UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

MÉTODOS DE OTIMIZAÇÃO: TEORIA E PRÁTICA

Versão preliminar

Eduardo Camponogara

Florianópolis, outubro de 2003

Sumário

1 As Subáreas da Otimização **1**

1.1 Conceitos fundamentais 1

1.1.1 Modelagem de Problemas 1

1.1.2 Elementos de um Problema de Otimização 2

1.1.3 Duas Exceções à Formulação Geral 3

1.2 Programação Linear 3

1.2.1 Problema Exemplo 3

1.3 Programação Linear Inteira 4

1.3.1 Problema Exemplo 5

1.4 Programação Quadrática 6

1.5 Mínimos Quadrados Não-linear 6

1.6 Mínimos Quadrados Linear 7

1.6.1 Problema 8

1.7 Equações Não-Lineares 9

1.7.1 Aplicação em Sistemas de Controle 9

1.8 Otimização Não-linear sem Restrições 9

1.8.1 Problema Exemplo 10

1.9 Otimização Não-linear com Limites Superiores/Inferiores 10

1.10 Otimização Não-linear com Restrições 11

1.10.1 Problema Exemplo 11

1.11 Programação Semi-Definita 12

1.11.1 Exemplo 12

1.12 Referências 13

1.13 Exercícios 13

2 Minimização de Funções com o Método de Descenso **17**

2.1 Problemas de Interesse 17

2.2 Fundamentos de Otimização Irrestrita 18

2.2.1 O Que É Uma Solução para P_1? 18
2.2.2 Reconhecendo Um Mínimo Local 19
2.3 O Algoritmo de Descenso 21
2.3.1 O Algoritmo de Descenso em Detalhes 21
2.3.2 Direção de Busca 21
2.3.3 Encontrando a Direção de Busca 22
2.3.4 Encontrando o Passo 22
2.3.5 Redução Suficiente e Retrocesso 25
2.3.6 Convergência do Método de Descenso 26
2.3.7 Taxa de Convergência do Método de Descenso Íngreme 28
2.4 Exercícios 29

3 Minimização de Funções e Solução de Equações Não-Lineares com o Método de Newton 31
3.1 Problemas de Interesse 31
3.2 O Método de Newton em uma Variável 31
3.2.1 Exemplo 32
3.3 O Método de Newton para Minimização em uma Variável 33
3.3.1 Exemplo 34
3.4 O Método de Newton em Múltiplas Variáveis 35
3.5 Minimização Irrestrita 36
3.6 Convergência 37
3.6.1 Convergência Linear 38
3.6.2 Convergência Quadrática 38
3.6.3 Convergência do Método de Newton 39
3.7 Métodos de Região de Confiança 40
3.8 Exercícios 42

4 Otimização Não-Diferenciável 45
4.1 Otimização Black-Box 45
4.2 Algoritmo Genético (GA) 46
4.2.1 Genética e Evolução 46
4.2.2 Adaptação Biológica 46
4.2.3 Hereditariedade com Evolução Simulada 47
4.2.4 Popularidade do Algoritmo Genético 47
4.2.5 Algoritmo Genético em Detalhes 48
4.2.6 Operador Genético *Cross-Over* 49

4.2.7 Exemplo de Aplicação 49
4.2.8 Questões Práticas 50
4.2.9 Schema Theorem 51
4.3 Simulated Annealing 53
4.3.1 O Processo de Annealing 53
4.3.2 O Algoritmo de Metropolis 54
4.3.3 Exemplo: O Problema do Caixeiro Viajante 55
4.4 Exercícios 56

5 Treinamento de Redes Neurais: Um Problema de Otimização 59
5.1 Elementos Básicos das Redes Neurais 60
5.1.1 O Problema de Treinamento 60
5.1.2 ALVINN: Um Exemplo de Aplicação 62
5.1.3 Problemas Apropriados para Redes Neurais 62
5.2 Perceptron: A Primeira Unidade Neural 63
5.2.1 Treinando um Perceptron 64
5.3 Regra Delta 66
5.3.1 Treinando uma Unidade Delta 66
5.4 A Unidade Sigmoid 67
5.5 Exercícios 68

6 Programação Linear 71
6.1 Problema Exemplo: Gerenciamento de Uma Unidade de Produção . . 71
6.1.1 Gerente de Produção Otimista 71
6.1.2 Gerente de Produção Pessimista 72
6.2 O Problema de Programação Linear 73
6.3 Algoritmo Simplex 76
6.3.1 Exemplo 76
6.3.2 Algoritmo Simplex em detalhes 79
6.3.3 Inicialização 81
6.4 Dualidade 83
6.4.1 Motivação 84
6.4.2 O Problema Dual 85
6.4.3 O Teorema Fraco da Dualidade 86
6.4.4 O Teorema Forte da Dualidade 87
6.4.5 Folga Complementar 89
6.5 Algoritmo Simplex em Notação Matricial 89

6.5.1 Dicionário em Forma Matricial 90
6.6 Exercícios 91

7 Teoria dos Jogos **93**
7.1 Jogos Matriciais 93
7.1.1 O Jogo da Tesoura, Pedra e Papel 93
7.1.2 Um Jogo Menos Trivial 94
7.2 Formalização 94
7.3 Estratégia Ótima para o Agente Coluna 95
7.4 Estratégia Ótima para o Agente Linha 97
7.5 Relação entre os Problema P_x e P_y 97
7.6 Teorema Minimax 98
7.7 Exercícios 99

8 Fluxo em Redes **101**
8.1 Dois Problemas Clássicos 101
8.1.1 O Problema de Transportes 101
8.1.2 O Problema de Alocação 102
8.2 O Problema de Fluxo em Redes de Custo Mínimo 103
8.3 Transformações 104
8.3.1 Convertendo Arestas em Arcos 104
8.3.2 Removendo Limites Inferiores 105
8.3.3 Removendo Capacidade de Arcos 105
8.4 Um exemplo 105
8.5 Redes Residuais 107
8.6 Algoritmo de Cancelamento de Circuitos Negativos 108
8.6.1 Complexidade do Algoritmo 108
8.6.2 Exemplo de Execução do Algoritmo 109
8.7 Matrizes Totalmente Unimodulares 110
8.8 Exercícios 112

9 Linguagens de Modelagem **115**
9.1 Linguagem Mosel 115
9.1.1 Qual Interface Devemos Utilizar? 116
9.1.2 Resolvendo um Problema 117
9.1.3 Indo Mais Longe 118
9.1.4 Trabalhando com o Optimizer 119
9.1.5 Construindo um Primeiro Modelo 119

9.1.6 Usando Cadeias de Caracteres como Índices 122
9.1.7 Modelagem Versátil 122
9.2 Linguagem AMPL 123
9.2.1 Modelo AMPL do Problema da Mochila 124
9.2.2 Comentários 126
9.3 Exercícios 126

10 Fundamentos de Programação Inteira 127
10.1 Introdução 127
10.1.1 Escalonamento de Trens 127
10.1.2 *Airline Crew Scheduling* 128
10.2 O Que É um Problema Inteiro? 128
10.2.1 Problema (Linear) Inteiro Misto 129
10.2.2 Problema (Linear) Inteiro 129
10.2.3 Problema Linear Binário 129
10.2.4 Problema de Otimização Combinatória 129
10.3 Programação Linear e Arredondamento 130
10.4 Formulação de PIs e PIBs 131
10.4.1 Exemplo 1: Formulando o Problema de Alocação 131
10.4.2 Exemplo 2: O Problema da Mochila 132
10.4.3 Exemplo 3: O Problema de Cobertura de Conjuntos 133
10.4.4 Exemplo 4: O Problema do Caixeiro Viajante (PCV) 134
10.5 Explosão Combinatória 135
10.6 Formulação de Problemas Inteiros Mistos (PIMS) 136
10.6.1 Exemplo 1: Modelando Custos Fixos 136
10.6.2 Exemplo 2: Localização de Depósitos sem Limites de Capacidade 137
10.6.3 Alternativas Discretas e Disjuntas 138
10.7 Formulações Alternativas 140
10.7.1 Formulações Equivalentes para o Problema da Mochila 140
10.7.2 Localização de Depósitos sem Limites de Capacidade 141
10.8 Formulações Apertadas e Ideais 141
10.8.1 Formulações Equivalentes para o Problema da Mochila 143
10.8.2 Formulação para o Problema de Localização de Depósitos . . . 144
10.9 Exercícios 144

11 Programação Inteira: Relaxações e Algoritmo *branch-and-bound* 147
11.1 Condições de Otimalidade 147

11.1.1 Limite Primal 148
11.1.2 Limite Dual 148
11.2 Relaxação Baseada em PL (Relaxação Linear) 148
11.3 Relaxação Combinatória 149
11.3.1 O Problema do Caixeiro Viajante 150
11.3.2 O Problema da Mochila 150
11.4 Relaxação Lagrangeana 151
11.5 Algoritmo *Branch-and-Bound* 152
11.5.1 Estratégia de Divisão e Conquista 152
11.5.2 Enumeração Implícita 154
11.5.3 Algoritmo *Branch-and-Bound* (B&B) 157
11.6 Exercícios 160

12 Algoritmos de Planos de Corte e Desigualdades Fortes 161
12.1 Introdução a Planos de Corte 161
12.2 Exemplos de Desigualdades Válidas 162
12.2.1 Um conjunto 0-1 puro 162
12.2.2 Um Conjunto 0-1 Misto 163
12.2.3 Um Conjunto Inteiro Misto 163
12.2.4 Conjunto Combinatório 163
12.2.5 Arredondamento Inteiro 164
12.2.6 Arredondamento Inteiro Misto 165
12.3 Teoria de Desigualdades Válidas 165
12.3.1 Desigualdades Válidas para Problemas Lineares 166
12.3.2 Desigualdades Válidas para Problemas Inteiros 166
12.3.3 Procedimento Chvátal-Gomory para Geração de Desigualdades Válidas 167
12.4 O Algoritmo de Planos de Corte 167
12.5 Algoritmo de Planos de Corte Usando Cortes de Gomory Fracionários 168
12.5.1 Exemplo 169
12.6 Desigualdades Disjuntivas 171
12.6.1 Exemplo 171
12.6.2 Desigualdades Disjuntivas para Problemas 0-1 172
12.7 Exercícios 174

13 Programação Dinâmica: Domínio Discreto 175
13.1 Um Exemplo de Programação Dinâmica 176

13.1.1 Calculando Números de Fibonacci 176
13.2 Programação Dinâmica para o Problema da Mochila 178
13.2.1 Complexidade do algoritmo 179
13.3 Elementos de um Algoritmo DP: Sequência Crescente Mais Longa . . 180
13.3.1 Construindo um Algoritmo 181
13.4 Edição Automática de Cadeias de Caracteres ("Approximate String Matching") 182
13.4.1 Projeto de um algoritmo DP 183
13.5 Exercícios 184

Lista de Figuras

1.1 Predição peso com base na altura 8

2.1 Exemplo de função com um número infinito de ótimos locais 18
2.2 Ilustração do gradiente de uma função, $\nabla f(x_k)$, que assume a direção ortogonal às curvas de nível. 22
2.3 Ilustração de passo satisfatório. 23
2.4 Ilustração das condições de Armijo e de curvatura. 24

3.1 Ilustração do processo iterativo de Newton 34
3.2 Função $\cos(x)$ 35

4.1 Operador *cross-over* de um ponto 49
4.2 Operador *cross-over* de dois pontos 50
4.3 Fenômeno de *crowding* 51
4.4 Empacotamento de componentes eletrônicos em circuitos integrados . . 53
4.5 Ilustração do processo de *annealing* 54
4.6 Comportamento típico do nível de energia conforme processo de *annealing* 54
4.7 Probabilidade de transição com a queda de temperatura 55
4.8 Exemplo de rota para uma instância particular 56
4.9 Ilustração dos operadores de perturbação 57

5.1 Exemplo de topologia de redes neurais 60
5.2 Unidade de processamento neural 61
5.3 Rede neural com camadas intermediárias 61
5.4 Rede neural do sistema ALVINN 63
5.5 Perceptron 64
5.6 Conjunto separáveis e não separáveis linearmente 65
5.7 Unidade delta 66
5.8 Função sigmoid 68

5.9 Sistema neural para reconhecimento de palavras 69
5.10 Topologia da rede neural 70

6.1 Exemplo de problema infactível 75
6.2 Exemplo de problema ilimitado 75
6.3 Método simplex como um processo iterativo 90

8.1 Transformação de Arcos em Arestas 105
8.2 Transformação de Arcos em Arestas 105
8.3 Eliminando Capacidade dos Arcos 106
8.4 Exemplo de problema de fluxo em redes 106
8.5 Exemplo de rede residual 107
8.6 Exemplo de rede residual 108
8.7 Primeira iteração do algoritmo de cancelamento de circuito negativo . 109
8.8 Segunda iteração do algoritmo de cancelamento de circuito negativo . 109
8.9 Terceira iteração do algoritmo de cancelamento de circuito negativo . . 110
8.10 Fluxo ótimo 110
8.11 Propriedade de unimodularidade das matrizes de incidência de grafos . 112
8.12 Grafo $G = (N, A)$ com a especificação do problema de fluxo em rede . 113

10.1 Ilustração da solução obtida através de arredondamento 130
10.2 Ilustração da restrição de conectividade 135
10.3 Função com custo fixo 137
10.4 Região factível não convexa 139
10.5 Formulações equivalentes alternativas 140
10.6 Formulações alternativas 142
10.7 Formulações apertadas 143
10.8 Grafo da instância do problema do caixeiro viajante 145

11.1 Árvore de enumeração 153
11.2 Árvore de enumeração 153
11.3 Árvore de enumeração completa, explícita 154
11.4 Eliminação por otimalidade 155
11.5 Eliminação por limite 155
11.6 Nenhum ramo da árvore pode ser eliminado 156
11.7 Quebra do primeiro nó da árvore de enumeração B&B 158
11.8 Dividindo S_1 em S_{11} e S_{12} 158
11.9 Cortando o nó S_{12} por meio da condição de otimalidade 159

12.1 Espaço de soluções factíveis e desigualdade válida $x \leq 14 - 4(2 - y)$. 164
12.2 Desigualdades disjuntivas 172

13.1 Árvore de recursão do algoritmo recursivo para cálculo do número de Fibonacci 178

Lista de Tabelas

1.1 Dados amostrais 7

3.1 Iterações do método de Newton. 33

4.1 Memória inicial 50

5.1 Exemplos para treinamento da rede neural 69

8.1 Capacidade de Produção dos Fornecedores 102
8.2 Demanda dos Clientes 102
8.3 Custo Unitário de Transporte 103

9.1 Dados do problema da mochila 120

10.1 Número de soluções em função do tamanho do problema 135
10.2 Crescimento de funções 136

13.1 Tabelas DP para o Problema da Mochila 180
13.2 Tabelas DP para o Problema de Subseqüência Mais Longa 182

Notação

Capítulo 1

As Subáreas da Otimização

Otimização é a área da Matemática Aplicada que se preocupa em calcular e computar valores ótimos para variáveis de decisão que induzem desempenho ótimo, ao mesmo tempo que satisfazem restrições, de um modelo matemático.

1.1 Conceitos fundamentais

1.1.1 Modelagem de Problemas

A representação da realidade é uma necessidade da sociedade moderna, seja pela impossibilidade de lidar diretamente com a realidade, seja por aspectos econômicos, ou seja pela complexidade. Assim busca-se a representação da realidade por meio de modelos que sejam bem estruturados e representativos desta realidade.

> *Modelos são representações simplificadas da realidade que preservam, para determinadas situações e enfoques, uma equivalência adequada.*

Dentre as características de um modelo, se destacam a sua capacidade representativa e a capacidade de simplificação da realidade. A capacidade representativa de um modelo deve ser validada por meio de experimentação, análise numérica, ensaios ou qualquer outro método que verifique a acurácia das predições obtidas com o modelo. A modelagem de um problema complexo não é uma tarefa trivial, invariavelmente dependendo de fatores subjetivos como intuição, experiência, criatividade e poder de síntese. A formulação consiste em traduzir o modelo em uma linguagem formal, normalmente expressa em notação matemática e compreendendo variáveis, equações, desigualdades e fórmulas. Os processos de formulação e validação são iterativos, pois envolvem múltiplas etapas de tentativa e erro, e interativos à medida que o modelador deve intervir continuamente objetivando refinar o modelo.

Uma abordagem freqüentemente empregada na formulação e resolução de problemas consite no emprego de modelos de otimização, os quais visam maximizar (minimizar) um critério de desempenho como, por exemplo, a produção de um dado insumo, sujeito a restrições que descrevem as condições operacionais. A linguagem utilizada pela otimização para expressar os problemas de uma forma declarativa é conhecida universalmente por *programação matemática*. No que segue, apresentamos a linguagem de programação matemática de exemplificamos o seu uso em uma série de problemas ilustrativos e de interesse geral.

1.1.2 Elementos de um Problema de Otimização

Variáveis de Decisão

Parâmetros cujos valores definem uma solução para o problema. Em um sistema de produção, esses parâmetros podem definir as quantidades produzidas e os recursos utilizados.

Função Objetivo

Uma função das variáveis de decisão a ser minimizada ou maximizada. No sistema de manufatura, podemos estar interessados em minimizar custos, reduzir o número de homens-hora, e consequentemente aumentar a produtividade.

Restrições

Um conjunto de funções que define o espaço factível de soluções. No sistema de manufatura, as restrições estabelecem limites para os recursos utilizados, restrições operacionais do processo de produção bem como limitações físicas e tecnológicas.

Formulação do Problema de Otimização Generalizado

O problema geral de otimização é expresso em programação matemática como:

$$
\begin{aligned}
&Minimize \quad f(x) \\
&Sujeito\ a: \\
&\qquad\qquad g(x) \leq 0 \\
&\qquad\qquad h(x) = 0 \\
&\qquad\qquad x \in \mathbb{R}^n
\end{aligned}
$$

Onde $f : \mathbb{R}^n \to \mathbb{R}$ é a função objetivo, $g : \mathbb{R}^n \to \mathbb{R}^p$ e $h : \mathbb{R}^n \to \mathbb{R}^q$ são restrições que limitam o espaço de soluções factíveis, e x é o vetor com as variáveis de decisão.

1.1.3 Duas Exceções à Formulação Geral

Problemas sem função objetivo

O usuário deseja apenas encontrar um conjunto de decisões que sejam viáveis isto é, encontre $x \in \mathbb{R}^n$, tal que:

$$\begin{aligned} g(x) &\leq 0 \\ h(x) &= 0 \end{aligned}$$

Problemas com múltiplos objetivos

Em problemas reais, não é incomum procurar otimizar mais do que um objetivo. No problema de manufatura, o usuário pode desejar maximizar o lucro, maximizar a qualidade dos itens manufaturados e ainda minimizar o tempo de produção. Usualmente, estes problemas são reduzidos a problemas envolvendo apenas um objetivo (combinando-se múltiplos objetivos em apenas um ou, alternativamente, escolhendo-se um objetivo e introduzindo restrições). Tais problemas são transcritos em programação matemática como:

$$\begin{aligned} &Minimize && f_1(x) \\ &\vdots \\ &Minimize && f_k(x) \\ &x \in \mathbb{R}^n \\ &Sujeito\ a: \\ & && g(x) \leq 0 \end{aligned}$$

1.2 Programação Linear

1.2.1 Problema Exemplo

Um atleta deseja encontrar uma dieta otimizada, ou seja, um programa alimentar com tipos e quantidades de alimentos que atendam às suas necessidades mínimas. Os alimentos devem ser escolhidos de forma a minimizar o preço total. Os dados do problema são:

- N alimentos, tais como arroz, feijão, alface, etc;

- M tipos de substâncias alimentares, como proteínas, lipídios, etc;
- c_n é o preço unitário do alimento n;
- $a_{m,n}$ é a quantidade de substância m contida em cada unidade de alimento n; e
- b_m é a quantidade mínima de substância m a ser ingerida pelo atleta.

Exercício: modele o problema em programação matemática.

Variáveis: x_n é quantidade de alimento n a ser comprada e ingerida, $n = 1, \ldots, N$.

Restrições:

$$\begin{array}{ccccccccc} a_{1,1}x_1 & + & a_{1,2}x_2 & + & \ldots & + & a_{1,N}x_N & \geq & b_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & \ldots & + & a_{2,N}x_N & \geq & b_2 \\ \vdots & & \vdots & & \ldots & & \vdots & & \vdots \\ a_{M,1}x_1 & + & a_{M,2}x_2 & + & \ldots & + & a_{M,N}x_N & \geq & b_M \end{array}$$

onde $x_1, x_2, \ldots, x_N \geq 0$.

Função objetivo: $f = c_1x_1 + c_2x_2 + \ldots + c_Nx_N$

Formulação compacta:

$$\begin{array}{lllll} PL: & Minimize & c^Tx & & \\ & Sujeito\ a: & & & \\ & & Ax & \geq & b \\ & & x & \geq & 0 \\ & & x \in \mathbb{R}^N & & \end{array}$$

1.3 Programação Linear Inteira

Uma variedade de problemas reais podem ser formulados com o emprego de variáveis discretas. Dentre eles, citamos o problema de agendamento de trens, o problema de agendamento de tripulações de aviões, e o problema de alocação em sistemas de telecomunicações. Todos este problema tem uma característica em comum: fazem uso de variáveis inteiras ou discretas. Não se pode por exemplo dividir um vagão de trem em frações; não se pode alocar meio piloto a uma aeronave; e não se pode instalar uma fração de um servidor de telecomunicações. Estes e muitos outros problemas fazem

parte do universo da programação linear inteira, que engloba problemas da forma:

$$
\begin{array}{lll}
Minimize & c^T x & \\
Sujeito\ a: & & \\
 & Ax & \geq\ b \\
 & Cx & =\ d \\
 & x & \geq\ 0 \\
 & x \in \mathbb{Z}^n &
\end{array}
$$

1.3.1 Problema Exemplo

Dados do Problema

- um número m de possíveis locais para instalação de depósitos e siderúrgicas;
- um número n de clientes;
- d_i é a demanda de aço do cliente i e esta deve ser suprida por precisamente um depósito ou siderúrgica;
- u_j é a capacidade de um possível depósito a ser instalado no local j;
- o custo de transporte do depósito j para o cliente i é $c_{i,j}$; e
- o custo de instalação do depósito j é f_j.

Tarefa

Formule o problema de definir quais depósitos devem ser instalados de maneira a suprir a demanda e, ao mesmo tempo, minimizar o custo total de instalação e transporte.

Variáveis

- $x_{ij} = 1$ se o cliente i é atendido pelo depósito j
- $x_{ij} = 0$ caso contrário
- $y_j = 1$ se o depósito j é instalado
- $y_j = 0$ caso contrário

Formulação

$$
\begin{array}{lll}
Minimize & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} f_j y_j & \\
Sujeito\ a: & & \\
& \sum_{i=1}^{n} d_i x_{ij} \leq u_j y_j & j = 1, \ldots, m \\
& \sum_{j=1}^{m} x_{ij} = 1 & i = 1, \ldots, n \\
& x_{ij} \in \mathbb{B} & i = 1, \ldots, n \\
& & j = 1, \ldots, m \\
& y_j \in \mathbb{B} & j = 1, \ldots, m
\end{array}
$$

1.4 Programação Quadrática

O problema geral de programação quadrática é formulado como segue:

$$
\begin{array}{lll}
Minimize & \frac{1}{2}x^T Q x + c^T x & \\
Sujeitoa: & & \\
& Ax \leq b & \\
& Cx = d &
\end{array}
$$

onde Q é uma matriz simétrica.

Programação quadrática tém aplicações em identificação de parâmetros para modelos de processos, modelos estruturais e sistemas de controle, e em algoritmos como SQP (*Sequential Quadratic Programming.*

A dificuldade de se resolver tais problemas depende da natureza da matriz Q. Quais características de Q tornam o problema difícil? A dificuldade de se resolver tais problemas depende da natureza da matriz Q. Se $Q \geq 0$ (positiva semi-definida) ou $Q > 0$ (positiva definida) o problema é relativamente fácil de ser resolvido (ou seja, encontrar a solução ótima global). Se Q é indefinida (ou negativa semi-definida ou definida) então o problema é muito difícil.

1.5 Mínimos Quadrados Não-linear

O problema dos mínimos quadrados não-linear consiste de um problema da seguinte forma:

$$
\begin{array}{ll}
Minimize & \frac{1}{2}\|f(x)\|^2 \\
x \in \mathbb{R}^n &
\end{array}
$$

onde $\| \bullet \|$ corresponde à norma Euclidiana e $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ é uma função qualquer, contínua e diferenciável. Tais problemas têm aplicações no casamento de modelos com dados experimentais, tipicamente encontrados em estudos econômicos, aprendizagem automática e engenharia. Dado uma série de exemplos de treinamento (pares entrada-saída), o problema de treinar uma rede neural a aproximar a função descrita pelos pares entrada-saída consiste em um problema de regressão linear.

1.6 Mínimos Quadrados Linear

Seja $w(h)$ um modelo que descreve a relação entre a altura e o peso médio das pessoas do sexo feminino. Suponha que o modelo escolhido é um polinômio da forma:

$$w(h) = x_3 h^3 + x_2 h^2 + x_1 h + x_0,$$

ou seja, $w(h)$ é um polinômio de terceira ordem que modela o peso como uma função da altura. Os dados amostrais são dados na Tabela 1.1

Tabela 1.1: Dados amostrais

Exemplo (i)	Altura (h_i)	Peso (w_i)
1	1.5	55
2	1.54	53
3	1.58	56
4	1.60	52
5	1.65	58
6	1.67	54
7	1.70	64
8	1.72	70
9	1.72	71
10	1.75	75
11	1.80	82
12	1.81	81

1.6.1 Problema

Encontre os parâmetros x_3, x_2, x_1 e x_0 que minimizam a função erro, a qual consiste na soma dos quadrados dos erros de predição:

$$\begin{array}{ll} Minimize & \frac{1}{2}\sum_{i=1}^{n} \|w(h_i) - w_i\|^2 \\ x_1, x_2, x_3, x_4 & \end{array}$$

A solução ótima para o problema acima, tomando como dados as entradas da Tabela 1.1, é:

$$\begin{aligned} x_1 &= 100.0000 \\ x_2 &= -33.1098 \\ x_3 &= -62.9920 \\ x_4 &= 41.7403 \end{aligned}$$

A Figura 1.1 ilustra os dados amostrais juntamente com a curva de aproximação $w(h)$ dada pelo polinômio $w(h) = x_3h^3 + x_2h^2 + x_1h + x_0$.

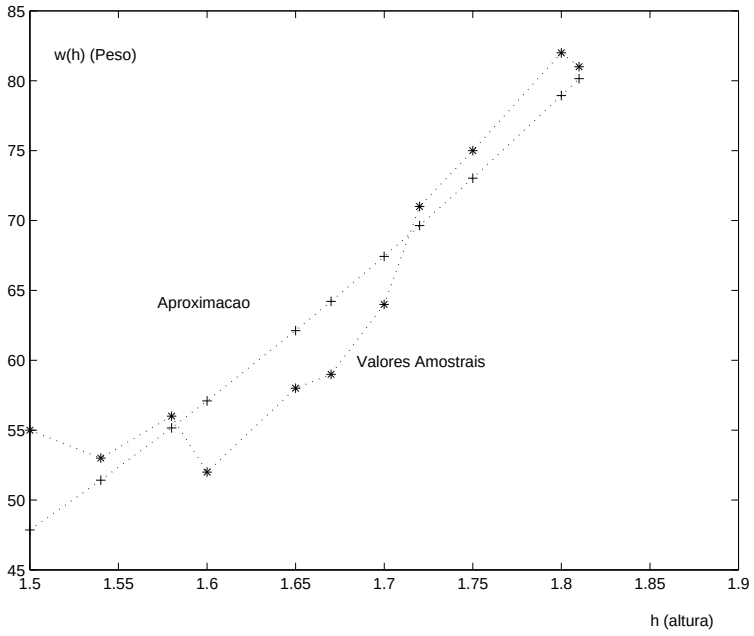


Figura 1.1: Predição peso com base na altura

1.7 Equações Não-Lineares

Sistemas de equações não-lineares aparecem em problemas de otimização, mas também em equações diferenciais e suas formas discretizadas, jogos dinâmicos e processos iterativos. Seja $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ uma função contínua e diferenciável, o problema de interesse é definido como segue:

$$Encontre\ x^*\ tal\ que\ f(x^*) = 0$$

Alguns algoritmos, transformam este problema em um problema de otimização irrestrita:

$$\begin{aligned} &Minimize \quad \|f(x)\|^2 \\ &x \in \mathbb{R}^n \end{aligned}$$

onde $\| \bullet \|$ é uma norma vetorial.

1.7.1 Aplicação em Sistemas de Controle

Dado um sistema de equações diferenciais, encontre um ponto de equilíbrio. Considere o sistema de equações diferenciais abaixo:

$$\begin{aligned} \dot{x}_1 &= -x_1 - 1 \\ \dot{x}_2 &= 2x_1x_2 - 2\sqrt{2}x_1 + 2x_2 - 2\sqrt{2} + sin(x_2 - \sqrt{2}) \\ \dot{x}_3 &= 2x_2 - 2\sqrt{2} \end{aligned}$$

De uma forma mais compacta, o sistema acima pode ser escrito como $\dot{x} = f(x)$ onde $x = [x_1, x_2, x_3]$. Um problema típico no domínio de controle não linear é a busca de um ponto de operação ou de equilíbrio, i.e., um x tal que $\dot{x} = 0$. O problema de encontrar um ponto de equilíbrio pode ser reduzido à solução um sistema de equações não lineares: $f(x) = 0$. Para o exemplo acima, um ponto de equilíbrio é $x^* = [-1, \sqrt{2}, x_3]$.

1.8 Otimização Não-linear sem Restrições

Otimização irrestrita constitui um bloco fundamental no desenvolvimento de software. Algoritmos para solução de problemas de otimização restrita fazem uso de otimização irrestrita. O problema de otimização irrestrita (sem restrições) é definido como:

$$\begin{aligned} &Minimize \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

Tipicamente procura-se um ótimo local x^*, ou seja, x^* tal que $f(x^*) \leq f(x)$ para todo $x \in B(x^*, \delta) = \{x : \|x - x^*\| \leq \delta\}$ e $\delta > 0$. Otimização global se preocupa em encontrar um vetor x^* cujo valor $f(x^*) \leq f(x)$ para todo $x \in \mathbb{R}^n$.

1.8.1 Problema Exemplo

Seja $z = (x, y)$ a coordenada onde será instalada uma central telefônica. Suponha que as chamadas são recebidas de um conjunto $S = \{z_1 = (x_1, y_1), \ldots, z_m = (x_m, y_m)\}$ de localidades com probabilidade uniforme. Seja Z a variável randômica associada com o local das chamadas. Portanto, Z assume valores do conjunto S tal que a probabilidade de uma chamada vir do local k é $Pr\{Z = z_k\} = 1/m$, para $k = 1, \ldots, m$.

Tarefa

Qual deve ser a localização da central telefônica para que $E[(Z - z)^2]$ seja minimizado? $E[f(Z)]$ é o valor esperado da função $f(Z)$ da variável randômica Z.

Solução

Desenvolvendo a expressão do valor esperado, deduzimos:

$$\begin{aligned} E[(Z - z)^2] &= \sum_{k=1}^{m} \left[(z_k - z)^2 \times Pr\{Z = z_k\}\right] \\ &= \sum_{k=1}^{m} \left[(x_k, y_k) - (x, y)\right]^2 / m \end{aligned}$$

Assim, o problema pode ser reduzido a um problema de mínimos quadrados.

1.9 Otimização Não-linear com Limites Superiores/Inferiores

Esta classe de problemas é dada pela formulação a seguir:

$$\begin{aligned} &Minimize \quad f(x) \\ &x \in \mathbb{R}^n \\ &Subjeito\ a: \quad l \leq x \leq u \end{aligned}$$

onde f é uma função contínua, diferenciável.

Estes modelos têm aplicações em engenharia e na identificação de modelos físicos, onde as grandezas e parâmetros são sujeitos a limites. Alguns algoritmos de otimização restrita resolvem sequências de problemas com limites superiores e inferiores.

1.10 Otimização Não-linear com Restrições

Os problemas de otimização não-linear com restrições consistem em minimizar uma função não-linear (contínua e diferenciável) sujeita a restrições não-lineares (contínuas e diferenciáveis). O problema geral é da forma:

$$\begin{aligned} &Minimize && f(x) \\ &Sujeito\ a: && \\ & && g(x) \leq 0 \\ & && h(x) = 0 \end{aligned}$$

onde $f : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^n \to \mathbb{R}^p$ e $h : \mathbb{R}^n \to \mathbb{R}^q$ são funções contínuas e diferenciáveis. Os modelos de otimização não-linear restritos são os mais gerais no domínio da otimização contínua.

1.10.1 Problema Exemplo

Desejamos instalar uma estação de bombeiros de forma que a mesma esteja dentro de um raio r (km) de um conjunto $S = \{(x_1, y_1), \ldots, (x_p, y_p)\}$ de prédios nas proximidades da unidade dos bombeiros. Além disso, desejamos localizá-la o mais afastado possível de outras estações de bombeiros $T = \{(\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_q, \tilde{y}_q)\}$.

Tarefa

Formule este problema em linguagem de otimização.

Formulação

$$\begin{aligned} &Maximize && d \\ &Sujeito\ a: && \\ & && \|(x,y) - (x_j, y_j)\| \leq r \qquad j = 1, \ldots, p \\ & && \|(x,y) - (\tilde{x}_j, \tilde{y}_j)\| \geq d \qquad j = 1, \ldots, q \end{aligned}$$

Observações:

$$\begin{aligned} d &\leq Min\{\|(x,y)-(\tilde{x}_j,\tilde{y}_j)\| : j=1,\ldots,q\} \\ r &\geq Max\{\|(x,y)-(x_j,y_j)\| : j=1,\ldots,p\} \end{aligned}$$

1.11 Programação Semi-Definita

Esta classe compreende problemas com função objetivo linear e restrições envolvendo matrizes e suas propriedades (tais como, positiva definida e semi-definida). Tipicamente, uma matriz $F_i(x)$ é definida como uma função afim com parâmetros dados pelo vetor x. Mais precisamente:

$$F_i(x) = F_{i,0} + F_{i,1}x_1 + \ldots + F_{i,n}x_n$$

onde $F_{ij} \in \mathbb{R}^{n\times n}$ é uma matriz simétrica, ou seja, $F_{i,j} = F_{i,j}^T$. Os problemas são expressos na forma a seguir:

$$\begin{aligned} &Minimize \quad c^T x \\ &Sujeito\ a: \\ &\qquad\qquad F_i(x) \geq 0 \qquad i=1,\ldots,m \end{aligned}$$

onde $F_i(x) \geq 0$ significa que a matriz $F_i(x)$ deve ser positiva semi-definida (*linear matrix inequality*).

1.11.1 Exemplo

Considere o seguinte sistema dinâmico

$$\dot{x} = Ax$$

Uma condição suficiente para que o sistema convirja para $x^* = 0$ à medida que $t \to \infty$, a partir de qualquer estado inicial $x(0)$, é a existência de uma função $V(x)$ com as seguintes propriedades:

- $V(x)$ é positiva definida, ou seja, para todo $x \neq 0$ tem-se $V(x) > 0$ e $V(0) = 0$
- $\dot{V}(x) < 0$ para $x \neq 0$.

Tal função é conhecida como função Lyapunov.

Ccomo encontrar tal função?

Seja $V(x) = x^T Px$ para $P > 0$. Neste caso $V(x)$ satisfaz a condição de ser positiva definida. Como fazer para satisfazer a segunda condição?

$$\begin{aligned}\frac{d}{dt}V &= \frac{d}{dt}(x^T Px) \\ &= \dot{x}^T Px + x^T P\dot{x} \\ &= (x^T A^T)Px + x^T P(Ax) \\ &= x^T(A^T P + PA)x\end{aligned}$$

Então a condição $\dot{V} < 0$ é equivalente a $A^T P + PA < 0$. O problema de encontrar uma função Lyapunov, quando esta existe, pode ser colocado em programação matemática:

$$\begin{aligned}&Minimize \quad c^T x \\ &Sujeito\ a: \\ &\qquad P(x) > 0 \\ &\qquad A^T P(x) + P(x)A < 0\end{aligned}$$

onde $P(x)$ gera o espaço de matrizes simétricas.

1.12 Referências

- Optimization Tree: http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/
- OR Notes: http://www.ms.ic.ac.uk/jeb/or/contents.html

1.13 Exercícios

EX 1.1 (Produção Industrial Otimizada): Uma companhia manufatura dois produtos (x e y) usando duas máquinas (M_1 e M_2). Para produzir uma unidade do item x, são necessários 50 minutos de processamento na máquina M_1 e 30 minutos de processamento na máquina M_2. Para produzir cada unidade do item y, são necessários 24 minutos de processamento na máquina M_1 e 33 minutos de processamento na máquina M_2. No início da semana, tem-se 30 unidades de x e 90 unidades de y em estoque. O tempo disponível para processamento na máquina M_1 durante a semana é de 40 horas, já para a máquina M_2 o tempo disponível é de 35 horas. A demanda por item x na semana corrente está prevista em 75

unidades, enquanto que para o item y a estimativa é de 95 unidades. O objetivo da empresa é maximiar a quantidade agregada de itens x e y no fim da semana.

Tarefa: formule em programação linear o problema de decidir quantas unidades de cada item devem ser produzidas.

EX 1.2 (Mínimos Quadrados): Formule os problemas a seguir como problemas de mínimos quadrados. Para cada problema forneça a matriz A e o vetor b tal que o problema possa ser expresso como:

$$\begin{array}{ll} Minimize & \|Ax-b\|^2 \\ x & \end{array}$$

Tarefas

- Minimize $x^Tx + \|Bx - d\|^2$, onde $B \in \mathbb{R}^{p\times n}$, e $d \in R^p$ e $x \in R^n$.
- Minimize $\|Bx-d\|^2 + \|Fx-g|^2$, onde $B \in \mathbb{R}^{p\times n}$, $d \in \mathbb{R}^p$, $F \in \mathbb{R}^{q\times n}$, $g \in \mathbb{R}^q$ e $x \in R^n$.
- Minimize $x_1^2 + 2x_2^2 + 3x_3^2 + (x_1 - x_2 + x_3 - 1)^2 + (-x_1 - 4x_2 + 2)^2$ (Sugestão: expresse os três termos principais na forma procurada, depois utilize os resultados obtidos em (a) e (b).)

EX 1.3 (Projeto de Latas): O preço do cm^2 de latão é R$ 2,00. Deseja-se produzir latas cilíndricas sem tampa, com volume igual ou superior a 500 cm^3 e de custo mais baixo possível. Formule em programação matemática (otimização não-linear com restrições) o problema de dimesionar-se a lata, ou seja, de encontrar-se a altura h (cm) e raio r (cm) da lata .

EX 1.4 (Rede de Abastecimento de Água): Considere uma região onde serão instaladas M caixas de água, sendo a caixa m localizada na posição (x_m, y_m) conhecida. Suponha que um tubo mestre deverá ser instalado horizontalmente, ou seja, ao longo da linha y_t, e que tubos verticais secundários serão construídos para conectar o tubo mestre a cada caixa. Desejamos encontrar a altura para instalar o tubo mestre (valor de y_t) tal que o comprimento total da rede de abastecimento seja o menor possível. Formule este problema em programação matemática.

EX 1.5 (*The Queens Problem*): Considere um tabuleiro de xadrez (8×8 posições). Deseja-se colocar o maior número possível de rainhas no tabuleiro, de maneira que nenhuma delas possa ser eliminada por nenhuma das demais rainhas. Formule

o problema de encontrar tais posições em programação matemática (otimização linear inteira).

EX 1.6 (Rastreamento de Objetos): Dois radares (R_1 e R_2) estão localizados nas posições $z_1 = (x_1, y_1)$ e $z_2 = (x_2, y_2)$ respectivamente. Cada radar informa o centro de rastreamento e controle da respectiva distância a um objeto desconhecido $z = (x, y)$. Sendo d_1 a distância de R_1 a z e d_2 a distância de R_2 a z, formule o problema de calcular as coordenadas $z = (x, y)$ do objeto desconhecido.

(Sugestão: expresse o problema em duas equações não-lineares.)

Capítulo 2

Minimização de Funções com o Método de Descenso

Este capítulo se concentra nas propriedades gerais dos problemas de otimização não-linear sem restrições e no algoritmo de descenso. São introduzidos conceitos de ótimo local e ótimo global, bem como condições necessárias e condições suficientes para que uma solução candidata seja um ótimo local. O algoritmo de descenso e as condições para convergência global a partir de um ponto qualquer para um ótimo local são apresentados na seqüência.

2.1 Problemas de Interesse

Consideraremos primeiramente o problema de minimização irrestrita:

$$\begin{aligned} P_1: \quad & Minimize\ f(x) \\ & x \in \mathbb{R}^n \\ & onde\ f: \mathbb{R}^n \to \mathbb{R} \end{aligned}$$

Também investigaremos a solução de equações não-lineares, mas com maior ênfase no próximo capítulo. Este problema pode ser formulado em programação matemática como:

$$\begin{aligned} P_2: \quad & Encontre\ x \in \mathbb{R}^n,\ tal\ que\ c(x) = 0 \\ & onde\ c: \mathbb{R}^n \to \mathbb{R}^m \end{aligned}$$

Vamos estudar conceitos básicos em otimização, apresentando os algoritmos de descenso e de Newton. Os dois algoritmos podem, em princípio, ser utilizados para resolver P_1 e P_2, todavia apresentam propriedades distintas. Assumiremos que as funções f e c são contínuas, diferenciáveis e algumas vezes duas vezes diferenciáveis. Aqui, nos

concentramos no algoritmo de descenso para solução de P_1.

2.2 Fundamentos de Otimização Irrestrita

2.2.1 O Que É Uma Solução para P_1?

Em general, gostaríamos de encontrar uma solução ótima global, ou seja, um ponto x^* tal que $f(x^*) \leq f(x)$ para todo $x \in \mathbb{R}^n$. Tipicamente, é extremamente difícil encontrar uma solução global porque o conhecimento de f é usualmente local. A grande maioria dos algoritmos é capaz de encontrar uma solução ótima para uma certa vizinhança de f. Formalmente, dizemos que x^* é um ótimo local se existe uma vizinhança N de x^* tal que $f(x^*) \leq f(x)$ para todo $x \in N$, onde $N = N(x^*, \epsilon) = \{x \in \mathbb{R}^n : \|x - x^*\| \leq \epsilon\}$ para algum $\epsilon > 0$ e qualquer norma vetorial $\| \bullet \|$. Um ponto x^* é um ótimo local estrito se $f(x^*) < f(x)$ para todo $x \in N$, $x \neq x^*$.

Exemplo

Considere a função $f(x) = x^4 \cos(1/x) + 2x^4$ para $x \neq 0$ e $f(0) = 0$. Note que f é contínua e duas vezes diferenciável. O ponto $x^* = 0$ é um ótimo global, mas existe um número infinito de ótimos locais.

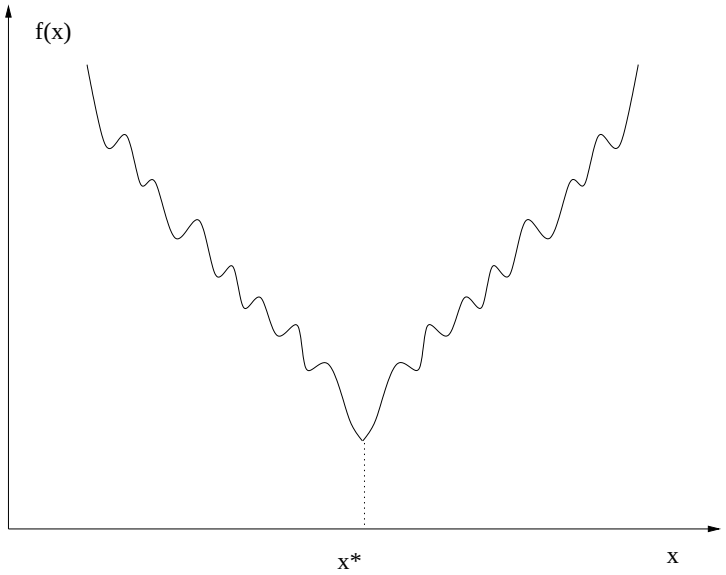


Figura 2.1: Exemplo de função com um número infinito de ótimos locais

2.2.2 Reconhecendo Um Mínimo Local

A partir das definições acima, parece que a única maneira de se identificar um mínimo local é através da comparação com os pontos na vizinhança. Quando a função é suave, entretanto, há formas mais eficientes e práticas de se identificar um mínimo local.

Expansão de Taylor de Primeira Ordem

Assumindo que a função f é diferenciável, a expansão de Taylor consiste em aproximar f em torno de um ponto $x \in \mathbb{R}^n$ como:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + O(\delta \|\Delta x\|^2).$$

onde $\nabla f(x) = [\partial f(x)/\partial x_1, \ldots, \partial f(x)/\partial x_n]$ é o gradiente de f e $O(\delta\|\Delta x\|^2)$ é o resíduo (erro da aproximação). Para pequenos passos ($\|\Delta x\|$ pequeno) podemos desprezar o resíduo, resultando na seguinte aproximação:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x$$

Teorema 1 *(Condição necessária de primeira ordem) Se x^* é um ótimo local e f é uma função continuamente diferenciável em uma vizinhança aberta, então $\nabla f(x^*) = 0$.*

Prova. Vamos supor o contrário. Seja $\Delta x = -\delta \nabla f(x^*)$ para algum $\delta > 0$ pequeno, tal que $x = (x^* + \Delta x)$ esteja na vizinhança de x^*. Portanto, $f(x) = f(x^* + \Delta x) = f(x^*) + \nabla f(x^*)^T[-\delta \nabla f(x^*)] < f(x^*)$. Isto configura uma contradição. □

Expansão de Taylor de Segunda Ordem

Se f é duas vezes diferenciável, podemos utilizar uma aproximação de segunda ordem como segue:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x) \Delta x + O(\delta \|\Delta x\|^3)$$

onde a matriz $\nabla^2 f(x)$ é denominada de Hessiana, sendo sua entrada i, j definida por $\partial^2 f(x)/\partial x_i \partial x_j = \partial[\partial f(x)/x_j]/\partial x_i$. Para um pequeno passo Δx, o resíduo pode ser desprezado resultando na expansão abaixo:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x) \Delta x$$

Teorema 2 *(Condição necessária de segunda ordem) Se x^* é um ótimo local de f e $\nabla^2 f$ é contínua em uma vizinhança aberta de x^*, então $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*)$ é positiva semi-definida.*

Prova. Pelo Teorema 1, $\nabla f(x^*) = 0$. Pela aproximação de Taylor de segunda ordem, podemos dizer que $f(x^*+\Delta x) = f(x^*)+\frac{1}{2}\Delta x^T \nabla^2 f(x^*)\Delta x$. Suponha que $\nabla^2 f(x^*)$ não é positiva semi-definida, o que implica que existe Δx pequeno tal que $\Delta x^T \nabla^2 f(x^*)\Delta x < 0$. Portanto, $f(x^* + \Delta x) = f(x^*) + \Delta x^T \nabla^2 f(x^*)\Delta x < f(x^*)$ e, conseqüentemente, x^* não pode ser um ótimo local de f. □

Teorema 3 *(Condição suficiente de segunda ordem) Suponha que $\nabla^2 f(x)$ seja contínua em uma vizinhança de x^*, $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*)$ seja positiva definida. Então x^* é um ótimo local estrito de f.*

Prova. Seja $x = x^* + \Delta x$ um ponto na vizinhança de x^*. Para Δx suficientemente pequeno, $f(x) = f(x^* + \Delta x) = f(x^*) + \nabla f(x^*)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x^*)\Delta x = f(x^*) + \frac{1}{2}\Delta x^T \nabla^2 f(x^*)\Delta x > f(x^*)$, porque $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*)$ é positiva definida. A última desigualdade implica que todo x na vizinhança de x^*, $x \in \mathbb{R}^n$ tal que $\|x - x^*\| \leq \delta$ para $\delta > 0$ pequeno o suficiente, induz $f(x) > f(x^*)$. □

Observação: as condições suficientes de segunda ordem não são necessárias, ou seja, uma solução pode ser um mínimo estrito e mesmo assim não passar no teste de suficiência. Um exemplo simples é a função $f(x) = x^4$, para a qual o ponto $x^* = 0$ é um mínimo local estrito cuja Hessiana não é positiva definida.

Teorema 4 *Se f é convexa, então todo o mínimo local é também um mínimo global.*

Prova. Seja x^* um mínimo local e suponha, por absurdo, que x^* não seja um mínimo global. Seja $\tilde{x}$ um mínimo global e considere o segmento de reta que une os pontos x^* e $\tilde{x}$, ou seja,

$$x = \alpha\tilde{x} + (1 - \alpha)x^* \text{ onde } \alpha \in (0, 1].$$

Por convexidade de f, temos que $f(x) \leq \alpha f(\tilde{x}) + (1 - \alpha)f(x^*) < \alpha f(x^*) + (1 - \alpha)f(x^*) = f(x^*)$. Qualquer vizinhança N de x^* contém um pedaço do segmento de reta acima definido e, portanto, sempre existirá um x em N que satisfaça a desigualdade $f(x) < f(x^*)$. Conclui-se que x^* não é um ótimo local. □

2.3 O Algoritmo de Descenso

A grande maioria dos algoritmos de otimização são iterativos. Dado um ponto x_0, eles produzem uma sequência de pontos $x_0, x_1, x_2, \ldots, x_T = \{x_k\}$ convergente para um mínimo local. Tipicamente, um processo iterativo consiste em obter uma nova solução a partir da solução anterior:

$$x_{k+1} = G_k(x_k) \qquad para\ k = 0, \ldots, T-1$$

Para que a seqüência convirja para um ótimo local, os algoritmos induzem uma redução (não necessariamente a cada iteração) da função objetivo, ou seja, $f(x_k) < f(x_{k-m})$.

2.3.1 O Algoritmo de Descenso em Detalhes

O algoritmo de descenso consiste em escolher uma direção p_k, a cada iteração k, e fazer uma busca ao longo desta direção a partir do iterando x_k. Em outras palavras, o algoritmo resolve o seguinte problema de uma dimensão:

$$\begin{array}{ll} Minimize & f(\alpha_k) = f(x_k + \alpha_k p_k) \\ \alpha_k > 0 & \end{array} \tag{2.1}$$

A solução exata de (2.1) induziria o benefício máximo da direção p_k; todavia, a minimização exata é cara computacionalmente e não necessária. Em vez disso, o método de descenso gera um número limitado de passos até encontrar um ponto α_k que aproxima o mínimo de (2.1). Um novo ponto é gerado, $x_{k+1} = x_k + \alpha_k p_k$, uma nova direção p_{k+1} é encontrada e o processo se repete.

2.3.2 Direção de Busca

A direção de maior descenso, $-\nabla f(x_k)$, é a opção óbvia para a busca direcional. A direção de maior descenso produz a maior taxa de redução no valor de f dentre todas as possíveis direções. O método de maior descenso (*steepest descent*) executa uma busca ao longo da direção $p_k = -\nabla f(x_k)$ a cada iteração. O passo α_k pode ser escolhido de várias formas. Uma vantagem do método de descenso é que ele requer apenas o gradiente, $\nabla f(x_k)$, não necessitando da Hessiana. Todavia, ele pode se tornar excessivamente lento em problemas difíceis.

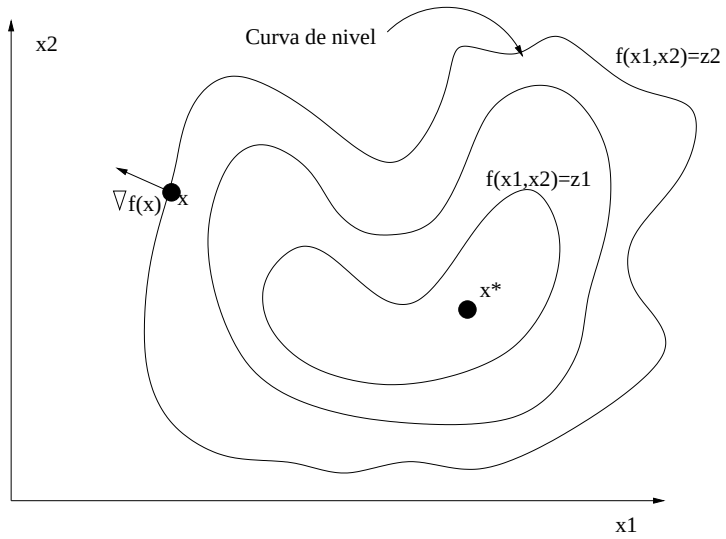


Figura 2.2: Ilustração do gradiente de uma função, $\nabla f(x_k)$, que assume a direção ortogonal às curvas de nível.

2.3.3 Encontrando a Direção de Busca

O processo iterativo do método de descenso é dado por: $x_{k+1} = x_k + \alpha_k p_k$, onde α_k é um escalar positivo (chamado de passo) e p_k é um vetor (chamado de direção de descenso). O sucesso do método depende da escolha adequada da direção e do passo. O algoritmo exige que p_k seja uma direção de descenso, ou seja, $\nabla f(x_k)^T p_k < 0$. Outros algoritmos utilizam uma direção da forma $p_k = -B_k^{-1}\nabla f(x_k)$, onde B_k é uma matriz simétrica não-singular. Note que quando B_k é positiva definida, p_k define uma direção de descenso.

2.3.4 Encontrando o Passo

Durante a computação do passo α_k, deparamo-nos com um conflito. Por um lado, desejamos escolher α_k que produza uma redução substancial em f mas, por outro lado, não desejamos gastar muito esforço computacional durante a escolha. O passo ideal minimizaria a função $\phi(\alpha)$, definida por: $\phi(\alpha) = f(x_k + \alpha p_k)$, onde $\alpha > 0$. Encontrar um minimizador local pode ser muito caro, necessitando várias computações de f e ∇f. Métodos práticos executam uma busca inexata, procurando obter uma redução satisfatória de f a um custo mínimo. Tipicamente, os métodos aproximados estabelecem condições para um decréscimo satisfatório.

Condições para Passo Satisfatório

A condição de Armijo é definida pela seguinte desigualdade:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k$$

onde $c_1 \in (0, 1)$. Em outras palavras, a condição estabelece que a redução em f deve ser proporcional ao passo α_k e à derivada direcional $\nabla f(x_k)^T p_k$. Seja $l(\alpha) = f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k$, então a condição de Armijo pode ser expressa como $f(x_k + \alpha p_k) \leq l(\alpha)$. Note que a função $l(\alpha)$ é uma função decrescente e linear em α. A Figura 2.3 ilustra a condição de Armijo.

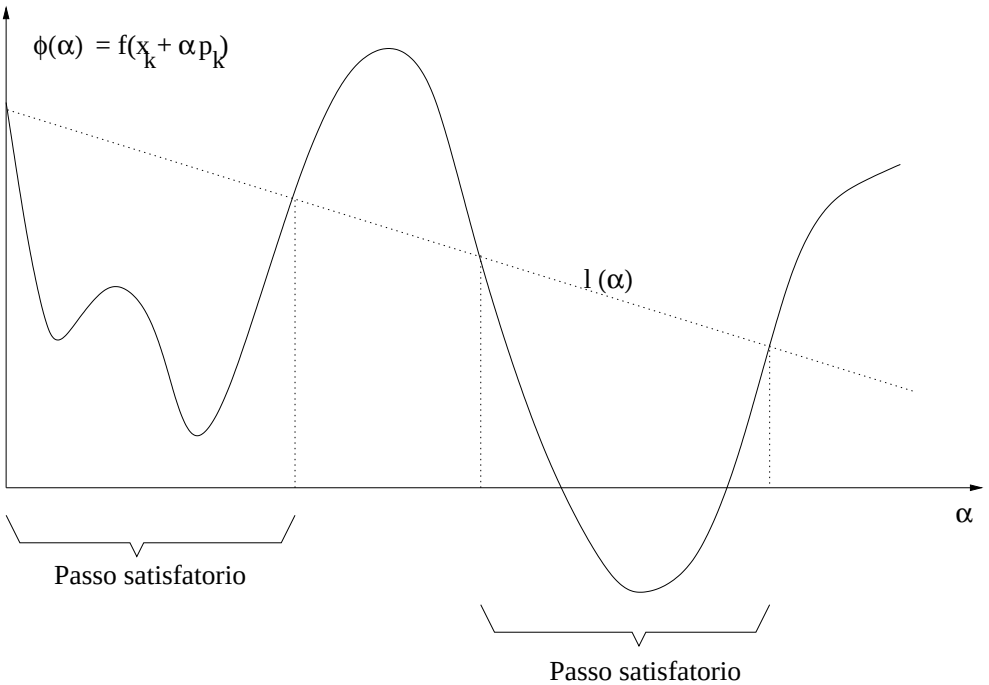


Figura 2.3: Ilustração de passo satisfatório.

A condição de Armijo não é, entretanto, suficiente para garantir progresso razoável e satisfatório do algoritmo, pois a condição pode ser satisfeita por passos extremamente pequenos. Para evitar passos muito pequenos, uma condição de curvatura é introduzida, a qual exige que:

$$\nabla f(x_k + a_k p_k)^T p_k \geq c_2 \nabla f(x_k)^T p_k,$$

para alguma constante $c_2 \in (c_1, 1)$. Note que o lado esquerdo é simplesmente

$d\phi(\alpha)/d\alpha = \phi'(\alpha)$[1], assim a condição de curvatura garante que a taxa de decrescimento $\phi'(\alpha)$ seja c_2 vezes o gradiente $\phi'(0)$. [Em outras palavras, $\phi'(\alpha)$ deve ser menos negativo do que $\phi'(0)$.] Isso faz sentido: se $\phi'(\alpha)$ é bastante negativo, então temos uma indicação de que podemos reduzir f substancialmente ao longo da direção escolhida; caso contrário, se $\phi'(\alpha)$ é pouco negativo, temos um sinal de que não podemos reduzir f substancialmente.

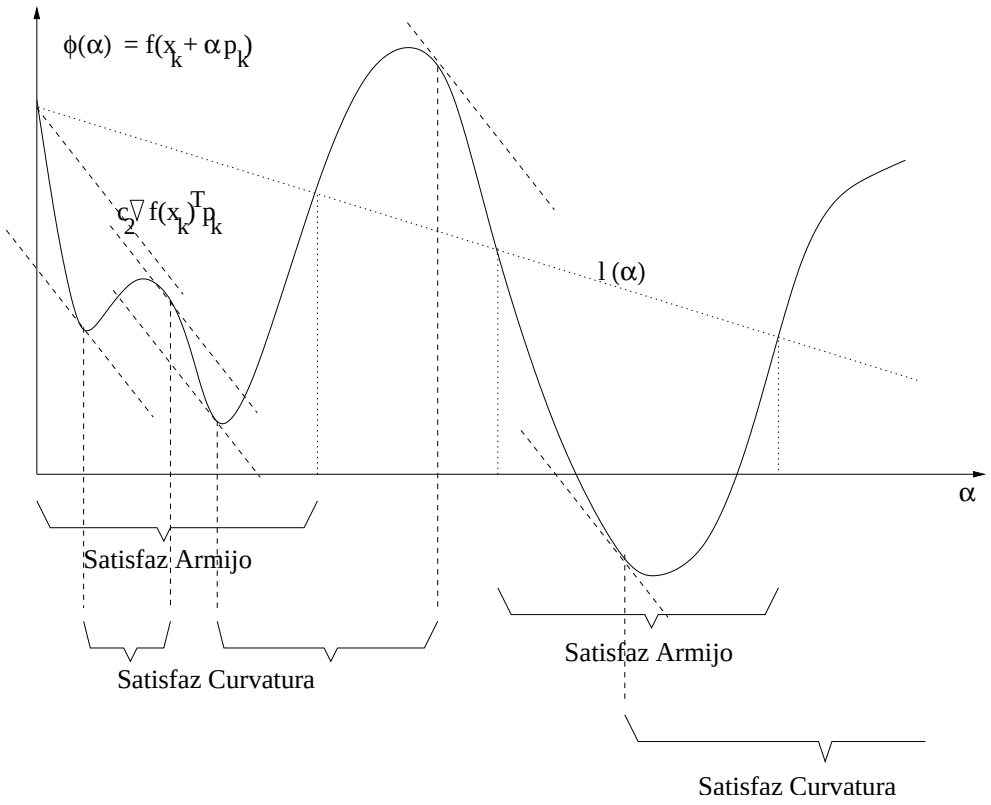


Figura 2.4: Ilustração das condições de Armijo e de curvatura.

As condições suficientes para redução da função objetivo e de curvatura são conhecidas como condições de Wolfe:

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k \\ \nabla f(x_k + a_k p_k)^T p_k &\geq c_2 \nabla f(x_k)^T p_k \end{aligned} \tag{2.2}$$

onde $0 < c_1 < c_2 < 1$. Não é difícil provar que existem passos α_k que satisfazem as condições de Wolfe se a função f é suave e possui um limite inferior.

[1]Use a regra da cadeia para verificar esta afirmação: $dz(y(x))/dx = (dz/dy)(dy/dx)$, onde $z(y)$ é uma função de y e $y(x)$ é uma função de x.

Lema 1 *Seja $f : \mathbb{R}^n \to \mathbb{R}$ uma função continuamente diferenciável. Seja p_k uma direção de descenso para o ponto x_k, e assuma que f é limitada inferiormente ao longo do raio $\{x_k + \alpha p_k : \alpha > 0\}$. Então, se $0 < c_1 < c_2 < 1$, existem intervalos para os passos que satisfazem as condições de Wolfe.*

Prova. Uma vez que $\phi(\alpha) = f(x_k + \alpha p_k)$ é limitada inferiormente para todo $\alpha > 0$ e uma vez que $0 < c_1 < c_2 < 1$, a linha $l(\alpha) = f(x_k) + \alpha c_1 \nabla f(x_k)^T p_k$ deve interceptar a curva $\phi(\alpha)$ pelo menos uma vez. Seja $\alpha' > 0$ o menor valor α tal que $l(\alpha)$ intercepta $\phi(\alpha)$, ou seja,

$$f(x_k + \alpha' p_k) \quad = \quad f(x_k) + \alpha' c_1 \nabla f_k^T p_k \tag{2.3}$$

A condição de Armijo é claramente satisfeita por todos os passos menores que α'.

A partir do teorema do valor médio[2], deve existir $\alpha'' \in (0, \alpha')$ tal que

$$f(x_k + \alpha' p_k) - f(x_k) = \alpha' \nabla f(x_k + \alpha'' p_k)^T p_k \tag{2.4}$$

Combinando (2.3) e (2.4), obtemos

$$\nabla f(x_k + \alpha'' p_k)^T p_k = c_1 \nabla f_k^T p_k > c_2 \nabla f_k^T p_k \tag{2.5}$$

uma vez que $c_1 < c_2$ e $\nabla f_k^T p_k < 0$. Assim, α'' satisfaz as condições de Wolfe, e as desigualdas (2.2) são estritas. Portanto, de acordo com a propriedade de que f é suave, deve existir um intervalo em torno de α'' para o qual as condições de Wolfe são satisfeitas. □

2.3.5 Redução Suficiente e Retrocesso

Utilizando-se o chamado método de retrocesso, podemos sistematicamente encontrar um passo α_k que satisfaz as condições as condições de Wolfe (2.2) sem considerar a segunda.

Procedimento

Escolha $\alpha' > 0$ (tipicamente $\alpha' = 1$)
Escolha ρ, $c \in (0, 1)$
Faça $\alpha \leftarrow \alpha'$

[2]O teorema do valor médio diz que $f(y) - f(x) = \nabla f(z)^T (y - x)$ sendo z um vetor que pertence ao segmento $x - y$.

Repita até que $f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^T p_k$
    Faça $\alpha \leftarrow \rho\alpha$
Fim-repita
Termine com $\alpha_k \leftarrow \alpha$

O procedimento acima encontra um passo satisfatório (i.e., que satisfaz as condições de Wolfe) após um número finito de iterações pois α_k eventualmente se torna pequeno o suficiente para satisfazer a condição de Armijo.

2.3.6 Convergência do Método de Descenso

Para que o algoritmo de descenso obtenha convergência global devemos escolher não apenas direções apropriadas, mas também passos adequados. O teorema a seguir descreve condições suficientes para que o método de descenso convirja, a partir de qualquer ponto inicial, para um mínimo local.

Teorema 5 *Considere uma iteração (2.1), onde p_k é uma direção de descenso e α_k satisfaz as condições de Wolfe (2.2). Suponha que f é limitada por baixo em $\mathbb{R}^n$ e que f é continuamente diferenciável em um cojunto aberto N que contém o conjunto $\mathcal{L} = \{x : f(x) \leq f(x_0)\}$, onde x_0 é o ponto inicial da iteração. Assuma ainda que ∇f é Lipschitz contínua em N, i.e., existe uma constante $L > 0$ tal que*

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{para todo } x, \tilde{x} \in N. \tag{2.6}$$

Então

$$\sum_{k \geq 0} \cos(\theta_k)^2 \|\nabla f_k\|^2 < \infty. \tag{2.7}$$

Prova. A partir da iteração $x_{k+1} = x_k + \alpha_k p_k$ e a segunda condição de Wolfe (2.2), $\nabla f(x_k + a_k p_k)^T p_k \geq c_2 \nabla f(x_k)^T p_k$, temos que

$$(\nabla f_{k+1} - \nabla f_k)^T p_k \geq (c_2 - 1)\nabla f_k^T p_k$$

enquanto que a condição Lipschitz e a desigualdade de Schwartz[3] implicam

$$\begin{aligned}(\nabla f_{k+1} - \nabla f_k)^T p_k &\leq |(\nabla f_{k+1} - \nabla f_k)^T p_k| \\ &\leq \|\nabla f_{k+1} - \nabla f_k\| \|p_k\| \\ &\leq L\|x_k + \alpha_k p_k - x_k\| \|p_k\| \\ &\leq \alpha_k L \|p_k\|^2.\end{aligned}$$

Combinando as duas relações acima, obtemos:

$$\alpha_k \geq \frac{c_2 - 1}{L} \frac{\nabla f_k^T p_k}{\|p_k\|^2}.$$

Substituindo esta desigualdade na primeira condição de Wolfe, verificamos

$$f_{k+1} \leq f_k - c_1 \frac{1 - c_2}{L} \frac{(\nabla f_k^T p_k)^2}{\|p_k\|^2}.$$

Fazendo θ_k ser o ângulo entre p_k e a direção de maior descenso $-\nabla f_k$, verifica-se que

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}.$$

Fazendo $c = c_1(1 - c_2)/L$ e combinando as duas últimas expressões obtemos

$$f_{k+1} \leq f_k - c \cos^2\theta_k \|\nabla f_k\|^2.$$

Somando a expressão acima para todos os índices menores ou iguais a k, obtemos

$$f_{k+1} \leq f_0 - c \sum_{j=0}^{k} \cos^2\theta_j \|\nabla f_j\|^2.$$

Uma vez que f é limitada inferiormente, sabemos que $f_0 - f_{k+1}$ é uma constante positiva, para todo k. Tomando o limite da expressão acima, concluímos que

$$\sum_{j=0}^{\infty} \cos^2\theta_j \|\nabla f_j\|^2 < \infty.$$

□

A consequência do Teorema 5 é a convergência global do método de descenso, isto se este sempre escolhe como direção de descenso p_k com ângulo θ_k que se afaste de 90^0.

[3]Para $\|\bullet\| = \|\bullet\|_2$, $|x^T y| \leq \|x\| \|y\|$

Assim $\cos\theta_k \geq \delta > 0$ para todo k. Consequentemente $\lim_{k\to\infty} \|\nabla f_k\| = 0$. Em outras palavras, a norma do gradiente $\|\nabla f_k\|$ converge para zero se a direção de descenso não é muito próxima da ortogonalidade e as condições de Wolfe são satisfeitas.

2.3.7 Taxa de Convergência do Método de Descenso Íngreme

Aqui vamos considerar a taxa de convergência de métodos de descenso em regiões próximas de um ótimo local. Uma função continuamente diferenciável f pode ser aproximada em torno de um ótimo local x^* (não singular) como uma função quadrática

$$f(x) = f(x^*) + \frac{1}{2}(x - x^*)^T \nabla^2 f(x^*)(x - x^*)$$

desde que $\|x - x^*\|$ seja pequeno. Podemos assumir sem perda de generalidade que $x^* = 0$ e $f(x^*) = 0$. Assim a aproximação quadrática de f fica

$$f(x) = \frac{1}{2}x^T Q x, \quad \nabla f(x) = Qx, \quad \nabla^2 f(x) = Q.$$

O método de descenso íngreme (*steepest descent*) pode ser colocado na forma:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) = (I - \alpha_k Q)x_k$$

o que nos leva a

$$\|x_{k+1}\|^2 = x_k^T (I - \alpha^k Q)^2 x_k.$$

Uma vez que $x^T(I - \alpha_k Q)^2 x \leq Max\{\lambda \in \sigma((I - \alpha_k Q)^2)\}\|x\|^2$, onde $\sigma(A)$ denota o conjunto de autovalores da matriz A, concluímos

$$\|x_{k+1}\|^2 \leq Max\{\lambda \in \sigma((I - \alpha_k Q)^2)\}\|x_k\|^2.$$

Os autovalores de $(I - \alpha_k Q)^2$ são iguais a $(1 - \alpha_k \lambda_i)^2$, onde λ_i são os autovalores de Q. Dessa forma podemos deduzir que

$$Max\{\lambda \in \sigma((I - \alpha_k Q)^2)\} = Max\{(1 - \alpha_k m)^2, (1 - \alpha_k M)^2\}$$

onde $m(M)$ é o menor (maior) autovalor de Q. Segue então que para $x_k \neq 0$

$$\frac{\|x_{k+1}\|}{\|x_k\|} \quad \leq \quad Max\{|1 - \alpha_k m|, |1 - \alpha_k M|\}. \tag{2.8}$$

Se $|1 - \alpha_k m| < |1 - \alpha_k M|$ então a desigualdade (2.8) é satisfeita como uma equação e x_k é proporcional ao autovetor correspondente a m. Se $|1 - \alpha_k m| > |1 - \alpha_k M|$ então a desigualdade (2.8) é satisfeita como uma equação e x_k é proporcional ao autovetor correspondente a M. O passo α_k que minimiza o limite dado em (2.8) é

$$\alpha^* = \frac{2}{M + m}$$

o que implica em

$$\frac{\|x_{k+1}\|}{\|x_k\|} \leq \frac{M - m}{M + m}, \tag{2.9}$$

que é a melhor taxa de convergência do *steepest descent* com passo constante. De acordo com (2.9), quanto menor for a diferença entre o maior e menor autovalor de Q, mais rápida é a convergência. Nas situações em que as curvas de nível em torno de x^* são alongadas, i.e., quando a diferença entre o maior e menor autovalor é grande, a taxa de convergência tende a diminuir.

2.4 Exercícios

EX 2.1 Calcule o gradiente $\nabla f(x)$ e a matriz Hessiana $\nabla^2 f(x)$ da função $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. Mostre que o ponto $x^* = (1, 1)$ é o único ótimo global. O ponto x^* satisfaz as condições necessárias para ser um ótimo local?

EX 2.2 Mostre que a função $f(x) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2$ possui apenas um ponto estacionário e que este não define um mínimo nem um máximo de f.

EX 2.3 Seja $c \in \mathbb{R}^n$ um vetor e $A \in \mathbb{R}^{n \times n}$ uma matriz simétrica. Calcule o gradiente e a matriz Hessiana de $f_1(x) = c^T x$ e de $f_2(x) = x^T Ax/2$.

EX 2.4 Define-se uma função $f(x)$ como convexa se, e somente se, $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1 - \alpha) f(y)$ para todo x, y e $\alpha \in [0, 1]$. Sejam $f(x)$ e $g(x)$ duas funções convexas. Responda às seguintes questões:

- $h(x) = f(x) + g(x)$ é convexa?
- $h(x) = f(x) - g(x)$ é convexa?
- $h(x) = f(x)^2$ é convexa?
- $h(x) = \sqrt{f(x)}$ é convexa, assumindo que $f(x) \geq 0$?

EX 2.5 Seja $f(x)$ uma função quadrática: $f(x) = \frac{1}{2}x^TQx - b^Tx$, onde Q é simétrica e positiva definida. O algoritmo de descenso pode ser descrito como segue:

Seja $x_0 \in \mathbb{R}^n$ o ponto inicial, $k = 0$;
Enquanto $(\|\nabla f(x_k)\| > \sigma)$ e $(k < itbound)$ faça
  $p_k = -\nabla f(x_k)$
  $x_{k+1} = x_k + \alpha_k p_k$, onde $\alpha_k = argmin_\alpha\{f(x_k + \alpha p_k) : \alpha \geq 0\}$
  $k = k + 1$
Fim

Implemente esse algoritmo em Matlab. Sua função script deve receber como parâmetros de entrada Q, b, x_0, e gerar a sequência $\{x_k\}$. Use como tolerância $\sigma = 10^{-6}$ e $itbound = 100$. Rode experimentos para exemplos onde $n = 2$, $b = 0$, com as seguintes matrizes:

i. uma matriz Q cujo *condition number* $\kappa(Q) = 10$, e

ii. uma matriz Q para a qual $\kappa(Q) = 100$.

Para construir a matriz Q tome $D = [0.10; 01]$, escolha uma matriz randômica e ortogonal P ($P^TP = I$), e finalmente faça $Q = P^TDP$ e $x_0 = P[1; 0.1]$; Para a parte (b) faça o mesmo, mas utilize 0.01 em vez de 0.1 nos dois lugares.

Apresente os resultados em tabelas bem como em figuras, juntamente com os dados utilizados durante os experimentos (Q, x_0, etc.).

Capítulo 3

Minimização de Funções e Solução de Equações Não-Lineares com o Método de Newton

Este capítulo trata do método de Newton e suas aplicações na solução sistemas de equações não-linares e na minimização de funções. São apresentados os fundamentos do algoritmo, exemplos e prova de convergência local em uma variável.

3.1 Problemas de Interesse

Desenvolveremos métodos de Newton para a solução de equações não-lineares, formuladas como segue:

$$\begin{aligned} P_1: \quad & \textit{Encontre } x \in \mathbb{R}^n, \textit{ tal que } c(x) = 0 \\ & \textit{onde } c : \mathbb{R}^n \to \mathbb{R}^m \end{aligned}$$

Também investigaremos a solução do problema de minimização irrestrita, ou seja:

$$\begin{aligned} P_2: \quad & \textit{Minimize } f(x) \\ & x \in \mathbb{R}^n \\ & \textit{onde } f : \mathbb{R}^n \to \mathbb{R} \end{aligned}$$

3.2 O Método de Newton em uma Variável

Nesta seção nos concentramos na solução do problema P_1 em uma variável, ou seja, desejamos encontrar $x \in \mathbb{R}$ tal que $c(x) = 0$, onde $c(x) : \mathbb{R} \to \mathbb{R}$. O método, como o próprio nome sugere, foi proposto por Isaac Newton. O método de Newton

é um processo iterativo de fundamental importância no desenvolvimento de muitos algoritmos de programação não-linear. Seja x^k a solução candidata na iteração k, sendo a solução inicial x^0 definida arbitrariamente. A idéia básica do método de Newton está na aproximação de $c(x)$ com uma série de Taylor de primeira ordem em torno do ponto x^k. Isso nos leva a seguinte aproximação:

$$c(x^{k+1}) = c(x^k) + c'(x^k)(x^{k+1} - x^k)$$

onde $c'(x) = dc/dx$. O método calcula a próxima solução candidata de forma a resolver a aproximação linear, mais precisamente:

$$c(x^k) + c'(x^k)(x^{k+1} - x^k) = 0$$

$$x^{k+1} = x^k - [c'(x^k)]^{-1} c(x^k)$$

Tipicamente, não se espera que $c(x^{k+1}) = 0$, mas que x^{k+1} defina uma solução candidata de "melhor qualidade" do que aquela definida por x^k. Formalmente, espera-se que:

$$|x^{k+1} - x^*| < |x^k - x^*| \quad e$$

$$|c'(x^{k+1})| < |c'(x^k)|$$

onde x^* é uma solução, i.e., $c(x^*) = 0$.

Uma seqüência de soluções $x^0, x^1, \ldots$ é dita convergente se $\lim_{k\to\infty} |x^k - x^*| = 0$.

Método de Newton

Dado um x inicial e uma tolerância $\epsilon > 0$
Repita
    Calcule $c(x)$ e $c'(x)$
    Se $|c(x)| < \epsilon$, retorne x
    $x \leftarrow x - c(x)/c'(x)$
Até que um número máximo de iterações seja atingido

3.2.1 Exemplo

Procuramos um zero para a função $c(x) = -10 - 2x + 4x^2$. A Tabela 3.1 mostra os resultados do método de Newton para $x^0 = 1000$. A Figura 3.1 ilustra o processo

iterativo de Newton.

Tabela 3.1: Iterações do método de Newton.

k	x^k	$c(x^k)$	$\lvert x^k - x^* \rvert$
0	1000	3.997990000000000e+006	9.981492189406418e+002
1	500.1262815703926	9.994949375065696e+005	4.982755005110344e+002
2	250.1907039194102	2.498711719029211e+005	2.483399228600520e+002
3	125.2254781755631	6.246523058084260e+004	1.233746971162049e+002
4	62.74799109896568	1.561374556562558e+004	6.089721003960747e+001
5	31.51949620842356	3.900875572514462e+003	2.966871514906535e+001
6	15.92572254113459	9.726631087465399e+002	1.407494148177637e+001
7	8.169595939998821	2.406299994113832e+002	6.318814880640609e+000
8	4.371580213652639	5.769969383029171e+001	2.520799154294426e+000
9	2.621653885664322	1.224896861354670e+001	7.708728263061098e-001
10	1.976061758605505	1.667156778081310e+000	1.252806992472928e-001
11	1.855327609771498	5.830693877868853e-002	4.546550413286443e-003
12	1.850787497645492	8.245047246546733e-005	6.438287280197130e-006
13	1.850781059371159	1.658051473896194e-010	1.294742091317858e-011
14	1.850781059358212	0	2.220446049250313e-016

É importante ressaltar que o método de Newton apresenta uma taxa de convergência quadrática. Isso significa, de uma forma informal, que o número de dígitos corretos na solução candidata duplica a cada iteração. (O método de Newton apresenta convergência Q-quadrática, ou seja, existe $M \in \mathbb{R}$ tal que $\|x^{k+1} - x^*\|/\|x^k - x^*\|^2 \leq M$ para k suficientemente grande se x^0 pertence à região de convergência.) Então, qual é o problema com o método de Newton? O método de Newton pode divergir. Isso ocorre quando a solução inicial não é suficientemente próxima da solução ótima. O método de Newton se comporta bem na região próxima da solução, todavia não apresenta convergência global como o método de descenso.

3.3 O Método de Newton para Minimização em uma Variável

Nesta seção, tratamos do problema de encontrar um valor $x \in \mathbb{R}$ tal que $f(x) : \mathbb{R} \to \mathbb{R}$ seja mínimo. O método inicia, como no caso anterior, com uma aproximação da função em torno da solução candidata corrente, x^k, mas desta vez utilizamos uma aproximação de segunda ordem. Poderíamos ter adotado uma aproximação de primeira

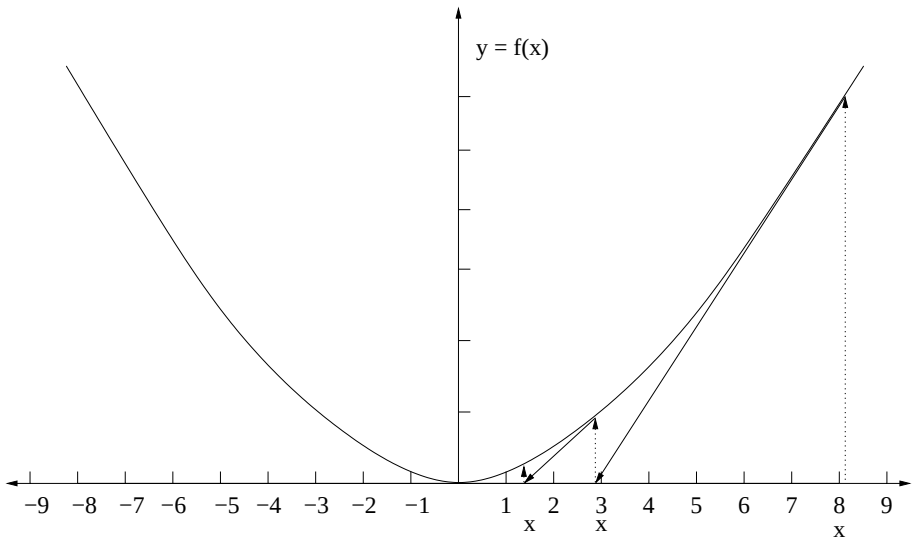


Figura 3.1: Ilustração do processo iterativo de Newton

ordem?

$$f(x^{k+1}) = f(x^k) + f'(x^k)[x^{k+1} - x^k] + \frac{1}{2}f''(x^k)[x^{k+1} - x^k]^2.$$

Para que x^{k+1} seja o mínimo da aproximação, necessitamos que:

$$\begin{aligned} df/dx^{k+1} = 0 \quad &\Rightarrow \quad f'(x^k) + f''(x^k)[x^{k+1} - x^k] = 0 \\ &\Rightarrow \quad x^{k+1} = x^k - [f''(x^k)]^{-1} f'(x^k). \end{aligned}$$

3.3.1 Exemplo

Encontre um mínimo de $f(x) = \cos(x)$ (conforme Figura 3.2). Se iniciamos o processo iterativo no ponto $x^0 = \pi/4$, o método converge para o ponto $x^* = 0$ ($\cos(x^*) = 1$). Se iniciamos no ponto $x^0 = 3\pi/4$, o método converge para o ponto $x^* = \pi$ ($\cos(\pi) = -1$). O que levou o método de Newton a convergir para um ponto de máximo? Será que o método apresenta uma falha fundamental?

Note que o método de Newton gera uma seqüência de soluções x^0, x^1, $\ldots$, x^* que converge para um ponto x^* estacionário, ou seja, $f'(x^*) = 0$. Todavia, x^* não é necessariamente um ponto de mínimo, podendo também definir um ponto de inflexão (e.g., o ponto estacionário da função $f(x) = x^3$) ou de máximo (e.g., o ponto estacionário da função $f(x) = -x^2$). Abaixo relembramos as condições necessárias e suficientes para mínimo local:

Condições Necessárias Para Mínimo Local: $f'(x^*) = 0$ *e* $f''(x^*) \geq 0$

Condições Suficientes Para Mínimo Local: $f'(x^*) = 0$ *e* $f''(x^*) > 0$.

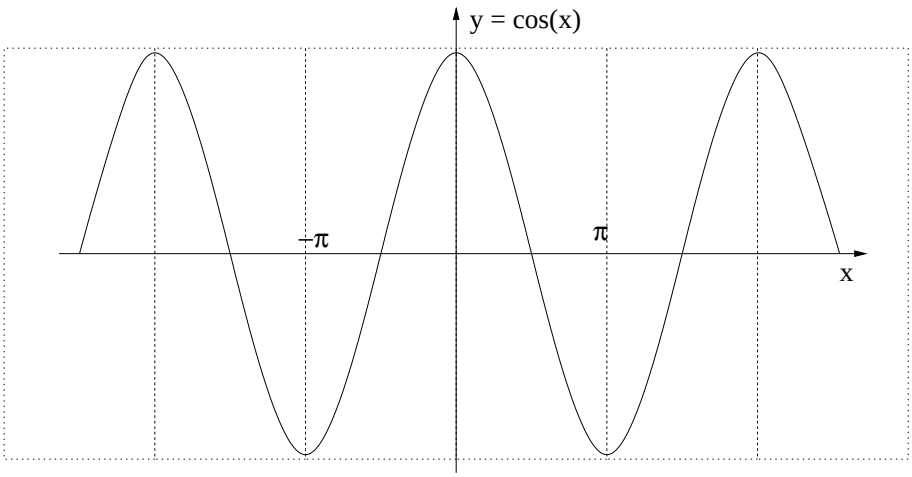


Figura 3.2: Função $\cos(x)$

3.4 O Método de Newton em Múltiplas Variáveis

Desejamos encontrar um vetor $x = (x_1, \ldots, x_n)$ tal que:

$$c(x) = \begin{bmatrix} c_1(x) \\ \vdots \\ c_m(x) \end{bmatrix} = 0 \tag{3.1}$$

Para simplificar a apresentação, suponha que $m = n$. Da mesma forma que no caso de uma variável, utilizamos uma aproximação linear de c^x em torno da solução candidata x^k.

$$c(x^{k+1}) = c(x^k) + G(x^k)[x^{k+1} - x^k] \tag{3.2}$$

sendo a matriz G chamada de Jacobiana e definida como segue:

$$G = \begin{bmatrix} \partial c_1/\partial x_1 & \ldots & \partial c_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial c_m/\partial x_1 & \ldots & \partial c_m/\partial x_n \end{bmatrix} \tag{3.3}$$

Para encontrar o próximo iterando, fazemos $c(x^k) + G[x^{k+1} - x^k] = 0$, resultando no processo iterativo abaixo:

$$x^{k+1} = x^k - G^{-1}c(x^k). \tag{3.4}$$

Lembramos que o método de Newton em múltiplas variáveis herda as mesmas pro-

priedades de sua versão univariada:

- o método apresenta convergência quadrática, desde que o ponto inicial x^0 esteja dentro de sua região de convergência;
- o método pode divergir se uma técnica de globalização não for empregada; e
- a matriz Jacobiana deve ser não-singular para que o método tenha sucesso.

Método de Newton em Várias Variáveis

Dado um x inicial e uma tolerância $\epsilon > 0$
Repita
    Calcule $c(x)$ e $G(x)$
    Se $\|c(x)\| < \epsilon$, retorne x
    Encontre uma solução para $G(x)\Delta x = -c(x)$
    $x \leftarrow x + \Delta x$
Até que um número máximo de iterações seja atingido

Enfatizamos que a iteração Newton, conforme equação (3.4), não é realizada através do cálculo direto da inversa de $G(x)$. Fazemos uso de algoritmos de solução de sistemas de equações lineares para resolver o sistema $G(x)\Delta x = -c(x)$. Um forma robusta para resolvê-lo se dá através da decomposição LU, a qual decompõe G em duas matrizes triangulares L (triangular inferior) e U triangular superior, de maneira que $G = LU$. Uma vez calculados os fatores L e U, a solução de (3.4) pode ser obtida com duas retrosubstituições. Quando a matrix G é positiva definida, o que ocorre no caso do método de Newton estar sendo utilizado como ferramenta para encontrar um ponto estacionário de uma função f, é recomendável realizar uma fatoração Cholesky. Esta encontra uma matriz triangular inferior L de forma que $G = LL^T$. Algoritmos eficientes e robustos podem ser encontrados na literatura para executar ambos os tipos de fatoração.

3.5 Minimização Irrestrita

Aqui a atenção se volta para o problema de minimização irrestrita em múltiplas variáveis. Desejamos encontrar um vetor $x = (x_1, \ldots, x_n)$ tal que o valor da função

$f(x)$ seja mínimo. Vamos aproximar $f(x)$ em torno da solução candidata corrente x^k como segue:

$$f(x^{k+1}) = f(x^k) + \nabla f(x^k)^T[x^{k+1} - x^k] + \frac{1}{2}[x^{k+1} - x^k]^T \nabla^2 f(x^k)[x^{k+1} - x^k]$$

Fazendo $x^{k+1} = x^+$, $x^k = x$, $\nabla f(x^k) = g$, $\nabla^2 f(x^k) = H$ e $p = [x^{k+1} - x^k]$, a aproximação pode ser reescrita como segue:

$$f(x^+) = f(x) + g^T p + \frac{1}{2} p^T H p.$$

Da mesma forma que no caso univariado, podemos encontrar um ponto estacionário para a aproximação, forçando x^+ a satisfazer a condição necessária de primeira ordem, formalmente:

$$\nabla f(x^+) = 0 \Rightarrow g + Hp = 0 \Rightarrow p = -H^{-1} g.$$

Portanto, o processo iterativo pode ser expresso como segue:

$$x^{k+1} = x^k - [\nabla^2 f(x^k)]^{-1} \nabla f(x^k).$$

Uma vez que o método de Newton se baseia na busca de um ponto para o qual o gradiente de $f(x)$ é nulo, não há garantia de que o processo iterativo convirja para um ponto de mínimo, podendo divergir ou ser atraído para um ponto de inflexão ou de máximo. Lembramos que as condições necessárias e suficientes para que x^k seja um mínimo local são:

Condições Necessárias: $\nabla f(x^k) = 0$ e $\nabla^2 f(x^k) \geq 0$ (positiva semi-definida)

Condições Suficientes: $\nabla^2 f(x^k) > 0$ (positiva definida)

3.6 Convergência

Algoritmos não-iterativos, como o método de Gauss para solução de sistemas de equações lineares, têm custo computacional bem-definido. O número de operações de ponto-flutuante é finito e pode ser expresso como uma função polinomial das dimensões do problema. O comportamento de algoritmos iterativos, por outro lado, é mais difícil de ser medido. O número de iterações depende do problema e do ponto inicial. A eficiência de um algoritmo iterativo é medida em termos de um limite superior de iterações necessárias para se atingir uma solução com certa precisão. Tais limites são obtidos por meio de uma análise de convergência.

3.6.1 Convergência Linear

Uma sequência x^k com limite x^* é linearmente convergente se existe uma constante $c \in (0,1)$ tal que:

$$|x^k - x^*| \leq c|x^{k-1} - x^*| \tag{3.5}$$

para k suficientemente grande. Convergência linear é algumas vezes definida como segue. A sequência x^k com limite x^* converge R-linearmente se existe um valor M e $c \in (0,1)$ tal que:

$$|x^k - x^*| \leq Mc^k \tag{3.6}$$

para k suficientemente grande.

3.6.2 Convergência Quadrática

Uma sequência x^k com limite x^* converge quadraticamente se existe uma constante $c > 0$ tal que:

$$|x^k - x^*| \leq c|x^{k-1} - x^*|^2 \tag{3.7}$$

para k suficientemente grande. A sequência $x^k = 1 + (\frac{1}{2})^{2^k}$ converge quadraticamente para $x^* = 1$ pois:

$$|x^{k+1} - x^*| = \left(\frac{1}{2}\right)^{2^{k+1}} = \left[\left(\frac{1}{2}\right)^{2^k}\right]^2 = |x^k - x^*|^2. \tag{3.8}$$

Definindo $r^k = -\log_{10}[|x^k - x^*|/|x^*|]$, quando $|x^*| \neq 0$, e dividindo por $|x^*|$ podemos reescrever (3.7) como

$$r^k \geq 2r^{k-1} - \log_{10}(|x^* c|). \tag{3.9}$$

Uma vez que $|x^k - x^*| \to 0$, $r^k \to +\infty$ e eventualmente o primeiro termo da parte à direita da desigualdade dominará o segundo. r^k é aproximadamente o número de dígitos corretos na iteração k, portanto o número de dígitos corretos praticamente dobra a cada iteração.

3.6.3 Convergência do Método de Newton

O método de Newton converge quadraticamente se $G(x^*)$ tem posto completo e se x^0 é suficientemente próximo de x^*. Para o caso unidimensional, a ser estudado nesta seção, a convergência quadrática é obtida se $f'(x^*) \neq 0$. Seja $I = [x^* - \delta, x^* + \delta]$ um intervalo em torno da solução x^* dentro do qual as seguintes condições são satisfeitas:

i. existe uma constante $m > 0$ tal que $|f'(x)| \geq m$ para todo $x \in I$; e

ii. existe uma constante $L > 0$ tal que $|f'(x) - f'(y)| \leq L|x - y|$ para todo $x, y \in I$.

Vamos mostrar que se $|x^k - x^*| \leq \delta$, então

$$|x^{k+1} - x^*| \leq \frac{L}{2m}|x^k - x^*|^2 \tag{3.10}$$

Em outras palavras, a desigualdade (3.7) é satisfeita com $c = L/(2m)$, garantindo dessa forma convergência quadrática para x^*. Denotando x^{k+1} por x^+ e x^k por x, temos que

$$\begin{aligned} x+ = x - \frac{f(x)}{f'(x)} \Rightarrow |x^+ - x^*| &= |x - \frac{f(x)}{f'(x)} - x^*| \\ &= \frac{|-f(x) - f'(x)(x^* - x)|}{|f'(x)|} \\ &= \frac{|f(x^*) - f(x) - f'(x)(x^* - x)|}{|f'(x)|} \end{aligned}$$

Lembre-se que $f(x^*) = 0$. Nós assumimos que $|f'(x)| \geq m$ e, portanto,

$$|x^+ - x^*| \leq \frac{|f(x^*) - f(x) - f'(x)(x^* - x)|}{m} \tag{3.11}$$

Para limitar o numerador, podemos proceder como segue

$$\begin{aligned} |f(x^*) - f(x) - f'(x)(x^* - x)| &= |\int_x^{x^*} (f'(u) - f'(x))du| \\ &\leq \int_x^{x^*} |f'(u) - f'(x)|du \\ &= L\int_x^{x^*} |u - x|du \\ &= L\frac{|x^* - x|^2}{2} \end{aligned} \tag{3.12}$$

Juntando (3.11) e (3.12) obtemos $|x^+ - x^*| \leq \frac{L}{2m}|x^* - x|^2$ o que prova (3.10). A desigualdade (3.10) por si só não garante que $|x^{k+1} - x^*| \to 0$. Entretanto, se assumirmos

que $|x^0 - x^*| \leq \delta$ e $\delta \leq m/L$, então convergência quadrática segue imediatamente. Uma vez que $|x^0 - x^*| \leq \delta$, podemos aplicar (3.10) e obter:

$$|x^1 - x^*| \leq \frac{L}{2m}|x^0 - x^*|^2 \leq \frac{L}{2m}\delta^2 \leq \delta/2.$$

Uma vez que $|x^1 - x^*| \leq \delta$, podemos aplicar (3.10) e obter a desigualdade:

$$|x^2 - x^*| \leq \frac{L}{2m}|x^1 - x^*|^2 \leq \frac{L}{2m}\delta^2/4 \leq \delta/8.$$

Repetindo este processo, concluímos que:

$$|x^{k+1} - x^*| \leq \frac{L}{2m}|x^k - x^*|^2 \leq 2\left(\frac{1}{4}\right)^{2^k}\delta.$$

Segundo o desenvolvimento acima, se $f(x^*) = 0$ e f' é uma função contínua, então é razoável assumir que as condições acima são satisfeitas para algum δ, m e L. Na prática, entretanto, é difícil encontrarmos valores para δ, m e L, portanto o resultado de convergência não nos oferece ajuda no sentido de escolhermos o ponto inicial x^0, mesmo porque isto demandaria conhecimento antecipado da existência de uma raiz. O resultado nos diz que se x^0 é suficientemente próximo de x^*, então o método de Newton converge com uma taxa quadrática.

3.7 Métodos de Região de Confiança

Como visto acima, o método de Newton puro para minimização de funções encontra $p = x - x^k$ que minimiza uma aproximação de f em torno de x^k dada pela expansão de Taylor de segunda ordem:

$$f^k(p) = f(x^k) + \nabla f(x^k)^T p + \frac{1}{2}p^T \nabla^2 f(x^k) p.$$

Sabemos que $f^k(p)$ é uma boa aproximação de $f(x^k + p)$ quando p está dentro de uma vizinhança próxima de zero, mas a dificuldade é que na minimização de $f^k(p)$ podemos obter um passo fora desta região. Se torna portanto relevante considerar um passo de Newton restrito p^k obtido por meio da minimização de $f^k(p)$ dentro de uma região pequena na vizinhança do zero, chamada região de confiança (*trust region*):

$$\begin{array}{rcll} p^k & = & ArgMin & f^k(p) \\ & & \|p\| \leq \gamma^k & \end{array} \tag{3.13}$$

onde γ^k é um escalar positivo. Explorando o fato de que o problema (3.13) tem apenas uma restrição, uma solução aproximada pode ser obtida rapidamente.

É importante enfatizar que mesmo quando $\nabla^2 f(x^k)$ não é positiva definida ou até mesmo quando a direção Newton não é uma direção de descenso, o passo de Newton restrito p^k reduz o custo da função, desde que $\nabla f(x^k) \neq 0$ e γ^k seja suficientemente pequeno. Note que para todo p tal que $\|p\| \leq \gamma^k$,

$$f(x^k + p) = f^k(p) + o((\gamma^k)^2)$$

e portanto

$$\begin{aligned} f(x^k + d^k) &= f^k(d^k) + o((\gamma^k)^2) \\ &= f(x^k) + Min_{\|p\| \leq \gamma^k}\{\nabla f(x^k)^T p + \frac{1}{2} p^T \nabla^2 f(x^k) p\} + o((\gamma^k)^2). \end{aligned}$$

Denotando

$$\tilde{p^k} = -\frac{\nabla f(x^k)}{\|\nabla f(x^k)\|}\gamma^k,$$

temos

$$\begin{aligned} f(x^k + d^k) &\leq f(x^k) + \nabla f(x^k)^T \tilde{p^k} + \frac{1}{2}\tilde{p^k}^T \nabla^2 f(x^k)\tilde{p^k}\} + o((\gamma^k)^2) \\ &= f(x^k) - \gamma^k \|\nabla f(x^k)\| + \frac{(\gamma^k)^2}{2\|\nabla f(x^k)\|^2} \nabla f(x^k)^T \nabla^2 f(x^k) \nabla f(x^k) + o((\gamma^k)^2) \end{aligned}$$

Para γ^k suficientemente pequeno o termo negativo $-\gamma^k \|\nabla f(x^k)\|$ domina os últimos dois termos, o que nos leva a concluir que: $f(x^{k+1}) < f(x^k)$. Podemos também observar que uma redução nos custos pode ser obtida mesmo quando $\nabla f(x^k) = 0$ desde que γ^k seja suficientemente pequeno e f tenha uma direção de curvatura negativa no ponto x^k, o que significa dizer que $\nabla^2 f(x^k)$ não é positiva semidefinida. Portanto, o procedimento só falha em reduzir o custo quando $\nabla f(x^k) = 0$ e $\nabla^2 f(x^k)$ é positiva semidefinita, que corresponde a dizer que x^k satisfaz as condições necessárias de segunda ordem.

Os desenvolvimentos acima nos levam a considerar passos da forma

$$x^{k+1} = x^k + p^k,$$

onde p^k é o passo de Newton restrito correspondendo a um escalar γ^k. A chave para a convergência rápida do método de Newton com região de confiança está em ajustar dinamicamente o parâmetro γ^k, garantindo redução significativa e se comportando como o método de Newton puro próximo de um mínimo local não singular. Uma

forma razoável de ajustar o valor inicial de γ^k é incrementar o valor quando o método parece estar gerando reduções no valor de f e reduzi-lo caso contrário. O progresso em direção a um ótimo local pode ser medido com base na razão entre a redução antecipada e a redução efetivamente observada:

$$r^k = \frac{f(x^k) - f(x^{k+1})}{f(x^k) - f^k(p^k)}.$$

Aumenta-se o valor de γ se esta razão está próxima ou acima da unidade, e reduzimos o seu valor caso contrário. Usando como parâmetros $0 < \sigma_1 \leq \sigma_2 \leq 1$ e $0 < \beta_1 < 1 < \beta_2$, um algoritmo pode ser proposto:

Passo 1: Encontre:

$$\begin{array}{rcll} p^k & = & ArgMin & f^k(p) \\ & & \|p\| \leq \gamma^k & \end{array}$$

Se $f^k(p^k) = f(x^k)$ pare (x^k satisfaz as condições necessárias de primeira e segunda ordem).

Passo 2: Se $f(x^k + p^k) < f(x^k)$ defina $x^{k+1} = x^k + p^k$ e calcule

$$r^k = \frac{f(x^k) - f(x^{k+1})}{f(x^k) - f^k(p^k)}$$

e vá para o Passo 3, caso contrário faça $\gamma^k = \beta_1 \|p^k\|$ e vá para o Passo 1.

Passo 3: Faça

$$\gamma^{k+1} = \begin{cases} \beta_1 \|p^k\| & se\ r^k < \sigma_1 \\ \beta_2 \gamma^k & se\ \sigma_2 \leq r^k\ e\ \|p^k\| = \gamma^k \\ \gamma^k & caso\ contrario \end{cases}$$

Volte ao Passo 1.

Se f é duas vezes diferenciável então é possível mostrar que o algoritmo acima converge, i.e., a seqüência $\{x^k\}$ é limitida e existe um ponto limite que satisfaz as condições necessárias de segunda ordem. Além disso, se $\{x^k\}$ converge para um mínimo local x^* não singular, então o método tem a mesma taxa de convergência do método de Newton puro.

3.8 Exercícios

EX 3.1 Seja $c(x) = \arctan(x)$. Encontre a região de convergência e a de divergência do método de Newton.

EX 3.2 Considere o sistema de equações não-lineares a seguir:

$$\begin{aligned} \log(x_1^2 + 2x_2^2 + 1) - \frac{1}{2} &= 0 \\ x_2 - x_1^2 + 0.2 &= 0 \end{aligned}$$

Implemente em Matlab o algoritmo de Newton para resolver o sistema acima. Para cada um dos seguintes iterandos iniciais, indique se o algoritmo converge e, em caso afirmativo, o ponto para o qual ele converge: $x^0 = (1, 1)$; $x^0 = (-1, 1)$; $x^0 = (1, -1)$; e $x^0 = (-1, -1)$.

EX 3.3 Considere uma transformação linear inversível $x = Sy$. Escreva o método de Newton no espaço da variável y e mostre que este gera uma sequência $y^k = S^{-1}x^k$, onde $\{x_k\}$ é a sequência gerada pelo método de Newton no espaço da variável x.

EX 3.4 Seja $A \in \mathbb{R}^{m\times n}$ uma matriz, $rank(A) = m$, $c \in \mathbb{R}^n$. Considere o problema abaixo:

$$\begin{aligned} P: \quad & Minimize && \|x - c\|^2 \\ & Sujeito\ a: && \\ & && Ax = 0 \\ & && x \in \mathbb{R}^n \end{aligned}$$

Transforme P em um problema de otimização irrestrita, permitindo o emprego do algoritmo de Newton. Podemos garantir que um ponto estacionário define um ótimo local? Podemos afirmar que um ótimo local é um ótimo global?

Capítulo 4

Otimização Não-Diferenciável

4.1 Otimização Black-Box

Neste capítulo focamos nossa atenção no problema geral de otimizaçção irrestrita, sendo este expresso em programação matemática como:

$$\begin{aligned} P: \quad & Minimize \quad f(x) \\ & x \in \mathbb{R}^n \end{aligned}$$

onde nada se assume sobre $f : \mathbb{R}^n \to \mathbb{R}$

Uma gama expressiva de problemas práticos e teóricos pode ser reduzida a P. Em particular, f pode representar:

- O custo de produção para um dado conjunto de parâmetros que representam indicadores econômicos, custos de matéria-prima e planejamento da produção, podendo este custo ser estimado por meio de simulação computacional.
- O valor da função objetivo de um problema de otimização, onde $f(x) = \infty$ se x é infactível.
- O prêmio recebido em um certo jogo quando o participante segue a estratégia induzida por x.

O termo otimização não-diferenciável tem suas raízes no fato que para uma função qualquer o gradiente não está definido, portanto não há informação local quanto à direção de descenso. Duas técnicas para otimização não-diferenciável:

- **Algoritmo Genético (AG).** O AG tem suas raízes na Teoria da Evolução de Charles Darwin, podendo este ser visto como um mecanismo de evolução simulada.

- **Simulated Annealing (SA).** Se inspira no processo físico de obtenção de estruturas cristalinas através do super-aquecimento e resfriamento lento da substância, o que induz o sistema a atingir o nível mais baixo de energia possível.

4.2 Algoritmo Genético (GA)

4.2.1 Genética e Evolução

A percepção de que certas características são hereditárias—i.e., transmitidas geneticamente—data de vários décadas passadas. A Teoria da Evolução é considerada um dos maiores avanços científicos até esta data. Ela tem implicações em várias áreas naturais, científicas e sociais como, por exemplo, Biologia, Matemática, Física e Sociologia.

4.2.2 Adaptação Biológica

Estudaremos maneiras pelas quais o processo de evolução natural pode ser empregado para se "evoluir" soluções computacionais (Algoritmo Genético) e até mesmo programas ou algoritmos (Programação Genética) para problemas interessantes.

A propriedade de adaptação é descrita pela seguinte expressão:

Adaptação = Variedade + Hereditariedade + Seleção

Variedade Se refere a como os indivíduos diferem uns dos outros, portanto só pode ocorrer se existirem múltiplos indivíduos, implicando em paralelismo e multiplicidade espacial (*população*).

Hereditariedade É uma forma de persistência temporal que se propaga de pai para filho, um processo iterativo que se repete (*processo Iterativo*).

Seleção Natural a capacidade de reprodução é exponencial, extremamente maior do que a quantidade de recursos disponíveis (limitada). Surge, portanto, a frase "a sobrevivência dos reprodutores". Levando em conta esta capacidade exponencial, somos um grupo seleto de indivíduos. (*Seleção*)

A frase mais conhecida é "a sobrevivência do mais dotado". Entretanto, a teoria diz que o que importa é a reprodução, o que implica a frase "a sobrevivência do reprodutor". Segundo a teoria da evolução, as características de uma pessoa (bonita, alta, inteligente, etc.) só importam se aumentam a capacidade de reprodução. A evolução das espécies pode levar a processos cíclicos, como por exemplo a dinâmica entre leões

e gazelas. Se os leões se tornam mais perspicazes ao caçarem gazelas, então as gazelas sobreviventes tendem a ser mais rápidas, que por sua vez vão gerar descendentes mais ágeis, dificultando a caça dos leões.

4.2.3 Hereditariedade com Evolução Simulada

Da mesma forma que em sistemas biológicos, as equações de adaptação biológica podem ser utilizadas para "evoluir" soluções de um problema (Algoritmo Genético) e até mesmo algoritmos (Programação Genética). O algoritmo genético representa soluções como estruturas semelhantes ao DNA, as quais expressam alguma forma de aptidão. Isto está inspirado na Natureza, onde estruturas de DNA aparecem em organismos com capacidade de se acasalarem, e onde haja mutação e *cross-over* do material genético. Estes princípios fundamentais foram apresentados através da teoria de Darwin, mais tarde, complementada pelos trabalhos de Gregor Mendel (pai da Genética) e posteriormente enriquecidos pela descoberta da estrutura de dupla hélice do DNA. Estes últimos trabalhos mostram que as características transmitidas são discretas, não são contínuas como anteriormente assumido. Mendel mostrou em seus experimentos que, se controlarmos o ambiente, as características são bem distintas (flores altas ou baixas). Conclusão:

A linguagem da natureza é um alfabeto discreto.

O AG expressa uma solução como uma cadeia de símbolos (usualmente 0s e 1s) de tamanho fixo, da mesma forma que o DNA codifica as características de um indivíduo. É necessário a existência de uma função de aptidão (*fitness function*) que mapeie a cadeia em uma forma útil. Por exemplo, a cadeia pode representar:

- um vetor com as variáveis de uma função f que se deseja minimizar; e
- uma estratégia para competir em um jogo.

4.2.4 Popularidade do Algoritmo Genético

A popularidade do algoritmo genético se dá em função de três propriedades:

Robustez Evolução natural é tida como um método bem-sucedido e robusto no meio biológico.

Generalidade AGs são capazes de tratar problemas complexos, que possuem um número grande de componentes, cujas contribuições para o conjunto não são entendidas.

Paralelização AGs são naturalmente paralelos, podem ser implementados em redes de computadores (assíncronos) e computadores paralelos (síncronos).

Antes de procedermos à descrição dos passos do Algoritmo Genético, assumiremos o seguinte:

- O problema de otimização a ser resolvido é de maximização, em vez de minimização. Para transformar minimização em maximização, basta substituir "min" por "max" e $f(x)$ por $-f(x)$.
- A função objetivo ($f(x)$ se maximização, $-f(x)$ se minimização) será utilizada como função de aptid]ão (*fitness function*, f_i). Portanto, quanto mais alto o valor da *fitness function* (f_i) melhor a qualidade da respectiva solução. Assumiremos também que a *fitness function* assume um valor não negativo. Isso pode ser contornado através da adição de uma constante, i.e., seja $m = Min\{f_i(x) : x$ pertence à população$\}$, então basta substituir $f_i(x)$ por $f_i(x) - m$ para que a *fitness function* se torne não negativa.

4.2.5 Algoritmo Genético em Detalhes

O algoritmo genético pode ser descrito como uma função $AG(f_i, f_t, p, r, m)$ cujos parâmetros de entrada definem um problema a ser resolvido. Abaixo segue o pseudo-código.

Algoritmo $AG(f_i, f_t, p, r, m)$

Parâmetros de entrada:
f_i - *fitness function*
f_t - *fitness threshold* (condição de parada)
p - tamanho da população
r - fração da população a ser substituída por *cross-over*
m - taxa de mutação

Inicialize a população P: gere p soluções candidatas aleatoriamente
Avalie: para cada $x \in P$, calcule $f_i(x)$

Enquanto $Max\{f_i(x) : x \in P\} < f_t$, execute:
Seja P' a nova população, iniciando com $P' = \emptyset$
Selecione $(1 - r)p$ elementos de P e adicione a P', sendo a probabilidade

de um eleemnto $x \in P$ dada por:

$Pr(x) = f_i(x) / \sum_{x_j \in P} f_i(x_j)$

Probabilisticamente selecione $rp/2$ pares de elementos de P,

de acordo com a distribuição dada por $Pr(x)$.

Para cada par (x_i, x_j) execute cross-over e adicione os "filhos" a P'.

Execute mutação de mp elementos de P'

$P \leftarrow P'$

Calcule $f_i(x)$ para todo $x \in P$

4.2.6 Operador Genético *Cross-Over*

Os operadores genéticos determinam como os membros de uma população são combinados, ou sofrem mutação, de forma a gerar novos candidatos. Na Figura 4.1 é ilustrado o operador *cross-over* de um ponto, enquanto que o operador *cross-over* de dois pontos é exemplificado na Figura 4.2. O operador de *cross-over* uniforme consistem em utilizar uma máscara cujos bits são selecionados aleatoriamente com probabilidade uniforme. Já o operador de mutação simplesmente troca um ou mais bits de um cromossomo de 0 para 1 (ou vice-versa). Os bits são escolhidos aleatoriamente com probabilidade uniforme.

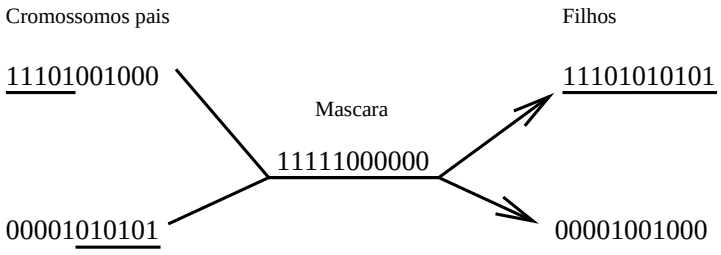


Figura 4.1: Operador *cross-over* de um ponto

4.2.7 Exemplo de Aplicação

O problema consiste em encontrar o valor máximo da função $f(x) = x^2$ onde x pode assumir valores do conjunto $A = \{0, \ldots, 31\}$. Para aplicação do algoritmo genético, os seguintes passos são sugeridos:

- Utilize 5 bits para representar uma solução candidata.

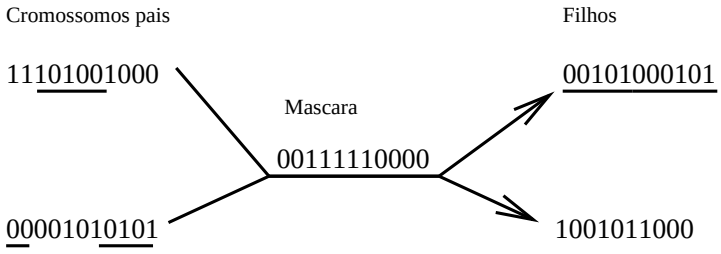


Figura 4.2: Operador *cross-over* de dois pontos

- Defina a função de aptidão $f_i(x) = f(x)$.
- Gere um conjunto de 4 soluções candidatas randomicamente, para compor a população inicial. (Tamanho da população $p = 4$).
- Adote o operador *cross-over* de um ponto como método de reprodução. (Taxa de substituição por *cross-over* $r = 50\%$).
- Adote como mecanismo de mutação a troca de um bit de uma solução candidata. (Taxa de mutação $m = 5\%$).

Na Tabela 4.1 é indicado os valores iniciais armazenados na memória do algoritmo genético. Com esta memória inicial e os parâmetros acima sugeridos, o algoritmo genético encontra a solução ótima para o problema: $x^* = 11111$ sendo $f(x^*) = 961$.

Tabela 4.1: Memória inicial

Solução Candidata	Fitness	Probabilidade de Reprodução
01101	169	14,4%
11000	576	49,2%
01000	64	5,5%
10011	361	30,9%

4.2.8 Questões Práticas

O fenômeno de *crowding* ocorre quando um indivíduo com alta aptidão se reproduz rapidamente, gerando cópias ou filhos muito semelhantes, que venham a constituir a maior parte da população. O aspecto negativo de *crowding* é a perda de diversidade, o que acarreta convergência prematura (ótimo local). O fenômeno de *crowding* é ilustrado na Figura 4.3

Algumas estratégias para evitar *crowding* são:

- seleção de pares para reprodução através de torneios, em vez de seleção probabilística;
- utilizar o valor de posição (*ranking*) no lugar da aptidão para calcular as probabilidades de selecionar os indivíduos; e
- o valor de aptidão de um indivíduo pode ser reduzido pela presença de outros indivíduos semelhantes.

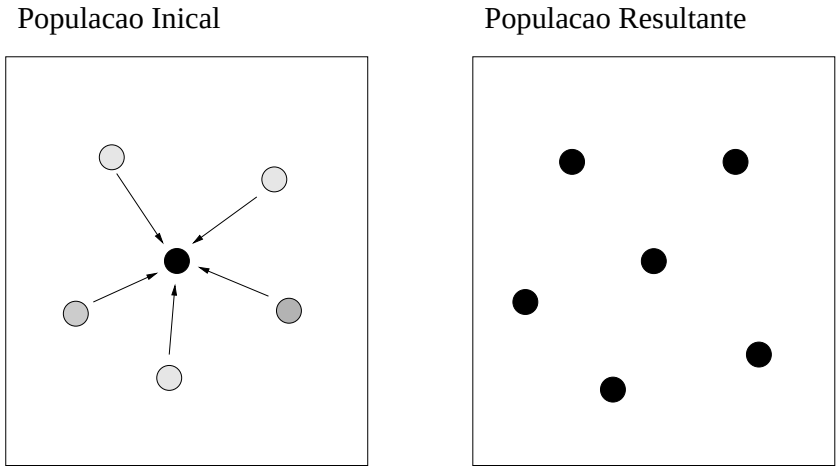


Figura 4.3: Fenômeno de *crowding*

4.2.9 Schema Theorem

É possível descrever matematicamente a evolução da população de um AG em função do tempo? O Schema Theorem de Holland (1975) oferece uma caracterização.

Definição 1 *Um Schema é uma cadeia de 0s, 1s e *s (curinga) que representa uma família de indivíduos.*

O *schema* "0*1" representa os indivíduos "001" e "011". *Schemas* podem ser utilizados para caracterizar a população de um algoritmo genético. Uma população de cadeias (vetores de bits 0s e 1s) pode ser vista em termos do conjunto de *schemas* que ela representa, e o número de indivíduos associados com cada *schema*. Seja $m(s,t)$ o número de indivíduos no instante t (t-ésima população) que pertençam ao *schema* s. O Schema Theorem descreve o valor esperado de $m(s,t+1)$ tomando como base:

$m(s,t)$; as propriedades de s; as propriedades da população; e os parâmetros do AG (e.g., métodos de recombinação e taxa de mutação).

Teorema 6 *Considerando apenas seleção, tem-se*

$$E[m(s,t+1)] = \frac{u(s,t)}{f_m(t)} m(s,t)$$

onde: $m(s,t)$ *é o número de indivíduos em* P_t *(t-ésima população) que pertençam ao* schema s*;* $f_m(t)$ *é o valor médio da função de aptidão dos elementos de* P_t*; e* $u(s,t)$ *é o valor médio da função de aptidão dos elementos de* P_t *que pertençam ao* schema s*.*

Se vemos o AG como um processo virtual paralelo que i) executa uma busca através do espaço de possíveis *schemas* e, ao mesmo tempo, ii) executa uma busca através do espaço de indivíduos, então a equação do Teorema 6 indica que *schemas* de maior aptidão crescerão em influência com o passar das gerações.

Teorema 7 *Considerando-se o operador cross-over de um ponto e mutação, tem-se*

$$E[m(s,t+1)] \geq \frac{u(s,t)}{f_m(t)} m(s,t) \left[1 - \frac{p_c d(s)}{l-1}\right] [1-p_m]^{o(s)}$$

onde:

- p_c *é a probabilidade do operador cross-over de um ponto ser aplicado a um indivíduo arbitrário;*
- p_m *é a probabilidade de que um bit arbitrário de um indivíduo arbitrário sofra mutação;*
- l *é o número de bits das cadeias;*
- $o(s)$ *é o número de bits definidos presentes no* schema s *(bits 0 ou 1); e*
- $d(s)$ *é a distância entre o bit definido mais à esquerda e o mais à direita no* schema s*.*

O Teorema 7 nos diz que os *schemas* de maior aptidão deverão aumentar sua influência, especialmente aqueles que possuem um pequeno número de bits definidos (contendo vários "*") e aqueles cujos bits definidos estão próximos uns dos outros.

4.3 Simulated Annealing

Simulated annealing é um método de otimização bastante geral (como o Algoritmo Genético), que tem demonstrado excelente desempenho em problemas que apresentam um número muito grande de ótimos locais. Uma das aplicações mais bem-sucedidas de SA é o problema de localização de componentes eletrônicos em microprocessadores, no qual procura-se minimizar o comprimento total das conexões. A Figura 4.4 ilustra o problema de localização e empacotamento de circuitos eletrônicos.

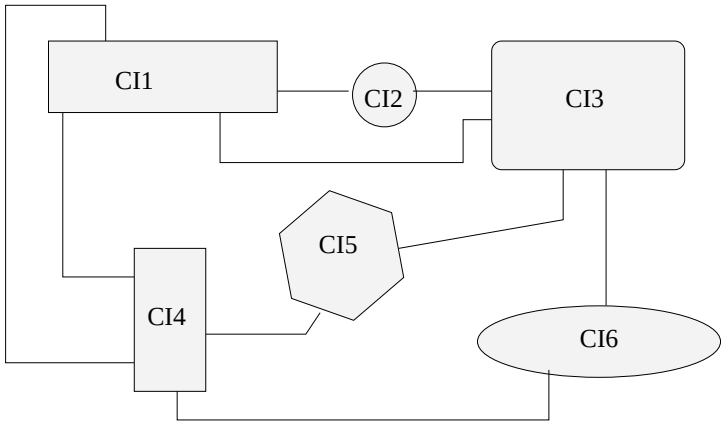


Figura 4.4: Empacotamento de componentes eletrônicos em circuitos integrados

4.3.1 O Processo de Annealing

O processo de *annealing* se refere ao método por meio do qual líquidos (ou metais) se resfriam e cristalizam. Aquecemos a substância até uma temperatura muito alta, depois resfriamos lentamente. Na natureza, se o resfriamento é bastante lento, a substância assume uma configuração de menor energia para a temperatura de equilíbrio.

A Natureza pode ser vista como um processo de minimização do nível de energia, sendo este probabilístico. Mais precisamente, a probabilidade de um sistema físico se encontrar em uma configuração com nível de energia E é dada por:

$$Pr(E) \approx e^{-\frac{E}{kT}} \tag{4.1}$$

onde K é a constante de Boltzman. De acordo com a equação (4.1), para uma dada temperatura T, quanto maior o nível de energia, menor a probabilidade da configuração. Para um dado nível de energia E, a probabilidade da configuração diminui com a redução da temperatura. Mesmo em baixa temperatura, é possível que o sistema esteja

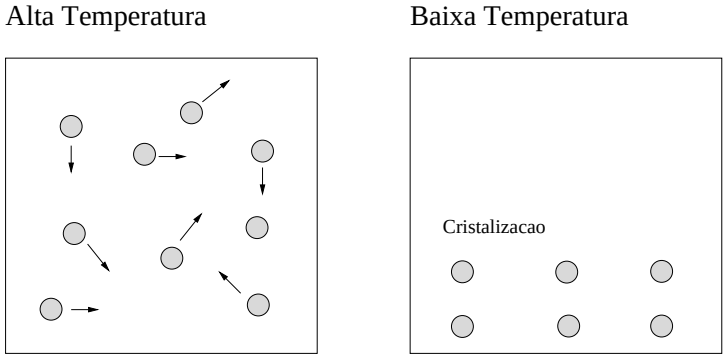


Figura 4.5: Ilustração do processo de *annealing*

em um nível elevado de energia, embora a probabilidade seja extremamente pequena. Em outros palavras, o sistema aumenta o nível de energia algumas vezes, outras vezes ele diminui, mas a probabilidade de aumentar diminui exponencialmente com a redução da temperatura.

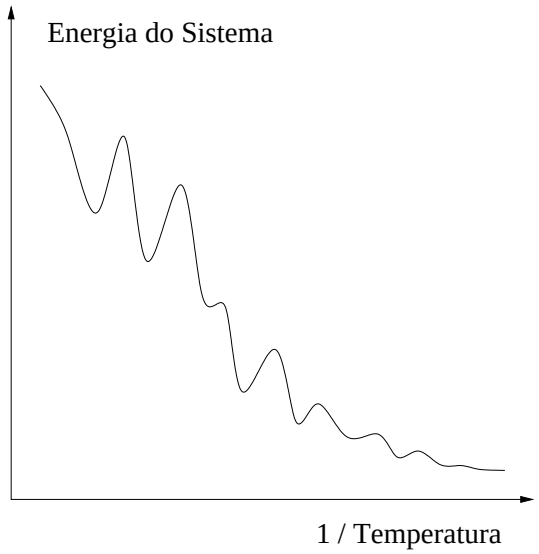


Figura 4.6: Comportamento típico do nível de energia conforme processo de *annealing*

4.3.2 O Algoritmo de Metropolis

Em 1950, Metropolis incorporou as idéias apresentadas em computações numéricas. Dado um conjunto de configurações para um sistema, a probabilidade do sistema as-

sumir a configuração E_2 a partir da configuração E_1 é dada por:

$$Pr = e^{-(E_2 - E_1)/kT} \; se \; E_2 > E_1$$
$$Pr = 1 \; se \; E_2 < E_1$$

A Figura 4.7 ilustra a queda de probabilidade de transição para estados de maior energia com a redução de temperatura. Para empregarmos o algoritmo de Metropolis em outros sistemas, não necessariamente termodinâmicos, precisamos de:

- uma descrição das possíveis configurações do sistema;
- um gerador randômico de perturbações do sistema; essas perturbações definem as "opções" de configuração;
- uma função objetivo (análoga à energia) cuja minimização desejamos executar; e
- um parâmetro de controle T (análogo à temperatura) e um procedimento de *annealing* que descreve a maneira pela qual T decresce.

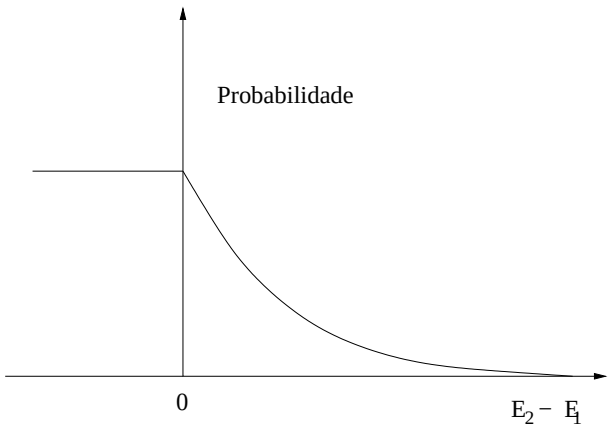


Figura 4.7: Probabilidade de transição com a queda de temperatura

4.3.3 Exemplo: O Problema do Caixeiro Viajante

Em uma instância do Problema do Caixeiro Viajante (PCV) são dados N cidades e uma matriz de distâncias M. Desejamos encontrar uma rota que, partindo da cidade 1, visite as demais cidades precisamente uma vez e retorne à cidade 1, tal que o caminho percorrido seja o menor possível. O PCV pertence à família dos problemas NP-Difíceis, o que significa que que não se conhece um algoritmo que encontre a rota ótima em tempo polinomial para qualquer instância.

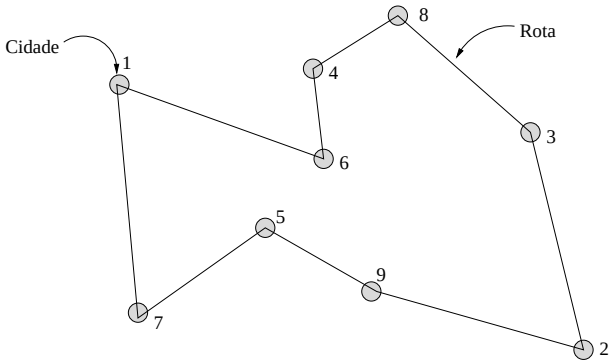


Figura 4.8: Exemplo de rota para uma instância particular

Abaixo seguem os passos para se aplicar o algoritmo SA:

- **Configuração:** utilize um vetor x com N posições, $x \in \{1, 2, \ldots, N\}^N$, para armazenar uma permutação das cidades.

- **Perturbação:** uma subseqüência de cidades é removida da rota e inserida em outra posição. A Figura 4.9 ilustra dois operadores de perturbação.

- **Função Objetivo:** a função objetivo é definida como a equação:

$$E = \sum_{n=1}^{N-1} \|(x_n, y_n) - (x_{n+1}, y_{n+1})\| + \|(x_N, y_N) - (x_1, y_1)\|$$

- **Procedimento de Annealing:** o procedimento é dado pelos passos a seguir:

  - Gere várias configurações randômicas, calcule o ΔE máximo, e defina T^0 (temperatura inicial) como algumas dezenas de vezes o valor de Δ_E.
  - Para cada T^k, execute $100N$ ou mais reconfigurações, aceitando-as ou não conforme a probabilidade de Boltzman.
  - Faça $T^{k+1} \leftarrow 0,98T^k$.

4.4 Exercícios

EX 4.1 (O Problema de Bi-Partição em Grafos): Seja $G = (V, E)$ um grafo não direcionado cujas arestas possuem pesos não-negativos. Uma aresta $(u, v) \in E$ possui peso $w(u, v)$. Para dois conjuntos A e B de vértices, definimos $w(A, B)$

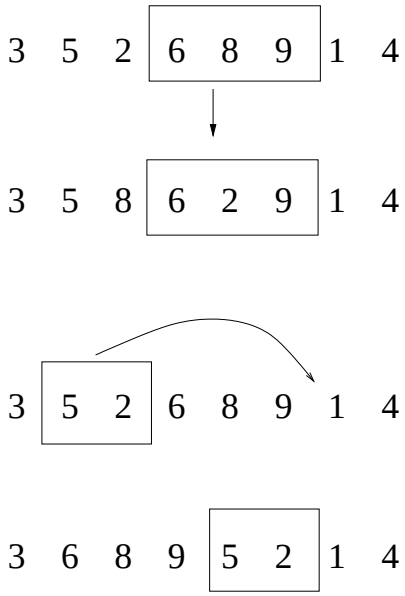


Figura 4.9: Ilustração dos operadores de perturbação

o peso total das arestas que têm um extremo em A e o outro, em B. Mais precisamente:

$$w(A,B) = \sum_{(u,v)\in E, u\in A, v\in B} w(u,v)$$

O problema é encontrar $S \subseteq V$ tal que $w(S, V-S)$ seja máximo.

Os dados de uma instância do problema da bi-partição são fornecidos em arquivos tipo texto, cujo formato consiste do número de vértices, do número de arestas e de uma lista de arestas e pesos.

Exemplo de instância:
5 6
1 3 0.3
2 4 0.5
2 5 1.2
1 5 0.8
3 4 0.5
2 3 3.0

Tarefas

i. implemente um Algoritmo Genético para resolver o problema de bi-partição e teste o algoritmo em alguns problemas;

ii. implemente e teste um algoritmo Simulated Annealing para o problema de bi-partição; e

iii. Compare os algoritmos e os resultados obtidos.

Capítulo 5

Treinamento de Redes Neurais: Um Problema de Otimização

Redes Neurais (RNs) constituem métodos "robustos" de aproximação de funções de valores vetoriais, reais ou discretos. Em outras palavras, as redes neurais toleram erros nos pares de treinamento. Uma rede neural pode ser vista como um aproximador universal ou, mais precisamente, um interpolador universal uma vez que elas não são capazes de inferir informações que estão fora do conjunto de treinamento. Dado um conjunto qualquer de treinamento podemos, em princípio, construir uma rede neural que produz os mesmos valores da função para os pares entrada-saída de treinamento. O algoritmo de treinamento de propagação reversa (*back-propagation*) nada mais é do que uma implementação especializada do algoritmo de descenso, sendo objeto de estudo neste capítulo.

Como o próprio nome sugere, redes neurais são inspiradas em sistemas biológicos de aprendizagem que tipicamente consistem de redes complexas de unidades elementares, conforme a Figura 5.1. Na figura, a primeira camada recebe os sinais de entrada, que são processados e transformados em novos sinais a serem transmitidos às unidades da camada intermediária, que por sua vez processa os sinais de entrada e os envia à camada de saída.

É interessante considerarmos que o cérebro humano possui aproximadamente 10^{11} neurônios (100 bilhões de neurônios), cada um conectado com cerca de 10^4 neurônios. O tempo de resposta de um neurônio é aproximadamente 10^{-3} segundos, muito mais lento do que um computador, cujas portas lógicas apresentam um tempo de resposta da ordem de 10^{-10} segundos. O cérebro humano reconhece uma figura em cerca de 10^{-1} segundos. Dado o tempo relativamente lento de resposta do neurônio, em comparação com a velocidade de processamento de informação, podemos concluir que o sistema de

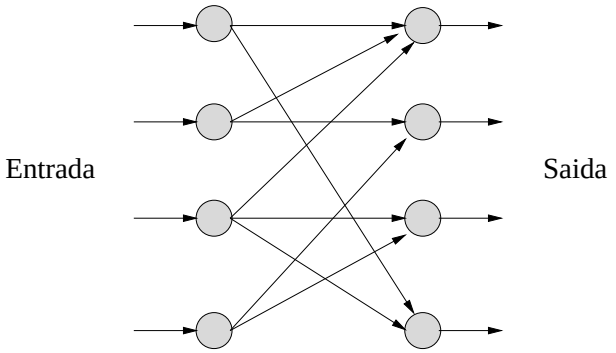


Figura 5.1: Exemplo de topologia de redes neurais

aprendizagem biológico é extremamente paralelo e distribuído, todavia não possui um número muito grande de camadas.

Existem, entretanto, incompatibilidades entre as redes neurais artificiais e as unidades do sistema biológico de aprendizagem. A saída de uma unidade artificial é um valor constante, enquanto que as unidades biológicas emitem uma série complexa de sinais, consistindo em pulsos com várias freqüências.

5.1 Elementos Básicos das Redes Neurais

O elemento básico de uma rede neural é a unidade de processamento. Na Figura 5.2 é ilustrada a estrutura geral de uma unidade de processamento neural, a qual consiste em um vetor com os sinais de entrada $x = (x_1, \ldots, x_n)$, uma unidade de combinação linear dos sinais de entrada cujos pesos são dados pelo vetor $w = (w_1, \ldots, w_n)$, uma função de ativação $f(s)$ onde $s = \sum_{i=1}^{n} w_i x_i$ que produz os sinais de saída tal que $y_i = f(s)$ para $i = 1, \ldots, m$. Unidades de processamento são organizadas em camadas e conectadas por arcos (sinapses) que simbolizam os sinais de comunicação de uma camada (ou sinais de entradas) para outra camada. Um exemplo de rede neural é ilustrada na Figura 5.3.

5.1.1 O Problema de Treinamento

Dados: uma rede neural (a sua topologia), ou seja, um grafo $G = (V, E)$; exemplos de treinamento, ou seja, uma lista de pares de entrada e saída desejados $D = \{(x^1, y^1), \ldots, (x^K, y^K)\}$.

O problema é encontrar pesos das sinapses tal que o erro de estimação das saídas seja o menor possível ("least squares problem"). Seja $f(w, x) : \mathbb{R}^n \to \mathbb{R}^m$ a função

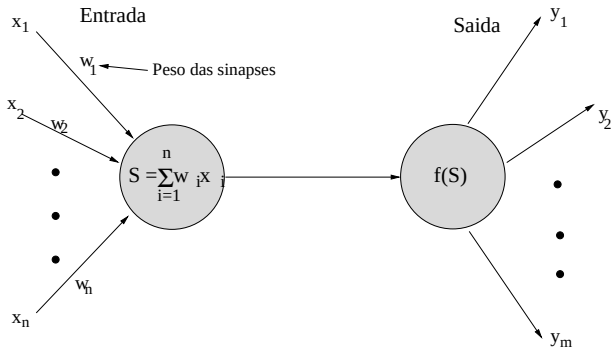


Figura 5.2: Unidade de processamento neural

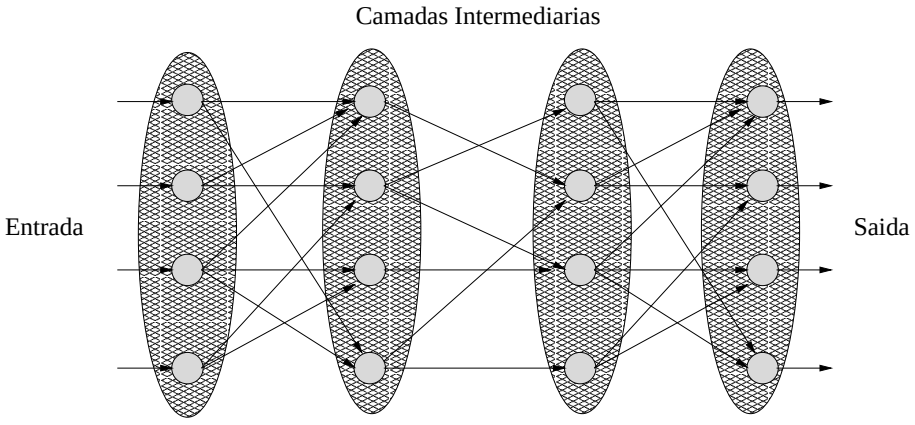


Figura 5.3: Rede neural com camadas intermediárias

representada pela rede neural com pesos dados por um vetor $w \in \mathbb{R}^r$. Dado x, a rede responde com um valor $y = f(x)$. Em programação matemática, o problema de treinamento da rede neural pode ser escrito como

$$\begin{aligned} &Minimize \quad \tfrac{1}{2} \sum_{k=1}^{K} \|f(w, x^k) - y^k\|^2 \\ &w \in \mathbb{R}^r \end{aligned}$$

5.1.2 ALVINN: Um Exemplo de Aplicação

Um exemplo de aplicação de redes neurais é o sistema ALVINN desenvolvido na Universidade de Carnegie Mellon, que utilizou com sucesso o modelo de redes neurais para dirigir de forma autônoma um veículo por cerca de 120 Km a uma velocidade de até 100 Km/h. Mais recentemente, este mesmo sistema foi aprimorado e desempenhou com sucesso a tarefa mais audaciosa de dirigir o mesmo veículo, de uma forma semi-autônoma, da cidade de Pittsburgh na Pennsylvania até a cidade de Berkeley na California. Uma ilustração simplificada da rede neural do sistema ALVINN é mostrada na Figura 5.4. Cada pixel da imagem é primeiramente transformado em um tonalidade de cinza e depois enviado à respectiva unidade neural, que por sua vez processa o sinal recebido e retransmite a outras unidades da camada intermediária. A camada neural de saída conta com várias unidades que indicam o grau de giro do volante, onde a unidade mais à esquerda indica giro máximo à esquerda enquanto que a unidade mais à direita indica giro oposto. A unidade cujo valor de saída é mais intenso terá seu comando executado pelo sistema robótico.

5.1.3 Problemas Apropriados para Redes Neurais

Em geral, as redes neurais são recomendadas para problemas de aproximações complexos tipicamente de natureza não-linear e sujeitos a ruídos, tais como dados e sinais provenientes de câmeras e microfones. Mais especificamente, RNs em combinação com o algoritmo de propagação reversa (*back-propagation*) são adequadas para problemas com as seguintes características:

- instâncias que são representadas por pares contendo muitos atributos (múltiplas entradas e múltiplas saídas), ou seja, as dimensões dos vetores x e y;

- o atributo de saída tenha um valor discreto, real ou um vetor de valores; no ALVINN, por exemplo, a saída é um vetor com cerca de 30 saídas, cada uma representando a probabilidade de um comando.

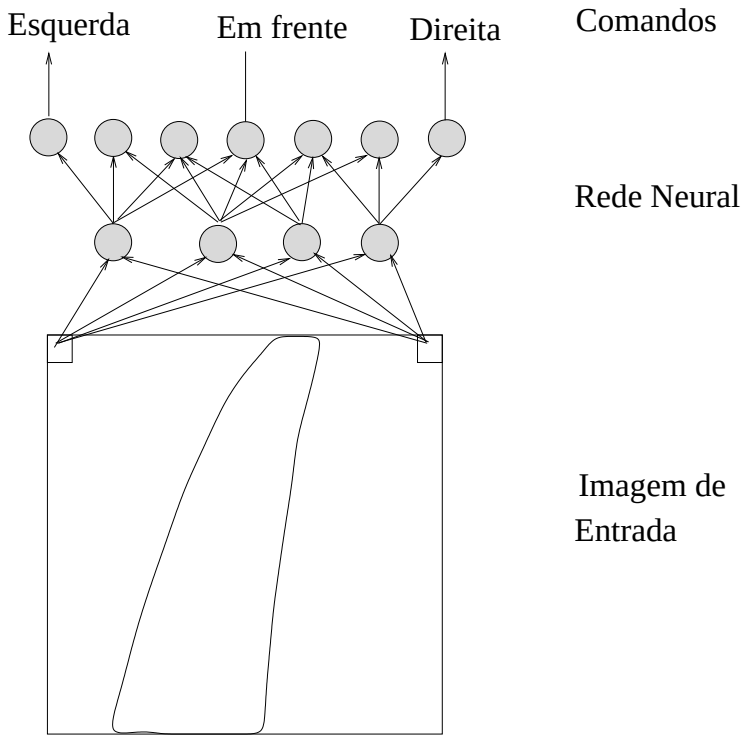


Figura 5.4: Rede neural do sistema ALVINN

- treinamento pode ser feito *off-line*;
- a avaliação da função deve ser muito rápida, para uso em aplicações de tempo-real;
- a habilidade de se entender o significado da aproximação não é necessária, o que pode ser um obstáculo para o uso de redes neurais em aplicações de diagnóstico médico, onde é imprescindível que o sistema justifique uma decisão e indique a seqüência de inferência que leva à conclusão; e
- os exemplos de treinamento podem conter erros.

5.2 Perceptron: A Primeira Unidade Neural

Marvin Minsky e Dean Edmonds construiram o primeiro computador neural em 1951. Ironicamente, Minsky mais tarde provou teoremas que contribuíram para o fim das pesquisas em redes neurais até meados de 1970. Embora a Ciência da Computação tenha desprezado o campo das redes neurais após a publicação do livro Perceptrons, por Minsky e Papert, trabalhos de pesquisa continuaram em outros domínios, espe-

cialmente na Física. A motivação principal veio nos anos 80 quando pelo menos 4 grupos re-inventaram o algoritmo de propagação reversa (*backpropagation algorithm*), inicialmente proposto por Bryson e Ho, em 1969. Os algoritmos foram aplicados a vários problemas de aprendizagem em Ciência da Computação e Psicologia. A estrutura básica da unidade perceptron é ilustrada na Figura 5.5. Em essência, a unidade recebe o sinais de entrada $x_1, \ldots, x_n$, pois assumimos que o sinal x_0 é sempre 1, e os combina linearmente de acordo com os pesos sinápticos $w_0, \ldots, w_n$, obtendo o sinal $w^T x$, onde $x = (x_1, \ldots, x_n)$ e $w = (w_0, \ldots, w_n)$. O sinal de saída enviado pelos canais $y_1, \ldots, y_m$ são idênticos, assumindo o valor 1 se o sinal de $w^T x$ é positivo e -1 caso contrário. A função de saída se comporta, portanto, como um *threshold*: se $(w_1, \ldots, w_n)^T (x_1, \ldots, x_n) > w_0$ onde w_0 contém o valor do *threshold* então o sinal de saída assume o valor máximo, 1, de outra forma o sinal assume o valor mínimo, -1.

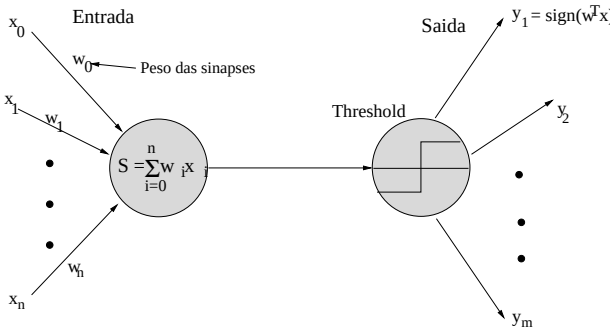


Figura 5.5: Perceptron

5.2.1 Treinando um Perceptron

Sendo dado um conjunto de pontos de treinamento $D = \{(x^1, t^1), \ldots, (x^M, t^M)\}$, considere o problema de treinamento de um perceptron, que consiste em minimizar o erro de predição:

$$P: \quad \begin{array}{ll} Minimize & \sum_{k=1}^{M} \left(sign(w^T x^k) - t^k\right)^2 \\ & w \in \mathbb{R}^n \end{array}$$

Quais são os poss'iveis obstáculos a serem encontrados na resolução de P? Um obstáculo fundamental é o fato de que a função $sign(x)$ não é diferenciável, o que inviabiliza o emprego de algoritmos eficientes baseados no gradiente da função. Outra limitação fundamental advém do fato de que o perceptron só consegue classificar corretamente, com erro nulo, aqueles conjuntos de dados que são linearmente separáveis. Note que a região $H = \{x \in \mathbb{R}^n : sign(w^T x) \leq 0\}$ forma um hiper-espaço, sendo a

região complementar $\bar{H}$ dada por $H = \{x \in \mathbb{R}^n : sign(w^T x) > 0\}$. Em outras palavras, O perceptron define um subespaço multidimensional, classificando como positivo os exemplos que estão de um lado e como negativo os demais. Esta limitação é ilustrada na Figura 5.6. O perceptron pode ser usado para representar as funções lógicas *and*, *or*, *nor* e *nand*. Por si só, entretanto, o perceptron não consegue implementar a porta lógica *xor*. Todavia, podemos aumentar o poder de representação do peceptron se utilizarmos uma rede. Qualquer função Booleana dada na forma normal disjuntiva (i.e., uma disjunção de clásulas) pode ser representada por uma rede de perceptrons.

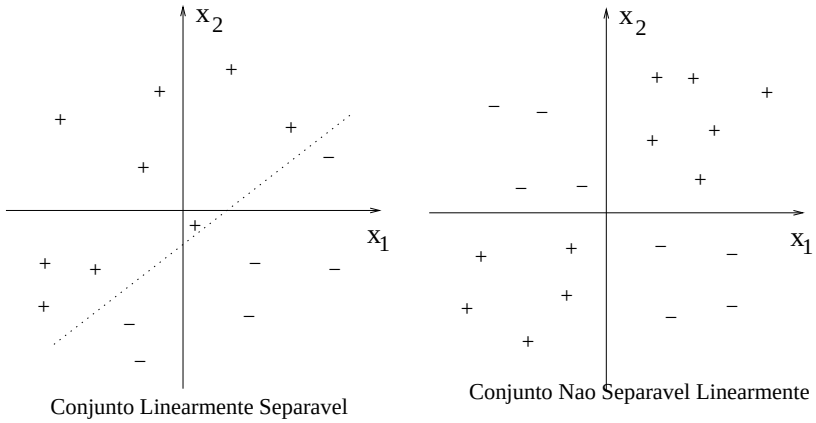


Figura 5.6: Conjunto separáveis e não separáveis linearmente

O procedimento de treinamento de um perceptron é o elemento básico para desenvolver o procedimento de treinamento de uma rede. Assim apresentamos abaixo um algoritmo de treinamento de um perceptron.

Algoritmo de Treinamento de um Perceptron

`O vetor com pesos sinapticos` $w = (w_0, \ldots, w_n)$
  `e inicializado randomicamente` $(-0,5 < w_i < 0,5)$
$\alpha \leftarrow$ `taxa de aprendizagem (e.g.,` $\alpha = 0,01$`)`
$m \leftarrow 1$ `(temos` M `exemplos de treinamento)`
Repita
  `Para` $i = 0, \ldots, n$
    $\Delta w_i = \alpha[t^m - sign(w^T x^m)]x_i^m$
    $w_i = w_i + \Delta w_i$
  `FimPara`
$m \leftarrow mod(m, M) + 1$
`FimRepita`

Uma propriedade interessante do algoritmo acima é que ele converge sempre que o conjunto de treinamento for linearmente separável.

5.3 Regra Delta

Um dos obstáculos ao uso do perceptron é a possibilidade de falha quando os exemplos de treinamento não são linearmente separáveis. A regra delta supera esta dificuldade, propiciando a concepção de um algoritmo que converge para a melhor aproximação linear da relação entre entrada e saída (a função entrada-saída desconhecida). Em síntese, a regra delta consiste em substituir a saída $o(x) = sign(w^T x)$ pela saída $o(x) = w^T x$, i.e., simplesmente eliminamos a função sinal deixando a função de saída o uma função linear e diferenciável. A unidade delta é ilustrada na Figura 5.7.

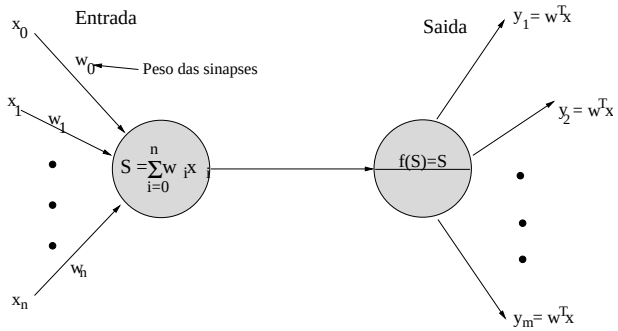


Figura 5.7: Unidade delta

5.3.1 Treinando uma Unidade Delta

Dado um conjunto fixo de exemplos $D = (x^1, t^1), \ldots, (x^M, t^M)$, onde $x^k \in \mathbb{R}^{n+1}$ e $t^k \in \mathbb{R}$, $k = 1, \ldots, M$, o problema de treinamento da unidade delta consiste em encontrar o vetor $w \in \mathbb{R}^{n+1}$ que minimize o erro de predição. (Note que assumimos que $x_0^k = 1$.) Em programação matemática, o problema é expresso como:

$$P: \quad Minimize \quad E(w) = \frac{1}{2} \sum_{k=1}^{M} \left(w^T x^k - t^k\right)^2$$
$$w \in \mathbb{R}^{n+1}$$

Portanto, o problema de minizar a função $E(w)$ está dentro do domínio da otmização diferenciável sem restrições. Podemos, por exemplo, aplicar o algoritmo de descenso ou algoritmo de Newton. Qualquer que seja a abordagem, será necessário calcular

o gradiente de E. Seja $\nabla E = [\partial E/\partial w_0, \partial E/\partial w_1, \ldots, \partial E/\partial w_n]$ o gradiente de E. Podemos calcular a derivada parcial de E com respeito a w_j como segue:

$$\begin{aligned}\frac{\partial E}{\partial w_j} &= \frac{\partial}{\partial w_j}\left\{\frac{1}{2}\sum_{k=1}^{M}\left(w^T x^k - t^k\right)^2\right\} \\ &= \sum_{k=1}^{M}\left(w^T x^k - t^k\right)\frac{\partial}{\partial w_j}\left(w^T x^k - t^k\right) \\ &= \sum_{k=1}^{M}\left(w^T x^k - t^k\right)x_j^k\end{aligned}$$

Note que ∇E pode ser convenientemente calculado em termos dos valores das saídas para as diferentes entradas, $w^T x^k - t^k$, e o próprio valor da k-ésima entrada x^k. Se utilizarmos um algoritmo de descenso para resolver o problema, então dada uma solução candidata w^l, o algoritmo de descenso busca a solução alternativa $w^{l+1} = w^l - \alpha^l \nabla E(w^l)$, sendo α^l a solução (aproximada) do problema:

$$\begin{aligned}&Minimize \quad f(\alpha) = E\left[w^l - \alpha^l \nabla E(w^l)\right] \\ &\alpha \in \mathbb{R} \\ &Sujeito\ a: \\ &\qquad \alpha \geq 0\end{aligned}$$

A capacidade representacional de uma rede com múltiplas camadas de unidades lineares é superior à capacidade de uma unidade linear? Não, pois uma composição linear de funções lineares continua sendo uma função linear. Assim, uma rede neural multi-camada de unidades delta não é mais representativa do que o espaço das funções lineares.

5.4 A Unidade Sigmoid

Objetivando contornar as limitações da unidade delta e, ao mesmo tempo, incorporar o poder representacional de uma rede de perceptrons, foi proposto a aproximação da função *threshold* por uma função diferenciável. Uma função com tais propriedades é a função sigmoid, cujo comportamento é ilustrado na Figura 5.8

Dados i) um grafo $G = (V, A)$ representando uma rede com múltiplas camadas, cujas unidades de processamento neural são do tipo sigmoid, e ii) uma lista com exemplos de treinamento $(x^1, t^1), \ldots, (x^M, t^M)$, onde $x^k \in \mathbb{R}^{n+1}$ e $t^k \in \mathbb{R}^m$, podemos utilizar o algoritmo de propagação reversa para treinar a rede. (Lembramos que $x^k = [x_0^k, x_1^k, \ldots, x_n^k]$,

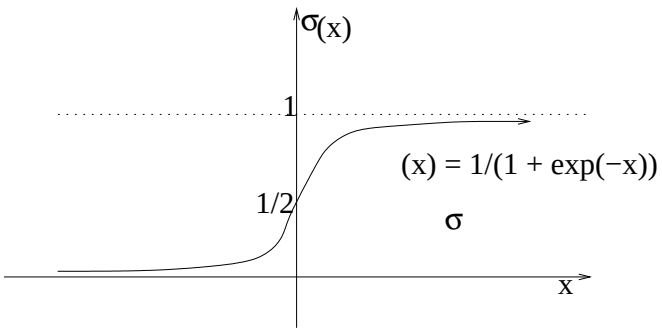


Figura 5.8: Função sigmoid

$w^k = [w_0^k, w_1^k, \ldots, w_n^k]$ e $x_0^k = 1$.) Este algoritmo consiste em um algoritmo de descenso que objetiva minimizar o erro entre as saídas da rede e os valores desejados. Em programação matemática, este problema pode ser especificado como:

$$P: \quad Minimize \quad E(w) = \frac{1}{2} \sum_{k=1}^{M} \sum_{j \in Saida} \left(o_j(x^k) - t^k\right)^2$$
$$w \in \mathbb{R}^r$$

onde w é o vetor com os pesos sinápticos de todos os arcos de G e $o_j(x)$ é a saída da j-ésima unidade da camada de saída da rede neural. Redes multi-camada com unidades sigmoid podem representar funções não lineares de elevada complexidade. Elas podem ser utilizadas em reconhecimento de voz, identificação e modelagem de processos não-lineares, e predição de grandezas econômicas. Na Figura 5.9 é ilustrado um sistema neural para reconhecimento de palavras que toma como entrada dois sinais com a intensidade de duas freqüências.

5.5 Exercícios

EX 5.1 Indique como se implementa com um perceptron as funções lógicas *AND*, *NAND*, *OR* e *NOR*.

EX 5.2 Considere o grafo $G = (V, E)$ da 5.10, o qual representa uma rede neural com três camadas: a camada de entrada com quatro unidades (vértices do grafo); a camada intermediária com duas unidades; e a camada de saída com quatro unidades. Seja o_j o valor da saída de uma unidade j. As unidades de entrada simplesmente copiam os valores de entrada para suas saídas, ou seja, $(o_1 = x_1, \ldots, o_4 = x_4)$. As saídas da rede neural correspondem às saídas das unidades da terceira camada

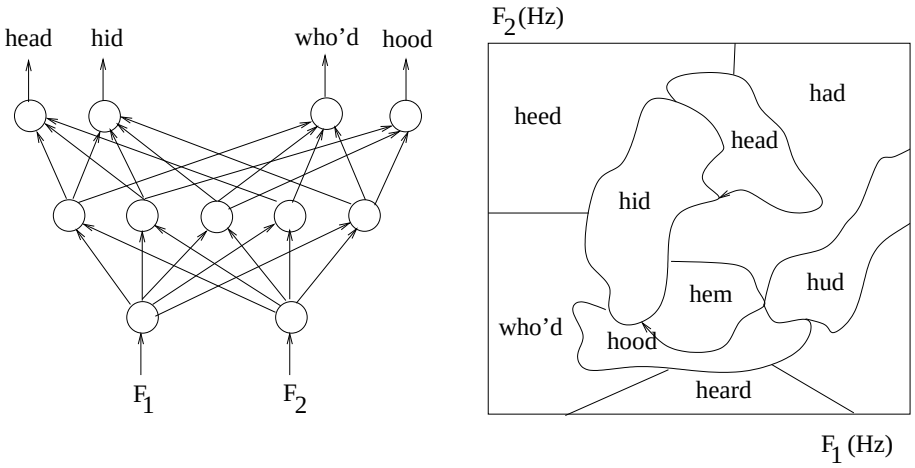


Figura 5.9: Sistema neural para reconhecimento de palavras

$(o_7 = y_1, \ldots, o_{10} = y_4)$.

Seja j uma unidade intermediária ou de saída. Esta unidade j calcula sua saída o_j como segue:

$$net_j = \sum_{(i,j)\in E} w_{ij} o_i$$

$$o_j = \sigma(net_j) = \frac{1}{1 + e^{-net_j}}$$

O conjunto de entradas e saídas desejadas é dado na Tabela 5.1. Dados os pesos dos arcos de G $(w_{15}, w_{16}, w_{25}, w_{26}, \ldots, w_{67}, w_{68}, w_{69}, w_{6,10})$, a rede neural pode ser vista como uma função $f(x) : \mathbb{R}^4 \to \mathbb{R}^4$ que, dado o valor de x, retorna o valor estimado de y.

Tabela 5.1: Exemplos para treinamento da rede neural

	Entrada	Saída
t	$x^t = (x_1^t, x_2^t, x_3^t, x_4^t)$	$(y_1^t, y_2^t, y_3^t, y_4^t)$
1	(0,0,0,1)	(0,0,0,1)
2	(0,0,1,0)	(0,0,1,0)
3	(0,1,0,0)	(0,1,0,0)
4	(1,0,0,0)	(1,0,0,0)

Tarefas

i. Formule o problema de treinar esta rede neural em programação matemática.

(Sugestão: defina restrições para calcular net_j^t e o_j^t para cada unidade j e exemplo de treinamento t como uma função dos pesos (w's) e das variáveis (net_j^t e o_j^t) que antecedem a unidade em questão; formule uma função objetivo para minimizar o erro entre as saídas da rede neural e os valores desejados–i.e, o erro entre o_{6+k}^t e o resultado esperado y_k^t, para $k = 1, \ldots, 4$.)

ii. Se você utilizou restrições, reescreva o problema em programação matemática sem utilizar restrições.

iii. Aplique o algoritmo de descenso para encontra o valores dos pesos sinápticos, i.e., o vetor w que minimiza o erro.

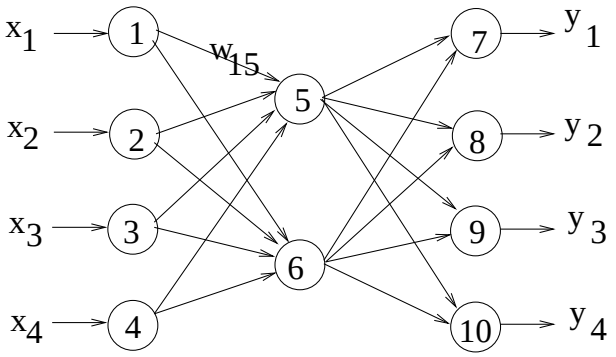


Figura 5.10: Topologia da rede neural

Capítulo 6

Programação Linear

6.1 Problema Exemplo: Gerenciamento de Uma Unidade de Produção

Uma unidade de produção pode manufaturar uma variedade de produtos $N = \{1, \ldots, n\}$, sendo estes produtos montados a partir de um conjunto de matérias-primas $M = \{1, \ldots, m\}$. As decisões gerenciais são dinâmicas, evoluindo conforme as condições do mercado. Considere um certo momento onde: b_i é a quantidade de matéria-prima i disponível; ρ_i é o custo unitário da matéria-prima i; a manufatura do produto j requer a_{ij} unidades da matéria-prima i; e o produto j pode ser vendido ao preço σ_j por unidade. A seguir investigaremos dois problemas relacionados.

6.1.1 Gerente de Produção Otimista

O problema envolve a alocação da matéria-prima na manufatura de itens acabados. Se o gerente decide produzir x_j unidades do produto j, então o lucro associado com a venda de uma unidade de produto j é $c_j = \sigma_j - \sum_{i=1}^{m} \rho_i a_{ij}$. O problema de determinar o número de unidades a serem vendidas de forma a maximizar o lucro pode ser expresso em programação matemática como segue:

$$
\begin{array}{lll}
P_0: & \mathit{Maximize} & \sum_{j=1}^{n} c_j x_j \\
& \mathit{Sujeito\ a}: & \\
& & x_j \geq 0 \qquad\qquad j = 1, \ldots, n \\
& & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m
\end{array}
\tag{6.1}
$$

Assim a tarefa do gerente é determinar as quantidades de produção, $\{x_j\}$, que maximizem as função objetivo de 6.1 enquanto que as restrições sejam satisfeitas.

6.1.2 Gerente de Produção Pessimista

Nesta situação o gerente assume a posição de contador tendo como tarefa avaliar o preço de venda de cada matéria-prima, em outras palavras, a unidade de manufatura não produzirá e passará a se comportar como um depósito. A unidade deve estar preparada para vender a matéria-prima. Assim seja w_i o valor unitário para vender a matéria-prima i, a ser definido pelo gerente. O valor total da venda de todo o estoque é $\sum_{i=1}^{m} w_i$. O objetivo do gerente é encontrar o valor mais baixo para venda que não incorra perdas financeiras. Portanto, duas restrições são:

- $w_i \geq \rho_i, i = 1, \ldots, m$, o valor do item i deve ser maior ou igual ao de mercado; e
- $\sum_{j=1}^{m} a_{ij} w_j \geq \sigma_j, j = 1, \ldots, n$, pois de outra forma seria mais lucrativo utilizar a matéria-prima para gerar produto j e vendê-lo no mercado.

O problema a ser resolvido pelo gerente então fica:

$$
\begin{aligned}
P_1: \quad & Minimize \quad \sum_{i=1}^{m} w_i \\
& Sujeito\ a: \\
& \qquad w_i \geq \rho_i \qquad\qquad i = 1, \ldots, m \\
& \qquad \sum_{i=1}^{m} a_{ij} w_i \geq \sigma_j \qquad j = 1, \ldots, n
\end{aligned}
$$

Agora, efetue a mudança de variável conforme segue:

$$
\begin{aligned}
w_i \geq \rho_i \quad &\Leftrightarrow \quad w_i - y_i = \rho_i \ e \ y_i \geq 0 && (6.2) \\
&\Leftrightarrow \quad w_i = y_i + \rho_i \ e \ y_i \geq 0 && (6.3)
\end{aligned}
$$

Em termos das variáveis $\{y_i\}$, o problema se torna:

$$
\begin{aligned}
P_2: \quad & Minimize \quad \sum_{i=1}^{m} b_i y_i + \sum_{i=1}^{m} b_i \rho_i \\
& Sujeito\ a: \\
& \qquad \sum_{i=1}^{m} a_{ij} y_i + \sum_{i=1}^{m} a_{ij} \rho_i \geq \sigma_j \qquad j = 1, \ldots, n \\
& \qquad y_i \geq 0 \qquad\qquad i = 1, \ldots, m
\end{aligned}
$$

Uma vez que $b_i\rho_i$ é uma constante para todo i, o problema pode ser expresso na forma abaixo:

$$\begin{aligned} P_3: \quad & Minimize \quad \sum_{i=1}^{m} b_i y_i \\ & Sujeito\ a: \\ & \qquad \sum_{i=1}^{m} a_{ij} y_i \geq c_j \qquad j = 1, \ldots, n \\ & \qquad y_i \geq 0 \qquad\qquad\quad i = 1, \ldots, m \end{aligned}$$

Veremos na seqüência que P_3 é o problema dual de P_0.

6.2 O Problema de Programação Linear

Programação linear (PL) envolve a escolha de valores para variáveis de forma ótima. As variáveis $\{x_j\}$ são ditas variáveis de decisão. O objetivo em PL é sempre maximizar (minimizar) uma função linear:

$$Minimize\ c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

sob restrições da forma:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b$$

É fácil converter uma restrição do tipo $\leq$ em outra equivalente do tipo $=$

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b \Leftrightarrow a_1x_1 + a_2x_2 + \ldots + a_nx_n + s = b,\ s \geq 0.$$

A variável s é dita variável de folga (*slack variable*), já que seu valor corresponde à quantidade de recurso b não utilizada. Podemos também converter uma restrição do tipo $=$ em restrições equivalentes do tipo $\leq$

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b \Leftrightarrow \left\{ \begin{array}{l} a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b \\ a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq b \end{array} \right.$$

Podemos ainda converter variáveis irrestritas em sinal em variáveis não negativas:

$$x \in \mathbb{R} \Leftrightarrow \left\{ \begin{array}{l} x = x^+ - x^- \\ x^+ \geq 0 \\ x^- \geq 0 \end{array} \right.$$

Portanto, podemos assumir que qualquer problema em PL pode ser especificado (re-expresso) na forma abaixo:

$$
\begin{array}{lll}
PL: & Maximize & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
& Sujeito\ a: & \\
& & a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n \leq b_1 \\
& & a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n \leq b_2 \\
& & \vdots \\
& & a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n \leq b_m \\
& & x_1, \ldots, x_n \geq 0
\end{array}
$$

Um vetor com valores para as variáveis de decisão $x = (x_1, \ldots, x_n)$ é dito uma solução candidata. A solução x é dita factível se satisfaz as restrições. A solução x é dita ótima se ela é factível e induz o valor ótimo da função objetivo.

Exemplo de Problema Infactível

Um problema é dito infactível se não existe uma solução candidata $x\mathbb{R}^n$ que satisfaça às restrições. O problema abaixo é um exemplo de problema infactível, sendo este ilustrado na Figura 6.1.

$$
\begin{array}{ll}
Maximize & 5x_1 + 4x_2 \\
Sujeito\ a: & \\
& \begin{array}{rcrcr} x_1 & + & x_2 & \leq & 1 \\ -2x_1 & - & 2x_2 & \leq & -9 \end{array} \\
& x_1, x_2 \geq 0
\end{array}
$$

Exemplo de Problema Ilimitado

Um problema é ilimitado se não existe um limite superior para o valor da função objetivo, ou seja, podemos crescer o valor da função objetivo arbitrariamente. Abaixo segue um exemplo de problema ilimitado cuja ilustração é feita na Figura 6.2.

$$
\begin{array}{ll}
Maximize & x_1 - 4x_2 \\
Sujeito\ a: & \\
& \begin{array}{rcrcr} -2x_1 & + & x_2 & \leq & -1 \\ -x_1 & - & 2x_2 & \leq & -2 \end{array} \\
& x_1, x_2 \geq 0
\end{array}
$$

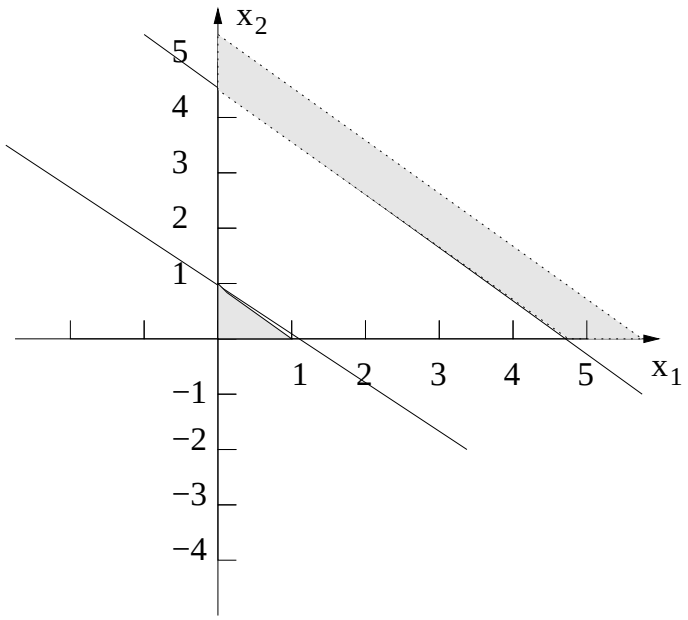


Figura 6.1: Exemplo de problema infactível

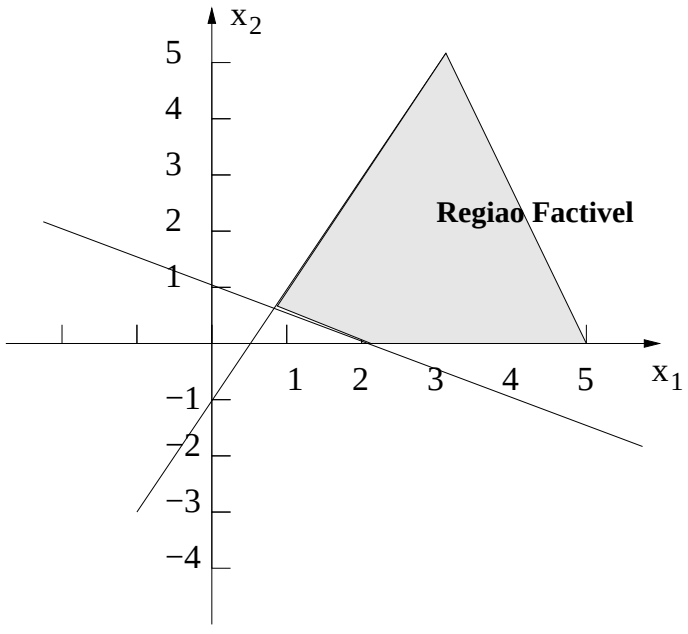


Figura 6.2: Exemplo de problema ilimitado

6.3 Algoritmo Simplex

O algoritmo simplex, proposto por George Dantzig nos anos 40, constituiu um grande avanço científico-tecnológico e deu grande impulso ao campo da pesquisa operacional que estava dando os primeiros passos. O nome do algoritmo tem suas raízes no conceito de simplex: um plano que corta os vetores unitários. O algoritmo como é conhecido atualmente difere da versão original e tem servido de base para versões estendidas para tarefas específicas como, por exemplo, o método dual simplex que é amplamente adotado em implementações *branch-and-bound* e *branch-and-cut* para resolução de problemas inteiros, e o método simplex adaptado para o problema de fluxo em redes de custo mínimo.

O algoritmo simplex pode ser visto como um processo combinatório que procurar encontrar as colunas da matriz de restrições que induzem uma base e, portanto, uma solução básica ótima. A dificuldade advém do fato que tipicamente existe um numéro exponencial de possíveis combinações de colunas, gerando portando um desempenho de pior caso de ordem exponencial. Apesar deste aspecto desfavorável, o algoritmo simplex é eficaz e para muitas instâncias continua sendo o algoritmo mais rápido, mesmo quando comparado com a algoritmos de ponto-interior que tem desempenho polinomial no pior caso.

No que segue apresentamos o algoritmo de simplex através de um exemplo. Na seqüência apresentamos o algoritmos em detalhes e tratamos de aspectos de inicialização.

6.3.1 Exemplo

Considere o problema de programação linear abaixo:

$$\begin{aligned} &Maximize && 5x_1 + 4x_2 + 3x_3 \\ &Sujeito\ a: && \\ & && 2x_1 + 3x_2 + x_3 \leq 5 \\ & && 4x_1 + x_2 + 2x_3 \leq 11 \\ & && 3x_1 + 4x_2 + 2x_3 \leq 8 \\ & && x_1, x_2, x_3 \geq 0 \end{aligned} \tag{6.4}$$

Após adicionarmos variáveis de folga, o problema toma a forma a seguir:

$$\begin{array}{lrcrcrcrcr} Maximize & \delta & = & 0 & & 5x_1 & + & 4x_2 & + & 3x_3 \\ Sujeito\ a: & & & & & & & & & \\ & w_1 & = & 5 & - & 2x_1 & - & 3x_2 & - & x_3 \\ & w_2 & = & 11 & - & 4x_1 & - & x_2 & - & 2x_3 \\ & w_3 & = & 8 & - & 3x_1 & - & 4x_2 & - & 2x_3 \\ & x_1, x_2, x_3, w_1, w_2, w_3 \geq 0 & & & & & & & & \end{array} \tag{6.5}$$

O método simplex é um processo iterativo que inicia com uma solução $y^0 = (x_1^0, \ldots, x_n^0, w_1^0, \ldots, w_m^0)$, onde $n = 3$ e $m = 3$, satisfazendo as equações de (6.4). O simplex procura uma nova solução y^1 tal que: $5x_1^1 + 4x_2^1 + 3x_3^1 > 5x_1^0 + 4x_2^0 + 3x_3^0$.

Algoritmo Simplex

Inicialização

Para iniciarmos o processo, necessitamos de uma solução factível, tal como

$$x_1^0 = 0,\ x_2^0 = 0,\ x_3^0 = 0,\ w_1^0 = 5,\ w_2^0 = 11,\ w_3^0 = 8$$

Esta solução y^0 induz o valor $\delta = 0$ para a função objetivo.

Passo 1

A solução corrente não é ótima. Qualquer acréscimo no valor de x_1 aumenta o valor de δ. Mas não podemos aumentar o valor de x_1 ilimitadamente já que este está limitado pelas desigualdades abaixo:

$$\left\{\begin{array}{lcl} w_1 = 5 - 2x_1 & \geq & 0 \\ w_2 = 11 - 4x_1 & \geq & 0 \\ w_3 = 8 - 3x_1 & \geq & 0 \end{array}\right. \Rightarrow \left\{\begin{array}{lcl} x_1 & \leq & 5/2 = 2.5 \\ x_1 & \leq & 11/4 = 2.75 \\ x_1 & \leq & 8/3 = 2.667 \end{array}\right.$$

Assim, o valor de x_1 na próxima iteração é o menor dentre $\{5/2, 11/4, 8/3\}$, o que leva a igualdade:

$$x_1 = \frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3 \tag{6.6}$$

Substituindo-se (6.6) no sistema (6.5) de maneira a transferir-se a variável w_1 para o lado direito, obtém-se:

$$\begin{array}{rcl} x_1 & = & \frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3 \\ \\ w_2 & = & 11 - 4\left(\frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3\right) - x_2 - 2x_3 \\ & = & 1 + 2w_1 + 5x_2 \\ \\ w_3 & = & 8 - 3\left(\frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3\right) - 4x_2 - 2x_3 \\ & = & \frac{1}{2} + \frac{3}{2}w_1 + \frac{1}{2}x_2 - \frac{1}{2}x_3 \\ \\ \delta & = & 5\left(\frac{5}{2} - \frac{1}{2}w_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3\right) + 4x_2 + 3x_3 \\ & = & \frac{25}{2} - \frac{5}{2}w_1 - \frac{7}{2}x_2 + \frac{1}{2} \end{array} \tag{6.7}$$

Agora, substituindo as equações (6.7) no "dicionário" (6.5), obtém-se o dicionário abaixo:

$$\begin{array}{rrcccc} Max & \delta & = & \frac{25}{2} & -\frac{5}{2}w_1 & -\frac{7}{2}x_2 & +\frac{1}{2}x_3 \\ & x_1 & = & \frac{5}{2} & -\frac{1}{2}w_1 & -\frac{3}{2}x_2 & -\frac{1}{2}x_3 \\ & w_2 & = & 1 & +2w_1 & +5x_2 & \\ & w_3 & = & \frac{1}{2} & +\frac{3}{2}w_1 & +\frac{1}{2}x_2 & -\frac{1}{2}x_3 \end{array} \tag{6.8}$$

A solução induzida pelo dicionário (6.8) é $y^1 = (x_1^1, x_2^1, x_3^1, w_1^1, w_2^1, w_3^1) = (\frac{5}{2}, 0, 0, 0, 1, \frac{1}{2})$ cujo valor da função objetivo é $\delta = \frac{5}{2}$. Neste dicionário, as variáveis x_1, w_2, e w_3 são ditas variáveis básicas tal que o conjunto $\mathbb{B} = \{x_1, w_2, w_3\}$ contém as variáveis básicas. As demais variáveis são ditas não básicas, sendo o conjunto $\mathbb{N} = \{x_2, x_3, w_1\}$ o conjunto das variáveis não básicas.

Passo 2

A solução corrente não é ótima! Note que um pequeno acréscimo no valor de x_3 invariavelmente aumenta o valor de δ. Mas não podemos aumentar o valor de x_3 ilimitadamente uma vez que isto poderia tornar a solução infactível (outras variáveis poderiam assumir valores negativos). Para que a solução resultante seja factível, as desigualdades abaixo devem ser respeitadas:

$$\left\{ \begin{array}{rcl} x_1 = \frac{5}{2} - \frac{1}{2}x_3 & \geq & 0 \\ w_3 = \frac{1}{2} - \frac{1}{2}x_3 & \geq & 0 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} x_3 \leq 5 \\ x_3 \leq 1 \end{array} \right.$$

Portanto, w_3 deve sair da base para que a variável x_3 possa entrar na base sem violar as restrições. Após substituirmos a equação $x_3 = 1 + 3w_1 + x_2 - 2w_3$ nas equações do

dicionário (6.8), obtemos:

$$\begin{aligned} x_1 &= \tfrac{5}{2} - \tfrac{1}{2}w_1 - \tfrac{3}{3}x_2 - \tfrac{1}{2}(1 + 3w_1 + x_2 - 2w_3) \\ &= 2 - 2w_1 - 2x_2 + w_3 \\ \delta &= \tfrac{25}{2} - \tfrac{5}{2}w - 1 - \tfrac{7}{2}x_2 + \tfrac{1}{2}(1 + 3w_1 - 2w_3) \\ &= 13 - w_1 - 3x_2 - w_3 \end{aligned} \tag{6.9}$$

Substituindo as equações do dicionário (6.8) pelas equações (6.9) obtemos um novo dicionário:

$$\begin{array}{lrcrrrr} Max & \delta & = & 13 & -w_1 & -3x_2 & -w_3 \\ & x_1 & = & 2 & -2w_1 & -2x_2 & +w_3 \\ & w_2 & = & 1 & +2w_1 & +5x_2 & \\ & x_3 & = & 1 & +3w_1 & +x_2 & -2w_3 \end{array} \tag{6.10}$$

cuja base é $\mathbb{B} = \{x_1, x_3, w_2\}$. A solução dada pelo dicionário (6.10) é ótima: $x_1 = 2$, $x_2 = 0$, $x_3 = 1$, $w_1 = 0$, $w_2 = 1$, $w_3 = 0$, e $\delta = 13$ é uma solução ótima pois os coeficientes das variáveis não básicas na equação de δ, no dicionário correspondente dado pelo sistema (6.10), são todos negativos; aumentando o valor de qualquer varíavel não básica reduzirá o valor da função objetivo.

6.3.2 Algoritmo Simplex em detalhes

Nesta seção generalizamos os passos acima ilustrados para o caso geral do problema de programação linear. Considere a forma geral do problema de programação linear:

$$\begin{aligned} &Maximize \quad \sum_{j=1}^{n} c_j x_j \\ &Sujeito\ a: \\ &\qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m \\ &\qquad x_j \geq 0 \qquad\qquad\quad j = 1, \ldots, n \end{aligned} \tag{6.11}$$

Inicialização: Obter Dicinoário Inicial

Inicialmente é necessário introduzir variáveis de folga na formulação (6.11) como segue:

$$\begin{aligned} \delta &= \sum_{j=1}^{n} c_j x_j \\ w_i &= b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, \ldots, m \end{aligned} \tag{6.12}$$

Com o intuito de facilitar os próximos desenvolvimentos, vamos assumir que $w_i = x_{i+n}$ para $i = 1, \ldots, m$, dessa forma gerando o dicionário:

$$\begin{aligned} \delta &= \sum_{j=1}^{n} c_j x_j \\ x_{n+i} &= b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, \ldots, m \end{aligned} \tag{6.13}$$

cuja base é dada por $=\{x_{n+1}, \ldots, x_{n+m}\}$. Seja $\mathcal{B} \subseteq \{1, \ldots, n+m\}$ o conjunto com os índices das variáveis básicas. Seja $\mathcal{N} = \{1, \ldots, n+m\} - \mathcal{B}$ o conjunto com os índices das variáveis não básicas. Inicialmente, vamos assumir que $x_1 = 0, \ldots, x_n = 0$ induz uma solução factível, o que equivale a dizer que $b_i \geq 0$ para $i = 1, \ldots, m$. Durante a aplicação do algoritmo simplex, o dicionário assumirá a forma:

$$\begin{aligned} \delta &= \bar{\delta} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j \\ x_i &= \bar{b}_i - \sum_{j \in N} \bar{a}_{ij} x_j \qquad i \in \mathcal{B} \end{aligned} \tag{6.14}$$

onde $\bar{c}_j$ é conhecido por custo reduzido da variável não básica x_j.

Laço de Iteração

No que segue descrevemos os passos do algoritmo simplex.

i. Escolha um $k \in \{j \in \mathcal{N} : \bar{c}_j > 0\}$. Se não existir nenhum k com tal propriedade, então a solução corrente é ótima. A variável x_k é dita *variável que entra na base*.

ii. A vairável que sairá da base é escolhida de forma que a solução subseqüente continue viável.

$$\begin{aligned} x_i = \bar{b}_i - \bar{a}_{ik} x_k \quad \forall i \in \mathcal{B} \quad &\Rightarrow \quad \bar{b}_i - \bar{a}_{ik} x_k \geq 0 \quad \forall i \in \mathcal{B} \\ &\Rightarrow \quad \bar{a}_{ik} x_k \leq \bar{b}_i \qquad \forall i \in \mathcal{B} \end{aligned}$$

Considerando a última desigualdade, surgem três casos:

(a) Se não existe $\bar{a}_{ik} > 0,\ i \in \mathcal{B}$, então o problema é ilimitado (*unbounded*).

(b) Se existe $\bar{b}_i = 0$ e $\bar{a}_{ik} > 0$ para algum $i \in \mathcal{B}$, então x_i é a variável que sai da base e o pivoteamento é dito *degenerado*. Pivoteamento compreende todos os passos envolvidos em uma mudança de base. O pivoteamento é dito degenerado porque uma variável entrou na base com valor nulo e, conseqüentemente, não aumentou o valor da função objetivo.

(c) Se os dois casos não ocorrem, então a variável que sai da base é $l \in \{i \in \mathcal{B} : \bar{a}_{ik}/\bar{b}_i$ é máximo, $\bar{a}_{ik} > 0$ e $\bar{b}_i > 0\}$. Isto é $l \in \{i \in \mathcal{B} : \bar{b}_i/\bar{a}_{ik}$ é mínimo$\}$.

iii. Uma vez escolhida a variável x_k que entra na base e a variável x_l que sai da base, executamos as operações apropriadas nas linhas do dicionário. A iteração é chamada de pivoteamento. Repetimos a partir do primeiro passo até que uma solução ótima seja atingida.

Não ocorrendo pivotemantos degenerados o algoritmo simplex converge em tempo finito para a solução ótima, ou este detecta que o problema é ilimitado. Na presença de pivoteamentos degenerados o algoritmo simplex pode entrar em ciclos e nunca convergir. Para que isto seja evitado, regras de pivoteamento foram concebidas que garantem convergência em um número exponencial no número de colunas da matriz.

6.3.3 Inicialização

Até este momento, consideramos problemas cujos b_i's são todos não negativos. Isso permitia a obtenção de uma solução viável fazendo $x_j = 0$ para $j = 1, \ldots, n$ e $x_{i+n} = b_i$ para $i = 1, \ldots, m$. *O que devemos fazer se algum b_i é negativo*? Contornamos esta dificuldade ao resolvermos um problema auxiliar, cuja solução inicial é viável e cuja solução ótima é viável para o problema original, isto se este é factível. O problema auxiliar é expresso como segue:

$$\begin{array}{lll} PA: & Maximize & -x_0 \\ & Sujeito\ a: & \\ & & \sum_{j=1}^{n} a_{ij}x_j - x_0 \leq b_i \quad i = 1, \ldots, m \\ & & x_j \geq 0 \qquad\qquad\qquad\quad j = 0, \ldots, n \end{array} \tag{6.15}$$

Não é difícil de ver que o problema (6.11) possui uma solução factível se e somente se a solução ótima do problema auxiliar (6.15) tem valor nulo como valor ótimo da função objetivo. É fácil de se obter uma solução para (6.15): basta definir $x_j = 0$ para $j = 1, \ldots, n$ e tomar um valor suficientemente grande para x_0.

Exemplo

Aqui ilustramos o emprego do problema auxiliar na busca de uma solução factível para um problema de programação linear.

Problema Original Considere como problema de programação linear a formulação:

$$\begin{array}{lrcrcrcr} Maximize & -2x_1 - x_2 & & & & & & \\ Sujeito\ a: & & & & & & & \\ & -x_1 & + & x_2 & \leq & -1 \\ & -x_1 & - & 2x_2 & \leq & -2 \\ & & & x_2 & \leq & 1 \\ & x_1, x_2 \geq 0 & & & & & & \end{array}$$

Problema Auxiliar De acordo com os desenvolvimentos anteriores, o problema auxiliar toma a forma:

$$\begin{array}{lrcrcrcrcr} Maximize & -x_0 & & & & & & & & \\ Sujeito\ a: & & & & & & & & & \\ & -x_1 & + & x_2 & - & x_0 & \leq & -1 \\ & -x_1 & - & 2x_2 & - & x_0 & \leq & -2 \\ & & & x_2 & - & x_0 & \leq & 1 \\ & x_0, x_1, x_2 \geq 0 & & & & & & & & \end{array}$$

Dicionário Inicial Tomando o dicionário inicial como se os elementos b_i's fossem todos não negativos, obtemos:

$$\begin{array}{lrcrcrcr} Max & \delta & = & & & & & -x_0 \\ & w_1 & = & -1 & +x_1 & -x_2 & & +x_0 \\ & w_2 & = & -2 & +x_1 & +2x_2 & & +x_0 \\ & w_3 & = & 1 & & -x_2 & & +x_0 \end{array}$$

Para tornar o dicionário viável, basta executar um pivoteamento com a variável x_0 fazendo a variável básica mais negativa entrar na base. Assim x_0 é a variável que entra na base enquanto que w_2 é a variável que sai da base. As equações a

seguir descrevem a operação de pivoteamento.

$$\begin{aligned} x_0 &= 2 - x_1 - 2x_2 + w_2 \\ \\ w_1 &= -1 + x_1 - x_2 + (2 - x_1 - 2x_2 + w_2) \\ &= 1 - 3x_2 + w_2 \\ \\ w_3 &= 1 - x_2 + x_0 \\ &= 1 - x_2 + (2 - x_1 - 2x_2 + w_2) \\ &= 3 - x_1 - 3x_2 + w_2 \end{aligned} \tag{6.16}$$

Executando o pivoteamento conforme equações (6.16), obtemos o dicionário abaixo:

$$\begin{array}{rlcrrrr} Max & \delta & = & -2 & +x_1 & +2x_2 & -w_2 \\ & w_1 & = & 1 & & -3x_2 & +w_2 \\ & x_0 & = & 2 & -x_1 & -2x_2 & +w_2 \\ & w_3 & = & 3 & -x_1 & -3x_2 & +w_2 \end{array}$$

Segundo Pivoteamento Fazendo x_1 entrar na base, x_0 deve obrigatoriamente deixar a base, gerando o dicionário:

$$\begin{array}{rlcrrrr} Max & \delta & = & 0 & -x_0 & & \\ & w_1 & = & 1 & & -3x_2 & +w_2 \\ & x_1 & = & 2 & -x_0 & -2x_2 & +w_2 \\ & w_3 & = & 1 & +x_0 & -x_2 & \end{array}$$

Note que o dicionário acima é ótimo para o problema auxiliar e, mais ainda, a solução candidata $x_1 = 2$, $x_2 = 0$, $x_3 = 0$, $w_1 = 1$, $w_2 = 0$ e $w_3 = 1$ é factível para o problema original dado que $\delta = 0$.

6.4 Dualidade

Associado a um problema de programação linear P (primal) está outro problema de programação linear chamado de dual e denotado por D. Há várias conseqüências teóricas e práticas resultantes da teoria da dualidade. Por exemplo, qualquer solução para D induz um limite para o valor ótimo da função objetivo de P, o problema primal, e vice-versa.

6.4.1 Motivação

Considere o problema exemplo abaixo:

$$\begin{array}{lll} P: & Maximize & 4x_1 + x_2 + 3x_3 \\ & Sujeito\ a: & \\ & & \begin{array}{lcl} x_1 + 4x_2 & \leq & 1 \\ 3x_1 - x_2 + x_3 & \leq & 3 \\ x_1, x_2, x_3 & \geq & 0 \end{array} \end{array} \tag{6.17}$$

Qualquer solução factível para P induz um limite inferior. Por exemplo, $x' = (1, 0, 0)$ nos diz que o valor ótimo da função objetivo $\delta^* \geq 4$. Usando a solução $x'' = (0, 0, 3)$ descobrimos que $\delta^* \geq 9$. Quão próximo do ótimo estão estes limites inferiores? Vamos multiplicar a primeira restrição de (6.17) por 2, multiplicar a segunda restrição por 3, e depois adicioná-las como segue:

$$\begin{array}{lcl} 2(x_1 + 4x_2) & \leq & 2(1) \\ 3(3x_1 - x_2 + x_3) & \leq & 3(3) \\ \hline 11x_1 + 5x_2 + 3x_3 & \leq & 11 \end{array}$$

Note que $4x_1 + 5x_2 + 3x_3 \leq 11x_1 + 5x_2 + 3x_3 \leq 11$. Assim concluímos que $9 \leq \delta^* \leq 11$. Para obtermos limites superiores mais apertados, utilizamos o mesmo procedimento mas desta vez com variáveis em vez de números específicos. Multiplicamos as restrições por duas variáveis não negativas:

$$\begin{array}{lcl} y_1(x_1 + 4x_2) & \leq & y_1 \\ y_2(3x_1 - x_2 + x_3) & \leq & 3y_2 \\ \hline (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 & \leq & y_1 + 3y_2 \end{array}$$

Agora, estipulamos que:

$$\begin{array}{lcl} y_1 + 3y_2 & \geq & 4 \\ 4y_1 - y_2 & \geq & 1 \\ y_2 & \geq & 3 \\ y_1, y_2 & \geq & 0 \end{array} \tag{6.18}$$

Valores para (y_1, y_2) satisfazendo as desigualdades (6.18) nos levam às seguintes desigualdades:

$$\begin{aligned} \delta &= 4x_1 + x_2 + 3x_3 \\ &\leq (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2 x_3 \\ &\leq y_1 + 3y_2 \end{aligned}$$

Dessa maneira obtemos o limite superior $(y_1 + 3y_2)$ para δ^*. Nossa tarefa é minimizar este limite superior através da solução do problema de programação linear que segue:

$$\begin{aligned} D: \quad & Minimize \quad y_1 + 3y_2 \\ & Sujeito\ a: \\ & \qquad\qquad y_1 + 3y_2 \geq 4 \\ & \qquad\qquad 4y_1 - y_2 \geq 1 \\ & \qquad\qquad y_2 \geq 3 \\ & \qquad\qquad y_1, y_2 \geq 0 \end{aligned}$$

6.4.2 O Problema Dual

Aqui desenvolvemos a maneira através da qual o problema dual pode ser obtido a partir do problema primal. Primeiramente, considere o problema primal que segue:

$$\begin{aligned} P: \quad & Maximize \quad \sum_{j=1}^{n} c_j x_j \\ & Sujeito\ a: \\ & \qquad\qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m \\ & \qquad\qquad x_j \geq 0 \qquad j = 1, \ldots, n \end{aligned}$$

O problema dual associado a P é:

$$\begin{aligned} D: \quad & Minimize \quad \sum_{i=1}^{m} b_i y_i \\ & Sujeito\ a: \\ & \qquad\qquad \sum_{i=1}^{m} a_{ij} y_i \geq c_j \qquad j = 1, \ldots, n \\ & \qquad\qquad y_i \geq 0 \qquad i = 1, \ldots, m \end{aligned}$$

Queremos mostrar que o dual do dual é o problema primal. Podemos expressar o dual como segue:

$$\begin{aligned} D: \quad & Maximize \quad -\sum_{i=1}^{m} b_i y_i \\ & Sujeito\ a: \\ & \sum_{i=1}^{m} -a_{ij} y_i \leq -c_j \qquad j = 1, \ldots, n \\ & y_i \geq 0 \qquad i = 1, \ldots, m \end{aligned}$$

Gerando o dual de D, obtemos:

$$\begin{aligned} DD: \quad & Minimize \quad \sum_{j=1}^{n} -c_j x_j \\ & Sujeito\ a: \\ & \sum_{j=1}^{n} -a_{ij} x_j \geq -b_i \qquad i = 1, \ldots, m \\ & x_j \geq 0 \qquad j = 1, \ldots, n \end{aligned}$$

que por sua vez pode ser colocado na forma:

$$\begin{aligned} DD: \quad & Maximize \quad \sum_{j=1}^{n} c_j x_j \\ & Sujeito\ a: \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m \\ & x_j \geq 0 \qquad j = 1, \ldots, n \end{aligned}$$

Os desenvolvimentos acima demonstram que $DD \equiv P$.

6.4.3 O Teorema Fraco da Dualidade

Teorema 8 *(Dualidade Fraca) Seja* $(x_1, \ldots, x_n)$ *uma solução primal factível e* $(y_1, \ldots, y_m)$ *uma solução dual factível. Então:*

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{i=1}^{m} b_i y_i$$

Prova:

$$\begin{aligned}
\sum_{j=1}^{n} c_j x_j &\leq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j \quad pois\ x_j \geq 0\ e\ \sum_{i=1}^{m} a_{ij} y_i \geq c_j \\
&= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) \\
&\leq \sum_{i=1}^{m} b_i y_i \qquad\qquad pois\ \sum_{j=1}^{n} a_{ij} x_j \leq b_i
\end{aligned}$$

□

Se obtemos uma solução primal $(x_1^*, \ldots, x_n^*)$ e uma dual $(y_1^*, \ldots, y_m^*)$ ambas factíveis, tal que

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

então podemos concluir que ambas as soluções são ótimas para seus respectivos problemas. Para o problema exemplo dado por (6.17) as soluções ótimas são $x^* = (0; 0,25; 3,25)$ e $y^* = (1; 3)$.

6.4.4 O Teorema Forte da Dualidade

O fato de que em programação linear não há folga entre o valor ótimo da função objetivo dos problemas primal e dual é conhecido como Teorema Forte da Dualidade.

Teorema 9 *(Dualidade Forte) Se o problema primal admite uma solução ótima x^* e o dual uma solução ótima y^*, então:*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

Ilustração do Teorema Forte

Abaixo seguem os problemas primal e dual.

$$\begin{array}{lll}
P: & Max & 4x_1 + x_2 + 3x_3 \\
& S.a: & \\
& & x_1 + 4x_2 \leq 1 \\
& & 3x_1 - x_2 + x_3 \leq 3 \\
& & x_1, x_2, x_3 \geq 0
\end{array}
\qquad
\begin{array}{lll}
D: & Min & y_1 + 3y_2 \\
& S.a: & \\
& & y_1 + 3y_2 \geq 4 \\
& & 4y_1 - y_2 \geq 1 \\
& & y_2 \geq 3 \\
& & y_1, y_2 \geq 0
\end{array}$$

Os dicionários iniciais são:

$$\begin{array}{lrcrrrr} Max & \delta & = & 0 & +4x_1 & +x_2 & +3x_3 \\ & w_1 & = & 1 & -x_1 & -4x_2 & \\ & w_2 & = & 3 & -3x_1 & +x_2 & -x_3 \end{array} \qquad \begin{array}{lrcrrr} Max & -\delta & = & 0 & -y_1 & -3y_2 \\ & z_1 & = & -4 & +y_1 & +3y_2 \\ & z_2 & = & -1 & +4y_1 & -y_2 \\ & z_3 & = & -3 & & +y_2 \end{array}$$

Os discionários acima podem ser representados por duas matrizes, as matrizes P e D abaixo:

$$P = \begin{bmatrix} 0 & 4 & 1 & 3 \\ 1 & -1 & -4 & 0 \\ 3 & -3 & 1 & -1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & -1 & -3 \\ -4 & 1 & 3 \\ -1 & 4 & -1 \\ -3 & 0 & 1 \end{bmatrix}$$

Note que $D = -P^T$. Se aplicarmos o método simplex ao problema primal e, ao mesmo tempo, executarmos pivoteamento análogo no dual podemos verificar que a propriedade $D = -P^T$ será mantida. Mais especificamente, considere uma solução primal factível a qual faremos x_3 entrar na base e w_2 a deixar a base. Uma vez que as colunas do primal correspondem às linhas do dual, concluímos que a "coluna x_3" correspondente à "linha z_3". Da mesma forma, a "linha w_2" corresponde à "coluna y_2" no dual. Portanto, a variável y_2 entra na base do dual enquanto que z_3 deixa a base.

Fazendo x_3 entrar na base e w_2 deixar a base, obtemos os dicionários abaixo:

$$\begin{array}{lrcrrrr} Max & \delta & = & 9 & -5x_1 & +4x_2 & -3w_2 \\ & w_1 & = & 1 & -x_1 & -4x_2 & \\ & x_3 & = & 3 & -3x_1 & +x_2 & -w_2 \end{array} \qquad \begin{array}{lrcrrr} Max & -\delta & = & -9 & -y_1 & -3z_3 \\ & z_1 & = & 5 & +y_1 & +3z_3 \\ & z_2 & = & -4 & +4y_1 & -z_3 \\ & y_2 & = & 3 & & +z_3 \end{array}$$

Deixando x_2 entrar na base e removendo w_1, produzimos os discionários abaixo:

$$\begin{array}{llcrrrr} (Primal): & Max & \delta & = & 10 & -6x_1 & -w_1 & -3w_2 \\ & & x_2 & = & 1/4 & -1/4x_1 & -1/4w_1 & \\ & & x_3 & = & 3,25 & -3,25x_1 & -1/4w_1 & -w_2 \end{array}$$

$$\begin{array}{llcrrr} (Dual): & Max & -\delta & = & -10 & -0,25z_2 & -3,25z_3 \\ & & z_1 & = & 6 & +1/4z_2 & +3,25z_3 \\ & & z_2 & = & +1 & +1/4z_2 & +1/4z_3 \\ & & y_2 & = & 3 & & +z_3 \end{array}$$

Ao analisarmos o dicionário do primal, verificamos que a solução é ótima, dado que os

coeficientes da linha da função objetivo são todos negativos. Analisando o dicionário do dual verificamos que este é factível e também ótimo. Para obter o dicionário do primal a partir do dicionário do dual, obtenha a matriz transposta negativa e troque x_j por z_j e w_i por y_i. Observe que enquanto o primal não for ótimo, a solução induzida pelo dicionário do dual é infactível.

6.4.5 Folga Complementar

Teorema 10 *Suponha que $x = (x_1, \ldots, x_n)$ é uma solução primal factível e que $y = (y_1, \ldots, y_n)$ é dual factível. Seja $w = (w_1, \ldots, w_m)$ o vetor com as variáveis de folga do primal e $z = (z_1, \ldots, z_n)$ o vetor com as variáveis de folga do dual. Então x e y são soluções ótimas para seus respectivos problemas se, e somente se,*

$$\begin{aligned} x_j z_j &= 0 \quad para\ j = 1, \ldots, n \\ y_i w_i &= 0 \quad para\ i = 1, \ldots, m. \end{aligned}$$

6.5 Algoritmo Simplex em Notação Matricial

Após inserirmos variáveis de folga sempre que necessário, podemos assumir a seguinte formulação do problema de programação linear:

$$\begin{array}{lrcl} Maximize & c^T x & & \\ Sujeito\ a: & & & \\ & Ax & = & b \\ & x & \geq & 0 \end{array} \tag{6.19}$$

O algoritmo simplex é um processo iterativo, consistindo em escolher variáveis básicas a cada iteração. A Figura 6.3 ilustra as operações de pivoteamento que produzem as iterações $x^0 \rightarrow x^1 \rightarrow \ldots \rightarrow x^*$, tendo como ponto de convergência a solução ótima x^*.

Conforme notação estabelecida acima, $\mathcal{B}$ é o conjunto de índices das variáveis básicas, enquanto $\mathcal{N}$ denota o conjunto dos índices das variáveis não básicas. Assim, cada linha i de $Ax = b$ pode ser escrita como

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \Leftrightarrow \sum_{j \in \mathcal{B}} a_{ij} x_j + \sum_{j \in \mathcal{N}} a_{ij} x_j = b_i.$$

Seja B a submatriz $m \times m$ de $A \in \mathbb{R}^{m \times n}$ correspondente às colunas básicas. Note que B tem posto completo, $rank(B) = m$, daí surge o nome de matriz básica já que esta pode gerar o espaço de todos os possíveis $b \in \mathbb{R}^m$. A dificuldade combinatória de

programação linear está em encontrar m colunas dentre $n > m$ colunas que induzem a solução básica ótima: o número de possíveis combinações de colunas é $n!/m!(n-m)!$. Seja N a submatriz $m \times (n-m)$ de A correspondente às colunas não básicas. Então, podemos escrever $Ax = b$ como $Bx_B + Nx_N = b$, onde x_B é o vetor com as variáveis básicas e x_N, o vetor com as variáveis não básicas. O problema (6.19) pode então ser expresso como:

$$\begin{array}{lrcl} Maximize & c_B^T x_B + c_N^T x_N & & \\ Sujeito\ a: & & & \\ & Bx_B + Nx_N & = & b \\ & x = (x_B, x_N) & \geq & 0 \end{array} \tag{6.20}$$

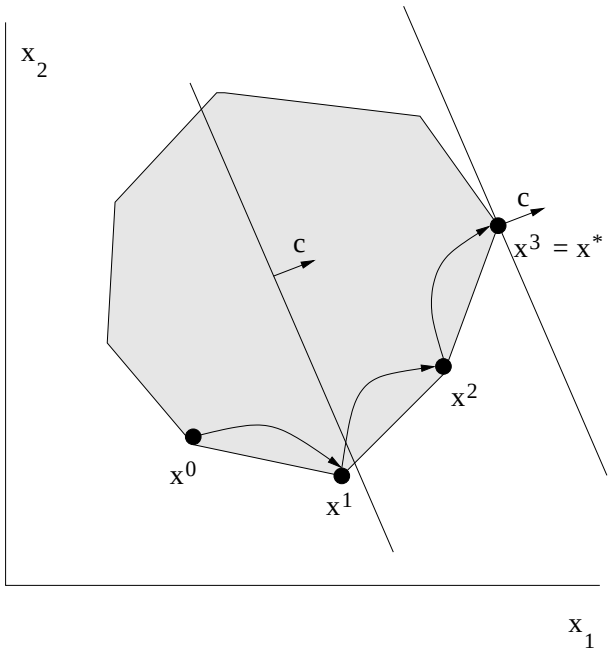


Figura 6.3: Método simplex como um processo iterativo

6.5.1 Dicionário em Forma Matricial

O dicionário tem a propriedade de que as variáveis básicas podem ser escritas como uma função das variáveis não básicas. Portanto,

$$Bx_B + Nx_N = b \Rightarrow x_B = B^{-1}b - B^{-1}Nx_N. \tag{6.21}$$

Substituindo (6.21) na equação da função objetivo, deduzimos que

$$\begin{aligned} \delta &= c_B^T x_B + c_N^T x_N = \\ &= c_B^T \left(B^{-1}b - B^{-1}Nx_N\right) + c_N^T x_N \\ &= c_B^T B^{-1} b + \left(c_N^T - c_B^T B^{-1} N\right) x_N. \end{aligned} \tag{6.22}$$

Se combinarmos (6.21) e (6.22), obtemos o dicionário:

$$\begin{aligned} \delta &= c_B^T B^{-1} b + \left(c_N^T - c_B^T B^{-1} N\right) x_N \\ x_B &= B^{-1}b - B^{-1}Nx_N. \end{aligned}$$

6.6 Exercícios

Ex 6.1 Utilize o método Simplex (executando as operações conforme ilustrado em sala de aula) para resolver o problema abaixo:

$$\begin{aligned} &Maximize && 2x_1 - 6x_2 \\ &Sujeito\ a: \\ &&& -x_1 - x_2 - x_3 \leq -2 \\ &&& 2x_1 - x_2 + x_3 \leq 1 \\ &&& x_1, x_2, x_3 \geq 0 \end{aligned}$$

Ex 6.2 Utilize o método Simplex (no papel ou fazendo uso de pacote de otimização linear) para resolver o problema abaixo:

$$\begin{aligned} &Maximize && 3x_1 + 2x_2 \\ &Sujeito\ a: \\ &&& x_1 - 2x_2 \leq 1 \\ &&& x_1 - x_2 \leq 2 \\ &&& 2x_1 - x_2 \leq 6 \\ &&& x_1 \leq 5 \\ &&& 2x_1 + x_2 \leq 16 \\ &&& x_1 + x_2 \leq 12 \\ &&& x_1 + 2x_2 \leq 21 \\ &&& x_2 \leq 10 \\ &&& x_1, x_2 \geq 0 \end{aligned}$$

Ex 6.3 Mostre em um gráfico a região factível do problema dado em Ex 6.2. Indique a seqüência de soluções básicas produzida pelo método Simplex.

Ex 6.4 Dê um exemplo mostrando que uma variável que se torna básica em uma iteração do Simplex pode se tornar não-básica na iteração seguinte.

Ex 6.5 Mostre que uma variável que se torna não-básica em uma iteração do Simplex não pode tornar-se básica na iteração seguinte.

Ex 6.6 Encontre o dual do seguinte problema de programação linear:

$$\begin{array}{lll} P: & Maximize & c^T x \\ & Sujeito\ a: & \\ & & a \leq Ax \leq b \\ & & l \leq x \leq u \\ & & x \geq 0 \end{array}$$

Ex 6.7 Ilustre o Teorema Forte da Dualidade no problema dado no exercício EX 6.1. (Se necessário, utilize o pacote de programação linear "lp_solve" disponível no diretório /users/professores/camponog/lp_solve_1.4)

Capítulo 7

Teoria dos Jogos

Aqui, nos concentraremos em jogos envolvendo dois agentes competidores, onde o ganho de um corresponde à perda do outro (*finite two-person zero-sum games*).

7.1 Jogos Matriciais

Nestes jogos, cada agente seleciona uma ação dentre um número finito de possibilidades que não depende da escolha do agente oponente. O agente 1, denotado por agente linha, escolhe a ação i enquanto que o agente 2, denotado por agente coluna, escolhe a ação j. Para o par de ações (i, j), o agente 1 paga a_{ij} unidades ao agente 2 se $a_{ij} > 0$; caso contrário, se $a_{ij} < 0$, o agente 2 paga $-a_{ij}$ unidades ao agente 1. Assim o jogo matricial é representado por uma matriz $A \in \mathbb{R}^{m \times n}$ onde on agente linha pode escolher dentre m ações e o agente coluna tem n ações.

7.1.1 O Jogo da Tesoura, Pedra e Papel

Conforme matriz dada em (7.1), cada iteração do jogo tem como resultado vitória para um agente (e derrota para o outro agente) ou empate. Neste jogo, nenhum agente possui uma estratégia determinística vencedora: se um agente sempre escolhe uma alternativa, então o outro agente pode selecionar a opção que o leva à vitória. Portanto, os agentes devem randomizar suas estratégias mas, em virtude da simetria do problema, cada agente deve selecionar uma das alternativas com probabilidade 1/3. Por exemplo, se o agente linha seleciona *papel* com probabilidade 1/2 e escolhe *pedra* e *tesoura* com probabilidade 1/4, então se o agente coluna selecionar *tesoura* com probabilidade 1 ele vencerá em 50% das vezes, empatará em 25% das vezes e será derrotado em apenas

25% das vezes.

$$A = \begin{array}{c} \\ Papel \\ Tesoura \\ Pedra \end{array} \begin{array}{c} \begin{array}{ccc} Papel & Tesoura & Pedra \end{array} \\ \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} \end{array} \tag{7.1}$$

7.1.2 Um Jogo Menos Trivial

Considere um jogo cuja matriz de ganhos e perdas é definida como segue:

$$A = \begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix}$$

Este jogo tem a propriedade de que qualquer estratégia fixa pode ser explorada pelo agente oponente. Por exemplo, se o agente linha escolha a primeira linha em todas as jogadas, então o agente coluna pode escolher a segunda coluna obtendo um ganho de 1 unidade a cada iteração. Novamente, os agentes devem randomizar suas estratégias. Mas agora a probabilidade uniforme de 1/3 não é mais ótima. Além disso, qual dos agentes tem vantagem neste jogo? Note que

$$\sum_{i,j:a_{ij}>0} a_{ij} = 10$$
$$-\sum_{i,j:a_{ij}<0} a_{ij} = 11$$

o que nos leva a suspeitar que o agente linha tem vantagem neste jogo.

7.2 Formalização

Se o agente linha segue uma estratégia randomizada, então o agente coluna observa que a probabilidade do agente linha escolher a opção i é y_i. O vetor de probabilidades $y \in \mathbb{R}^m$ do agente linha é chamado de vetor estocástico, o qual satisfaz as propriedades

$$y \geq 0,\ e^T y = 1$$

onde e é um vetor com todas as entradas iguais a 1. Se a estratégia do agente coluna for modelada por um vetor estocástico x, então o ganho esperado do agente coluna é

$$\sum_{i=1}^{m}\sum_{j=1}^{n} y_i a_{ij} x_j = y^T A x.$$

7.3 Estratégia Ótima para o Agente Coluna

Suponha que o agente coluna decide jogar de acordo com a estratégia x. Nesta situação, a melhor defesa do agente linha é a estratégia y^* que resolve o seguinte problema:

$$\begin{array}{rcll} y^* & = & Argmin & y^T A x \\ & & y \in \mathbb{R}^m & \\ & & Sujeito\ a: & \\ & & & e^T y = 1 \\ & & & y \geq 0 \end{array} \tag{7.2}$$

A partir do Teorema Fundamental da Programção Linear, sabemos que existe pelo menos uma solução básica ótima. Por exemplo, suponha que $x = (1/3, 1/3, 1/3)$. Então

$$\begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix} x = \begin{bmatrix} -1/3 \\ 1/3 \\ -1/3 \end{bmatrix}$$

Assim a melhor estratégia para o agente linha é $y' = (1, 0, 0)$, $y'' = (0, 0, 1)$, ou qualquer combinação convexa de y' e y''. Uma vez que dado x o agente linha escolhe uma estratégia y que produz o mínimo de (7.2), então o agente coluna escolhe a estratégia x^* que produz o máximo do seguinte problema:

$$\begin{array}{rcllll} x^* & = & Argmax & Min & y^T A x & \\ & & x & y & & \\ & & & S.a: & & \\ & & & & e^T x = 1 & \\ & & & & e^T y = 1 & \\ & & & & x, y \geq 0 & \end{array} \tag{7.3}$$

Enquanto que o problema (7.2) pode ser resolvido através de programação linear, o problema (7.3) envolve dois operadores de otimização. De que forma poderíamos resolver (7.3)? O problema (7.3) pode ser resolvido por meio de programação linear. Lembre que já verificamos como que o problema interno (*Min*) pode ser resolvido por

meio de uma estratégia determística

$$\underset{x}{Max}\ \underset{y}{Min}\ y^T Ax = \underset{x}{Max}\ Min\ \{e_1^T Ax, e_2^T Ax, \ldots, e_m^T Ax\} \tag{7.4}$$

onde e_i é um vetor com todas as entradas iguais a zero, com exceção da i-ésima entrada. O problema (7.4) pode ser expresso em programação linear como segue:

$$\begin{array}{lll} P_x: \ Max & z & \\ \quad S.a: & & \\ & z \leq e_1^T Ax & \equiv \\ & \vdots & \\ & z \leq e_m^T Ax & \\ & e^T x = 1 & \\ & x \geq 0 & \\ & z \text{ irrestrito em sinal} & \end{array} \qquad \begin{array}{ll} Max & z \\ S.a: & \\ & ez \leq Ax \\ & e^T x = 1 \\ & x \geq 0 \\ & z \text{ irrestrito em sinal} \end{array}$$

O problema acima pode se colocado em uma forma matricial:

$$\begin{array}{lll} P_x: \ Max & \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} & \\ \quad S.a: & & \\ & \begin{bmatrix} -A & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \leq = \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \\ & x \geq 0 & \\ & z \text{ irrestrito em sinal} & \end{array} \tag{7.5}$$

Os desenvolvimentos acima mostram que a estratégia ótima x^* do agente coluna é precisamente a solução do problema P_x.

7.4 Estratégia Ótima para o Agente Linha

Por simetria, pode ser verificado que o agente linha procura uma estratégia y^* que produz a solução ótima do seguinte problema min-max:

$$\begin{array}{lllll} Min & Max & y^T Ax & \equiv & Min \quad Max \quad \{y^T Ae_1, y^T Ae_2, \ldots, y^T Ae_n\} \\ y & x & & & y \\ & S.a: & & & S.a: \\ & & e^T y = 1 & & e^T y = 1 \\ & & e^T x = 1 & & y \geq 0 \\ & & x, y \geq 0 & & \end{array} \tag{7.6}$$

O problema (7.6) pode ser expresso em programção linear como segue:

$$\begin{array}{lllll} P_y: & Min & w & & Min \quad w \\ & S.a: & & & S.a: \\ & & w \geq y^T Ae_1 & \equiv & ew \geq A^T y \\ & & \vdots & & e^T y = 1 \\ & & w \geq y^T Ae_n & & y \geq 0 \\ & & e^T y = 1 & & w \text{ irrestrito em sinal} \\ & & y \geq 0 & & \\ & & w \text{ irrestrito em sinal} & & \end{array}$$

Da mesma forma que para o agente coluna, podemos expressar o problema de encontrar uma estratégia ótima em um forma matricial:

$$\begin{array}{lll} P_y: & Min & \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} \\ & S.a: & \\ & & \begin{bmatrix} -A^T & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} \begin{matrix} \geq \\ = \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ & & y \geq 0 \\ & & w \text{ irrestrito em sinal} \end{array} \tag{7.7}$$

7.5 Relação entre os Problema P_x e P_y

Considere o problema P_x na sua forma matricial (7.5). Vamos obter o problema dual de P_x introduzindo as variáveis duais y (para as linhas da matriz A) e a variável w para a última restrição de P_x, conforme desenvolvimento abaixo:

$$
\begin{aligned}
P_x: \quad & Max \quad \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \\
& S.a: \\
& \qquad (y) \quad \begin{bmatrix} -A & e \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \leq \begin{bmatrix} 0 \end{bmatrix} \\
& \qquad (w) \quad \begin{bmatrix} e^T & 0 \end{bmatrix} \begin{bmatrix} z \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \\
& \qquad\qquad x \geq 0 \\
& \qquad\qquad z \text{ irrestrito.}
\end{aligned}
\tag{7.8}
$$

O dual do problema (7.8) é:

$$
\begin{aligned}
\bar{P}_x: \quad & Min \quad \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} \\
& S.a: \\
& \qquad \begin{bmatrix} y^T & w^T \end{bmatrix} \begin{bmatrix} -A & e \\ e^T & 0 \end{bmatrix} \geq = \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T \\
& \qquad y \geq 0 \\
& \qquad w \text{ irrestrito}
\end{aligned}
\tag{7.9}
$$

que pode ser manipulado de forma a se obter:

$$
\begin{aligned}
\bar{P}_x: \quad & Min \quad \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} \\
& S.a: \\
& \qquad \begin{bmatrix} -A^T & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} \geq \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\
& \qquad y \geq 0 \\
& \qquad w \text{ irrestrito}
\end{aligned}
\tag{7.10}
$$

Assim concluímos que o dual de P_x corresponde ao problema P_y, ou seja, $\bar{P}_x \equiv P_y$. Ao resolver seu problema P_x, o agente coluna obtém sua estratégia ótima x^* e também a estratégia ótima do agente linha y^*.

7.6 Teorema Minimax

Inicialmente demonstrado por John Von Neumann em 1928, pode ser facilmente demonstrado através da Teoria da Dualidade Linear.

Teorema 11 *Existem vetores estocásticos x^* e y^* para os quais*

$$\underset{x}{Max} \quad y^{*T}Ax \quad \equiv \quad \underset{y}{Min} \quad y^{T}Ax^*.$$

O valor ótimo $z^* = w^*$ dos problemas primal e dual são chamados de valores ótimos do jogo. Ao adotar a estratégia y^*, o agente linha garante que não perderá mais do que w^* unidades em média. Similarmente, o agente coluna assegura que receberá pelo menos z^* unidades em média se adotar a estratégia x^*.

Um jogo com valor $z^* = w^* = 0$ é dito justo. Jogos cujos papéis dos agentes podem ser invertidos são ditos jogos simétricos. Esses jogos são justo e caracterizados por $a_{ij} = -a_{ji}$ para todo i, j, isto é, $A = -A^T$.

7.7 Exercícios

EX 7.1 Primeiramente considere o jogo matricial dado pela matriz:

$$A = \begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix}.$$

Encontre o vetor estocástico com a política de decisão do agente coluna resolvendo o problema P_x com um pacote de otimização linear tal como `lp_solve`. Encontre a solução dual de P_x diretamente a partir da solução de P_x. Repita os passos anteriores para o agente linha. O jogo dado por A é justo?

Ex 7.2 Dois jogadores, A e B, escondem independentemente uma moeda de 5 ou 10 centavos. Se as moedas escondidas possuem a mesma denominação, então o jogador A recebe ambas; se as moedas são de denominação diferente, então o jogador B recebe ambas.

- Encontre as estratégias ótimas.
- Qual dos jogadores tem vantagem?
- Resolva o problema para quaisquer denominações a e b.

Ex 7.3 Dois jogadores, A e B, escolhem independentemente um número entre 1 e 100. Os jogadores empatam se eles escolhem o mesmo número. De outra forma, o jogador que escolheu o menor número ganha, digamos x, a menos que o oponente tenha escolhido precisamente o número $x + 1$. Encontre a estratégia ótima para

os jogadores. (Sugestão: utilize um pacote de programação linear, tal como "lp_solve").

Capítulo 8

Fluxo em Redes

Neste capítulo nos concentraremos em problemas de programação linear que apresentam uma estrutura de redes, ou seja, as variáveis advém de valores a serem definidos para os arcos de um grafo direcionado. Na literatura existem algoritmos específicos para resolver problemas desta classe de uma forma mais eficiente. O problema mais geral desta classe é o problema de fluxo em redes de custo mínimo, para o qual uma variedade de problemas práticos podem ser reduzidos e resolvidos eficientemente.

8.1 Dois Problemas Clássicos

No que segue apresentamos dois problemas clássicos que podem ser reduzidos ao problema de fluxo em redes de custo mínimo.

8.1.1 O Problema de Transportes

Neste problema nos são dados um conjunto de fornecedores e um conjunto de clientes. Os fornecedores têm suas capacidades de produção limitadas, já os clientes possuem demandas a serem supridas. O custo de transporte do depósito do fornecedor s_i para o cliente t_j é dado por c_{ij}. Desejamos então encontrar a quantidade a ser produzida por cada fornecedor e as quantidades a serem enviadas aos clientes, de maneira que o custo total seja o menor possível ao mesmo tempo que as restrições sejam atendidas. Os dados de uma instância do problema são:

- $S = \{s_1, \ldots, s_m\}$ é o conjunto de fornecedores;
- $T = \{t_1, \ldots, t_n\}$ é o conjunto de clientes;
- u_i é a capacidade de produção do fornecedor s_i;

- d_j é a demanda do cliente t_j; e

- c_{ij} é o custo unitário de transporte do fornecedor s_i para o cliente t_j.

O problema de encontrar a alocação de produção e transporte entre fornecedores e clientes que minimiza o custo agregado de transporte (problema de transporte) pode ser formulado em programação matemática:

$$
\begin{aligned}
&\textit{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
&\textit{Sujeito a}: \\
&\qquad \sum_{i=1}^{m} x_{ij} = d_j \qquad\qquad j = 1,\ldots,n \\
&\qquad \sum_{j=1}^{n} x_{ij} \leq u_i \qquad\qquad i = 1,\ldots,m \\
&\qquad x_{ij} \geq 0 \qquad\qquad i = 1,\ldots,m \\
&\qquad\qquad\qquad\qquad\quad j = 1,\ldots,n
\end{aligned}
$$

onde x_{ij} é a quantidade produzida/enviada pelo fornecedor s_i ao cliente t_j. Uma instância exemplo do problema de transporte é dada nas Tabelas 8.1, 8.2 e 8.3.

Tabela 8.1: Capacidade de Produção dos Fornecedores

Fornecedores	s_1	s_2	s_3
Capacidade	135	56	93

Tabela 8.2: Demanda dos Clientes

Cliente	t_1	t_2	t_3	t_4
Demanda	62	83	39	91

8.1.2 O Problema de Alocação

O problema e alocação (*assignment problem*) consiste em designar n pessoas a n tarefas com vistas a maximizar algum critério de competência/aptidão. Seja c_{ij} o nível de competência/aptidão da pessoa i ao executar a tarefa j. Podemos então expressar

Tabela 8.3: Custo Unitário de Transporte

	Fornecedores		
Cliente	s_1	s_2	s_3
t_1	132	85	106
t_2	$+\infty$	91	89
t_3	97	$+\infty$	100
t_4	103	$+\infty$	98

o problema em programação matemática:

$$
\begin{aligned}
&Maximize \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
&Sujeito\ a: \\
&\qquad \sum_{i=1}^{n} x_{ij} = 1 \qquad\qquad j = 1,\ldots,n \\
&\qquad \sum_{j=1}^{n} x_{ij} = 1 \qquad\qquad i = 1,\ldots,n \\
&\qquad x_{ij} \in \{0,1\} \qquad\quad i = 1,\ldots,n \\
&\qquad\qquad\qquad\qquad\quad j = 1,\ldots,n
\end{aligned}
$$

onde x_{ij} assume o valor 1 se a pessoa i é designada à tarefa j e 0 caso contrário

8.2 O Problema de Fluxo em Redes de Custo Mínimo

Um universo amplo de problemas teóricos e práticos podem ser transformados em problemas de fluxo em redes. Os dois problemas acima apresentados são dois exemplos. Os dados que formam uma instância do problem são:

- um grafo direcionado $G = (N, A)$ consistindo em um conjunto de vértices (ou nós) e um conjunto de arcos;
- o custo unitário c_{ij} de transporte através do arco (i, j);
- o limite inferior l_{ij} e superior u_{ij} para fluxo através do arco (i, j); e
- a quantidade de fluxo $b(i)$ que é injetada ou consumida pelo nó i; se $b(i) > 0$, então i é um nó fornecedor; se $b(i) < 0$, então i é um nó consumidor; e se $b(i) = 0$, então i é um nó de transporte.

Em programação matemática, o problema de fluxo em redes de custo mínimo é expresso como:

$$
\begin{aligned}
&\textit{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \\
&\textit{Sujeito a}: \\
&\qquad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \qquad \textit{para todo } i \in N \\
&\qquad l_{ij} \leq x_{ij} \leq u_{ij} \qquad \textit{para todo } (i,j) \in A
\end{aligned}
$$

As seguintes classes de problemas podem ser convertidas em problemas de fluxo em redes (de custo mínimo):

- fluxo máximo em grafos;
- problema de caminhos mínimos em grafos;
- problema de alocação;
- problema de transporte; e
- fluxo em redes com função custo convexa (através de linearização por partes).

8.3 Transformações

Há um conjunto de transformações que tipicamente são úteis quando se trabalha com problemas de fluxo em redes. Por exemplo, é possível converter um problema de fluxo em redes cujos arcos são não direcionados em outro problema de fluxo em redes cujos arcos são direcionados. Abaixo apresentamos algumas transformações que serão apresentadas por meio de figuras.

8.3.1 Convertendo Arestas em Arcos

Considere o arco (i, j) da rede ilustrada na Figure 8.1. Assumindo que $l_{ij} = 0$ e $c_{ij} > 0$, é fácil verificar que uma solução ótima do problema de fluxo em redes terá necessariamente fluxo em uma direção ou outra, mas não em ambas. Assim podemos transformar a aresta não direcionada em dois arcos com sentidos opostos, cada um deles com o custo e capacidade da aresta.

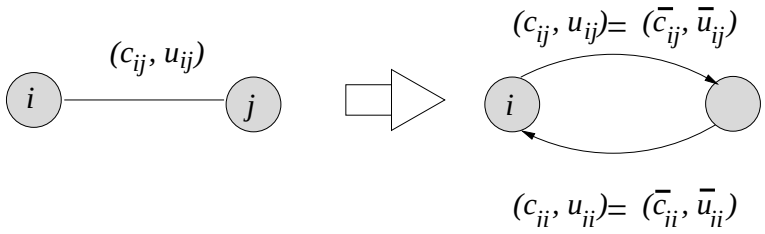


Figura 8.1: Transformação de Arcos em Arestas

8.3.2 Removendo Limites Inferiores

Para eliminarmos o limite inferior de um arco (i, j), basta enviar precisamente a quantidade l_{ij} através do arco, reduzir o suprimento do nó i de l_{ij} unidades, $b(i) \rightarrow b(i) - l_{ij}$, aumentar em l_{ij} unidades a quantidade de fluxo que deve emanar do nó j, $b(j) \rightarrow b(j) + l_{ij}$, e finalmente reduzir a capacidade do arco (i, j) em l_{ij} unidades. Note que o custo final deve ser acrescido de $l_{ij}c_{ij}$ de forma a refletir o envio antecipado do fluxo. O fluxo na rede original, x_{ij}, e o fluxo na rede modificada, $\bar{x}_{ij}$, obedecem a relação $x_{ij} = \bar{x}_{ij} + l_{ij}$.

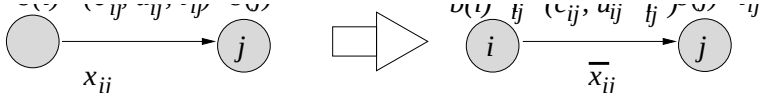


Figura 8.2: Transformação de Arcos em Arestas

8.3.3 Removendo Capacidade de Arcos

A Figura 8.3 ilustra como que um arco com capacidade pode ser substituído por dois arcos sem capacidade por meio da inserção de um nó auxiliar. É fácil verificar que o fluxo $\bar{x}_{ik}$ através do arco (i, k) da rede modificada corresponde precisamente ao fluxo x_{ij} através do arco (i, j) da rede original. Note que os fluxos satisfazem a relação $x_{ik} + s_{jk} = u_{ij}$, portanto s_{jk} é o resíduo de capacidade de transmissão através do arco (i, j).

8.4 Um exemplo

Na Figura 8.4 é ilustrado um problema de fluxo em redes, consistindo em enviar 10 unidades de fluxo do nó 1 (produtor) para o nó 4 (consumidor).

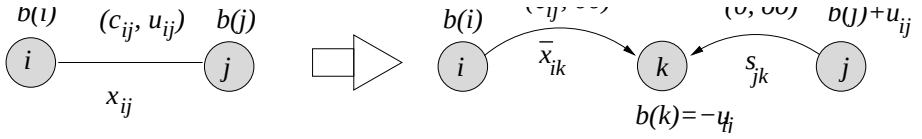


Figura 8.3: Eliminando Capacidade dos Arcos

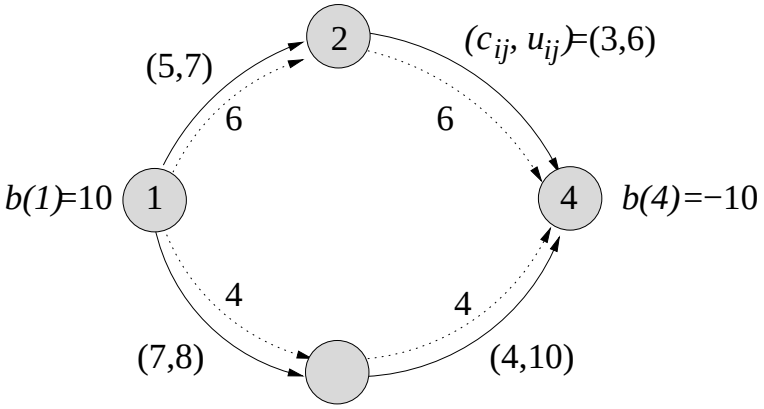


Figura 8.4: Exemplo de problema de fluxo em redes

O problema pode ser expresso de uma forma mais compacta:

$$\begin{array}{ll} Min & c^T x \\ s.a: & Nx = b \\ & l \leq x \leq u \\ & x \in \mathbb{R}^{|}A| \end{array}$$

onde c é vetor de custo unitário de transporte, N é a matriz de incidência do grafo, b é o vetor com as taxas de injeção e consumo de fluxo, l é o vetor com limites inferiores, u é o vetor de limites superiores, e x é o vetor com os fluxos dos arcos. A solução ótima está indicada na figura conforme arcos tracejados. Para o problema dado na figura acima, os dados são:

$$N = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} (1,2) & (1,3) & (2,4) & (3,4) \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & -1 \end{bmatrix} \end{array}$$

$$b = \begin{bmatrix} 10 \\ 0 \\ 0 \\ -10 \end{bmatrix} \quad l = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad u = \begin{bmatrix} 7 \\ 8 \\ 6 \\ 10 \end{bmatrix} \quad c = \begin{bmatrix} 5 \\ 7 \\ 3 \\ 4 \end{bmatrix} \quad x = \begin{bmatrix} x_{12} \\ x_{13} \\ x_{24} \\ x_{34} \end{bmatrix}$$

8.5 Redes Residuais

Dados uma rede $G = (N, A)$ e um fluxo $x^0 \in \mathbb{R}_+^{|A|}$, a rede residual denotada por $G(x^0)$ descreve a capacidade incremental de fluxo relativa ao fluxo x^0 já transportado através da rede. Considere o arco (i, j) e o fluxo inicial x_{ij}^0. Há duas possibilidades:

- podemos enviar uma quantidade adicional $u_{ij} - x_{ij}^0$ de fluxo através do arco (i, j), incorrendo um custo unitário de c_{ij}; e
- podemos enviar de volta uma quantiade x_{ij}^0 através do arco (j, i), a custo $-c_{ij}$ por unidade.

Para uma rede G com fluxo x^0, a rede residual $G(x^0)$ é ilustrada na Figura 8.5.

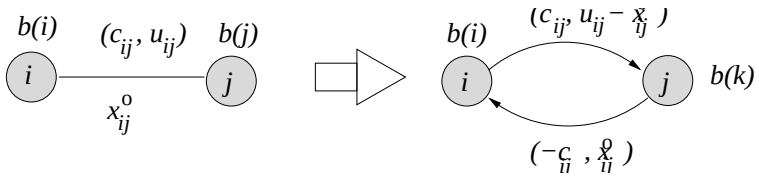


Figura 8.5: Exemplo de rede residual

Os passos para computo da rede residual $G(x^0)$ dado um fluxo inicial x^0 são:

i. substitua cada arco original (i, j) por dois arcos: (i, j) e (j, i);

ii. o arco (i, j) tem custo c_{ij} e capacidade residual $r_{ij} = u_{ij} - x_{ij}^0$;

iii. o arco (j, i) tem custo $-c_{ij}$ e capacidade residual $r_{ij} = x_{ij}^0$.

Proposição 1 *Um fluxo x é factível para a rede G se e somente se o fluxo correspondente $\bar{x}$, definido por $\bar{x} = \bar{x}_{ij} - \bar{x}_{ji} = x_{ij} - x_{ij}^0$, é factível na rede residual $G(x^0)$. Além disso, $c^T x = \bar{c}^T \bar{x} + c^T x^0$.*

Na Figura 8.6 é ilustrada uma rede G com fluxo x e sua respectiva rede residual $G(x)$.

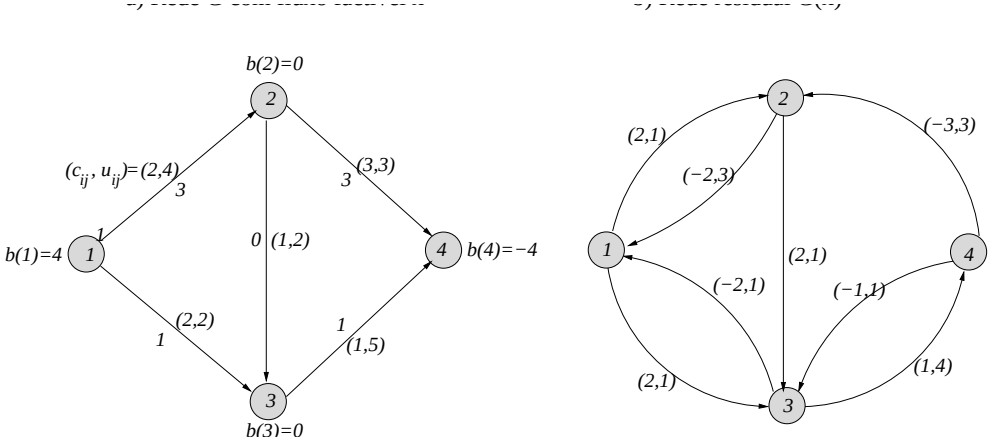


Figura 8.6: Exemplo de rede residual

8.6 Algoritmo de Cancelamento de Circuitos Negativos

Teorema 12 *(Condição de Otimalidade) Um fluxo factível x^* induz uma solução ótima para o problema de fluxo em redes de custo negativo se, e somente se, a rede residual $G(x^*)$ não contém nenhum circuito de custo negativo.*

Algoritmo: Cancelamento de Circuitos Negativos
`Obtenha um fluxo viável` x
`Enquanto` $G(x)$ `contém um circuito negativo faça`
  `Utilize um algoritmo para encontrar circuito negativo` w
  $\delta = min\{r_{ij} : (i,j) \in w\}$
  `Aumente o fluxo a longo do circuito` w `de` δ `unidades`
    `e atualize` $G(x)$
`Fim-enquanto`

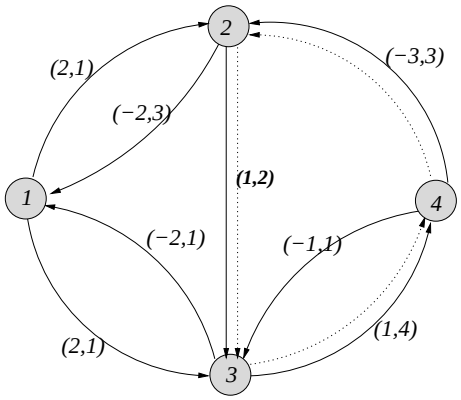
8.6.1 Complexidade do Algoritmo

Um limite superior no número de operações elementares executadas pelo algoritmo pode ser facilmente estabelecido. Seja $C = Max\{c_{ij} : (i,j) \in A\}$ assumindo que $c_{ij} \geq 0$ e seja $U = Max\{u_{ij} : (i,j) \in A\}$. Então mCU é um limite superior para o custo do fluxo inicial, onde $|A| = m$. Além disso, zero é um limite inferior para o custo do fluxo ótimo. Portanto, o algoritmo termina em $O(mcU)$ iterações. Se utilizarmos um algoritmo com tempo de execução $O(nm)$ para detectar circuitos negativos, verificamos que o tempo de execução do algoritmo de cancelamento de circuitos negativos é

$O(nm^2CU)$. Podemos dizer que o algoritmo é de tempo polinomial?

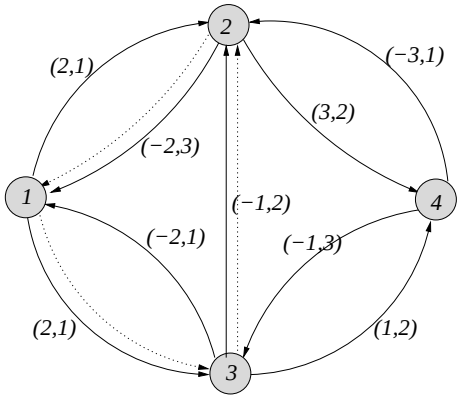
8.6.2 Exemplo de Execução do Algoritmo

As Figuras 8.7, 8.8, 8.9, e 8.10 ilustram a dinâmica do algoritmo.



w =<(4,2), (2,3), (3,4)>
c(w) = −1
δ(w) = 2

Figura 8.7: Primeira iteração do algoritmo de cancelamento de circuito negativo



w =<(1,3), (3,2), (2,1)>
c(w) = −1
δ(w) = 1

Figura 8.8: Segunda iteração do algoritmo de cancelamento de circuito negativo

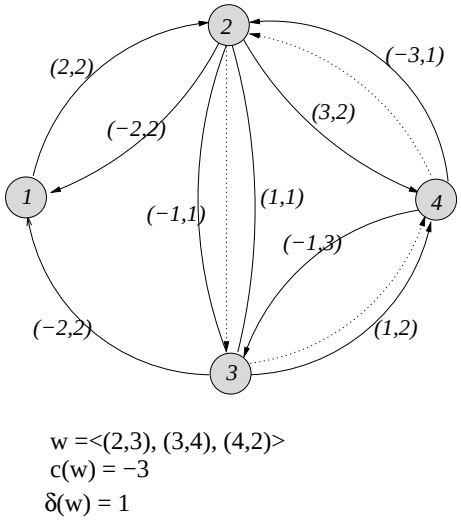


Figura 8.9: Terceira iteração do algoritmo de cancelamento de circuito negativo

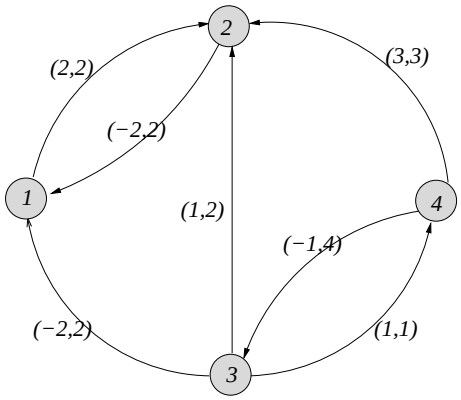


Figura 8.10: Fluxo ótimo

8.7 Matrizes Totalmente Unimodulares

Considere o problema inteiro abaixo:

$$\begin{aligned} IP: \quad & Max \quad c^T x \\ & S.a: \\ & \qquad Ax \leq b \\ & \qquad x \in \mathbb{Z}^n_+ \end{aligned}$$

cujos dados (A, b) são inteiros. Sob quais condições podemos esperar que a relaxação linear LP de IP, $\{max\ c^T x : Ax \leq b,\ x \in \mathbb{R}^n_+\}$, terá solução inteira? A partir da teoria

da programação linear, sabemos que as soluções básicas têm form:

$$x = (x_B, x_N) = (B^{-1}b, 0)$$

onde B é uma submatriz $m \times m$ não-singular da matriz (A, I) e I é uma matriz identidade de dimensão $m \times m$.

Proposição 2 *(Condição Suficiente) Se a base ótima B tem determinante $det(B) = +/-1$, então a relaxação linear resolve IP.*

Prova: Pela regra de Cramer, $B^{-1} = \frac{B^*}{det(B)}$ onde B^* é a matriz adjunta. As entradas de B^* são produtos dos termos de B, portanto, B^* é uma matriz de inteiros, e $det(B) = +/-1$, o que nos leva a concluir que $B^{-1}b$ é inteiro para qualquer b inteiro. □

Definição 2 *Uma matriz A é totalmente unimodular (TU) se toda a submatriz quadrada de A tem determinante -1, $+1$ ou 0.*

Dois exemplos de matrizes que não são TU:

$$A_1 = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

pois $det(A_1) = 2$ e $det(A_2) = 2$. Dois exemplos matrizes TU são:

$$A_1 = \begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad A_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Note que se A é TU então $a_{ij} \in \{+1, -1, 0\}$ para todo i, j.

Proposição 3 *Uma matriz A é TU se e somente se:*

- *a matriz transposta A^T é TU; e*
- *a matriz (A, I) é TU.*

A condição suficiente dada pela proposição a seguir é simples e nos permite verificar que uma das matrizes acima é TU.

Proposição 4 *(Condição suficiente) A matriz A é TU se:*

i. $a_{ij} \in \{+1, -1, 0\}$ *para todo* i, j*;*

ii. cada coluna contém no máximo dois coeficientes não nulos, $\sum_{i=1}^{m} |a_{ij}| \leq 2$*; e*

iii. existe uma partição (M_1, M_2) *do conjunto* M *das linhas tal que cada coluna* j *que contenha dois coeficientes não nulos satisfaz* $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$.

Prova: Suponha que A não é TU e seja B a menor submatriz quadrada de B para a qual $det(B) \notin \{0, 1, -1\}$. B não pode conter uma coluna com uma única entrada não nula, pois B não seria mínima. Portanto, B contém duas entradas não nulas em cada coluna. Pela condição (iii), adicionamos as linhas de M_1 e subtraímos as linhas de M_2 de forma que obtemos o vetor nulo e, portanto, $det(B) = 0$, o que constitui uma contradição. □

Retornando à questão de IP, fica claro que quando A é TU a relaxação linear LP de IP produz a solução ótima.

Proposição 5 *O programa linear* $max\{c^T x : Ax \leq b, x \in \mathbb{R}^n_+\}\}$ *tem solução ótima inteira para um vetor b inteiro, para o qual existe uma solução ótima de valor finito, se e somente se A é TU.*

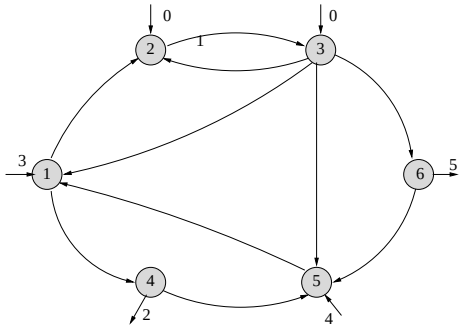


Figura 8.11: Propriedade de unimodularidade das matrizes de incidência de grafos

A Figura 8.11 ilustra uma rede cuja matriz de incidência é dada abaixo.

8.8 Exercícios

EX 8.1 Obtenha o problema dual do problema de fluxo em redes.

	x_{12}	x_{14}	x_{23}	x_{31}	x_{32}	x_{35}	x_{36}	x_{45}	x_{51}	x_{53}	x_{65}	$b(i)$	
1	1	1	0	-1	0	0	0	0	-1	0	0	=	3
2	-1	0	1	0	-1	0	0	0	0	0	0	=	0
3	0	0	-1	1	1	1	1	0	0	-1	0	=	0
4	0	-1	0	0	0	0	0	1	0	0	0	=	-2
5	0	0	0	0	0	-1	0	-1	1	1	-1	=	4
6	0	0	0	0	0	0	-1	0	0	0	1	=	-5

EX 8.2 Formule o caminho de caminhos mínimos de um vértice s para um vértice t como um problema de fluxo em redes. Formule o problema de encontrar a árvore de caminhos mínimos a partir de um v'ertice s para os demais vértices do grafo.

EX 8.3 Aplique o algoritmo de cancelamento de circuito negativo ao problema de fluxo em rede de custo mínimo especificado na Figura 8.12. (Encontre um fluxo viável x_0, obtenha a rede residual $G(x_0)$ e depois obtenha fluxos $x_1, x_2, \ldots, x^*$ até que a rede residual $G(x^*)$ não contenha nenhum circuito com custo negativo.)

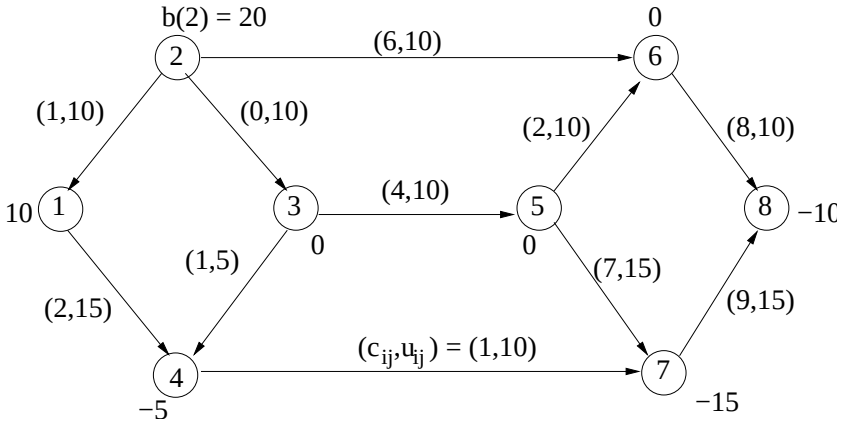


Figura 8.12: Grafo $G = (N, A)$ com a especificação do problema de fluxo em rede

EX 8.4 Demonstre a verdade ou falsidade da seguinte afirmação: suponha que todas as demandas e suprimentos (i.e., os valores de $b(i)$) bem como as capacidades dos arcos são valores inteiros e pares, em um problema de fluxo em rede de custo mínimo. Então, existe um fluxo ótimo x^* onde cada fluxo $x^*_{i,j}$ é um número par.

EX 8.5 O conjunto de soluções ótimas para um problema qualquer de fluxo em redes, de custo mínimo, se altera se multiplicarmos o custo de cada arco por uma constante k? O conjunto se altera se adicionarmos uma constante k?

EX 8.6 Em um problema de fluxo em redes, suponha que além das capacidades dos arcos os nós também apresentam limites superiores para o fluxo de entrada. Seja $w(i)$ o fluxo máximo de entrada para o nó i, $i \in N$. Como que você resolveria esta generalização do problema de fluxo em redes?

EX 8.7 O problema de fluxo máximo em redes tem como dados uma rede $G = (N, A)$, a capacidade u_{ij} de transmissão para cada arco $(i, j) \in A$, um nó origem s e um nó destino t. O problema se resume a encontrar o fluxo x_{ij} através de cada arco (i, j) que maximize a quantidade de fluxo enviada de s para t, enquanto satisfazendo às restrições de capacidade nos arcos. Em programação matemática, o problema fica:

$$
\begin{aligned}
&Maximize && \sum_{\{j:(s,j)\in A\}} x_{sj} && \\
&Sujeito\ a: && && \\
& && \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0 && for\ all\ i \in N - \{s,t\} \\
& && 0 \le x_{ij} \le u_{ij} && for\ all\ (i,j) \in A
\end{aligned}
$$

Mostre como se pode encontrar um fluxo factível para o problema de fluxo em redes de custo mínimo fazendo uso do problema de fluxo máximo.

Capítulo 9

Linguagens de Modelagem

Neste capítulo apresentamos os elementos fundamentais de duas linguagens de modelagem que permitem especificar problemas de otimização em uma linguagem muito semelhante à programação matemática. Em particular, desenvolveremos os conceitos básicos das linguagens AMPL e Mosel. Tais linguagens procuram separar o modelo dos dados, dessa forma permitindo que um modelo possa ser utilizados na resolução de diferentes instâncias de uma mesma classe de problemas. Elas também são responsáveis pela parte de pré-processamento e interface com algoritmos de otimização, tais como ILOG CPLEX, MINOS e XPress-MP.

9.1 Linguagem Mosel

A linguagem Mosel faz parte do pacote de software de otimização Xpress-MP. Xpress-MP é uma ferramenta de software para modelagem matemática e solução de problemas de otimização linear, quadrática e linear inteira. As ferramentas do Xpress-MP compreendem uma coleção de interfaces, objetivando atender necessidades de usuários diversos e permitindo a solução de problemas bem como a integração com outros produtos de software. Os dois componentes básicos do Xpress-MP são o Xpress-Mosel e o Xpress-Optimizer, mas há outros componentes também relevantes destacando-se Xpress-IVE, Console Xpress e Xpress-MP Libraries.

Xpress-Mosel É um ambiente para modelagem e solução de problemas, tomando como entrada um modelo que descreve um problema em programação matemática (LP, QP, MIP), escrito na linguagem de modelagem Mosel. Mosel permite separar modelo dos parâmetros que definem uma instância e tem facilidades de transferência de dados. Por exemplo, dados podem ser transferidos através de arquivos tipo texto e em formato ODBC.

Xpress-Optimizer Optimizer é a parte central da ferramenta Xpress-MP, representando décadas de pesquisa e desenvolvimento de métodos de solução de problemas lineares, quadráticos e inteiros mistos.

Xpress-IVE Xpress-IVE (*The Xpress Interactive Visual Environment*) é um ambiente para modelagem e solução de problemas de otimização que roda em sistemas MS-Windows. IVE apresenta Mosel dentro de uma interface gráfica amigável com editor embutido.

Console Xpress Consiste em interfaces tipo texto para Mosel e Optimizer. Este módulo permite a carga de modelos armazenados em arquivos e resolução de problemas, podendo ser executado em plataformas computacionais diversas, incluindo Unix e MS-Windows. O Console Xpress pode ser aplicado em ambientes de produção fazendo uso de arquivos *shell-script*.

Xpress-MP Libraries Para implementações especializadas, bibliotecas Xpress-MP são oferecidas para prover acesso direto a Mosel e Optimizer a partir código C/C++, Java e Visual Basic implementado e customizado pelo usuário. A vantagem principal das bibliotecas é a possibilidade de interagir diretamente com a funcionalidade de Xpress-MP e, dessa forma, desenvolver aplicações customizadas como, por exemplo, heurísticas projetadas para problemas particulares e algoritmos tipo *branch-and-cut*.

Uma versão limitada no número de variáveis e restrições, mas completa em termos de funcionalidade, do Xpress-IVE pode ser obtida para uso acadêmico e educacional no site da Dash Optimization: `http://www.dashoptimization.com`.

9.1.1 Qual Interface Devemos Utilizar?

Esta decisão depende das necessidades do usuário.

Usuários iniciantes devem preferencialmente utilizar Xpress-IVE, uma vez que esta é a interface mais simples para o software Xpress-MP. O ambiente integrado permite rápida modelagem ao mesmo tempo que facilita a comunicação com os demais componentes do Xpress-MP

Já em Ambientes de produção, a interface Console Xpress é mais adequada pois esta permite que se desenvolva aplicações tipo *batch*. Console Xpress provê uma interface tipo texto poderosa para os componentes Xpress-MP, permitindo desenvolvimento interativo e processamento *batch* de modelos de usuários de várias plataformas

Por outro lado, as bibliotecas devem ser reservadas para os usuários avançados que desejem desenvolver e testar algoritmos (e.g., heurísticas inteligentes e algoritmos *branch-and-bound*).

9.1.2 Resolvendo um Problema

Aqui descrevemos os passos necessários para se resolver um problema através da interface Mosel. Considere o problema de programação linear abaixo:

$$\begin{aligned} &Maximize && a + 2b \\ &Sujeito\ a: && 3a + 2b \leq 400 \\ & && a + 3b \leq 200 \\ & && a, b \geq 0 \end{aligned}$$

Abaixo segue a especificação do problema acima na linguagem Mosel.

```
model Simple
  uses ‘‘mmxprs’’
  declarations
    a:  mpvar
    b:  mpvar
  end-declarations

  first := 3*a + 2*b <= 400
  second := a + 3*b <= 200
  profit := a + 2*b

  maximize(profit)

  writeln(‘‘profit is ’’, getobjval)

end-model
```

O modelo acima é quase auto-explicativo. A diretiva `uses` é utilizada para declarar a biblioteca de otimização a ser utilizada. Na seção `declarations` são declaradas as variáveis do modelo. As restrições, denominadas `first` e `second`, são descritas na seqüência sendo ambas as desigualdades do tipo "$\leq$". A função objetivo é denominada de `profit` e utilizada pelo comando `maximize`, que executará o algoritmo de otimização adequado para o problema, buscando uma solução que respeite às restrições e maximize

o lucro. No momento em que a solução ótima for encontrada, o valor da função objetivo ótima é impresso na saída padrão.

A solução de um problema em Mosel é um processo de 3 estágios consistindo em i) compilar o programa modelo, ii) carregar o programa compilado, e iii) resolver o programa. Os comandos que realizam estas três etapas são: `compile`, `load`, e `run`.

Abaixo ilustramos a execução dos passos para se resolver o problema de interesse. O modelo está armazenado no arquivo "`simple.mos`".

```
C:> mosel
** Xpress-Mosel **
© Copyright Dash Associates 1998-2002
> compile simple
Compiling simple
> load simple
> run
Profit is 171.429
Returned value:  0
```

Para obter informações sobre os comandos disponíveis e parâmetros, simplesmente digite o comando `help` na janela de entrada da interface Mosel.

9.1.3 Indo Mais Longe

O formato MPS é um padrão para representação e armazenamento de problemas de programação matemática. Vários otimizadores aceitam problemas especificados neste formato. Mosel e Xpress-optimizer não podiam ser diferentes e ambos suportam a entrada e saída de problemas no formato MPS. Para gerar o arquivo MPS correspondendo ao problema escrito em Mosel, basta executar o comando:

```
> export -m simple.mps
```

Mosel também é capaz de gerar o problema no formato LP, que pode ser resolvido com certos pacotes de otimização como, por exemplo, `lp_solve` (`ftp://ftp.es.ele.tue.nl/pub/lp_solve`) e `glpk` da GNU (`http://www.gnu.org/software/glpk/glpk.html`). Para gerar o arquivo LP correspondente ao problema escrito em Mosel, execute o comando:

```
> export simple.lp
```

9.1.4 Trabalhando com o Optimizer

Nos casos anteriores, carregamos modelos dentro do ambiente Mosel e invocamos optimizer como um módulo da biblioteca. Optimizer pode também ser executado isoladamente. Ele aceita problemas armazenados em arquivos tipo matriz, em formato MPS ou LP. Com um arquivo "pb.mat" válido, tudo que precisa ser feito é carregar a matriz completamente no optimizer e maximizar/minimizar a função objetivo. Abaixo segue a seqüência de comandos que carregam e resolvem um problema armazenado em um arquivo tipo MPS:

```
C:> optimizer
Xpress-MP Integer Barrier Optimizer Release xx.yy
© Copyright Dash Associates 1998-2002
Enter problem name> simple
> readprob
⋮
> maxim
⋮
> writeprtsol (ou prtsol)
> quit
```

9.1.5 Construindo um Primeiro Modelo

Trataremos aqui de flexibilidades da modelagem com Mosel bem como a separação da estrutura do modelo e dos dados, que juntos formam uma instância do problema. Utilizaremos o problema da mochila para ilustrar tal flexibilidade.

$$
\begin{aligned}
&\textit{Maximize} \quad \sum_{i=1}^{n} c_i x_i \\
&\textit{Sujeito a}: \\
&\qquad \sum_{i=1}^{n} a_i x_i \leq b \\
&\qquad x_i \in \{0,1\},\ i = 1, \ldots, n
\end{aligned}
$$

Considere a instância do problema da mochila obtida com os dados da Tabela **??**.

A especificação Mosel do problema da mochila para os dados acima pode ser realizada como segue:

```
model knapsack
uses ‘‘mmxprs’’
```

Tabela 9.1: Dados do problema da mochila

Objeto	Peso	Valor
Camera	2	15
Necklace	20	100
Vase	20	90
Picture	30	60
TV	40	40
Video	30	15
Chest	60	15
Brick	10	1

```
declarations
   camera, necklace, vase,
   picture, tv, video, chest,
   brick :  mpvar
end-declarations

camera is_binary
necklace is_binary
vase is_binary
⋮
brick is_binary

Totalweight := 2*camera + 20*necklace + 20*vase + 20*picture
   + 40*tv + 30*video + 60*chest + 10*bricks <= 102
Totalvalue := 15*camera + 100*necklace + 90*vase + 60*picture
   + 40*tv + 15*video + 10*chest + 1*brick

maximize(TotalValue)
writeln(‘‘Objective value is ’’,getobjval)

end-model
```

A construção de modelos como visto acima é conveniente para problemas com um número pequeno de variáveis, mas isso pode se tornar inviável em problemas de grande porte. Foi tedioso, por exemplo, ter que especificar que cada variável é binária. É

usual na modelagem, mesmo de problemas pequenos, o uso de vetores de variáveis ou variáveis subscritas, como também são conhecidas. Tal política de modelagem facilita enormemente a especificação de modelos e futuras alterações. Abaixo segue a segunda versão Mosel do problema da mochila.

```
model knapsack2
uses ‘‘mmxprs’’

declarations
  Items = 1..8
  Weight:  array(Items) of real
  Value:  array(Items) of real
  x :  array(Items) of mpvar
end-declarations

!  Items
Weight:= [2, 20, 20, 30, 40, 30,
     60, 10]
Value:= [15, 100, 90, 60, 40,
     15, 10, 1]

!  All x are binary
forall(i in Items) x(i) is_binary
!  Objective
TotalValue := sum(i in Items) x(i)*Value(i)
!  Constraint weight restriction
TotalWeight := sum(i in Items) x(i)*Weight(i) <= 102

maximize(TotalValue)

writeln(‘‘Objective value is ’’, getobjval)
forall(i in Items) writeln(‘‘x(’’,i,‘‘) =’’,getsol(x(i)))

end-model
```

A modelagem com vetores trouxe algumas facilidades, dentre elas destacam-se:

Items é um conjunto indexado para os elementos do vetor

forall permite varrer elementos de um vetor indexado

sum permite somar elementos de um vetor indexado

9.1.6 Usando Cadeias de Caracteres como Índices

O modelo anterior é consideravelmente mais simples do que o primeiro modelo para o problema da mochila, mas sua interpretação é mais difícil—temos que traduzir os identificadores em seus respectivos nomes. Contudo, podemos melhorar a legibilidade do código Mosel por meio da seguinte substituição:

```
Items = {''camera'', ''necklace'', ''vase'', ''picture'',
   ''tv'', ''video'', ''chest'', ''brick''}
```

9.1.7 Modelagem Versátil

Os exemplos acima são apenas alguns de um número de problemas similares, conhecidos como problemas da mochila. Seria útil generalizar o modelo acima, permitindo o aumento do número de itens e a alteração dos pesos/valores. Com o problema já especificado em termos de vetores, as dificuldades estão no acoplamento do modelo com os dados. Aqui introduzimos algumas definições que facilitam o entendimento dessas questões:

Modelos são geralmente especificados usando símbolos para representar as diversas variáveis de decisão, bem como o relacionamento entre as variáveis por meio de desigualdades e igualdades.

Modelos Genéricos são obtidos através de uma descrição sistemática dessas restrições e dos objetivos.

Instância é o resultado da combinação do modelo genérico com um conjunto de dados, a qual pode ser otimizada.

Modelo: "knapsack3.mos"

```
model knapsack3
uses ''mmxprs''

declarations
   Items = set of strings
   MAXWT:real
```

```
   Weight:  array(Items) of real
   Value:  array(Items) of real
   x :  array(Items) of mpvar
end-declarations

!  Read in data from external file
initializations from ‘‘knapsack3.txt’’
   Items MAXWT Weight Value
end-initializations

!  Create Variables
forall(i in Items) create(x(i))
!  All x are binary
forall(i in Items) x(i) is_binary
!  Objective
TotalValue := sum(i in Items) x(i)*Value(i)
!  Constraint weight restriction
TotalWeight := sum(i in Items) x(i)*Weight(i) <= MAXWT
maximize(TotalValue)
writeln(‘‘Objective value is ’’, getobjval)
end-model
```

Arquivo de Dados: “knapsack3.txt”

```
Items:  {‘‘camera’’, ‘‘necklace’’, ‘‘vase’’, ‘‘picture’’,
   ‘‘tv’’, ‘‘video’’, ‘‘chest’’, ‘‘brick’’}
Weight:  [2, 20, 20, 30, 40, 30,
   60, 10]
Value:  [15, 100, 90, 60, 40,
   15, 10, 1]
MAXWT: 120
```

9.2 Linguagem AMPL

AMPL pode ser vista como uma linguagem de computador utilizada para descrever de uma forma declarativa problemas de planejamento, escalonamento e distribuição da produção e muitos outros problemas conhecidos em geral como problemas de otimização

em larga escala ou programação matemática. A notação algébrica de AMPL e o seu ambiente de comandos interativo foram projetados para auxiliar na formulação de modelos, comunicar com uma variedade de pacotes de otimização e examinar o resultado de soluções. A flexibilidade de AMPL a torna ideal para prototipação rápida e desenvolvimento de modelos, enquanto que sua velocidade e generalidade provêem os recursos necessários para resolução em regime de produção.

AMPL é um entre vários sistemas de modelagem para otimização que são projetados tomando como base linguagens de modelagem algébricas. Essas linguagens empregam uma notação matemática muito semelhante à programação matemática, tipicamente utilizada para descrever problemas de otimização como a minimização (ou maximização) de uma expressão algébrica envolvendo variáveis de decisão algébricas, subjeita a restrições expressas como igualdades e desigualdades entre expressões algébricas e variáveis de decisão. Interpretadores e interfaces para essas linguagem provêem suporte para simplificação e análise de modelos. Algumas linguagens também oferecem extensões para descrisão de métodos algorítmicos para atacar problemas difíceis por meio da resolução de subproblemas relacionados. MOSEL e AMPL são dois exemplos de linguagens de modelagem algébricas distribuídas comercialmente.

AMPL foi projetada para combinar e estender as habilidades expressivas de linguagens de modelagem, mas sem perder a facilidade de ser utilizada em aplicações elementares. AMPL é notável pela simplicidade e naturalidade de sua sintaxe e pela generalidade dos seus conjuntos e expressões de indexação. AMPL provê forte suporte à validação, verificação e análise de soluções ótimas, através de um conjunto de alternativas para apresentação de dados e resultados. O pré-processador de AMPL é capaz de executar automaticamente transformações que reduzem o tamanho do problema e substituem variáveis. AMPL se distingue ainda pela continuidade do seu desenvolvimento que visa atender às necessidades dos usuários. Adições recentes incluem construtores de laços e testes para escrita de *scripts* na linguagem de comandos AMPL, e facilidades para definição e manipulação de vários problemas inter-relacionados.

No que segue apresentamos um exemplo de AMPL para o problema da mochila.

9.2.1 Modelo AMPL do Problema da Mochila

A descrição de um problema é feita através da especificação de três arquivos: o arquivo com o modelo genérico, o arquivo com os dados e arquivo de comandos.

Modelo Genérico

```
#-------------------------------------
```

```
# knapsack problem

set Items;

param c {j in Items}; # c[j] is the value of the item j
param w {j in Items}; # w[j] is the weight of the item j

param b >= 0;

var x {j in Items} integer >= 0, <= 1;

# objective function
maximize value:  sumj in Items c[j]*x[j];

# subject to knapsack constraint
subject to knapsack:
   sumj in Items w[j]*x[j] <= b;
```

Arquivo de Dados

```
set Items := camera necklace vase picture
             tv video chest brick;

param:                 w   c :=
             camera    2   15
             necklace  20  100
             vase      20  9
             picture   30  60
             tv        40  40
             video     30  15
             chest     60  10
             brick     10  1;

param b := 120;
```

Arquivo de Comandos

```
solve;
display value;
display w;
display c;
display x;
```

9.2.2 Comentários

A linguagem AMPL rapidamente se tornou um padrão entre pesquisadores e engenheiros, hoje disponível na maior parte dos pacotes de otimização. Há um site exclusivamente dedicado à AMPL, `http://www.ampl.com`, onde estão depositados exemplos além de inúmeras informações e ponteiros. A propósito, existe um livro sobre AMPL e suas aplicações bem como uma versão educacional da linguagem que pode ser empregada para resolver problemas com um número pequeno de variáveis e restrições.

9.3 Exercícios

EX 9.1 Codifique em AMPL os exercícios EX 1.1, EX 1.3, EX 1.4, EX 1.5 e EX 1.6. Encontre as soluções executando os respectivos modelos no servidor NEOS (`http://www-neos.mcs.anl.gov/neos/server-solvers.html`).

Capítulo 10

Fundamentos de Programação Inteira

Este capítulo inicia o estudo da programação inteira. Após apresentarmos alguns problemas motivadores, descrevemos formalmente alguns casos particulares de problemas de programação inteira e apresentamos um roteiro para modelagem de problemas. Ao final introduzimos conceitos fundamentais para formulação eficaz de problemas, em particular formas de comparação e avaliação da qualidade de formulações alternativas para um problema.

10.1 Introdução

Uma variedade expressiva de problemas de cunho prático e teórico podem ser formulados em programação linear inteira, incluindo problemas combinatórios freqüentemente encontrados em teoria dos grafos, problemas de lógica, situações práticas compreendendo a logística de empresas, entre outros. No que segue delineamos dois problemas práticos tipicamente resolvidos por meio de modelos e algoritmos de programação inteira.

10.1.1 Escalonamento de Trens

Considere uma empresa de transporte ferroviário que deseja encontrar um escalonamento de seus trens, ou seja, rotas incluindo datas de partida, chegadas e tempos de parmanência em estações ao longo das rotas. Um aspecto relevante é o fato de que as rotas se repetem diariamente, justificando portanto a otmização dos escalonamento. Dados acerca do problema também estão disponíveis e devem ser levados em consideração: os tempos de percurso de uma estação para outra são conhecidos; dois trens

não podem trafegar na mesma linha, a menos que separados por alguns minutos; e para facilitar conexões, o horário de partida de um trem A deve suceder o horário de chegada de um trem B por alguns minutos. Enfim, o problema é encontrar um escalonamento factível que minimize algum critério econômico.

10.1.2 *Airline Crew Scheduling*

O problema de alocar tripulações a vôos sujeito a restrições físicas, temporais, de segurança e trabalhistas constitui uma tarefa árdua de grande impacto econômico para empresas aéreas, tanto que a indústria de aviação comercial americana possui departamentos de pesquisa operacional responsáveis por problemas desta natureza. Dados o itinerário dos vôos para um tipo particular de avião, um problema é projetar a escala semanal das tripulações, onde cada tripulação deve ser designada para um período consistindo de um conjunto de vôos que satisfaça uma série de restrições. Dentre elas, ressaltamos: o número máximo de horas de vôo, os períodos de descanso mínimo, e a necessidade de que os vôos sejam cíclicos permitindo o retorno periódico das tripulações às suas respectivas bases. O problema é encontrar uma escala das tripulações que minimize o custo total, que é função do tempo de vôo, período de descanso e muitos outros fatores.

10.2 O Que É um Problema Inteiro?

Em palavras, um problema inteiro se refere ao problema de encontrar um vetor de inteiros que minimize uma função linear deste vetor, ao mesmo tempo que restrições lineares são satisfeitas. Formalmente, o problema de programação linear pode ser expresso em programação matemática como segue:

$$PL: \qquad Max\left\{c^T x : Ax \leq b, x \geq 0\right\}$$

onde $x \in \mathbb{R}^{n\times 1}$, $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^{n\times 1}$ e $b \in \mathbb{R}^{m\times 1}$. O problema geral de programação inteira pode ser divido em classes, dependendo da natureza das decisões, da existência de variáveis contínuas e discretas e do tipo de restrições. Abaixo apresentamos algumas dessas classes.

10.2.1 Problema (Linear) Inteiro Misto

O problema inteiro misto nada mais é do que um problema de programação com variáveis discretas e outras contínuas, sendo expresso na forma:

$$\begin{array}{rll} PIM: & Max & c^T x + h^T y \\ & s.a: & Ax + Gy \leq b \\ & & x \geq 0,\ y \geq 0 \text{ e } y \text{ inteiro.} \end{array}$$

10.2.2 Problema (Linear) Inteiro

Nesta versão do problema, todas as variáveis devem ser inteiras:

$$\begin{array}{rll} PI: & Max & c^T x \\ & s.a: & Ax \leq b \\ & & x \in \mathbb{Z}_+^n \end{array}$$

10.2.3 Problema Linear Binário

Uma versão particular do problema inteiro é o problema binário, no qual todas as variáveis só podem assumir os valores 0 ou 1:

$$\begin{array}{rll} PIB: & Max & c^T x \\ & s.a: & Ax \leq b \\ & & x \in \{0,1\}^n \end{array}$$

A relevância de PIB surge do desenvolvimento de resultados teóricos e algoritmos específicos para esta classe de problema. Um ponto interessante é o fato que um PI pode ser reduzido em um problema equivalente do tipo PIB.

10.2.4 Problema de Otimização Combinatória

Problemas de natureza combinatória compreende aqueles em que se deseja encontrar um arranjo particular, dentre um número comumente combinatório e exponencial de arranjos factíveis, que minimize/maximize algum critério de seleção para os diferentes arranjos. Seja $N = \{1, \ldots, n\}$ um conjunto finito, c_j o custo de cada elemento j de N, e $\mathcal{F}$ uma família de subconjuntos de N dita família de subconjuntos viáveis, e.g., $\mathcal{F} \subseteq 2^N$. Então o problema de otimização combinatória pode ser definido em

programação matemática:

$$POC: \ Max_{S \subseteq N} \left\{ \sum_{j \in S} c_j \ : \ S \in \mathcal{F} \right\}$$

Freqüentemente, POC pode ser formulado ser formulado como PI ou PIB.

10.3 Programação Linear e Arredondamento

Por que não utilizar PL? Poderíamos desconsiderar as restrições de variáveis inteiras, obter uma solução ótima x^* para PL e depois arredondar x^* de forma a se obter uma solução para PI, por exemplo. Infelizmente, esta abordagem não funciona como o PI contra-exemplo abaixo demonstra:

$$\begin{array}{rrcrcr} Max & x_1 & + & 0.6x_2 & & \\ s.a: & 50x_1 & + & 31x_2 & \leq & 250 \\ & 3x_1 & - & 2x_2 & \geq & -4 \end{array}$$

$$x_1, \, x_2 \geq 0 \text{ e inteiros}$$

A solução ótima para PL, $x_{PL} = (\frac{376}{193}, \frac{950}{193})$, poderia ser arredondada para a solução $\overline{x}_{PL} = (2, 4)$, que é bastante "distante" da solução ótima $x^* = (5, 0)$. Uma ilustração é dada na Figura 10.4.

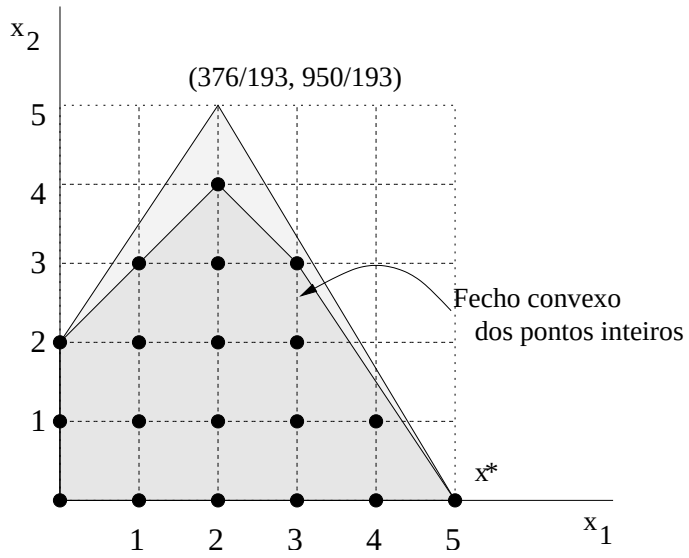


Figura 10.1: Ilustração da solução obtida através de arredondamento

10.4 Formulação de PIs e PIBs

A formulação de um problema é tanto uma arte quanto uma ciência. Existem técnica e metodologias, contudo a experiência e a intuição continuam e provavelmente continuarão a desempenhar um papel fundamental. Aqui apresentamos alguns passos metodológicos para formulação de problemas que são exemplificados. A transcrição de um problema em uma formulação deve ser conduzida de uma forma sistemática, mantendo sempre claro a distinção entre os *dados do problema* e as *variáveis de decisão*. Passos sugeridos:

i. Defina quais variáveis são necessárias

ii. Utilize as variáveis para definir restrições tal que os pontos factíveis correspondam às soluções viáveis da formulação

iii. Utilize as variáveis para definir a função objetivo

Se as dificuldades persistirem, defina conjuntos de variáveis adicionais alternativas, repetindo os passos metodológicos.

10.4.1 Exemplo 1: Formulando o Problema de Alocação

No problema de alocação, há n pessoas para executar n tarefas, sendo cada pessoa obrigatoriamente designada a precisamente a uma tarefa. Alguns indivíduos são mais eficientes em certas tarefas, incorrendo um custo c_{ij} se a pessoa i executar a tarefa j. Portanto o problema é encontrar a alocação pessoa-tarefa de menor custo agregado.

Definindo as Variáveis

$$x_{ij} = \begin{cases} 1 & \text{se a pessoa } i \text{ é alocada à tarefa } j \\ 0 & \text{caso contrário} \end{cases}$$

Definindo as Restrições

a) Cada pessoa i executa uma tarefa

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$$

b) Cada tarefa j é executada por uma pessoa

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n$$

c) As variáveis são 0-1

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n \quad j = 1, \ldots, n$$

Definindo a Função Objetivo

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

10.4.2 Exemplo 2: O Problema da Mochila

O problema da mochila é chave em várias aplicações teórico-práticas. Um exemplo é o problema enfrentado por um investidor que dispões de uma quantidade b para investimentos. Há n projetos sob consideração, sendo a_j o custo estimado do projeto j e c_j é o retorno esperado. O objetivo do investidor se resume a encontrar um conjunto de projetos tal que a quantidade de investimento não exceda a quantidade disponível e, ao mesmo tempo, maximize o retorno.

Definindo as Variáveis

$$x_{ij} = \begin{cases} 1 & \text{se o projeto } i \text{ é selecionado} \\ 0 & \text{caso contrário} \end{cases}$$

Definição de Restrições

a) Limitação de recursos: $\sum_{j=1}^{n} a_j x_j \leq b$

b) Variáveis são 0-1: $x_j \in \{0, 1\}$ $j = 1, \ldots, n$

Definindo a Função Objetivo

$$Max \sum_{j=1}^{n} c_j x_j$$

10.4.3 Exemplo 3: O Problema de Cobertura de Conjuntos

O problema de cobertura de conjuntos define outra classe empregada com freqüência na prática. Para se ter uma idéia, o problema de escalonamento de tripulações pode ser resolvido por meio da decomposição em alguns problemas, um dos quais é um problema de cobertura de conjuntos, onde os conjuntos são vôos a serem operados pela empresa de transporte aéreo e os conjuntos possíveis configurações de tripulações. Aqui vamos considerar o caso particular de localização de unidades de emergência. Dado um certo número de regiões, o problema é decidir onde devemos instalar um conjunto de serviços de emergência. Para cada ponto possível de instalação de um centro, sabemos o custo de instalação e as regiões que a central de serviços pode atender (e.g., em menos de 8 minutos) Seja $M = \{1, \ldots, m\}$ o conjunto de regiões e $N = \{1, \ldots, n\}$ o conjunto de possíveis centrais de emergência. Seja ainda $S_j \subseteq M$ o conjunto de regiões que podem ser atendidas pela central j e c_j o custo de instalação. Então, obtemos o seguinte problema de otimização combinatória:

$$POC: \ Min_{T \subseteq N}\{\sum_{j \in T} c_j : \bigcup_{j \in T} S_j = M\}$$

Formulando o Problema Como um PIB

Primeiro construímos a matriz A tal que $a_{ij} = 1$ se $i \in S_j$ e $a_{ij} = 0$ caso contrário, portanto $A \in \{0,1\}^{m \times n}$.

Definindo as Variáveis

$$x_j = \begin{cases} 1 & \text{se o centro } j \text{ será instalado} \\ 0 & \text{caso contrário} \end{cases}$$

Definindo as restrições

a) Pelo menos um centro deve servir a região i

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad \text{para } i = 1, \ldots, m$$

b) As variáveis são binárias

$$x_j \in \{0,1\} \qquad \text{para} j = 1, \ldots, n$$

Definindo a Função Objetivo

$$Min \sum_{j=1}^{n} c_j x_j$$

10.4.4 Exemplo 4: O Problema do Caixeiro Viajante (PCV)

Outro problema clássico é o problema do caixeiro viajante, o qual consiste em escolher uma ordem para um viajante partir da sua cidade base, digamos cidade 1, visitar as demais $n-1$ cidades precisamente uma vez, e depois retornar à cidade base de maneira que a distância percorrida se a menor possível. Nos é dado um conjunto de n cidades e o custo (distância) c_{ij} do deslocamento da cidade i para a cidade j, e a tarefa de encontrar uma rota (circuito) que visite cada cidade precisamente uma vez. As aplicações são diversar, em particular o roteamento de veículos, a soldagem de pontos de contato em placas de circuitos integrados, e a coleta de lixo sistematizada.

Definindo as Variáveis

$$x_{ij} = \begin{cases} 1 & \text{se o viajante se desloca da cidade } i \text{ para a cidade } j \\ 0 & \text{caso contrário} \end{cases}$$

Definindo as Restrições

a) O viajante deixa a cidade i exatamente uma vez: $\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$

b) O viajante entra na cidade j exatamente uma vez: $\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n$

c) Restrições de conectividade:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \qquad \forall S \subset N,\ S \neq \emptyset$$

ou eliminação de subrotas

$$\sum_{i \in S} \sum_{j \in S,\, j \neq i} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq N,\ 2 \leq |S| \leq n-1$$

A Figura 10.2 ilustra a restrição de conectividade.

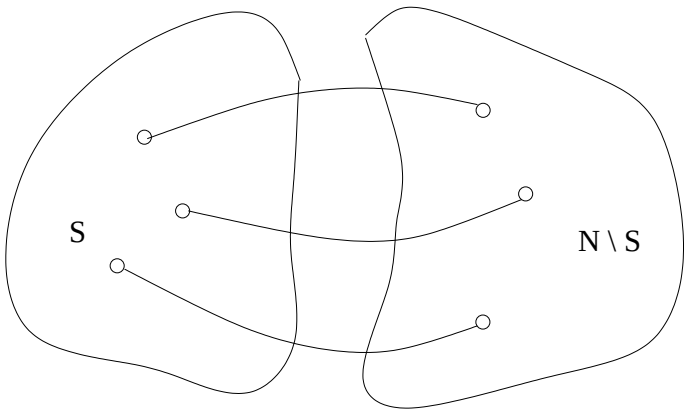


Figura 10.2: Ilustração da restrição de conectividade

Definindo a função objetivo

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

10.5 Explosão Combinatória

Todos os problemas até então vistos são combinatoriais: a solução é um subconjunto de um conjunto finito. Em princípio, tendo em vista que o conjunto de soluções factíveis é finito, poderíamos enumerar todas as soluções factíveis. As Tabelas 10.1 e 10.2 ilustram o crescimento do número de soluções factíveis para alguns problemas. A partir desta análise preliminar, podemos concluir que uma abordagem baseada em enumeração pode ser eficaz apenas em instâncias pequenas.

Tabela 10.1: Número de soluções em função do tamanho do problema

Tipo	Problema	Número de soluções
1	Alocação (10.4.1)	$n!$
2	Mochila e Cobertura de conjuntos (10.4.2) e (10.4.3)	2^n
3	Problema assimétrico do caixeiro viajante	$(n-1)!$

Tabela 10.2: Crescimento de funções

n	$log_2 n$	$\sqrt{n}$	n^2	2^n	$n!$
10	3.32	3.16	10^2	10^3	3.6×10^6
10^2	6.64	10	10^4	10^{30}	9.3×10^{157}
10^3	9.97	31.62	10^6	10^{301}	4×10^{2567}

10.6 Formulação de Problemas Inteiros Mistos (PIMS)

Modelos contendo variáveis discretas e contínuas apresentam propriedades sui generis, necessitando tratamento particular. Por exemplo, nos próximos capítulos veremos os cortes de Gomory que são diferenciados dependendo do problema ser inteiro ou inteiro misto. No que segue será apresentado um conjunto de truques de modelagem e exemplos que podem ser úteis.

10.6.1 Exemplo 1: Modelando Custos Fixos

Imagine a situação onde se deseja modelar uma função não-linear dada por:

$$h(x) = \begin{cases} f + px & \text{se } 0 < x \leq c \\ 0 & \text{se } x = 0 \end{cases}$$

A função $h(x)$ é ilustrada na Figura 10.3.

Definição de uma variável adicional (y)

$$y = \begin{cases} 1 & \text{se } x > 0 \\ 0 & \text{se } x = 0 \end{cases}$$

Introduzindo restrições e a função objetivo

$$h(x) = fy + px$$
$$x \leq cy$$
$$y \in \{0, 1\}$$

O modelo matemático acima é válido apenas para o caso de minimização. minimização. Note também que $x = 0$ e $y = 1$ podem ocorrer.

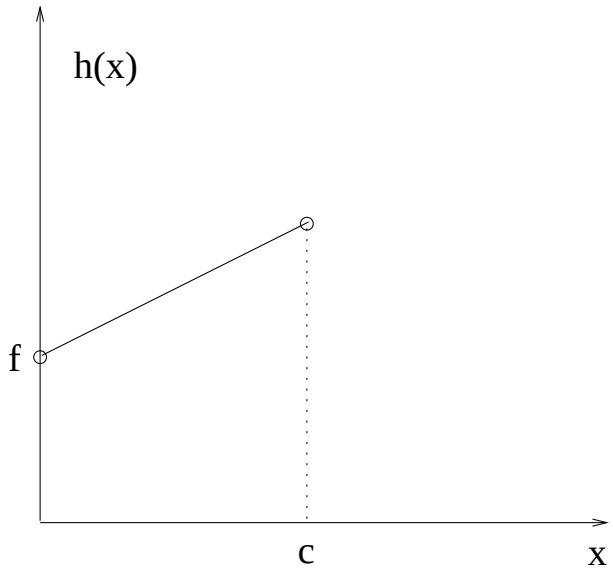


Figura 10.3: Função com custo fixo

10.6.2 Exemplo 2: Localização de Depósitos sem Limites de Capacidade

Mais um exemplo de problema clássico que se preocupa com a localização de depósitos, e conseqüentemente o custo de instalação deste bem como a posição estratégica dos mesmos, além dos custos de transporte incorridos no transporte de mercadorias dos depósitos aos clientes. Nos é dado um conjunto $N = 1, \ldots, n$ de possíveis localizações de depósitos, um conjunto $M = 1, \ldots, m$ de clientes, o custo fixo f_j para a instalação do depósito j, e custo de transporte c_{ij} caso o suprimento da demanda do cliente i se realizada pelo depóstiso j. Assim o problema consistem em decidir que depósitos serão instalados e quais depósitos atenderão quais clientes, de maneira a minimizar o custo total de instalação e transporte.

Definindo as variáveis

$$y_j = \begin{cases} 1 & \text{se o depósito } j \text{ será construído} \\ 0 & \text{caso contrário} \end{cases}$$

x_{ij} é a fração da demanda do cliente i suprida pelo depósito j

Definindo as restrições

a) Depósito j pode operar somente se instalado: $\sum_{i=1}^{n} x_{ij} \leq m y_j \qquad j = 1, \ldots, n$

b) A demanda do cliente i deve ser satisfeita: $\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, m$

c) Variáveis 0-1

$$y_j \in \{0,1\} \qquad j = 1, \ldots, m$$
$$0 \leq x_{ij} \leq 1 \qquad i = 1, \ldots, m \quad \text{e} \quad j = 1, \ldots, n$$

Especificando a função objetivo

$$Min \sum_{j=1}^{n} f_j y_j + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Problema completo

$$\begin{aligned} &Minimize \quad \sum_{j=1}^{n} f_j y_j + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ &Sujeito\ a: \\ &\qquad \sum_{j=1}^{n} x_{ij} = 1 \qquad\qquad i = 1, \ldots, m \\ &\qquad \sum_{i=1}^{n} x_{ij} \leq m y_j \qquad\qquad j = 1, \ldots, n \\ &\qquad y_j \in \{0,1\} \qquad\qquad j = 1, \ldots, m \\ &\qquad 0 \leq x_{ij} \leq 1 \qquad\qquad i = 1, \ldots, m \quad \text{e} \quad j = 1, \ldots, n \end{aligned} \tag{10.1}$$

10.6.3 Alternativas Discretas e Disjuntas

Uma área que tem recebido atenção de pesquisadores e está tendo resultados promissores na prática é a programação disjuntiva, ou seja, modelagem e algoritmos com base em disjunções. Para entender a programação disjuntiva, suponha que $x \in \mathbb{R}^n$ satisfaz

$$\begin{aligned} &0 \leq x \leq u \ \text{ e} \\ &a_1^T x \leq b_1 \ \text{ ou } \ a_2^T x \leq b_2 \end{aligned} \tag{10.2}$$

Em outras palavras, x deve satisfazer uma ou outra restrição linear, não sendo necessário mas possível que ele satisfaça ambas as restrições. A região factível de uma disjunção com duas restrições é ilustrada na Figura 10.4. Note que a região factíve não é convexa, o que aconteceria se estivemos

Podemos representar as equações em (10.2), ou melhor, a *disjunção ou* em programação inteira mista? Sim é possível. Seja $M = Max_{i=1,2}\{a_i^T x - b_i : 0 \leq x \leq u\}$.

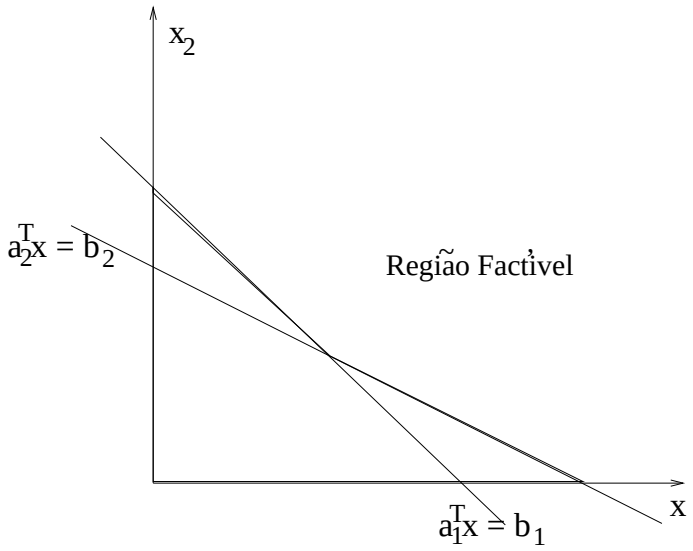


Figura 10.4: Região factível não convexa

Primeiramente, introduzimos duas variáveis binárias y_1 y_2 cuja semântica é explicitada abaixo:

$$y_1 = \begin{cases} 1 & \text{se } x \text{ satisfaz } a_1^T x \leq b_1 \\ 0 & \text{caso contrário} \end{cases}$$
$$y_2 = \begin{cases} 1 & \text{se } x \text{ satisfaz } a_2^T x \leq b_2 \\ 0 & \text{caso contrário} \end{cases}$$

Uma vez introduzidas as variáveis acima, podemos apresentar a formulação completa:

$$a_1^T x \leq b_1 + M(1 - y_1)$$
$$a_2^T x \leq b_2 + M(1 - y_2)$$
$$y_1 + y_2 = 1$$
$$y_1,\ y_2\ \in \{0, 1\}$$
$$0 \leq x \leq u$$

Disjunções aparecem naturalmente em problemas de escalonamento. Suponha que as tarefas 1 e 2 devam ser processadas em uma mesma máquina, mas não simultaneamente. Seja p_i o tempo de processamento da tarefa i e t_i o instante em que o processamento é iniciado. Então, podemos expressar a precedência temporal de uma tarefa em relação a outra por meio de uma disjunção:

$$t_1 + p_1 \leq t_2 \ \text{ ou } \ t_2 + p_2 \leq t_1.$$

10.7 Formulações Alternativas

Nesta e na próxima seção tentaremos entender o que leva uma formulação a ser melhor do que outra. Há situações em que uma formulação é superior a outra, em especial se leva a uma solução algorítmica mais eficiente.

Definição 3 *Um subconjunto do $\mathbb{R}^n$ descrito por um conjunto finito de restrições lineares $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ é dito um poliedro.*

Definição 4 *Um poliedro $P \subseteq \mathbb{R}^{n+p}$ é uma formulação para um conjunto $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ se, e somente se, $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.*

Na Figura são ilustradas duas formulações equivalentes para o conjunto $X = \{(1,1), (2,1), (3,1), (1,2), (2,2), (3,2), (2,3)\}$.

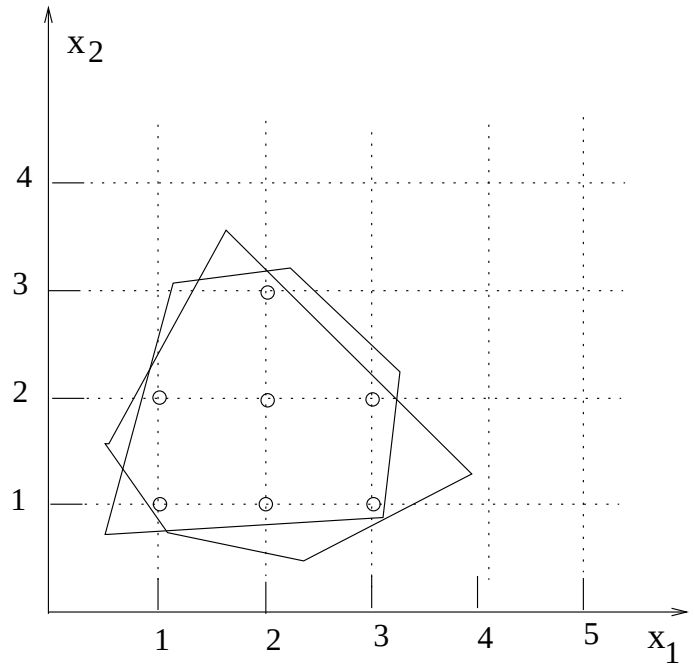


Figura 10.5: Formulações equivalentes alternativas

10.7.1 Formulações Equivalentes para o Problema da Mochila

Considere uma instância do problema da mochila com $n = 4$, onde o conjunto de soluções factíveis é $X = \{(0,0,0,0), (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1),$

$(0,1,0,1)$, $(0,0,1,1)\}$. Os poliedros abaixo induzem formulações válidas para o conjunto X:

$$\begin{array}{lrcrcrcrcr}
P_1 = \{x \in \mathbb{R}^4 :\ 0 \le x \le 1, & 83x_1 & + & 61x_2 & + & 49x_3 & + & 20x_4 & \le & 100\} \\
P_2 = \{x \in \mathbb{R}^4 :\ 0 \le x \le 1, & 4x_1 & + & 3x_2 & + & 2x_3 & + & x_4 & \le & 4\} \\
P_3 = \{x \in \mathbb{R}^4 :\ 0 \le x \le 1, & 4x_1 & + & 3x_2 & + & 2x_3 & + & x_4 & \le & 4\,, \\
 & x_1 & + & x_2 & + & x_3 & & & \le & 1\,, \\
 & x_1 & & & & & + & x_4 & \le & 1\}
\end{array}$$

10.7.2 Localização de Depósitos sem Limites de Capacidade

Relembrando o problema de localização de depósitos sem capacidade, considere um depósito j e a restrição

$$\sum_{i=1}^{n} x_{ij} \le m y_j, \quad y_j \in \{0,1\}, \quad 0 \le x_{ij} \le 1 \text{ para } i \in M \tag{10.3}$$

Logicamente, estas restrições expressam a condição: se $x_ij > 0$ então $y_j = 1$. Podemos então utilizar o seguinte conjunto de restrições alternativas:

$$\begin{array}{l}
0 \le x_{ij} \le y_j \quad \text{para } i \in M \\
y_j \in \{0,1\}
\end{array} \tag{10.4}$$

Isto nos leva à formulação alternativa:

$$\begin{array}{lll}
Min & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} f_j y_j & \\
s.a: & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,m \\
 & 0 \le x_{ij} \le y_j & i = 1,\ldots,m \text{ e } j = 1,\ldots,n \\
 & y_j \in \{0,1\} & j = 1,\ldots,n
\end{array} \tag{10.5}$$

Qual das duas formulações do problema de localização de facilidades é melhor, a formulação (10.1) ou a formulação (10.5)?

10.8 Formulações Apertadas e Ideais

Já observamos que o número de formulações válidas pode ser infinito. *Dentre um número potencialmente infinito de formulações, qual é a melhor formulação?* Por exemplo, considere o conjunto e as formulações equivalentes ilustrados na Figura 10.6.

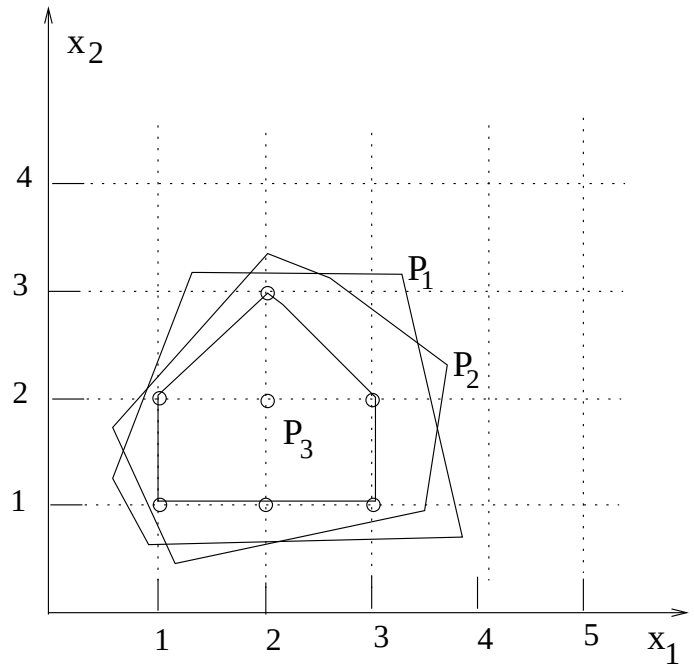


Figura 10.6: Formulações alternativas

Conforme a figura 10.6, a formulação P_3 é ideal: podemos resolver o PL usando P_3 (sem restrições inteiras) e sempre obtermos um ponto extremo inteiro; portanto, com P_3 o problema pode ser resolvido em tempo polinomial.

Definição 5 *Dado um conjunto $X \subseteq \mathbb{R}^n$, o fecho convexo de X, denotado por $conv(X)$ é definido como:*

$$conv(X) = \{x : x \text{ é combinação convexa dos elementos de } X\}$$

Proposição 6 *$conv(X)$ é um poliedro.*

Proposição 7 *Todos os pontos extremos de $conv(X)$ pertencem a X.*

Em princípio, podemos substituir o problema $\{Max\ c^T x : x \in X\}$ por $\{Max\ c^T x : x \in conv(X)\}$. *Então, qual é a dificuldade prática em se realizar esta substituição?* Tipicamente, necessitamos de um número exponencial de desigualdades para descrever o poliedro de $conv(X)$ e, além disso, não existe uma caracterização simples dos mesmos.

Definição 6 *Dado um conjunto $X \subseteq \mathbb{R}^n$ e duas formulações, P_1 e P_2 para X, dizemos que P_1 é melhor (mais apertada) do que P_2 se $P_1 \subset P_2$.*

10.8.1 Formulações Equivalentes para o Problema da Mochila

Vamos relembrar as três formulações alternativas para o problema da mochila sugerido na Seção 10.7.1.

$$
\begin{array}{lrcrcrcrcr}
P_1 = \{x \in \mathbb{R}^4 : 0 \leq x \leq 1, & 83x_1 & + & 61x_2 & + & 49x_3 & + & 20x_4 & \leq & 100 \ \} \\
P_2 = \{x \in \mathbb{R}^4 : 0 \leq x \leq 1, & 4x_1 & + & 3x_2 & + & 2x_3 & + & x_4 & \leq & 4 \ \} \\
P_3 = \{x \in \mathbb{R}^4 : 0 \leq x \leq 1, & 4x_1 & + & 3x_2 & + & 2x_3 & + & x_4 & \leq & 4 \\
 & x_1 & + & x_2 & + & x_3 & & & \leq & 1 \\
 & x_1 & & & & & + & x_4 & \leq & 1 \ \}
\end{array}
$$

Podemos verificar que $P_3 \subset P_2 \subset P_1$. Primeiro vamos mostrar que $P_2 \subset P_1$:

$$
\begin{gathered}
25(4x_1 + 3x_2 + 2x_3 + x_4) \leq 25 \times 4 \Rightarrow \\
100x_1 + 75x_2 + 50x_3 + 25x_4 \leq 100 \Rightarrow \\
83x_1 + 61x_2 + 49x_3 + 20x_4 \leq 100x_1 + 75x_2 + 50x_3 + 25x_4 \leq 100
\end{gathered}
$$

Portanto $x \in P_1 \not\Rightarrow x \in P_2$.

Agora mostraremos que $P_3 \subset P_2$: claramente, $P_3 \subseteq P_2$; o ponto $x = (\frac{1}{2}, 0, 1, 0) \in P_2$ mas $x \notin P_3$. As relações de inclusão entre as três formulações alternativas são ilustradas na Figura 10.7

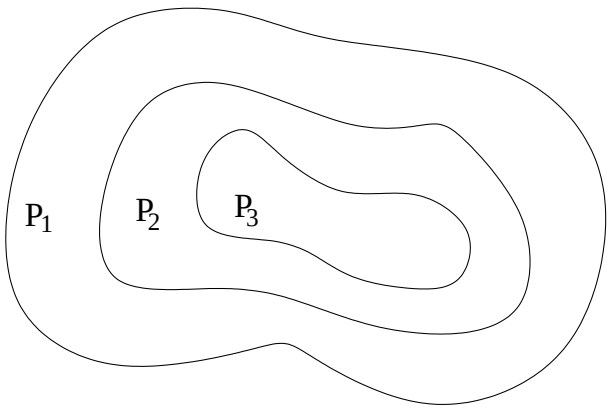


Figura 10.7: Formulações apertadas

10.8.2 Formulação para o Problema de Localização de Depósitos

As duas formulações só diferem quanto às restrições que permitem suprimento de clientes somente se um depósito estiver instalado. Relembrando as formulações.

$$P_1 : \qquad \sum_{i \in M} x_{ij} \leq m y_j \tag{10.6}$$

$$P_2 : x_{ij} \leq y_j \text{ para } i \in M \tag{10.7}$$

Claramente, (10.7) $\Rightarrow$ (10.6) e portanto $P_2 \subseteq P_1$. Considere o ponto x definido como:

$$x_{1j} = 1, x_{2j} = x_{3j} = \ldots . = x_{mj} = 0$$
$$y_j = \tfrac{1}{m}$$

Verifica-se que o ponto $x \in P_1$, mas $x \notin P_2$

10.9 Exercícios

Ex 10.1 Suponha que um investidor deseja investir em um conjunto $S = \{1, \ldots, 7\}$ de ações. Usando variáveis 0-1 formule as seguintes restrições:

i. não pode-se investir em todas as ações;

ii. pelo menos uma ação deve ser selecionada;

iii. a ação 1 não pode ser escolhida se a ação 3 é escolhida;

iv. a ação 4 pode ser escolhida apenas se a ação 2 é escolhida;

v. ambas as ações 1 e 5 devem ser escolhidas ou, exclusivamente, nenhuma delas; e

vi. pelo menos uma das ações do conjunto $\{1, 2, 3\}$ ou pelo menos duas ações do conjunto $\{2, 4, 5, 6\}$.

Ex 10.2 Formule os seguintes problemas em programação inteira mista:

i. $u = min\{x_1, x_2\}$, assumindo que $0 \leq x_j \leq C$ para $j = 1, 2$;

ii. $v = |x_1 - x_2|$ com $0 \leq x_j \leq C$ para $j = 1, 2$; e

iii. o conjunto $X - \{x^*\}$ onde $X = \{x \in Z^n : Ax \leq b\}$ e $x^* \in X$.

EX 10.3 Sejam:

$$
\begin{aligned}
X_1 &= \{x \in \mathbb{B}^4 : 97x_1 + 32x_2 + 25x_3 + 20x_4 \leq 139\} \\
X_2 &= \{x \in \mathbb{B}^4 : 2x_1 + x_2 + x_3 + x_4 \leq 3\} \\
X_3 &= \{x \in \mathbb{B}^4 : \quad x_1 + x_2 + x_3 \leq 2 \\
&\qquad\qquad\quad\; x_1 + x_2 + x_4 \leq 2 \\
&\qquad\qquad\quad\; x_1 + x_3 + x_4 \leq 2\}
\end{aligned}
$$

onde $\mathbb{B} = \{0, 1\}$. Mostre que $X_1 = X_2 = X_3$.

EX 10.4 Considere a instância do problema do caixeiro viajante simétrico conforme Figura 10.8. As arestas omissas possuem um custo proibitivo. Encontre uma solução ótima utilizando um pacote de software capaz de resolver problemas de programação inteira mista. (Sugestão utilize o software `lp_solve` instalado no diretório: `/home/pesquisa/camponog/lp_solve_1.4`.)

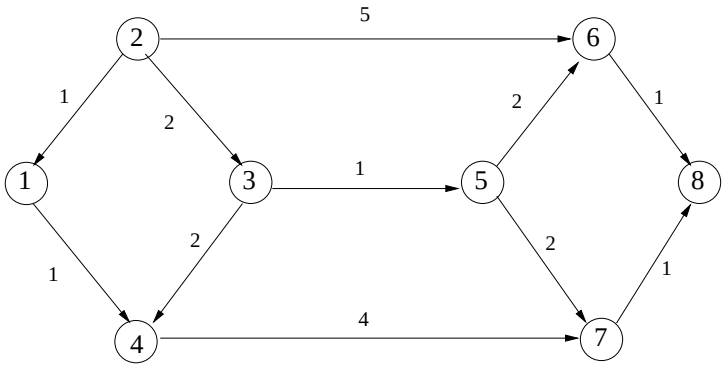


Figura 10.8: Grafo da instância do problema do caixeiro viajante

Capítulo 11

Programação Inteira: Relaxações e Algoritmo *branch-and-bound*

Da mesma forma que em problemas de otimização diferenciáveis, é de grande valor conhecermos condições sob as quais uma solução candidata pode ter sua qualidade estimada em relação a solução ótima, mesmo sem conhecermos a solução ótima. Este tipo de estimação é comumente obtido através da solução ótima de problemas mais fáceis, todavia com espaço de soluções mais abrangente do que o espaço do problema em questão. Estes problemas são ditos relaxações, podendo ser obtidos por meio da desconsideração de algumas restrições, da desconsideração das restrições de integralidade, e da simplificação do problema.

Neste capítulo investigaremos questões relacionadas à otimalidade, diferentes tipos de relaxações, e por fim apresentaremos o algoritmo de *branch-and-bound* que pode ser visto como um procedimento de enumeração implícita que faz uso de uma mais relaxações.

11.1 Condições de Otimalidade

Inicialmente, considere o problema PI que segue:

$$PI: \qquad z = \text{Max}\{c^T x : x \in X \subseteq Z^n\}$$

Dado PI e uma solução candidata x^*, como podemos provar que x^* é uma solução ótima? Em outras palavras, procuramos condições de otimalidade que provêm uma condição de parada para algoritmos. Um método que a primeira vista parece "ingênuo", consiste em encontrar um limite inferior $\underline{z} \leq z$ e um limite superior $\overline{z} \geq z$ tal que se $\underline{z} = \overline{z}$, então z é ótimo. Tipicamente, algoritmos produzem duas seqüências de limites:

- uma seqüência de limites superiores $\overline{z}_1 > \overline{z}_2 > \ldots > \overline{z}_t \geq z$
- uma seqüência de limites inferiores $\underline{z}_1 < \underline{z}_2 < \ldots < \underline{z}_k \leq z$

Assim dado um $\epsilon \geq 0$, podemos definir um critério de parada como $|\overline{z}_t - \underline{z}_k| \leq \xi$. No momento que o algoritmo termina, teremos certeza que a solução candidata z é no máximo ξ unidades inferior a solução ótima. A habilidade de se estimar a qualidade de uma solução candidata é de fundamental importância para otimização.

11.1.1 Limite Primal

Encontrar uma solução factível pode ser uma tarefa fácil ou árdua, dependendo intrinsecamente do problema e da instância particular. Qualquer solução viável x, $x \in X$, induz um limite inferior (limite primal para o caso de maximização) já que $c^T x \leq c^T x^* = z^*$. No caso de problema NP-Completos, como o de encontrar um circuito Hamiltoniano em um grafo, a busca de uma solução factível se resulta a resolver o problema enquanto, por outro lado, encontrar uma rota factível do problema do caixeiro viajante é fácil, estando a dificuldade em encontrar a rota mais curta.

11.1.2 Limite Dual

Um método para encontrar limites superiores faz uso de uma relaxação do problema original, ou seja, um problema mais simples cuja solução ótima não é inferior à solução ótima do problema original.

Definição 7 *Um problema (RP)* $z^R = Max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ *é uma relaxação do problema (PI)* $z = Max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ *se:*

i) $X \subseteq T$

ii) $f(x) \geq c(x)$ *para todo* $x \in X$.

Proposição 8 *Se (RP) é uma relaxação de (PI) então* $z^R \geq z$.

11.2 Relaxação Baseada em PL (Relaxação Linear)

Definição 8 *Para o problema de programação inteira* $Max\{c^T x : x \in P \cap \mathbb{Z}^n\}$ *com formulação* $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ *a relaxação em programação linear é* $z^{PL} = Max\{c^T x : x \in P\}$.

Exemplo:

$$\begin{array}{rrrrrr} z = & Max & 4x_1 & -x_2 & & \\ & s.a: & 7x_1 & -2x_2 & \leq & 14 \\ & & & x_2 & \leq & 3 \\ & & 2x_1 & -2x_2 & \leq & 3 \end{array}$$

$$x \in \mathbb{Z}^2_+$$

limite inferior: observe que $\underline{x} = (2, 1)$ é factível, portanto $z \geq 7$.

limite superior: a solução ótima da relaxação linear é $x^* = \left(\frac{20}{7}, 3\right)$ com $z^{LP} = \frac{59}{7}$, portanto concluímos que $z \leq 8 \leq \frac{59}{7}$, onde $8 = \lfloor \frac{59}{7} \rfloor$.

Proposição 9 *Sejam P_1, P_2 duas formulações para o problema inteiro $z = Max\{c^T x : x \in X \cap \mathbb{Z}^n\}$ sendo P_1 mais apertada do que P_2 (ou seja $P_1 \subseteq P_2$). Se $z_i^{PL} = Max\{c^T x : x \in P_i\}$ para $i = 1, 2$, então $z_1^{PL} \leq z_2^{PL}$.*

Proposição 10

(i) Se a relaxação RP é infactível, então o problema original PI é infactível

(ii) Seja x^* uma solução ótima para RP. Se $x^* \in X \cap \mathbb{Z}^n$ e $f(x^*) = c(x^*)$, então x^* é uma solução ótima para PI.

11.3 Relaxação Combinatória

Sempre que a relaxação é um problema de otimização combinatória, dizemos que ela é uma relaxação combinatória. A seguir exemplificamos a relaxação combinatória no problema do caixeiro viajante e no da mochila.

11.3.1 O Problema do Caixeiro Viajante

Lembrarmos incialmente a formulação em programação matemática do problema:

$$
\begin{aligned}
(PCV) \quad Minimize \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} && && (11.1)\\
s.a: \quad & \sum_{j=1}^{n} x_{ij} = 1 && i = 1,..,n && (11.2)\\
& \sum_{i=1}^{n} x_{ij} = 1 && j = 1,..,n && (11.3)\\
& \sum_{i\in S}\sum_{j\notin S} x_{ij} \geq 1 && \forall S \subset \{1,\ldots,n\}, |S| \geq 2 && (11.4)\\
& x_{ij} \in \{0,1\} && \forall (i,j) \in A && (11.5)
\end{aligned}
$$

Eliminando a família de restrições (11.4), obtém-se um problema de alocação, conforme o problema de alocação de pessoas a tarefas visto em capítulo anterior. Note que o problema de alocação pode ser eficientemente resolvido fazendo uso de algoritmos para fluxo em redes de custo mínimo.

11.3.2 O Problema da Mochila

O problema da mochila é expresso em programação matemática na forma:

$$
\begin{aligned}
(PM) \quad Max \quad & \sum_{j=1}^{n} c_j x_j \\
s.a: \quad & \sum_{j=1}^{n} a_j x_j \leq b \\
& x_j \in \{0,1\}, j = 1,\ldots,n
\end{aligned}
\qquad (11.6)
$$

Uma primeira relaxação pode ser obtida ao tomarmos os valores inteiros imediatamente inferiores aos pesos dos objetos:

$$
\begin{aligned}
(RP_1) \quad Max \quad & \sum_{j=1}^{n} c_j x_j \\
s.a: \quad & \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \leq b \\
& x_j \in \{0,1\}, j = 1,\ldots,n
\end{aligned}
\qquad (11.7)
$$

que por sua vez é equivalente ao problema:

$$\begin{array}{lll} (RP_2) & Max & \sum_{j=1}^{n} c_j x_j \\ & s.a: & \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \leq \lfloor b \rfloor \\ & & x_j \in \{0,1\}, j = 1, \ldots, n \end{array} \tag{11.8}$$

11.4 Relaxação Lagrangeana

Primeiramente, considere o problema em programação inteira na forma abaixo:

$$(PI) \quad z = \quad Max\ c^T x \tag{11.9}$$

$$s.a:\ Ax \leq b \tag{11.10}$$

$$x \in X \subseteq \mathbb{Z}^n \tag{11.11}$$

Como visto acima, uma relaxação pode ser obtida ao eliminarmos o conjunto de restrições $\{Ax \leq b\}$. Em vez de eliminar $\{Ax \leq b\}$, a relaxação Lagrangeana incorpora estas restrições na função objetivo como segue:

$$(RL) \quad z = \quad Max\ c^T x + u^T(b - Ax) \tag{11.12}$$

$$s.a: \quad x \in X \tag{11.13}$$

$$u \geq 0 \tag{11.14}$$

Proposição 11 *Seja* $z(u) = Max\ \{c^T x + u^T(b - Ax) : x \in X\}$. *Então* $z(u) \geq z$ *para todo* $u \geq 0$.

De acordo a Proposição 11, a solução ótima do problema RL para $u \geq 0$ qualquer induz um limite superior $z(u) \geq z^*$, onde z^* é o valor da solução ótima de PI. Não é difícil de se verificar que $z(u) \geq c^T x$, para todo $x \in \mathcal{P} = \{x : x \in X\ eAx \leq b\}$: $c^T x \leq c^T x + u^T(b - Ax)$ uma vez que $b - Ax \geq 0$ para x factível e $u \geq 0$; já que as soluções factíveis de PI também fazem parte do espaço de soluções factíveis de RL, verifica-se que $z(u) \geq z^*$. Uma vez que $z(u)$ é um limite superior para z^* naturalmente desejamos minimizá-lo, dando origem ao problema Lagrangeano dual:

$$\begin{array}{lllll} LD: & Min & z(u) & \cong & Min \quad Max \quad c^T x + u^T(b - Ax) \\ & u \geq 0 & & & u \geq 0 \\ & s.a: & & & s.a: \\ & & x \in X & & x \in X \end{array}$$

Um aspecto relevante do Lagrangeano dual é o fato do problema ser convexo mas não diferenciável. Um algoritmo que pode ser usado para encontrar uma solução aproximada para LD, e até mesmo uma solução ótima quando certas condições são satisfeitas, é o algoritmo subgradiente. Este algoritmo opera de forma semelhante ao algoritmo de descenso. Dada um vetor de multiplicadores de Langrange u_k, o algoritmo encontra um subgradiente d_k que indica a direção na qual os multiplicadores devem ser aumentados/diminuídos. Assim, o iterando seguinte $u_{k+1} = x_k + \alpha_k d_k$ é obtido por meio do passo $\alpha_k \geq 0$. Em virtude da natureza não-diferenciável do problema, não é possível obter condições como as de Armijo.

11.5 Algoritmo *Branch-and-Bound*

O algoritmo "branch-and-bound" (B&B) pode ser entendido como uma extensão da estratégia de divisão e conquista para problemas de natureza inteira mista, ou seja, divida um problema P em um conjunto de subproblemas $\{SP_k\}$ de forma que a solução de P possa ser obtida através da solução dos subproblemas, resolva os subproblemas e, no final, obtenha a solução do problema de interesse. De acordo com o algoritmo B&B, as divisões são feitas iterativamente, sempre observando que os subproblemas devem ser mais fáceis de serem resolvidos que o problema original, além de se procurar descartar subproblemas por meio de enumeração implícita—isto equivale a dizer que, de alguma forma, podemos garantir que a solução ótima não é solução de um certo subproblema e, por conseguinte, podemos descartá-lo.

11.5.1 Estratégia de Divisão e Conquista

Considere o problema:

$$z = Max\{c^T x : x \in S\}$$

Como que o problema acima pode ser "quebrado"em subproblemas menores e depois "recombinados"de maneira a se obter uma solução para o problema original? Abaixo enumeramos algumas proposições que servem de regras gerais para que a "quebra" seja feita de uma forma sistemática e com embasamento teórico.

Proposição 12 *Seja $S = S_1 \cup \ldots \cup S_K$ uma decomposição de S em K conjuntos menores. Seja também $z^k = Max\{c^T x : x \in S_k\}$ para $k = 1, \ldots, K$. Então, $z = Max_k z^k$.*

Uma maneira típica de se ilustrar a estratégia de divisão e conquista, obedecendo as premissas da Proposição 12, é por meio de uma árvore de enumeração.

Exemplo: Para $S \subseteq \{0,1\}^3$, podemos construir a árvore de enumeração da Figura 11.1. Claramente $S = S_0 \cup S_1$, onde $S_0 = \{x \in S : x_1 = 0\}$ e $S_1 = \{x \in S : x_1 = 1\}$. Estendendo este princípio a mais um nível, podemos subdividir cada um dos subproblemas em subproblemas ainda menores, fazendo $S_0 = S_{00} \cup S_{01}$ e $S_1 = S_{10} \cup S_{11}$, onde $S_{i_1 i_2} = \{x \in S_{i_1} : x_2 = i_2\}$.

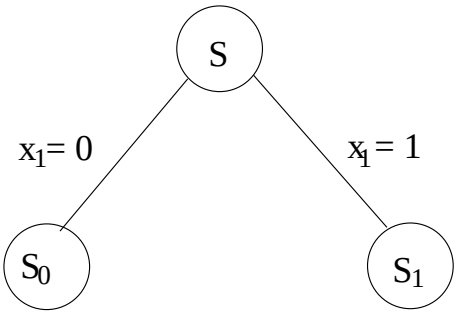


Figura 11.1: Árvore de enumeração

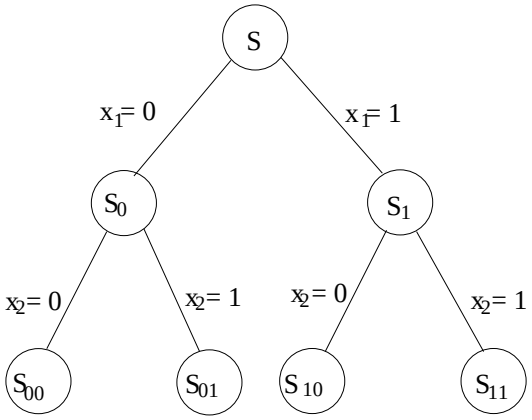


Figura 11.2: Árvore de enumeração

Na Figura 11.3 ilustramos a árvore de enumeração completa. Note que uma folha da árvore $S_{i_1 i_2 i_3}$ é não-vazia se, e somente se, $x = (i_1, i_2, i_3) \in S$. Portanto, as folhas da árvore correspondem precisamente às soluções candidatas que seriam examinadas se fosse conduzida uma enumeração completa.

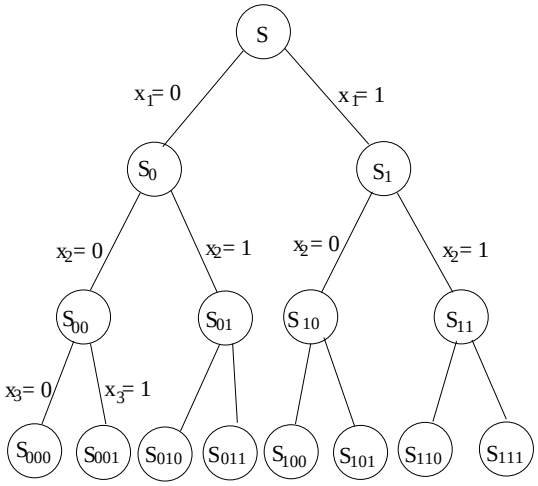


Figura 11.3: Árvore de enumeração completa, explícita

11.5.2 Enumeração Implícita

Enumeração completa é inviável para problemas práticos—o número de nós cresce exponencialmente. Necessitamos de uma alternativa que vai além da simples divisão do problema em subproblemas. Uma abordagem frequentemente encontrada na prática consiste em se utilizar os limites de $\{z^k\}$ de uma forma inteligente, tanto os limites superiores quanto os inferiores. A proposição 13 explicita formalmente como que os limites inferiores e superiores (primais e duais) podem ser empregados de maneira a se realizar enumeração implítica.

Proposição 13 *Seja $S = S_1 \cup \ldots \cup S_K$ uma decomposição de S em K subconjuntos, $z^k = Max\{c^T x : x \in S_k\}$ para $k = 1, \ldots, K$, $\overline{z}^k$ um limite superior para z^k e $\underline{z}^k$ um limite inferior para z^k.*

Então:

a) $\overline{z} = Max\{\overline{z}^k : k = 1, \ldots, K\}$ define um limite superior para z

b) $\underline{z} = Max\{\underline{z}^k : k = 1, \ldots, K\}$ define um limite inferior para z

Exemplo de corte por otimilidade (maximização)

Vamos ilustrar o funcionamento do algoritmo de *branch-and-bound* em um problema qualquer. Seja S o conjunto inicial contendo todas as soluções do problema, conforme Figura 11.4, sendo o limite inferior $lb = 13$ e o limite superior $ub = 27$. Dividindo o

espaço de soluções em dois subconjuntos, $S = S_1 \cup S_2$, e calculando os limites superiores e inferiores para S_1 e S_2, verifica-se que a solução obtida em S_1 é ótima para o problema restrito ao conjunto S_1, já que $l_1 = u_1 = 20$. Portanto, o ramo (S, S_1) foi eliminado por otimalidade. O nó da árvore de enumeração correspondente ao conjunto S_2 de soluções terá de ser examinado uma vez que o limite superior $u_2 = 25 > 20 = l_1$.

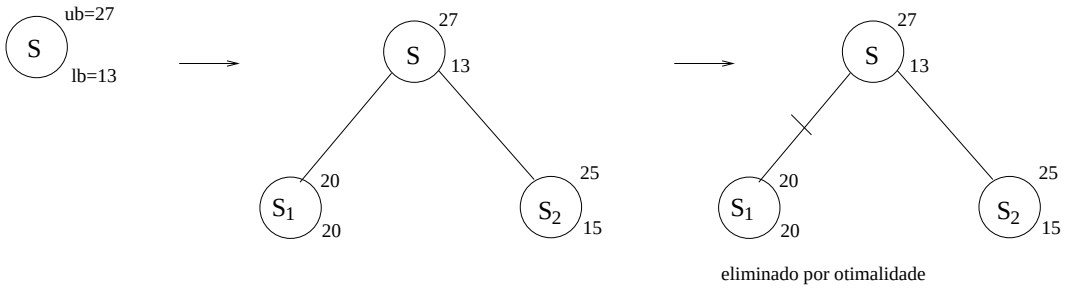


Figura 11.4: Eliminação por otimalidade

Exemplo de corte por limite (maximização)

Considere um problema semelhante ao acima, onde S foi dividido em dois subproblemas tais que $S = S_1 \cup S_2$ conforme Figura 11.5. Limites inferiores e superiores também foram computados e estão indicados ao lado de cada nó da árvore de enumeração. Note que $l_2 > u_1$, o que significa que a melhor solução possível de ser obtida no espaço de soluções S_1 é inferior a melhor solução primal encontrada até o momento. Portanto, podemos eliminar a busca a partir de S_1 dessa forma eliminando muitas soluções de forma implícita. Este corte é ilustrado na figura.

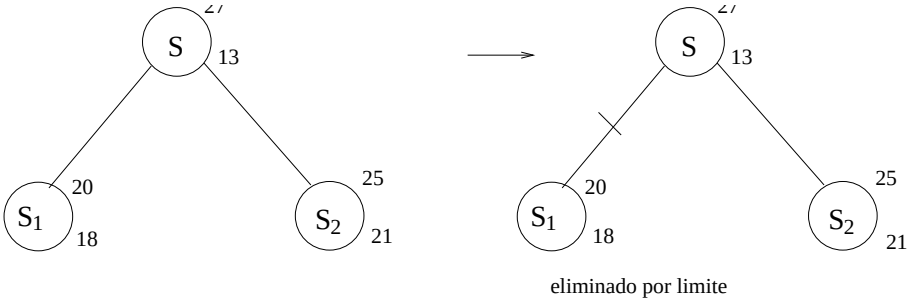


Figura 11.5: Eliminação por limite

Exemplo de situação sem possibilidade de corte

Agora tome como exemplo a árvore de enumeração dada à esquerda da Figura 11.6. Note que o limite superior de S é maior do que o limite superior obtido em S_1 e em S_2, podemos então substituir o limite anterior de 40 para 37, já que $S = S_1 \cup S_2$. O melhor limite inferior obtido até então é 13 que, neste momento, se torna o melhor limite primal para o problema, como ilustrado na árvore à direita. Todavia, ambos os nós S_1 e S_2 devem ser quebrados em subproblemas menores.

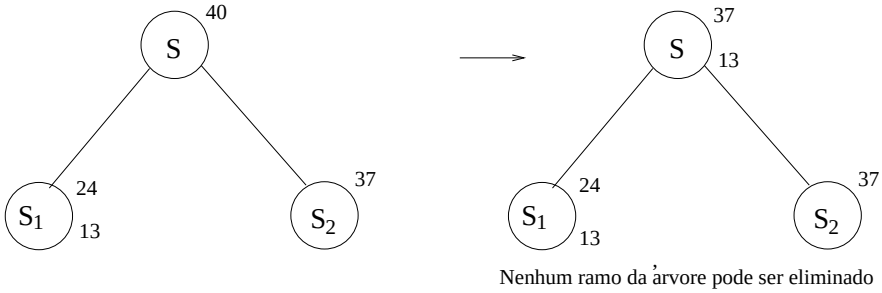


Figura 11.6: Nenhum ramo da árvore pode ser eliminado

Conclusões

Baseado nos exemplos acima, podemos listar três razões para cortar ramos da árvore:

i) Cortando por razão de otimalidade: $z_t = Max\{c^T x : x \in S_t\}$ foi resolvido

ii) Cortando por meio dos limites: $\overline{z}_t < \underline{z}$

iii) Cortando devido à infactibilidade do subproblema: $S_t = \emptyset$

Lembramos aqui que os limites *primais (inferiores)* são obtidos com soluções factíveis, enquanto que os limites *duais (superiores)* são obtidos através de relaxação ou dualidade. É imediata a aplicação da técnica de enumeração com base nas idéias acima apresentadas. No entanto, as seguintes questões são de relevância:

i. Que problema de relaxação ou dual deve ser empregado para obter-se limites superiores? Às vezes podemos tirar vantagem da estrutura do problema na concepção de procedimentos de dualidade como, por exemplo, a relaxação Lagrangeana das restrições de grau 1 do problema do caixeiro viajante.

ii. Como devemos decompor S em $S_1 \cup \ldots \cup S_K$? Não é incomum se fazer valer de uma heurística que explora a natureza particular de um problema.

iii. Em que ordem devemos examinar os subproblemas? Duas estratégias empregadas são a busca em profundidade (*depth-first search*) e a busca pelo melhor ramo (*best-first search*).

11.5.3 Algoritmo *Branch-and-Bound* (B&B)

B&B é um método de enumeração implícita para resolver problemas de programação inteira que utiliza programação linear para calcular os limites duais. Ilustramos o funcionamento do algoritmo através de um exemplo. Consideraremos o problema exemplo abaixo:

$$\begin{array}{rrrrrrr} S: & z = & Max & 4x_1 & -x_2 & & \\ & & s.a: & 7x_1 & -2x_2 & \leq & 14 \\ & & & & x_2 & \leq & 3 \\ & & & 2x_1 & -2x_2 & \leq & 3 \\ & & & x \in \mathbb{Z}^2_+ & & & \end{array}$$

A aplicação do algoritmo B&B ao problema pode ser ilustrada através dos passos a seguir.

***Bounding*:** Obtemos o primeiro limite superior ao resolvermos a relaxação linear, $R(S)$, obtendo $\overline{z} = \frac{59}{7}$ com $(\overline{x}_1, \overline{x}_2) = (\frac{20}{7}, 3)$. Assumimos que $\underline{z} = -\infty$.

***Branching*:** Uma vez que $\underline{z} < \overline{z}$, devemos quebrar S em subproblemas. Uma idéia é quebrar S no ponto onde uma variável é fracionária, fazendo:

$$S_1 = S \cap \{x : x_j \leq \lfloor \overline{x}_j \rfloor\} \; S_2 = S \cap \{x : x_j \leq \lceil \overline{x}_j \rceil\}$$

Claramente, $S = S_1 \cup S_2$. Estas operações do algoritmo B&B podem ser observadas na Figura 11.7, onde a quebra foi feita com a variável x_3, fazendo $x_3 \leq 2$ ou $x_3 \geq 3$. A lista de nós ativos passa a ser $L = \{S_1, S_2\}$.

Escolhendo um nó: A lista de nós ativos $L = \{S_1, S_2\}$ contém dois conjuntos. Arbitrariamente, escolhemos S_1.

***Bounding*:** Resolvemos a relaxação linear $R(S_1)$ associada a S, ou seja, resolvemos o problema de programação linear abaixo:

$$\begin{array}{rrrrrrr} S_1: & \overline{z}_1 = & Max & 4x_1 & -x_2 & & \\ & & s.a: & 7x_1 & -2x_2 & \leq & 14 \\ & & & & x_2 & \leq & 3 \\ & & & 2x_1 & -2x_2 & \leq & 3 \\ & & & x_1 & & \leq & 2 \end{array}$$

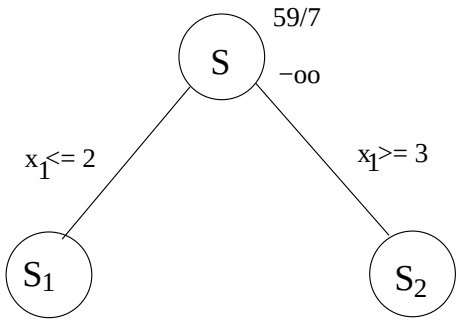


Figura 11.7: Quebra do primeiro nó da árvore de enumeração B&B

cuja solução ótima é $(x_1^1, x_2^1) = (2, \frac{1}{2})$ que por sua vez induz o limite superior $\overline{z} = \frac{15}{2}$.

***Branching*:** Quebramos S_1 em dois conjuntos: $S_{11} = S_1 \cap \{x : x_2 \leq 0\}$ e $S_{12} = S_1 \cap \{x : x_2 \geq 1\}$, o que faz com que a lista de nós ativos se torne $L = \{S_2, S_{11}, S_{12}\}$. O resultado desta subdivisão está indicado na Figura 11.8.

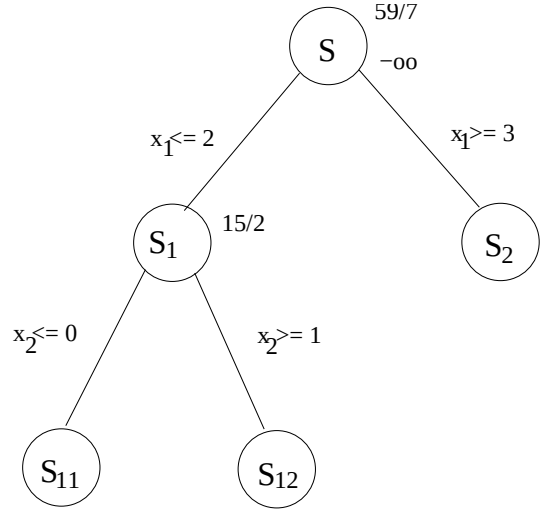


Figura 11.8: Dividindo S_1 em S_{11} e S_{12}

Escolhendo um nó: Arbitrariamente escolhemos o nó identificado por S_2 dentre a lista de nós ativos $L = \{S_{11}, S_{12}, S_2\}$.

***Bounding*:** Resolvemos a relaxação linear $R(S_2)$, correspondendo a relaxação do prob-

lema:

$$\begin{array}{rrrrrrr} S_2: & z_2 = & Max & 4x_1 & -x_2 & & \\ & & s.a: & 7x_1 & -2x_2 & \leq & 14 \\ & & & & x_2 & \leq & 3 \\ & & & 2x_1 & -2x_2 & \leq & 3 \\ & & & x_1 & & \geq & 3 \\ & & & x \in \mathbb{Z}^2_+ & & & \end{array}$$

Uma vez que o problema S_2 é infactível, ou seja, $\overline{z}_2 = -\infty$, podemos cortar o nó S_2 em função da inviabilidade.

Escolhendo um nó: A lista de nós ativos passou a ser $L = \{S_{11}, S_{12}\}$. Arbitrariamente escolhemos S_{12}.

***Bounding*:** Resolvemos a relaxação $R(S_{12})$ do problema cujo espaço de soluções factíveis é $S_{12} = S \cap \{x : x_1 \leq 2 \text{ e } x_2 \geq 1\}$, obtendo a solução $\overline{x}_{12} = (2, 1)$ a qual gera o limite superior $\overline{z}_{12} = 7$. Já que a solução obtida para $R(S_{12})$ é inteira, produzimos o primeiro limite inferior, ou seja, $underline z_1 2 = 7$ que pode ser progado para os demais nós da árvore. Portanto o nó S_{12} foi cortado por otimalidade como está indicado na Figura 11.9.

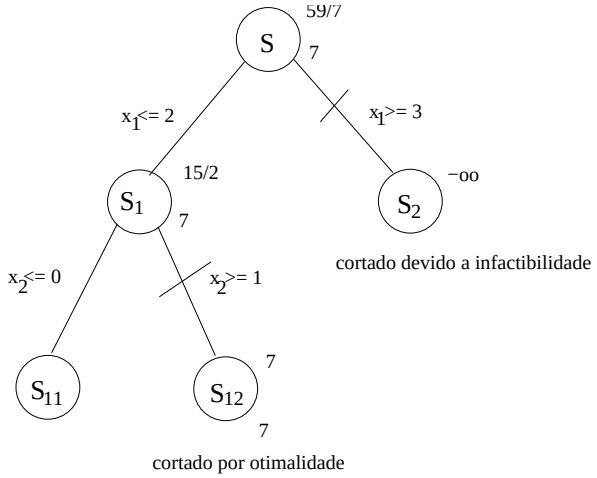


Figura 11.9: Cortando o nó S_{12} por meio da condição de otimalidade

Escolhendo um nó: Apenas S_{11} está ativo.

***Bounding*:** Observe primeiramente que $S_{11} = S \cap \{x : x_1 \leq 2, x_2 = 0\}$. Resolvendo are relaxação linear $R(S_{11})$, obtemos a solução $\overline{x}^{11} = (\frac{3}{2}, 0)$ a qual produz o limite superior $\overline{z}_{11} = 6$. Levando em conta que $\overline{z}_{11} = 6 < 7 = \underline{z}$, o nó é cortado.

Escolhendo um nó: Já que a lista está vazia, concluímos que a solução ótima é $x^* = (2, 1)$ com $z^* = 7$.

11.6 Exercícios

Ex 11.1 Aplique o algoritmo "Branch-and-Bound"ao problema abaixo descrito. Faça uso de um pacote de otimização linear como, por exemplo, `lp_solve` na resolução das relaxações lineares. Desenhe a árvore B&B e escreva os limites (superior e inferior) para cada nó. Tente pré-processar o problema, em outras palavras, tente simplificá-lo.

$$
\begin{array}{lrcl}
Maximize & 77.9x_1 + 76.8x_2 + 89.6x_3 + 97.1y_1 + 31.3y_2 & = & x_0 \\
Sujeito\ a: & & & \\
 & 60.9x_1 + 68.9x_2 + 69.0x_3 - 56.9y_1 + 22.5y_2 & = & 86.5 \\
 & -86.8x_1 + 32.7x_2 + 24.3x_3 + 13.8y_1 - 12.6y_2 & \leq & 77.3 \\
 & 10.9x_1 + 3.6x_2 - 40.8x_3 + 43.9y_1 + 7.1y_2 & \leq & 82.3 \\
 & x_1, x_2, x_3 \geq 0 \ e\ inteiro & & \\
 & y_1, y_2 \geq 0 & &
\end{array}
$$

Capítulo 12

Algoritmos de Planos de Corte e Desigualdades Fortes

Aqui iniciaremos o estudo de algoritmos *branch-and-cut* (planos de corte) que adicionam desigualdades válidas à relaxação linear até que se obtenha uma solução inteira. Em outras palavras, o algoritmo *branch-and-cut* se assemelha ao algoritmo *branch-and-bound* mas procurar aproximar o poliedro inteiro (o fecho convexo das soluções inteiras) de uma forma iterativa. Estudaremos ainda os chamados cortes de Gomory, que podem ser aplicados a qualquer problema linear inteiro (ou misto), e cortes específicos para alguns problemas de interesse.

12.1 Introdução a Planos de Corte

Inicialmente, lembramos o problema inteiro na sua forma geral:

$$IP: \qquad Max\{c^T x : x \in X\}, \quad \text{onde } X = \{x : Ax \leq b, x \in \mathbb{Z}^n_+\}$$

Um resultado de significância prática e teória pode ser resumido na propoposição a seguir.

Proposição 14 *$conv(X) = \{x : \tilde{A}x \leq \tilde{b}, x \geq 0\}$ é um poliedro.*

Este resultado nos diz que, pelo menos em teoria, IP pode ser reformulado como um problema de programação linear:

$$LP: \qquad Max\{c^T x : \tilde{A}x \leq \tilde{b}, x \geq 0\}$$

Note que qualquer ponto extremo de LP é uma solução ótima de IP. Para alguns problemas, tal como o problema de fluxo em redes, conhecemos uma descrição completa de

$conv(X)$. Em geral, e particularmente para problemas NP-difíceis, não há esperança de se encontrar uma descrição completa de $conv(X)$, mesmo porque esta pode conter um número exponencial de restrições. Dado um problema NP-difícil, aqui nos preocupamos em encontrar uma aproximação para $conv(X)$ para uma dada instância, podendo esta aproximação ser construída gradualmente à medida que se insere novas restrições válidas e não triviais, preferencialmente desigualdades que toquem no poliedro que descreve $conv(X)$. O conceito fundamental que já utilizamos informalmente é o de desigualdade válida, agora formalizado pela definição abaixo.

Definição 9 *Uma desigualdade $\pi^T x \leq \pi_0$ é uma desigualdade válida para $x \subseteq \mathbb{R}^n$ se $\pi^T x \leq \pi_0$ para todo $x \in X$.*

Surgem imediatamente duas questões relevantes:

a) Quais desigualdades são "boas" e "úteis"?

b) Se conhecemos uma família de desigualdades para um certo problema, como podemos utilizá-las de uma forma eficaz?

Ambas as questões acima serão respondidas nas próximas seções, cujas respostas implicam diretamente na eficácia de uma abordagem por planos de cortes.

12.2 Exemplos de Desigualdades Válidas

No que segue, apresentamos através de exemplos alguns tipos de desigualdades válidas que expressam condições lógicas.

12.2.1 Um conjunto 0-1 puro

Considere o conjunto da mochila 0-1 onde o conjunto de soluções factíveis X (i.e., as soluções que satisfazem a restrição *knapsack*) é dado por:

$$X = \{x \in B^5 : 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leq -2\}$$

Para $x_2 = x_4 = 0$, temos que a desigualdade $3x_1 + 2x_3 + x_5 \leq -2$ se torna impossível de ser satisfeita. Portanto, qualquer concluímos que uma solucção factível deve satisfazer $x_2 + x_4 \geq 1$. Se $x_1 = 1$ e $x_2 = 0$, então a desigualdade que resulta $2x_3 - 3x_4 + x_5 \leq -5$ não poderá ser satisfeita. Portanto $x_1 \leq x_2$ é uma desigualdade válida, podendo esta

ser introduzida na formulação de X. A partir das deduções desenvolvidas podemos propor uma formulação revisada para o problema em questão:

$$\begin{aligned} X = \{x \in B^5 : \ & 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leq -2 \\ & x_2 + x_4 \geq 1 \\ & x_1 \leq x_2\} \end{aligned}$$

12.2.2 Um Conjunto 0-1 Misto

O exemplo de conjunto de soluções mistas (contínuas e discretas) consiste no conjunto X:

$$X = \{(x, y) : x \leq 9999y, 0 \leq x \leq 5, y \in \mathbb{B}\}$$

É fácil verificar que a desigualdade $x \leq 5y$ é válida.

12.2.3 Um Conjunto Inteiro Misto

Considere o conjunto

$$X = \{(x, y) : x \leq x \leq 10y, 0 \leq x \leq 14, y \in \mathbb{Z}_+\}$$

O leitor pode se convencer da validade da desigualdade

$$x \leq 14 - 4(2 - y)$$

O espaço de soluções factíveis X juntamente com a desigualdade válida é ilustrado na Figura 12.1.

12.2.4 Conjunto Combinatório

Seja X o conjunto dos vetores de incidência de problemas de emparelhamento ("matchings"), mais precisamente:

$$X = \{x \in \mathbb{Z}_+^{|E|} : \sum_{e \in \delta(i)} x_e \leq 1 \qquad \text{para todo } i \in V\}$$

onde $G = (V, E)$ é um grafo não direcionado, e $\delta(i) = \{e \in E : e = (i, j) \text{ para algum } j \in V\}$. Seja ainda $T \subseteq V$ um conjunto qualquer com cardinalidade ímpar. O número de arestas tendo ambos os extremos em T é no máximo $(|T| - 1)/2$,

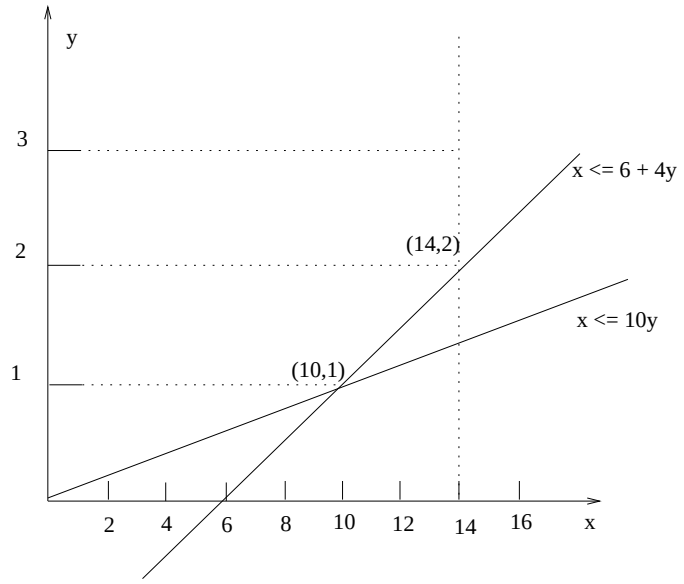


Figura 12.1: Espaço de soluções factíveis e desigualdade válida $x \leq 14 - 4(2 - y)$

portanto obtemos a desigualdade:

$$\sum_{e \in E(T)} x_e \leq \frac{|T| - 1}{2}$$

É propício mencionarmos que $conv(X)$ pode ser obtido através da inserção de todas as desigualdades da forma acima, isto é, $conv(X)$ é precisamente o poliedro dado por:

$$conv(X) = \{x \in \mathbb{R}_+^{|E|} : \begin{array}{ll} \sum_{e \in \delta(i)} x_e \leq 1 & \forall i \in V \\ \sum_{e \in E(T)} x_e \leq \frac{|T|-1}{2} & \forall T \subseteq V, |T| \text{ ímpar e } |T| \geq 3\} \end{array}$$

12.2.5 Arredondamento Inteiro

Considere a região $X = P \cap \mathbb{Z}^4$ e $P = \{x \in \mathbb{R}_+^4 : 13x_1 + 20x_2 + 11x_3 + 6x_4 \geq 72\}$. Dividindo a desigualdade por 11, obtemos a seguinte desigualdade válida para P:

$$\frac{13}{11}x_1 + \frac{20}{11}x_2 + \frac{11}{11}x_3 + \frac{6}{11}x_4 \geq \frac{72}{11}$$

uma vez que $x \geq 0$, podemos arredondar os coeficientes de x para o inteiro mais próximo:

$$\begin{array}{rcrcrcrcl} \lceil \frac{13}{11} \rceil x_1 & + & \lceil \frac{20}{11} \rceil x_2 & + & x_3 & + & \lceil \frac{6}{11} \rceil x_4 & \geq & \frac{72}{11} \quad \Rightarrow \\ 2x_1 & + & 2x_2 & + & x_3 & + & x_4 & \geq & \frac{72}{11} \quad \Rightarrow \\ 2x_1 & + & 2x_2 & + & x_3 & + & x_4 & \geq & \lceil \frac{72}{11} \rceil \quad \Rightarrow \\ 2x_1 & + & 2x_2 & + & x_3 & + & x_4 & \geq & 7 \end{array}$$

Note que um inteiro que é maior ou igual a $6 + \frac{6}{11}$ deve ser maior ou igual a 7.

12.2.6 Arredondamento Inteiro Misto

Considere o mesmo exemplo anterior mas com a adição de uma variação contínua. Seja $X = P \cap (\mathbb{Z}^4 \times \mathbb{R})$ onde

$$P = \{(y, s) \in \mathbb{R}^4_+ \times \mathbb{R}_+ : 13y_1 + 20y_2 + 11y_3 + 6y_4 + s \geq 72\}$$

Novamente dividindo a desigualdade por 11, obtemos

$$\begin{array}{rcrcrcrcrcl} \frac{13}{11} y_1 & + & \frac{20}{11} y_2 & + & \frac{11}{11} y_3 & + & \frac{6}{11} y_4 & + & \frac{s}{11} & \geq & \frac{72}{11} \Rightarrow \\ \frac{13}{11} y_1 & + & \frac{20}{11} y_2 & + & \frac{11}{11} y_3 & + & \frac{6}{11} y_4 & & & \geq & \frac{72-s}{11} \end{array}$$

Podemos observar que:

$$\begin{array}{rcrcrcrcrcll} 2y_1 & + & 2y_2 & + & y_3 & + & y_4 & \geq & \lceil \frac{72}{11} \rceil & = & 7 & \text{se } s = 0 \\ 2y_1 & + & 2y_2 & + & y_3 & + & y_4 & \geq & \lceil \frac{72-6}{11} \rceil & = & 6 & \text{se } s = 6 \end{array}$$

Isto sugere que a desigualdade abaixo é válida:

$$2y_1 + 2y_2 + y_3 + y_4 + \alpha s \geq 7$$

para algum α. A desigualdade acima pode ser verificada como válida para $\alpha \geq \frac{1}{6}$.

12.3 Teoria de Desigualdades Válidas

A presente seção aprofunda os conceitos sobre desigualdades válidas apresentados por meio de exemplos nas seções anteriores.

12.3.1 Desigualdades Válidas para Problemas Lineares

Dado um poliedro $P = \{x : Ax \leq b, x \geq 0\}$ e uma desigualdade $\pi^T x \leq \pi_0$, a primeira questão é se a desigualdade (π, π_0) é válida para P. A proposição abaixo provê condições que garantem a validade de (π, π_0).

Proposição 15 *$\pi^T x \leq \pi_0$ é válida para $P = \{x : Ax \leq b, x \geq 0\} \neq \emptyset$ se, e somente se,*

a) existe $u \geq 0$ e $v \geq 0$ tal que $u^T A - v^T = \pi^T$ e $u^T b \leq \pi_0$, ou

b) existe $u \geq 0$ tal que $u^T A \geq \pi^T$ e $u^T b \leq \pi_0$

Prova (b $\Rightarrow$) Se existe $u \geq 0$ tal que $u^T A \geq \pi^T$ e $u^T b \leq \pi_0$, então para qualquer $x \in P$,

$$Ax \leq b \Leftrightarrow u^T Ax \leq u^T b \Leftrightarrow$$
$$\pi^T x \leq u^T Ax \leq u^T b \leq \pi_0 \Rightarrow (\pi, \pi_0) \text{ é uma desigualdade válida.}$$

□

12.3.2 Desigualdades Válidas para Problemas Inteiros

Proposição 16 *A desigualdade $y \leq \lfloor b \rfloor$ é válida para $X = \{y \in \mathbb{Z} : y \leq b\}$.*

Exemplo: Podemos utilizar a Proposição 15 para gerar desigualdades válidas para um poliedro dado pelas restrições abaixo:

$$\begin{aligned} 7x_1 - 2x_2 &\leq 14 && (12.1)\\ x_2 &\leq 3 && (12.2)\\ 2x_1 - 2x_2 &\leq 3 && (12.3)\\ x &\geq 0, \quad x \text{ inteiro} && (12.4) \end{aligned}$$

i) Multiplicamos as restrições por um vetor de pesos não negativos $u = (\frac{2}{7}, \frac{37}{63}, 0)$ e obtemos a desigualdade válida:

$$2x_1 + \frac{1}{63}x_2 \leq \frac{121}{21}$$

ii) Reduzimos os coeficientes do lado esquerdo ao inteiro mais próximo e obtemos:

$$2x_1 + 0x_2 \leq \frac{121}{21}$$

iii) Uma vez que o lado esquerdo só pode assumir valores inteiros, podemos reduzir o valor do lado direito ao inteiro mais próximo, obtendo a desigualdade

$$2x_1 \leq \lfloor \frac{121}{21} \rfloor = 5 \;\Rightarrow\; x_1 \leq \frac{5}{2} \;\Rightarrow\; x_1 \leq 2$$

12.3.3 Procedimento Chvátal-Gomory para Geração de Desigualdades Válidas

O procedimento CG (Chvátal-Gomory) é uma formalização dos passos ilustrados na Seção 12.3.1 que permite *gerar todas as desigualdades válidas de um problema inteiro.* Seja $X = P \cap \mathbb{Z}^n$ o conjunto de soluções onde $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ é um poliedro, e $A \in \mathbb{R}^{m \times n}$ é uma matriz com colunas $\{a_1, a_2, \ldots, a_n\}$. Dado $u \in \mathbb{R}^m_+$, o procedimento consiste dos passos que seguem.

Passo 1: A desigualdade $\sum_{j=1}^{n} u^T a_j x_j \leq u^T b$ é válida para P uma vez que $u \geq 0$ e $\sum_{j=1}^{n} a_j x_j \leq b$

Passo 2: A desigualdade $\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j \leq u^T b$ é válida para P já que $x \geq 0$.

Passo 3: A desigualdade $\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j \leq \lfloor u^T b \rfloor$ é válida para P uma vez que x é inteiro, e portanto $\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j$ é inteiro

Importante: um fato surpreendente é que o procedimento acima é capaz de gerar todas as desigualdades válidas para o problema inteiro.

Teorema 13 *Toda desigualdade válida para X pode ser obtida por meio da aplicação de um número finito de vezes do procedimento Chavátal-Gomory.*

12.4 O Algoritmo de Planos de Corte

Suponha que $X = P \cap \mathbb{Z}^n$ e que uma família $\mathcal{F}$ de desigualdades válidas $\pi^T x \leq \pi_0$, $(\pi, \pi_0) \in \mathcal{F}$, seja conhecida para X. Tipicamente $\mathcal{F}$ contém um número muito grande de elementos (possivelmente exponencial) que não podem ser introduzidas na formulação a priori. Além disso, do ponto de vista prático, não estamos interessados em encontrar uma representação completa do fecho convexo, $conv(X)$, mas sim uma boa aproximação em torno da solução ótima.

Abaixo descreveremos o algoritmo básico de planos de corte para IP, $max\{c^T x; x \in X\}$, que gera cortes "úteis" a partir de $\mathcal{F}$

Inicialização Defina $t = 0$ e $P^0 = P$

Iteração t Resolva o problema linear $\overline{z}^t = Max\{c^T x : x \in P^t\}$
Seja x^t a solução ótima
Se $x^t \in \mathbb{Z}^n$, pare pois x^t é uma solução ótima para IP
Se $x^t \notin \mathbb{Z}^n$, encontre uma desigualdade $(\pi, \pi_0) \in \mathcal{F}$
tal que $\pi^T x^t > \pi_0$
Se uma desigualdade (π, π_0) foi encontrada,
então faça $P^{t+1} = P^t \cap \{x : \pi^T x \leq \pi_0\}$,
incremente t e repita
Caso contrário, pare.

Se o algoritmo termina sem encontrar uma solução inteira, pelo menos $P^t = P \cap \{x : \pi_i^T \leq \pi_{i0}, i = 1, 2, \ldots, t\}$ é uma formulação mais "apertada" do que P. Podemos então proceder a partir de P^t com um algoritmo de *branch-and-bound*.

12.5 Algoritmo de Planos de Corte Usando Cortes de Gomory Fracionários

Aqui consideraremos o problema inteiro:

$$Max\{c^T x : Ax = b, x \geq 0 \text{ e inteiro}\}$$

O princípio básica é resolver a relaxação linear e encontrar uma base ótima; a partir desta base se escolhe uma variável básica que não seja inteira, e então geramos uma desigualdade Chavátal-Gomory associada a esta variável básica visando "cortá-la", ou seja, eleminá-la do poliedro de relaxação. Supomos que, dada uma base ótima, o problema pode ser reescrito na forma:

$$\begin{aligned} Max \quad & \overline{a}_{oo} + \sum_{j \in NB} \overline{a}_{oj} x_j \\ \text{Sujeito a:} \quad & x_{Bu} + \sum_{j \in NB} \overline{a}_{uj} x_j = \overline{a}_{uo} \text{ para } u = 1, \ldots, m \\ & x \geq 0 \text{ e inteiro} \end{aligned}$$

onde 1) $\overline{a}_{oj} \leq 0$ para $j \in NB$, 2) $\overline{a}_{uo} \geq 0$ para $u = 1, \ldots, m$, e 3) NB é o conjunto de variáveis não básicas, portanto $\{B_u : u = 1, \ldots, m\} \cup NB = \{1, \ldots, n\}$.

Se a solução básica ótima x^* não for inteira, então deve existir uma linha u tal que $\overline{a}_{uo} \notin \mathbb{Z}$. Escolhendo esta linha, o corte de Chavátal-Gomory para a linha u fica:

$$x_{Bu} + \sum_{j \in NB} \lfloor \overline{a}_{uj} \rfloor x_j \leq \lfloor \overline{a}_{uo} \rfloor \tag{12.5}$$

Reescrevendo (12.5) de forma a eliminar x_{Bu}, obtemos:

$$x_{Bu} = \overline{a}_{uo} - \sum_{j \in NB} \overline{a}_{uj} x_j \tag{12.6}$$

$$\overline{a}_{uo} - \sum_{j \in NB} \overline{a}_{uj} x_j + \sum_{j \in NB} \lfloor \overline{a}_{uj} \rfloor x_j \leq \lfloor \overline{a}_{uo} \rfloor \quad \Rightarrow \tag{12.7}$$

$$\sum_{j \in NB} \left(\overline{a}_{uj} - \lfloor \overline{a}_{uj} \rfloor \right) x_j \geq \overline{a}_{uo} - \lfloor \overline{a}_{uo} \rfloor \tag{12.8}$$

De uma forma mais compacta, podemos reescrever o corte (12.8) como:

$$\sum_{j \in NB} f_{uj} x_j \geq f_{uo} \tag{12.9}$$

onde $f_{uj} = \overline{a}_{uj} - \lfloor \overline{a}_{uj} \rfloor$ e $f_{uo} = \overline{a}_{uo} - \lfloor \overline{a}_{uo} \rfloor$. Uma vez que $0 \leq f_{uj} < 1$ e $0 < f_{uo} < 1$, e $x_j^* = 0$ para toda a variável $j \in NB$ na solução x^*, a desigualdade (12.9) corta a solução corrente x^*.

12.5.1 Exemplo

Considere o problema inteiro a seguir

$$\begin{array}{rrrrrrl} z = Max & 4x_1 & - & x_2 & & & \\ s.a.: & 7x_1 & - & 2x_2 & \leq & 14 & \\ & & & x_2 & \leq & 3 & \\ & 2x_1 & - & 2x_2 & \leq & 3 & \\ & x_1, & & x_2 & \geq & 0, & \text{inteiros} \end{array} \tag{12.10}$$

Após adicionarmos variáveis de folga x_3, x_4 e x_5, podemos aplicar o método simplex

e obter a solução ótima

$$\begin{array}{rrrrrrrrrrrrr} z = Max & \frac{59}{7} & & & - & \frac{4}{7}x_3 & - & \frac{1}{7}x_4 & & & & & \\ s.a: & & x_1 & & + & \frac{1}{7}x_3 & + & \frac{4}{7}x_4 & & & = & \frac{20}{7}\ (=2.8571) & \\ & & & x_2 & & & + & x_4 & & & = & 3 & (12.11) \\ & & & & - & \frac{2}{7}x_3 & + & \frac{10}{7}x_4 & + & x_5 & = & \frac{23}{7}\ (=3.2857) & \\ & & x_1, & x_2, & & x_3, & & x_4, & & x_5 & \geq & 0 \text{ e inteiros} & \end{array}$$

A solução ótima da relaxação linear é $x^* = (\frac{20}{7}, 3, \frac{27}{7}, 0, 0) \notin \mathbb{Z}^5_+$, portanto usamos a primeira linha de (12.11), na qual a variável básica x_1 é fracionária, para gerar o corte

$$x_1 + \lfloor \tfrac{1}{7} \rfloor x_3 + \lfloor \tfrac{2}{7} \rfloor x_4 \leq \lfloor \tfrac{20}{7} \rfloor \quad \Rightarrow \quad x_1 \leq 2$$

$$\begin{array}{rl} x_1 + s = 2,\ x_1 = \frac{20}{7} - \frac{1}{7}x_3 - \frac{2}{7}x_4 & \Rightarrow \quad \frac{20}{7} - \frac{1}{7}x_3 - \frac{2}{7}x_4 + s = 2 \\ & \Rightarrow \quad s = 2 - \frac{20}{7} + \frac{1}{7}x_3 + \frac{2}{7}x_4 \\ & \Rightarrow \quad s = -\frac{6}{7} + \frac{1}{7}x_3 + \frac{2}{7}x_4 \\ & \quad \text{com } s, x_3, x_4 \geq 0 \text{ e inteiros.} \end{array}$$

Adicionando a variável s e o corte $s = -\frac{6}{7} + \frac{1}{7}x_3 + \frac{2}{7}x_4$, podemos obter a nova solução ótima:

$$\begin{array}{rrrrrrrrrrrrr} z = Max & \frac{15}{2} & & & & & - & \frac{1}{2}x_5 & - & 3s & & & \\ s.a: & & x_1 & & & & & & + & s & = & 2 & \\ & & & x_2 & & & - & \frac{1}{2}x_5 & + & s & = & \frac{1}{2} & \\ & & & & x_3 & & - & x_5 & - & 5s & = & 1 & (12.12) \\ & & & & & x_4 & + & \frac{1}{2}x_5 & + & 6s & = & \frac{5}{2} & \\ & & x_1, & x_2, & x_3, & x_4, & & x_5, & & s & \geq & 0 \text{ e inteiros.} & \end{array}$$

A solução ótima da relaxação linear acima (12.12), $x = (2, \frac{1}{2}, 1, \frac{5}{2}, 0, 0)$, continua fracionária uma vez que x_2 e x_4 são fracionários. A aplicação do corte de Gomory fracionário na segunda linha produz:

$$\begin{array}{rl} x_2 + \lfloor -\frac{1}{2} \rfloor x_5 + \lfloor 1 \rfloor s \leq \lfloor \frac{1}{2} \rfloor & \Rightarrow \quad x_2 - x_5 + s \leq 0 \\ & \Rightarrow \quad x_2 - x_5 + s + t = 0,\ t \geq 0 \\ & \Rightarrow \quad (\frac{1}{2} + \frac{1}{2}x_5 - s) - x_5 + s + t = 0 \\ & \Rightarrow \quad t - \frac{1}{2}x_5 = -\frac{1}{2},\ t \geq 0 \end{array}$$

Após adicionarmos a varivél $t \geq 0$ e o corte $t - \frac{1}{2}x_5 = -\frac{1}{2}$ e reotimizarmos, obtemos a

tabela simplex:

$$
\begin{array}{llccccccccccl}
z = Max & 7 & & & & & & - & 3s & - & t & & \\
s.a: & & x_1 & & & & & + & s & & & = & 2 \\
& & & x_2 & & & & + & s & - & t & = & 1 \\
& & & & x_3 & & & - & 5s & - & 2t & = & 2 \\
& & & & & x_4 & & + & 6s & + & t & = & 2 \\
& & & & & & x_5 & & & - & t & = & 1 \\
& & x_1, & x_2, & x_3, & x_4, & x_5, & & s, & & t & \geq & 0 \text{ e inteiros}
\end{array}
$$

A solução obtida é ótima pois os valores de todas as variáveis são inteiros. A solução ótima é $x^* = (2, 1, 2, 2, 1)$ cujo valor da função objetivo é $z^* = 7$.

12.6 Desigualdades Disjuntivas

Seja $X = X^1 \cup X^2$, onde $X^i \subseteq \mathbb{R}^n_+$, uma disjunção (união) de dois conjuntos X^1 e X^2. Alguns resultados importantes serão enunciados abaixo.

Proposição 17 *Se $\sum_{j=1}^n \pi_j^i x_j \leq \pi_0^i$ é uma desigualdade válida para X^i, $i = 1, 2$, então a desigualdade*

$$\sum_{j=1}^{n} \pi_j x_j \leq \pi_0$$

é válida para X se $\pi_j \leq Min\{\pi_j^1, \pi_j^2\}$ para $j = 1, \ldots, n$ e $\pi_0 \geq Max\{\pi_0^1, \pi_0^2\}$

Proposição 18 *Se $P^i = \{x \in \mathbb{R}^n_+ : A^i x \leq b^i\}$ para $i = 1, 2$ são poliedros não-vazios, então (π, π_0) é uma desigualdade válida para $conv(P^1 \cup P^2)$ se e somente se existem $u_1, u_2 \geq 0$ tal que:*

$$\begin{aligned}
\pi &\leq u_1^T A^1 \\
\pi &\leq u_2^T A^2 \\
\pi_0 &\geq u_1^T b^1 \\
\pi_0 &\geq u_2^T b^2
\end{aligned}$$

12.6.1 Exemplo

Sejam $P^1 = \{x \in \mathbb{R}^2 : -x_1 + x_2 \leq 1,\ x_1 + x_2 \leq 5\}$ e $P^2 = \{x \in \mathbb{R}^2 : x_2 \leq 4,\ -2x_1 + x_2 \leq -6,\ x_1 - 3x_2 \leq -2\}$ dois poliedros. Fazendo $u_1 = (2, 1)$ e $u_2 = (\frac{5}{2}, \frac{1}{2}, 0)$ e depois aplicando a Proposição 18 obtemos:

$$u_1^T A^1 = \begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 3 \end{bmatrix} \qquad u_1^T b^1 = 7$$

$$u_2^T A^2 = \begin{bmatrix} \frac{5}{2} & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -2 & 1 \\ 1 & -3 \end{bmatrix} = \begin{bmatrix} -1 & 3 \end{bmatrix} \qquad u_2^T b^2 = 7$$

Isto nos permite obter a desigualdade $-x_1 + 3x_2 \leq 7$ válida para $P^1 \cup P^2$. A Figura 12.2 ilustra esta desigualdade.

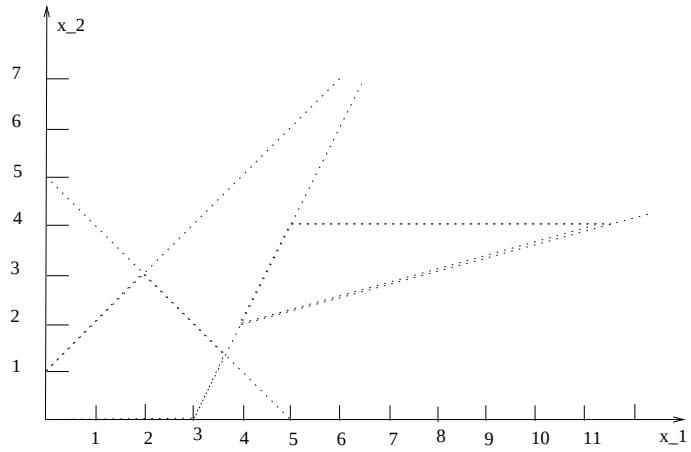


Figura 12.2: Desigualdades disjuntivas

12.6.2 Desigualdades Disjuntivas para Problemas 0-1

Especializando ainda mais, podemos nos restringir a problemas 0-1, onde $X = P \cap \mathbb{Z}^n \subseteq \{0,1\}^n$ e $P = \{x \in \mathbb{R}^n : Ax \leq b, 0 \leq x \leq 1\}$. Seja $P^0 = P \cap \{x \in \mathbb{R}^n : x_j = 0\}$ e $P^1 = P \cap \{x \in \mathbb{R}^n : x_j = 1\}$ para algum $j \in \{1, \ldots, n\}$.

Proposição 19 *A desigualdade (π, π_0) é válida para $conv(P^0 \cup P^1)$ se existe $u_i \in \mathbb{R}^m_+$, $v_i \in \mathbb{R}^n_+$, $w_i \in \mathbb{R}^1_+$ para $i = 0, 1$ tal que*

$$\begin{aligned} \pi &\leq u_0^T A + v_0 + w_0^T e_j \\ \pi &\leq u_1^T A + v_1 + w_1^T e_j \\ \pi_0 &\geq u_0^T b + 1^T v_0 \\ \pi_0 &\geq u_1^T b + 1^T v_1 - w_1 \end{aligned}$$

Prova: Aplique a Proposição 18 com $P^0 = \{x \in \mathbb{R}^n_+ : Ax \leq b, x \leq 1, x_j \leq 0\}$ e $P^1 = \{x \in \mathbb{R}^n_+ Ax \leq b, x \leq 1, -x_j \leq -1\}$ [0,7,7)

Exemplo

Considere a seguinte instância do problema da mochila

$$\begin{array}{lrcrcrcrcl} Max & 12x_1 & + & 14x_2 & + & 7x_3 & + & 12x_4 & & \\ s.a: & 4x_1 & + & 5x_2 & + & 3x_3 & + & 6x_4 & \leq & 8 \end{array}$$

$$x \in \mathbb{B}^4$$

cuja solução linear ótima é $x^* = (1, 0.8, 0, 0)$. Uma vez que $x_2^* = 0.8$ é fracionário, escolhemos $j = 2$ e definimos P^0 e P^1, e procuramos a desigualdade (π, π_0) que é violada conforme a Proposição 19. Para isso, resolvemos um problema de programação linear maximizando $\pi x^* - \pi_0$ sobre o poliedro que descreve os coeficientes das desigualdades válidas dadas pela Proposição 19.

$$Max \quad 1.0\pi_1 + 0.8\pi_2 - \pi_0$$

$$s.a: \begin{cases} \pi_1 \leq 4u^0 + v_1^0 \\ \pi_1 \leq 4u^1 + v_1^1 \\ \pi_2 \leq 5u^0 + v_2^0 + w^0 \\ \pi_2 \leq 5u^1 + v_2^1 - w^1 \end{cases}$$

$$\begin{cases} \pi_3 \leq 3u^0 + v_3^0 \\ \pi_3 \leq 3u^1 + v_3^1 \end{cases}$$

$$\begin{cases} \pi_4 \leq 6u^0 + v_4^0 \\ \pi_4 \leq 6u^1 + v_4^1 \end{cases}$$

$$\begin{cases} \pi_0 \geq 8u^0 + v_1^0 + v_2^0 + v_3^0 + v_4^0 \\ \pi_0 \geq 8u^1 + v_1^1 + v_2^1 + v_3^1 + v_4^1 - w^1 \end{cases}$$

$$u^0, u^1, v^0, v^1, w^0, w^1 \geq 0$$

Objetivando tornar o espaço de soluções factíveis limitado, devemos introduzir um critério de normalização. Duas possibilidades são:

a) $\sum_{j=1}^{n} \pi_j \leq 1$

b) $\pi_0 = 1$

Obtemos então a seguinte desigualdade de corte:

$$x_1 + \frac{1}{4}x_2 \leq 1.$$

Para P^0, a desigualdade é uma combinação das restrições $x_1 \leq 1$ e $x_2 \leq 0$ com $v_1^0 = 1$ e $w^0 = \frac{1}{4}$ respectivamente. Para P^1, ela é uma combinação da desigualdade $4x_1 + 5x_2 + 3x_3 + 6x_4 \leq 8$ e $-x_2 \leq -1$ com $u^1 = \frac{1}{4}$ e $w^1 = 1$, respectivamente.

12.7 Exercícios

Ex 12.1 Para os três conjuntos X abaixo, encontre algebricamente uma desigualdade válida cuja adição nos dá $conv(X)$. Verifique graficamente que a adição da desigualdade nos dá $conv(X)$.

i. $X = \{x \in \mathbb{B}^2 : 3x_1 - 4x_2 \leq 1\}$

ii. $X = \{(x, y) \in \mathbb{R}_+ \times \mathbb{B} : x \leq 20y, x \leq 7\}$

iii. $X = \{(x, y) \in \mathbb{R}_+ \times \mathbb{Z}_+ : x \leq 6y, x \leq 16\}$

Ex 12.2 Em cada um dos exemplos abaixo, são dados um conjunto X e um ponto x ou (x, y). Encontre uma desigualdade válida para X que corte (elimine) o ponto.

i. $X = \{(x, y) \in \mathbb{R}_+^2 \times \mathbb{B} : x_1 + x_2 \leq 2y, x_1 \leq 1; x_2 \leq 1\}$ e $(x_1, x_2, y) = (1, 0, 1/2)$

ii. $X = \{(x, y) \in \mathbb{R}_+ \times \mathbb{Z}_+ : x \leq 9, x \leq 4y\}$ e $(x, y) = (9, 9/4)$

iii. $X = \{(x_1, x_2, y) \in \mathbb{R}_+^2 \times \mathbb{Z}_+ : x_1 + x_2 \leq 25, x_1 + x_2 \leq 8y\}$ e $(x_1, x_2, y) = (20, 5, 25/8)$

iv. $X = \{x \in \mathbb{Z}_+^5 : 9x_1 + 12x_2 + 8x_3 + 17x_4 + 13x_5 \geq 50\}$ e $x = (0, 25/6, 0, 0, 0)$

v. $X = \{x \in \mathbb{Z}_+^4 : 4x_1 + 8x_2 + 7x_3 + 5x_4 \leq 33\}$ e $x = (0, 0, 33/7, 0)$

Ex 12.3 Prove que a desigualdade $y_2 + y_3 + 2y_4 \leq 6$ é válida para $X = \{y \in \mathbb{Z}_+^4 : 4y_1 + 5y_2 + 9y_3 + 12y_4 \leq 34\}$

Capítulo 13

Programação Dinâmica: Domínio Discreto

Uma técnica poderosa para resolver problemas consiste em quebrá-los em subproblemas menores, mais fáceis de serem resolvidos. Quando quebramos um problema em subproblemas do mesmo tipo, tipicamente podemos produzir um algoritmo recursivo. Dois paradigmas de concepção de algoritmos por meio da quebra são:

Divisão e Conquista Este paradigma é amplamente adotado na concepção de algoritmos tipicamente encontrados na Ciência da Computação, podendo ser dividido em três etapas:

- quebre o problema em duas metades;
- resolva cada metade separadamente; e
- combine as metades de forma a obter uma solução completa

O algoritmo *merge-sort*, por exemplo, ordena uma cadeia de n números em $\Theta(n \log n)$ operações de comparação segundo o princípio de divisão e conquista.

Programação Dinâmica Este paradigma consiste em quebrar um problema em uma seqüência de decisões que são tomadas em estágios. Em outras palavras, o paradigma segue os passos:

- remova um elemento do problema;
- resolva o problema menor; e
- use a solução do problema menor para adicionar o elemento removido de maneira adequada, produzindo uma solução para o problema maior.

Alguns aspectos importantes sobre DP (Programação Dinâmica):

1) problemas cujas instâncias apresentam uma ordem da esquerda para a direita, como cadeias de caracteres e vértices de polígonos, são candidatos a uma solução por DP;

2) sem uma ordem, DP pode resultar em um algoritmo de tempo exponencial; e

3) DP produz uma solução ótima global.

13.1 Um Exemplo de Programação Dinâmica

Programação dinâmica é uma técnica difícil de entender, mas cuja solução, uma vez obtida, é de fácil compreensão. Em algoritmos para problemas como o de ordenação de números, corretude é mais fácil de verificar do que eficiência. Este não é o caso de *problemas de otimização*, onde deseja-se provar que um algoritmo sempre produz uma solução ótima. Algoritmos gulosos, heurísticas que executam uma busca local são *eficientes* mas apenas *ocasionalmente* encontram uma solução ótima. Algoritmos baseados em enumeração, por outro lado, avaliam direta ou indiretamente todas as possíveis soluções e portanto encontram a solução ótima, todavia o custo computacional é proibitivo.

Programação Dinâmica combina estes dois universos (algoritmos gulosos e de enumeração): a técnica sistematicamente considera todas as possíveis decisões e sempre seleciona a ótima, armazenando as consequências de todas as decisões até o presente estágio, e usando estas informações de uma forma sistemática, o trabalho computacional pode ser minimizado.

A melhor maneira de aprender DP é por meio de exemplos.

13.1.1 Calculando Números de Fibonacci

A sequência de Fibonacci é dada por

$$\begin{aligned} F_k &= F_{k-1} + F_{k-2} \\ F_0 &= 0 \\ F_1 &= 1 \end{aligned}$$

A série de Fibonacci foi inicialmente proposta e estudada por um matemático no contexto de reprodução animal. Ela apresenta aplicações em Teoria dos Números e Computação.

Uma vez que a fórmula é recursiva, podemos desenvolver um algoritmo recursivo para calcular F_k.

```
Algoritmo F(k)

if k = 0
   return 0
else
   if k = 1
      return 1
   else
      return F(k − 1) + F(k − 2)
```

Qual é a complexidade do algoritmo? Para calcularmos o número de operações elementares, inicialmente obtemos uma recorrência:

$$\begin{cases} T(n) = 1 & \text{para } n = 0, 1 \\ T(n) = T(n-1) + T(n-2) & \text{para } n \geq 2 \end{cases}$$

Limite inferior para $T(n)$:

$$\begin{aligned} T(n) &\geq T(n-2) + T(n-2) && (13.1)\\ &= 2T(n-2) && (13.2)\\ &\geq 4T(n-4) && (13.3)\\ &\geq 8T(n-6) && (13.4)\\ &\vdots && (13.5)\\ &\geq 2^k T(n-2k) && (13.6) \end{aligned}$$

Note que

$$n - 2k = 1 \Leftrightarrow k = \frac{n-1}{2}$$

portanto, $T(n) \geq 2^{\lfloor \frac{n-1}{2} \rfloor} \Rightarrow T(n) \in \Omega(2^n)$. Conlcuímos então que o algoritmo recursivo é ineficiente, levando um tempo exponencial para calcular F_k. Veja a árvore de recursão dada na Figura 13.1.

Podemos desenvolver um algoritmo mais eficiente, de tempo linear se armazenarmos os valores das instâncias menores e não recalcularmos os seus valores. O algoritmo alternativo é dado abaixo.

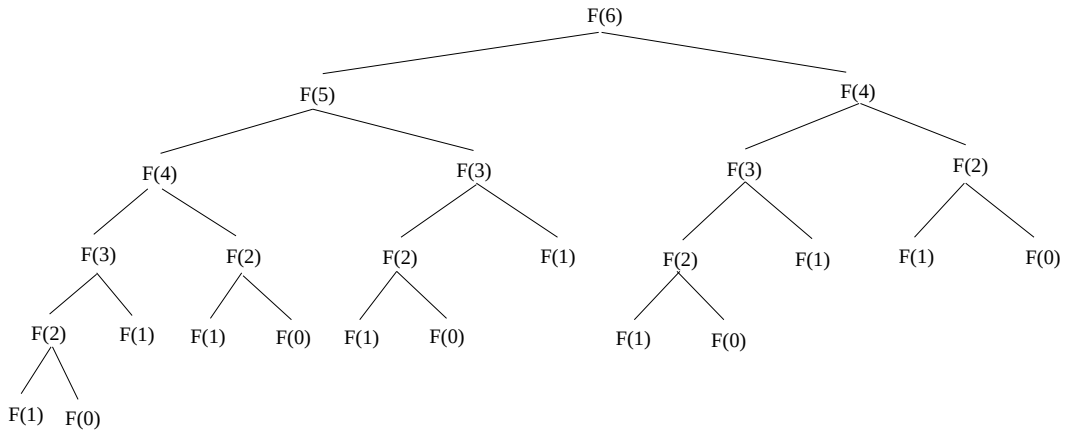


Figura 13.1: Árvore de recursão do algoritmo recursivo para cálculo do número de Fibonacci

`Algoritmo` $F_b(n)$

$F_0 = 0$
$F_1 = 1$
`for` $i \leftarrow 2$ `to` n
    $F_i \leftarrow F_{i-2} + F_{i-1}$
Return F_n

Uma vez que calculamos os números de Fibonacci, começando com os menores e armazenando os resultados, teremos já calculado os valores de F_{i-1} e F_{i-2} quando computarmos o valor de F_i. O algoritmo é executado em tempo $\Theta(n)$.

13.2 Programação Dinâmica para o Problema da Mochila

Inicialmente relembramos o problema da mochila:

$$\begin{aligned} P: z = \quad & Max \quad \sum_{j=1}^{n} c_j x_j \\ & s.\,a: \quad \sum_{j=1}^{n} a_j x_j \leq b \\ & \qquad x_j \in \{0,1\},\ j = 1, \ldots, n \end{aligned}$$

onde os coeficientes $\{a_j\}_{j=1}^{n}$ e b são inteiros positivos.

Imagine que o lado esquerdo da desigualdade assume um valor λ que varia de 0 até b, o que define estágios do problema de programação dinâmica, cujas soluções ótimas são $x_0, \ldots, x_k, \ldots, x_b$. Isto nos leva a definir o problema $P_k(\lambda)$, a solução ótima $x_k(\lambda)$ e o respectivo valor objetivo $f_k(\lambda)$ como segue:

$$\begin{aligned} P_k(\lambda) : f_k(\lambda) = \quad & Max \quad \sum_{j=1}^{k} c_j x_j \\ & s.\,a: \quad \sum_{j=1}^{k} a_j x_j \leq \lambda \\ & \qquad\quad x_j \in \{0,1\},\ j = 1, \ldots, k \end{aligned}$$

Assim, $z = f_n(b)$ nos dá a solução ótima para P. Nos resta então obter uma recursão que permite calcular $f_k(\lambda)$ em termos dos valores de $f_s(u)$ com $s \leq k$ e $u \leq \lambda$.

O que podemos dizer sobre a solução ótima x^* para o problema $P_k(\lambda)$ com valor $f_k(\lambda)$? Claramente, $x_k^* = 0$ ou $x_k^* = 1$. Considerando cada caso, temos:

1) Se $x^* = 0$, então concluímos que a solução ótima satisfaz $f_k(\lambda) = f_{k-1}(\lambda)$

2) Se $x^* = 1$, então concluímos que a solução ótima satisfaz $f_k(\lambda) = c_k + f_{k-1}(\lambda - a_k)$

Combinando os casos (1) e (2), obtemos a recorrência abaixo:

$$f_k(\lambda) = Max\{f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k)\} \tag{13.7}$$

Definindo-se os valores iniciais como $f_0(\lambda) = 0$ para $\lambda \geq 0$, pode-se utilizar a recorrência (13.7) para calcular sucessivamente os valores de $f_1, f_2, \ldots, f_n$ para todos os valores inteiros de $\lambda \in \{0, \ldots, b\}$.

A questão que resta é como encontrar a solução ótima associada ao valor ótimo. Podemos manter um indicador $P_k(\lambda)$ que assume valor 0 se $f_k(\lambda) = f_{k-1}(\lambda)$, e valor 1 caso contrário. A solução pode então ser encontrada por meio dos passos abaixo:

- se $P_n(b) = 0$, então como $f_n(b) = f_{n-1}(b)$, definimos $x_n^* = 0$ e continuamos o processo com o valor $f_{n-1}(b)$;
- se $P_n(b) = 1$, então $f_n(b) = c_n + f_{n-1}(b - a_n)$, definimos $x_n^* = 1$ e repetimos este procedimento para $f_{n-1}(b - a_n)$;
- após n iterações, obteremos a solução ótima.

13.2.1 Complexidade do algoritmo

Calculando o número de operações necessárias para obtermos $z = f_n(b)$, verificamos que o cálculo de $f_k(\lambda)$ para $\lambda = 0, 1, \ldots, b$ e $k = 1, \ldots, n$ necessita de um número

constante de operações. O algoritmo roda em tempo $\Theta(nb)$, sendo portanto pseudo-polinomial.

Exemplo: Considere a instância do problema da mochila que segue abaixo.

$$\begin{array}{rcrcrcrcrcl} z & = & \text{Maximize} & 10x_1 & + & 7x_2 & + & 25x_3 & + & 24x_4 & \\ & & \text{sujeito a:} & 2x_1 & + & x_2 & + & 6x_3 & + & 5x_4 & \leq \ 7 \end{array}$$

Para aplicar DP, calculamos os valores de $f_0(\lambda)$. A próxima coluna ($k = 1$) é calculada usando a fórmula (13.7).

Tabela 13.1: Tabelas DP para o Problema da Mochila

λ	f_0	f_1	f_2	f_3	f_4	P_1	P_2	P_3	P_4
0	0	0	0	0	0	0	0	0	0
1	0	0	7	7	7	0	1	0	0
2	0	10	10	10	10	**1**	**0**	**1**	0
3	0	10	17	17	17	1	1	0	0
4	0	10	17	17	17	1	1	0	0
5	0	10	17	17	24	1	1	0	1
6	0	10	17	25	31	1	1	1	1
7	0	10	17	32	34	1	1	1	**1**

Relembrando que $f_k(\lambda) = Max\{f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k)\}$, podemos reconstruir a solução ótima:

$$P_4(7) = 1 \Rightarrow x_4^* = 1$$
$$P_3(7-5) = P_3(2) = 0 \Rightarrow x_3^* = 0$$
$$P_2(2) = 0 \Rightarrow x_2^* = 0$$
$$P_1(2) = 1 \Rightarrow x_1^* = 1$$

A solução ótima é $x^*(1,0,0,1)$ e $f(x^*) = 34$.

13.3 Elementos de um Algoritmo DP: Sequência Crescente Mais Longa

Um algoritmo DP possui 3 componentes:

1) Formule uma resposta como uma relação de recorrência

2) Mostre que o número de valores distintos de sua recorrência é limitada por um polinômio (de preferência de baixa ordem)

3) Especifique uma ordem para avaliar a recorrência de forma que sempre se faça cálculos sobre valores já conhecidos.

Problema: dada uma sequência de n números (e.g. $(9, 5, 2, 8, 7, 3, 1, 6, 4)$), desejamos encontrar a "subsequência" mais longa cujos números estão em ordem crescente.

No problema em consideração, a sequência mais longa para $s = (9, 5, 2, 8, 7, 3, 1, 6, 4)$ é de comprimento 3, podendo ser $s_1 = (2, 3, 4)$ ou $s_2 = (2, 3, 6)$. Se estivéssemos procurando uma sequência contígüa, o problema seria facilmente resolvido. **Qual seria o número de soluções candidatas?** O número de soluções candidatas pode ser calculado pela recorrência:

$$\begin{aligned} T(n) &= n + (n-1) + (n-2) + \ldots + 1 && (13.8)\\ &= \frac{n(n-1)}{2} \Rightarrow T(n) \in \Theta(n^2) && (13.9) \end{aligned}$$

Para a situação sob consideração, qual é o número de possíveis soluções candidatas?

$$\begin{aligned} T(n) &= \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n} && (13.10)\\ &= \sum_{k=1}^{n} \binom{n}{k} && (13.11)\\ &= (1+1)^n && (13.12)\\ &= 2^n \Rightarrow T(n) \in \Theta(2^n) && (13.13) \end{aligned}$$

13.3.1 Construindo um Algoritmo

Suponha que para a sequência $(s_1, s_2, \ldots, s_{n-1})$ conhecemos o comprimento l_k, $k \in \{1, \ldots, n-1\}$ da sequência mais longa que termina no elemento s_k. **Como podemos encontrar a sequência mais longa em** $(s_1, \ldots, s_n)$**?** Isto pode ser realizado por meio da recorrência $l_n = Max\{l_k + 1 : s_k \leq s_n\}$, cujo algoritmo é dado a seguir.

```
Algoritmo LS(s)
l_0 ← 0
s_0 ← 0
p_0 ← 0 // predecessor, p_i é o índice do elemento que aparece
        imediatamente antes de s_i na sequência que termina em s_i.
for k = 1 to n do
```

$l_k = Max_{j=0,\ldots,k-1}\{e_j + 1 : s_j < s_k\}$
$p_k = ArgMax_{j=0,\ldots,k-1}\{l_j + 1 : s_j < s_k\}$
`return` $Max\{l_k : k = 1, \ldots, n\}$

Tabela 13.2: Tabelas DP para o Problema de Subseqüência Mais Longa

		1	2	3	4	5	6	7	8	9
Sequência s_i	0	9	5	2	8	7	3	1	6	4
Comprimento l_i	0	1	1	1	2	2	2	1	3	3
Predecessor p_i	0	0	0	0	2	2	3	0	6	6

O comprimento da cadeia mais longa é, portanto, $3 = Max\{l_i : i = 1, \ldots, n\}$. **Qual é o tempo de execução do nosso algoritmo?** Podemos determinar o tempo de execução através da recorrência $T(n) = T(n-1) + n$, o que nos leva a concluir que $T(n) \in O(n^2)$. Utilizando estruturas de dados como dicionários de uma forma inteligente é possível encontrar a "sequência" mais longa em tempo $O(n \log n)$.

13.4 Edição Automática de Cadeias de Caracteres ("Approximate String Matching")

Uma tarefa importante em processamento de texto é a busca por ocorrências de uma palavra no texto. Infelizmente, muitas palavras muitas vezes não são escritas corretamente.

Como poderíamos efetuar uma busca pela cadeia mais próxima de um padrão fornecido pelo usuário?

Mais especificamente, seja P uma cadeia padrão e T uma cadeia que representa o texto. A "distância" entre P e T é o menor número de operações de edição necessárias para transformar T em P, sendo as operações permitidas:

a) Substituição - dois caracteres podem diferir: KAT → CAT

b) Inserção - adiciona-se um caracter de P que não aparece em T: CT → CAT

c) Deleção - elimina-se um caracter de T que não aparece em P: CAAT → CAT

Exemplo: $abcdefghijkl$ (P) pode ser mapeado para $T = bcdeffghixkl$ usando três operações.

13.4.1 Projeto de um algoritmo DP

Que informação seria necessária para tomarmos a decisão final? Isto é, o que deve acontecer na última posição de T?

Possibilidades

a) o último caracter pode ser "emparelhado" com o último caracter de P, se eles são idênticos

b) de outra forma, o último caracter pode ser substituído

c) as outras operações são de deleção e inserção

Seja $D[i, j]$ a "distância" (i.e., menor número de operações para emparelhamento) entre $< P_1, P_2, \ldots, P_i >$ e $< T_1, T_2, \ldots, T_i >$. Então, $D[i, j]$ é o mínimo das três possíveis formas de estender as sub-cadeias.

1) Se $(P_i = T_j)$, então $D[i, j] \leftarrow D[i-1, j-1]$. Caso contrário, $D[i, j] \leftarrow D[i-1, j-1] + 1$. (emparelhamento ou substituição)

2) $D[i, j] \leftarrow D[i-1, j] + 1$, o que significa que há um caracter adicional na cadeia de busca P, portanto pagamos o custo de uma inserção e não avançamos o ponteiro do texto T

3) $D[i, j] \leftarrow D[i, j-1] + 1$, o que significa que há um caracter extra na cadeia T, portanto pagamos o custo de uma deleção em T e não avançamos o ponteiro de P

Entretanto, ainda falta especificar as condições iniciais. É crítico inicializar os valores corretamente para que o programa retorne o resultado correto. O valor $D[0, j]$ corresponde a emparelhar os j primeiros caracteres do texto com nenhum caracter do padrão T.

a) Se desejamos um emparelhamento entre todo o padrão P e todo o texto T, então $D[0, j]$ é o custo de eliminarmos j caracteres de T. Então, $D[0, j] = j$

b) Se desejamos "emparelhar" P com qualquer subcadeia de T, então o custo de iniciarmos o emparelhamento na posição j do texto deve ser o mesmo de iniciarmos na posição 1. Então, $D[0, j] = 0$.

Em ambos os casos, $D[i, 0] = i$ pois não podemos eliminar símbolos de P sem pagarmos o preço de deleção.

Antes de apresentarmos o algoritmo, assumimos que $m = |T|$ e $n = |P|$.

Algoritmo EditDistance(P,T)
for i=0 to n do $D[i,0] = i$
for j=0 to m do $D[0,j] = j$
for i=1 to n do
for j=1 to m do
$D[i,j] = Min(D[i-1,j-1] + matchcost(P_i, t_i), D[i-1,j] + 1, D[i,j-1] + 1)$
Fim algoritmo

Qual é a complexidade do algoritmo? $\Theta(mn)$.

Exemplo: A tabela abaixo traz o resultado da operação EditDistance(P,T) onde $P = abcdefghijkl$ e $T = bcdeftghixkl$

Tabela 1: Matriz $D[i,j]$ $i = 0, \ldots, n$ e $j = 1, \ldots, m$.

TABELA

α_1 - insert (custo 1)
α_2 - match (custo 0)
α_3 - match (custo 0)
α_4 - match (custo 0)
α_5 - match (custo 0)
α_6 - match (custo 0)
α_7 - delete (custo 1)
α_8 - match (custo 0)
α_9 - match (custo 0)
α_10 - match (custo 0)
α_11 - substitute $x \rightarrow j$ (custo 1)
α_12 - match (custo 0)
α_13 - match (custo 0)

13.5 Exercícios

Ex 14.1 Implemente o algoritmo que calcula a distância entre uma cadeia de caracteres padrão $p = (p_1, p_2, \ldots, p_n)$ e um texto $t = (t_1, t_2, \ldots, t_m)$ em qualquer linguagem (e.g., awk, shell script, C/C++, Pascal ou até mesmo Matlab). Produza uma versão onde p pode aparecer em qualquer posição de t e outra onde o "emparelhamento"deve ser perfeito. Ilustre a execução dos algoritmos para uma/duas instâncias.

Ex 14.2 Considere a seguinte técnica de compressão de texto. Nós temos uma tabela

com m cadeias de caracteres, cada uma de comprimento máximo k. Desejamos representar uma cadeia de dados $s = (s_1, \ldots, s_n)$ usando apenas cadeias da tabela. Por exemplo, se nossa tabela contém $(a, ba, abab, b)$ e a cadeia de dados é *bababbaababa*, então a melhor forma de codificá-la é $(b, abab, ba, abab, a)$, consumindo um total de 5 palavras. Desenvolva um algoritmo de tempo $O(nmk)$ para encontrar a codificação ótima. Assuma que os símbolos elementares (e.g., a e b) aparecem como cadeias na tabela.