$ entradas da segunda linha é conhecido como *código de Prüfer* da árvore. Por conseguinte, este código é uma lista de comprimento $n-2$ onde cada entrada é um número inteiro entre 0 e $n-1$, o que é bem semelhante ao código de pai, sendo apenas um dígito mais curto. Mas a principal característica do código de Prüfer é que ele é ótimo, como diz a Proposição 2.2.3

Proposição 2.2.3 *Toda lista de números entre* 0 *e* $n-1$*, de comprimento* $n-2$*, é um código de Prüfer de alguma árvore sobre* n *vértices.*

Demonstração: É bem simples, considere uma de $n-2$ entradas com cada entrada assumindo valores inteiros entre 0 e $n-1$. Então acrescenta-se 0 após a última entrada aumentando o tamanho da lista. Após isso, escreve-se uma linha acima desta lista de mesmo comprimento, de maneira que acima de cada entrada na primeira linha seja o menor inteiro que não ocorre na primeira linha antes dele e nem na segunda abaixo ou depois dele (o que é sempre possível pois esta condição exclui no máximo $n-1$ valores de n possíveis). O resultado é um código de Prüfer estendido que é uma árvore que é única. De fato, pois se duas listas distintas representassem a mesma árvore, então as matrizes teriam que ser iguais, o que implicaria que cada entrada teria que ser igual a sua correspondente na outra lista, sendo então as duas listas iguais.

■

2.2.2 Número de árvores rotuladas e não rotuladas

Teorema 2.2.5 ***(Teorema de Cayley)*** *O número de árvores rotuladas sobre* n *vértices é* n^{n-2}*.*

Demonstração: De fato, toda lista de $n-2$ elementos com cada entrada entre 0 e $n-1$ corresponde a uma árvore sobre n vértices, assim sendo o número de sequências é n^{n-2}.

■

O número de árvores não rotuladas sobre n vértices, que será denotado por T_n, é bem mais difícil de se trabalhar. Nenhuma fórmula simples como a do Teorema 2.2.5 é conhecida para se determinar T_n.

É claro que o número de árvores não rotuladas é menor do que o número de árvores rotuladas; toda árvore rotulada pode ser rotulada de muitas maneiras. Se for desenhada uma árvore não rotulada, pode-se rotular seus vértices de $n!$ maneiras. As árvores rotuladas que se obtém dessa maneira não são necessariamente todas diferentes. Por exemplo, se a árvore for um estrela, então independentemente de como se permutam os rótulos das folhas, obtém-se a mesma árvore rotulada.

Proposição 2.2.4 *O número de* T_n *de árvores não rotuladas sobre* n *vértices satisfaz*

$$\frac{n^{n-2}}{n!} \leq T_n$$

Demonstração: Se sabe que cada árvore rotulada pode ser rotulada de no máximo $n!$ maneiras. Como o número de árvores rotuladas é n^{n-2}, segue que:

$$\frac{n^{n-2}}{n!} \leq T_n$$

■

Note que a Proposição 2.2.4 representa apenas um limitante inferior para T_n. Como calcular então um limitante superior? Informalmente, como deve-se descrever uma árvore, se deseja-se apenas o "formato"dela e não se está preocupado com qual vértice possui qual rótulo?

Tome uma árvore G sobre n vértices, e especifique uma de suas folhas como sendo a raiz. Em seguida desenhe G no plano sem cruzar as arestas, isso pode sempre ser feito e quase sempre se desenham árvores dessa maneira. Agora imagine que as arestas da árvore são paredes perpendiculares ao plano. Começando na raiz, caminhe ao redor desse sistema de paredes, mantendo sempre a parede à sua direita. Chama-se o passeio sobre uma aresta de um "passo". Como existem $n-1$ arestas e cada arestas tem dois lados, será dado $2(n-1)$ passos antes de se retornar a raiz.

A cada vez que se dá um passo para longe da raiz (isto é, um passo de um pai para um de seus filhos), escreve-se 1, e cada vez que se dá um passo em direção a raiz, escreve-se 0. Dessa maneira se determina uma lista binária com $2(n-1)$ elementos. Esse código é conhecido como *código planar* da árvore não rotulada. O código planar tem a seguinte propriedade: toda árvore não rotulada é univocamente determinada pelo seu código planar (Lovász, 2013).

Proposição 2.2.5 *O número T_n de árvores não rotuladas satisfaz*

$$T_n \leq 4^{n-1}$$

Demonstração: Como o número de possíveis códigos planares é no máximo $2^{2(n-1)}$, obtém-se que o número de árvores não rotuladas é no máximo 4^{n-1}

■

Teorema 2.2.6 *O número T_n de árvores não rotuladas sobre n vértices satisfaz:*

$$\frac{n^{n-2}}{n!} \leq T_n \leq 4^{n-1}$$

Demonstração: Pela Proposição 2.2.4 se obtém a primeira parte da desigualdade e pela Proposição 2.2.5 se obtém a segunda parte.

■

2.2.3 Árvore ótima

Um país com n cidades deseja construir uma nova rede telefônica para ligar todas as cidades. É claro que não é necessária a construção de uma linha para cada par de cidades, mas de fato a rede tem que ser conexa. Suponha que por algum motivo o país não deseja construir uma linha direta entre cidades que podem ser atingidas por outro meio. Do ponto de vista da teoria dos grafos, o que se está a procura é de um grafo conexo "mínimo"com n vértices, isto é, uma árvore.

Sabe-se que independentemente de qual árvore será feita, será necessário pelo menos $n-1$ linhas. Apesar deste número ser fixo, não será qualquer árvore que será escolhida, pois algumas linhas com certeza não são tão fáceis de se construir, e todas não possuem o mesmo custo. O

objetivo então é construir uma árvore geradora cujo custo total (a soma dos custos de suas arestas) seja mínima. Suponha que se esteja de mão de todos os custos.

Uma ideia bem natural para se resolver este problema, seria gerar todas as árvores possíveis, calcular o custo de cada uma delas e simplesmente optar pela melhor. Mas essa com certeza não é a melhor ideia. Pelo Teorema 2.2.5 que o número de árvores rotuladas sobre n vértices é n^{n-2}. Portanto, para 10 cidades teriam que ser analisadas 10^8 (cem milhões) de árvores possíveis, e para 20 cidades o número é absurdamente grande. Logo, analisar árvore por árvore não é a melhor solução.

Suponha que o governo queira fazer a árvore de maneira otimista, ou seja, a cada passo, escolhe-se o melhor passo possível. A cada passo, o governo levanta o investimento necessário para a construção da linha, esse método é conhecido como *método guloso* ou *Algoritmo Guloso*. Após o término da construção, o resultado é uma árvore onde cada vértice representa uma cidade e cada aresta representa uma linha telefônica. Essa árvore é de fato tão barata quanto possível, mas se a tarefa for um pouco diferente, a mesma abordagem pode levar a resultados muito ruins.

Suponha que por razões de confiabilidade, o governo exige que entre quaisquer duas cidades deve haver pelo menos dois caminhos possíveis sem qualquer aresta em comum. Para isso, $n-1$ linhas não são suficientes, mas n linhas bastam, o que se tem que fazer é criar um ciclo através de todas as cidades. Com isso, a nova tarefa é encontrar um ciclo com n arestas passando pelos n vértices de modo que o custo total de construir suas arestas é mínimo.

Esse problema é bastante conhecido na Otimização Matemática, é chamado Problema do Caixeiro Viajante. Neste caso, o método guloso não retorna a melhor solução possível (Lovász, 2013).

Retornando ao problema de se encontrar a árvore de custo mínimo. No contexto de árvores geradoras, o algoritmo guloso é conhecido como *Algoritmo de Kruskal*. A árvore obtida pelo método guloso será denotada por F.

Teorema 2.2.7 *Seja F uma árvore construída pelo método guloso. Então qualquer outra árvore G custará no mínimo tanto quanto a árvore gulosa.*

Demonstração: Seja G uma árvore diferente da árvore gulosa F. Considere o processo de construção de F, e o passo quando primeiro se toma uma aresta de F que não é uma aresta de G. Seja α essa aresta. Se α for adicionada a G, obtém-se um ciclo C. Esse ciclo não está inteiramente contido em F, logo existe uma aresta f que não é aresta de F. Se for adicionada a aresta α a G e então remove-se f, obtém-se uma terceira árvore H. Será mostrado que H é no máximo tão cara quanto G. Isso quer dizer que α é no máximo tão cara quanto f. Suponha que f seja mais barata que α. Ora, pelo método otimista (ou método guloso) então f teria que ter sido escolhida ao invés de α, mas já que não escolhida, então foi descartada porque estaria formando um ciclo C' com as arestas de F já selecionadas. Mas todas essas arestas previamente selecionadas são arestas de G, pois inspeciona-se o passo em que a primeira aresta que não está em G foi adicionada a F. Como a própria f é uma aresta de G, segue que todas as arestas de C' são arestas de G, o que é impossível pos G é uma árvore. Portanto, f não pode ser mais barata que α e assim G não pode ser mais barata que H. Assim sendo, substitui-se G por H que não é mais cara, e que tem a vantagem de coincidir com a F em mais arestas. Isso implica que se H for diferente de F, repete-se o mesmo argumento, obtendo-se árvores que não são mais caras que G, e que coincidem com a F em mais e mais arestas. Como o número de arestas é finito, em um determinado passo se obterá a própria F, que não é mais cara que a G.

■

Retornando-se à questão de se encontrar um ciclo mais barato possível através de todas as cidades dadas: tem-se n cidades (vértices), e para quaisquer duas delas se tem o "custo"de se

conectá-las diretamente. Tem-se que encontrar um ciclo com esses vértices tal que o custo do ciclo (a soma dos custos de suas arestas) seja o mínimo possível.

Este problema é um dos mais importantes na área de *Otimização Combinatória*, conhecido como *Problema do Caixeiro Viajante*, e pode aparecer sob vários aspectos diferentes. Este nome vem da versão para o problema onde um vendedor viajante tem que visitar todas as cidades na região e depois retornar a própria casa,e obviamente ele deseja minimizar os custos dessa viagem. É fácil imaginar que este problema aparece em conexão com o "desenho"de rotas de entregas ótimas para os correios, rotas ótimas para coleta de lixo, etc.

A seguinte questão importante leva ao mesmo problema matemático. Uma máquina tem que perfurar um número de buracos em uma placa de circuitos impressos, e então retornar para o ponto inicial. Nesse caso, a quantidade importante é o tempo em que a máquina leva para mover a ponta perfurante de um buraco para o próximo, pois o tempo total que uma dada placa tem que gastar na máquina determina o número de placas que podem ser processadas em um dia. Portanto, se for tomado o tempo necessário para mover a ponta de um buraco para outro como o "custo"dessa aresta, precisa-se encontrar um ciclo com os buracos como vértices e com o custo mínimo.

O problema do caixeiro viajante é intimamente relacionado aos ciclos hamiltonianos. Antes de tudo, uma volta do caixeiro viajante é simplesmente um ciclo hamiltoniano no grafo completo sobre um conjunto de vértices. Mas há uma conexão mais interessante: *o problema de se descobrir se um dado grafo tem um ciclo hamiltoniano pode ser reduzido ao Problema do Caixeiro Viajante.*(Lovász, 2013)

2.3 Redes

Quando se associam valores e/ou arestas, o grafo designa-se geralmente por **rede**. Neste caso, fala-se em *nós* e *arcos* em vez de vértices e arestas, respectivamente.

Uma rede pode ser interpretada por $G = (N, A, C)$, em que (N, A) é um grafo e C corresponde ao conjunto de valores associados aos arcos "comprimentos": ao arco (i, j) está associado o valor c_{ij}. De uma maneira geral, os conceitos utilizados em grafos são extensíveis às redes.

Considere um caminho p de S para T, na rede G. O comprimento do caminho p corresponde à soma dos comprimentos dos arcos que pertencem àquele caminho, ou seja,

$$C(p) = \sum_{(i,j)\in p} c_{ij}$$

O conjunto de todos os caminhos de S para T, numa rede qualquer G será identificado por P.

Define-se **árvore mínima** (**árvore de caminhos mais curtos**) com origem em S, como a árvore que contém todos os nós de N acessíveis a partir de S, em que para cada nó n_2 é o caminho mais curto (de comprimento mínimo) na rede G que liga S a n_2.

2.3.1 O Problema do Caminho Mais Curto

Os problemas de caminho mais curto, são fundamentais e frequentes quando se estuda redes de transportes e de comunicação. Este problema surge quando se pretende determinar o caminho mais curto, entre um ou vários pares de nós de uma rede.

Existem três tipos de problemas de caminho mais curto:

(i) de um nó para outro;

(ii) de um nó para todos os outros;

(iii) entre todos os pares de nós;

Sejam S e T dois nós de uma rede $G = (N, A, C)$, em que a cada arco é associado apenas um valor (chamaremos de comprimento do arco). O comprimento de um caminho de S para T, é a soma dos comprimentos dos arcos que o compõem.

O problema do caminho mais curto entre os nós S e T, tem por objetivo determinar o caminho de valor mínimo existente em P, ou seja, determinar $p \in P$ tal que $C(p) \leq C(q)$, $\forall q \in P$.

Algumas observações relacionadas com este tipo de problema:

- o comprimento de um caminho é maior do que o de qualquer dos seus subcaminhos;
- qualquer subcaminho de um caminho mais curto, é ele próprio um caminho mais curto;
- para uma rede com n nós, qualquer caminho mais curto tem no máximo $n-1$ arcos (no caminho mais curto entre dois nós, não existem nós repetidos);

Matematicamente, este problema pode ser reformulado na seguinte maneira:

$$Z = \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$$

$$h_1 = \sum_{j \in N} xs_j - 1$$

$$h_2 = \sum_{i \in N} x_{ij} - \sum_{k \in N} x_{jk}, \forall j \in N - \{S, T\}$$

$$h_3 = \sum_{i \in N} x_{iT} - 1$$

onde $h_1 = h_2 = h_3 = 0$ e

$x_{ij} = \begin{cases} 1, \text{ se (i,j) pertence ao caminho} \\ 0, \text{ se (i,j) não pertence ao caminho} \end{cases}$

Existem vários algoritmos eficientes para resolver problemas de caminho mais curto, dentre eles um dos mais conhecidos é o de Dijkstra.

2.3.2 Algoritmo de Dijkstra

Este algoritmo, que foi apresentado por Dijkstra e que só pode ser aplicada a redes cujos arcos têm associados comprimentos não negativos, baseia-se num processo de rotulação dos nós da rede e classificação dos respectivos rótulo. A cada nó i é atribuído um rótulo $[\xi_i, \pi_i]$, o qual pode ser permanente ou temporário. Isto quer dizer o seguinte:

- $[\xi_i, \pi_i]$ permanente, representa o caminho mais curto de S para i;
  - $\xi_i \leftarrow$ nó que antecede i no caminho mais curto de S para i;
  - $\pi_i \leftarrow$ valor do caminho mais curto de S para i;
- $[\xi_i, \pi_i]$ temporário, representa um caminho mais curto de S para i;
  - $\xi_i \leftarrow$ nó que antecede i no melhor caminho, até a etapa atual de S para i;
  - $\pi_i \leftarrow$ valor do melhor caminho até a etapa atual de S para i;

O rótulo temporário de um nó representa um limite superior da distância mais curta de S ao nó determinado, uma vez que o caminho que lhe está associado pode ser ou não o caminho mais curto.

O algoritmo consiste em rotular os nós da rede, começando por S (origem), de uma forma ordenada, segundo as distâncias de cada nó a S: escolher o nó com rótulo temporário com menor valor de π, que passa a ser um nó com rótulo permanente, para depois serem "varridos"todos os nós adjacentes, de forma a atualizar os rótulos destes (temporários). O algoritmo termina quando não existirem nós com rótulos temporários. Inicialmente apenas o nó S é permanente, sendo os restantes temporários.

Algoritmo

Passo 1

$[\xi_S, \pi_S] = [S, 0]$ *(caminho mais curto para S custa 0 e não tem nós intermediários)*

$[\xi_i, \pi_i] = [S, C_{S,i}]$, $\forall i \in N - \{S\}$ e $(S, i) \in A$ *(os nós adjacentes a S são rotulados com $\xi_i = S$ e $\pi_i = C_{S,i}$ que é o custo do arco (S, i))*

$[\xi_S, \pi_S] = [S, \infty]$, $\forall i \in N - \{S\}$ e $(S, i) \in \complement A$ *(os nós que não são adjacentes a S são rotulados com $\xi_i = S$ e $\pi_i = \infty$ para representar que não existe o arco (S, i))*

Temporários $= N - \{S\}$ *(Temporários é o conjunto de nós com rótulos temporários)*

Permanentes $= \{S\}$ *(Permanentes é o conjunto de nós com rótulos permanentes)*

Passo 2

Se Temporários $= \emptyset$ *(todos os nós tem rótulos permanentes)* então

STOP *(critério de parada)*

Senão

$v \in Temporarios$ é tal que π_v é mínimo *($v : \pi_v = min\{\pi_x, x \in Temporarios\}$)*

$Temporarios = Temporarios - \{v\}$ *(v deixa de ser temporário)*

$Permanentes = Permanentes \cup \{v\}$ *(v passa a ser permanente)*

Passo 3

Para todo $j \in N$ tal que $(v, j) \in A$ e $j \in Temporarios$ faz

Se $\pi_v + C_{v,j} < \pi_j$ então

$\pi_j = \pi_v + C_{v,j}$ *(o custo atual de se chegar a j é substituído pelo custo de se chegar a v mais o custo do arco (v, j))*

$\xi_j = v$ *(o nó antecessor de j no caminho mais curto de S para j passa a ser v)*

Senão

Regressar ao Passo 2

Fim do Algoritmo

O algoritmo apresentado, determina o caminho mais curto entre um dado nó S e todos os outros nós da rede. Portanto, no fim do algoritmo, para se verificar se existe caminho entre S e um nó qualquer v, basta analisar o valor de π_v: se $\pi_v = \infty$, então não existe caminho. Se existir caminho mais curto de S para v, este pode ser determinado percorrendo (em sentido inverso) a primeira parte dos rótulos dos nós (a saber ξ) de v até S, da seguinte forma:

$Caminho = \{v\}$

$i = v$

Enquanto $i \neq S$ faz

$Caminho \leftarrow Caminho \cup \{i\}$

Exemplo 1: Encontrar o caminho mínimo da rede apresentada na Figura 2.7 entre o nó $S = 1$ e todos os outros nós da rede.

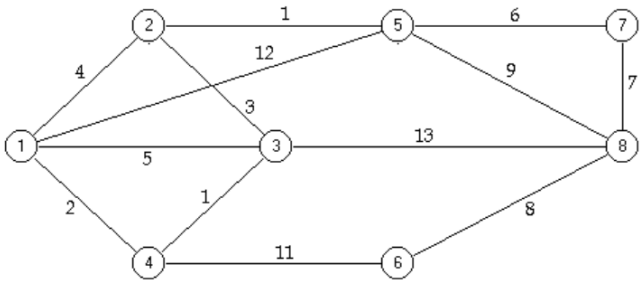

Figura 2.7: Rede com 8 nós

Passo 1:
Colocar rótulo permanente no nó 1 e rótulos temporários nos nós restantes.

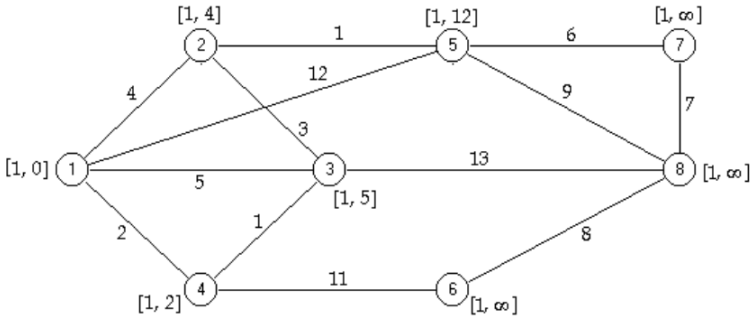

Figura 2.8: Rede após a primeira iteração, apenas a origem com nó permanente

$Permanentes = \{1\}$
$Temporarios = \{2, 3, 4, 5, 6, 7, 8\}$

Passo 2:
Como o conjunto de nós temporários não é vazio, então dentre os nós adjacentes à origem procura-se o nó com o menor valor de $\pi_j, j \in Temporarios$. Neste caso, o nó escolhido é $v = 4$, pois $\pi_4 = min\{\pi_2, \pi_3, \pi_4\}$.
$Permanentes = Permanentes \cup \{4\} = \{1, 4\}$
$Temporarios = Temporarios - \{4\} = \{2, 3, 5, 6, 7, 8\}$

Passo 3:
Varrer todos os nós adjacentes à 4, com rótulos temporários e atualizar os seus rótulos. Os nós adjacentes ao nó 4 são os nós 3 e 6.
$\pi_3 = min\{\pi_3, \pi_4 + C_{43}\} = min\{5, 2 + 1\} = 3 \Rightarrow [\xi_3, \pi_3] = [4, 3]$
$\pi_6 = min\{\pi_6, \pi_4 + C_{46}\} = min\{\infty, 2 + 11\} = 13 \Rightarrow [\xi_6, \pi_6] = [4, 13]$

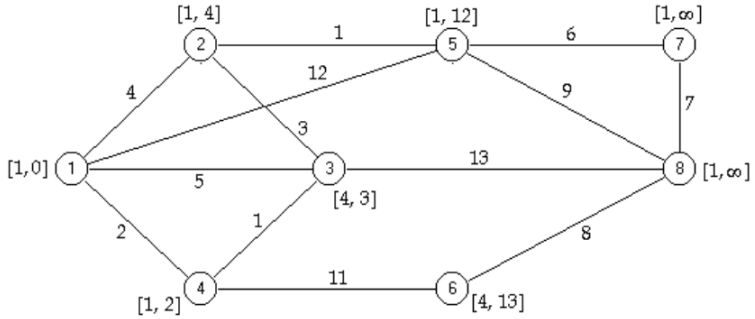

Figura 2.9: Rede com os rótulos dos nós 3 e 6 atualizados

Passo 2:

Como o conjunto de nós temporários não é vazio, então dentre os nós temporários procura-se o nó com o menor valor de $\pi_j, j \in Temporarios$. Neste caso, o nó escolhido é $v = 3$, pois $\pi_3 = min\{\pi_2, \pi_3, \pi_5, \pi_6, \pi_7, \pi_8\}$.
$Permanentes = Permanentes \cup \{3\} = \{1, 3, 4\}$
$Temporarios = Temporarios - \{3\} = \{2, 5, 6, 7, 8\}$

Passo 3:

Varrer todos os nós adjacentes à 3, com rótulos temporários e atualizar os seus rótulos. Os nós adjacentes ao nó 4 com rótulos temporários são os nós 2 e 8.
$\pi_2 = min\{\pi_2, \pi_3 + C_{32}\} = min\{4, 3 + 3\} = 4 \Rightarrow [\xi_2, \pi_2] = [1, 4]$ *(sem alteração)*
$\pi_8 = min\{\pi_8, \pi_3 + C_{38}\} = min\{\infty, 3 + 13\} = 16 \Rightarrow [\xi_8, \pi_8] = [3, 16]$

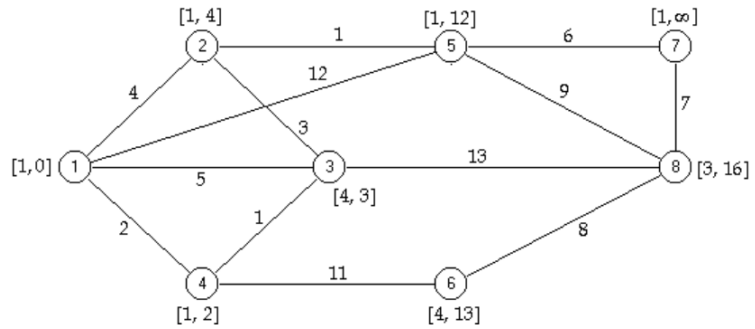

Figura 2.10: Rede com os rótulos dos nós 2 e 8 atualizados

Passo 2:

Como o conjunto de nós temporários não é vazio, então dentre os nós temporários procura-se o nó com o menor valor de $\pi_j, j \in Temporarios$. Neste caso, o nó escolhido é $v = 2$, pois $\pi_2 = min\{\pi_2, \pi_5, \pi_6, \pi_7, \pi_8\}$.
$Permanentes = Permanentes \cup \{2\} = \{1, 2, 3, 4\}$
$Temporarios = Temporarios - \{3\} = \{5, 6, 7, 8\}$

Passo 3:

Varrer todos os nós adjacentes à 2, com rótulos temporários e atualizar os seus rótulos. O nó adjacentes ao nó 2 com rótulo temporário é apenas o nó 5.
$\pi_5 = min\{\pi_5, \pi_2 + C_{25}\} = min\{12, 4 + 1\} = 5 \Rightarrow [\xi_5, \pi_5] = [2, 5]$

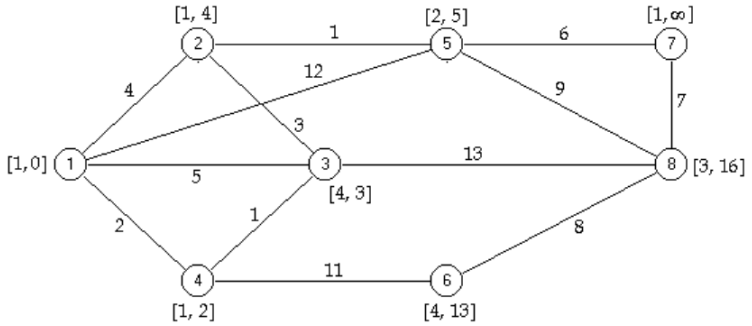

Figura 2.11: Rede com o rótulo do nó 5 atualizado

Passo 2:

Como o conjunto de nós temporários não é vazio, então dentre os nós temporários procura-se o nó com o menor valor de $\pi_j, j \in Temporarios$. Neste caso, o nó escolhido é $v = 5$, pois $\pi_5 = min\{\pi_5, \pi_6, \pi_7, \pi_8\}$.
$Permanentes = Permanentes \cup \{5\} = \{1, 2, 3, 4, 5\}$
$Temporarios = Temporarios - \{5\} = \{6, 7, 8\}$

Passo 3:

Varrer todos os nós adjacentes à 5, com rótulos temporários e atualizar os seus rótulos. Os nós adjacentes ao nó 5 com rótulos temporários são os nós 7 e 8.
$\pi_7 = min\{\pi_7, \pi_5 + C_{57}\} = min\{\infty, 5 + 6\} = 11 \Rightarrow [\xi_7, \pi_7] = [5, 11]$
$\pi_8 = min\{\pi_8, \pi_5 + C_{58}\} = min\{16, 5 + 9\} = 14 \Rightarrow [\xi_8, \pi_8] = [5, 14]$

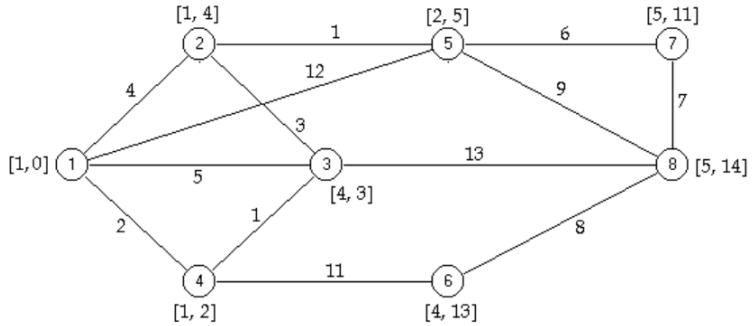

Figura 2.12: Rede com os rótulos dos nós 7 e 8 atualizados

Passo 2:

Como o conjunto de nós temporários não é vazio, então dentre os nós temporários procura-se o nó com o menor valor de $\pi_j, j \in Temporarios$. Neste caso, o nó escolhido é $v = 7$, pois $\pi_7 = min\{\pi_6, \pi_7, \pi_8\}$.
$Permanentes = Permanentes \cup \{7\} = \{1, 2, 3, 4, 5, 7\}$
$Temporarios = Temporarios - \{7\} = \{6, 8\}$

Passo 3:

Varrer todos os nós adjacentes à 7, com rótulos temporários e atualizar os seus rótulos. O nó adjacente ao nó 7 com rótulo temporário é apenas o nó 8.
$\pi_8 = min\{\pi_8, \pi_7 + C_{78}\} = min\{14, 11 + 7\} = 14 \Rightarrow [\xi_8, \pi_8] = [5, 14]$ *(sem alteração)*

Passo 2:

Como o conjunto de nós temporários não é vazio, então dentre os nós temporários procura-se o nó com o menor valor de $\pi_j, j \in Temporarios$. Neste caso, o nó escolhido é $v = 6$, pois

$\pi_6 = min\{\pi_6, \pi_8\}$.
$Permanentes = Permanentes \cup \{6\} = \{1, 2, 3, 4, 5, 6, 7\}$
$Temporarios = Temporarios - \{6\} = \{8\}$

Passo 3:

Varrer todos os nós adjacentes à 6, com rótulos temporários e atualizar os seus rótulos. O nó adjacente ao nó 6 com rótulo temporário é apenas o nó 8.
$\pi_8 = min\{\pi_8, \pi_6 + C_{68}\} = min\{14, 13 + 8\} = 14 \Rightarrow [\xi_8, \pi_8] = [5, 14]$ *(sem alteração)*

Passo 2:

Como o conjunto de nós temporários não é vazio, então dentre os nós temporários procura-se o nó com o menor valor de $\pi_j, j \in Temporarios$. Neste caso, o nó escolhido é $v = 8$, pois $\pi_8 = min\{\pi_8\}$.
$Permanentes = Permanentes \cup \{8\} = \{1, 2, 3, 4, 5, 6, 7, 8\}$
$Temporarios = Temporarios - \{8\} = \emptyset$

Passo 3:

Não existem nós adjacentes a 8 com rótulos temporários.

Passo 2:

$Temporarios = \emptyset \Rightarrow$ Fim do algoritmo *(critério de parada)*

O caminho mais curto do nó 1 para todos os demais nós é apresentado na Figura 2.13.

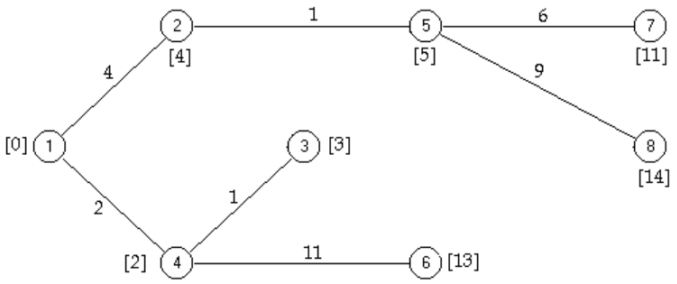

Figura 2.13: Árvore mínima

Conclusão

Através dos estudos desenvolvidos, as metas previstas para o projeto foram todas alcançadas. Inicialmente foi feito o desenvolvimento teórico para problemas de Otimização Combinatória. Mais adiante, ainda na primeira etapa do projeto, foram exploradas as principais propriedades relacionadas à Programação Linear Inteira e Teoria dos Grafos.

O estudo em Programação Linear Inteira foi de grande importância, visto que vários problemas relacionados à teoria dos grafos podem ser modelados e resolvidos via programação linear inteira. Na segunda etapa foi feito um levantamento de aplicações que beneficiaram-se dos estudos realizados, dentre elas o Problema do Caminho Mínimo, com o intuito de avançar nos estudos relacionados ao assunto; abordando aspectos no presente trabalho, serão feitas algumas considerações para trabalhos futuros, utilizando uma ferramenta da Pesquisa Operacional chamada Teoria das Filas juntamente com o estudo realizado sobre PCMC para fazer uma aplicação na Engenharia de Transportes.

Referências Bibliográficas

GOLDBARG, Marco; LUNA, Henrique. Otimização Combinatória e Programação Linear. 2a ed. Rio de Janeiro: Editora Campus, 2005.

LOVÁSZ, Lászlo et al. Matemática Discreta. 2 ed. Rio de Janeiro: Sociedade Brasileira de Matemática, 2013.

MACULAN, Nelson; FAMPA, Márcia. Otimização Linear. 1a ed. Rio de Janeiro: Universidade Federa do Rio de Janeiro, 2002.

PAPADIMITRIOU, Christos; STEIGLITZ, Kenneth. Combinatorial Optimization: Algorithms and Complexity. 1a ed. New York: Dover Publications, 1998.