1. Computação Evolutiva

Prof. Renato Tinós



Departamento de Computação e Matemática
Fac. de Filosofia, Ciência e Letras de Ribeirão Preto



Universidade de São Paulo
BRASIL

1.1. Introdução

1.1.1. Introdução à otimização

1.1.2. Técnicas clássicas utilizadas em otimização

1.1.3. Heurísticas e Metaheurísticas

1.1.4. Classes de problemas

1.1.5. A inspiração biológica

1.1.6. Uma introdução aos algoritmos evolutivos

1.1.1. Introdução à Otimização

- **Otimização**
  - Procura melhorar o desempenho na resolução de um problema em direção aos pontos ótimos
  - Teoria de otimização engloba os estudos quantitativos dos ótimos e os métodos para buscá-los

1.1.1. Introdução à Otimização

- **Instância de um problema de otimização**
  - é um par (Ω ,*f*), no qual
    - Ω é o conjunto dos pontos factíveis (candidatos a pontos ótimos), ou domínio
    - *f* é uma função de avaliação (ou custo)
  - O desafio é achar a variável de decisão $x \in \Omega$ tal que

$$f(x) \leq f(y) \quad \text{para todo} \quad y \in \Omega$$

considerando que o problema seja de minimização

1.1.1. Introdução à Otimização

- **Problema de otimização**
  - é um conjunto de instâncias de um problema de otimização

1.1.1. Introdução à Otimização

- **Exemplo: Problema do Caixeiro Viajante (*Traveling Salesman Problem* – TSP)**
  - um *tour* é um caminho fechado passando por todas as *n* cidades somente uma vez
  - o domínio Ω é dado por todas a permutações **x** das *n* cidades, sendo que *x*(*j*) indica a cidade visitada após a cidade *j*, sendo *j* =1,…,*n*
  - a função de avaliação é dada por $f(\mathbf{x}) = \sum_{j=1}^{n} D(j, x(j))$
  - Em uma instância do TSP são dadas
    - um inteiro *n*>0
    - as distâncias entre cada par das *n* cidades na forma de uma matriz

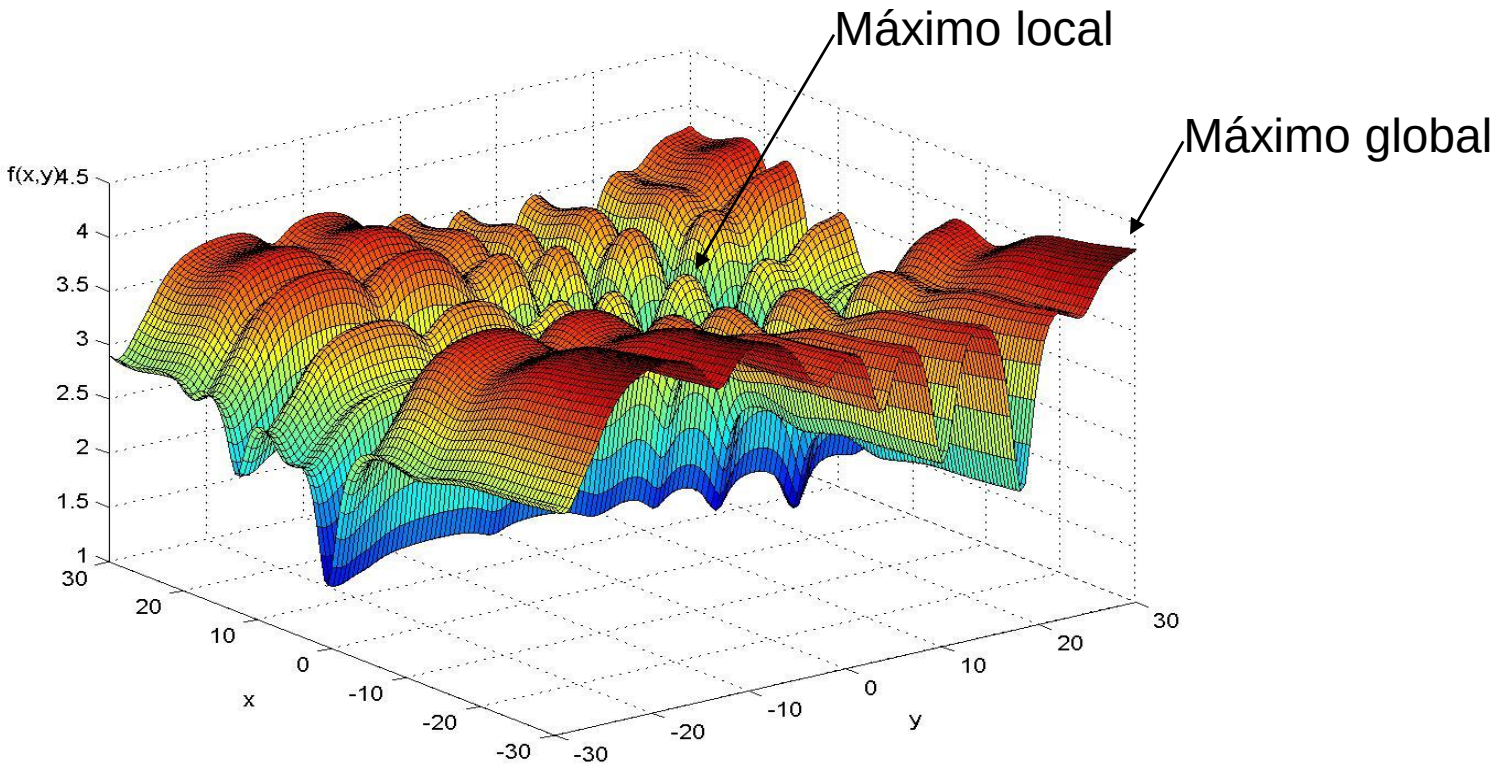
$$D \in Z^{n \times n}$$

1.1.1. Introdução à Otimização

- **Função objetivo (função de avaliação)**
  - Função que queremos maximizar ou minimizar
  - Não é necessariamente conhecida
- **Ótimo global (mínimo global ou máximo global)**
  - Ponto de ótimo no espaço de busca
    - No caso do máximo global, é o ponto factível mais alto da superfície da função objetivo
  - Pode existir mais de um
- **Ótimo local (mínimo local ou máximo local)**
  - Ponto de ótimo em um vale (mínimo) ou em uma elevação (máximo)
  - Não é o ponto factível mais alto (ou mais baixo) da superfície da função objetivo

1.1.1. Introdução à Otimização

- Espaço de busca



1.1.1. Introdução à Otimização

- **Sub-áreas comuns em otimização**
  - **Otimização Irrestrita**

➢ Sem restrições no domínio das variáveis de decisão

➢ Exemplo de otimização (minimização) irrestrita

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \in \mathrm{R} \\ &\text{sujeito a} && \mathbf{x} \in \mathrm{R}^n \end{aligned}$$

– Ou seja $\Omega = \mathrm{R}^n$

1.1.1. Introdução à Otimização

- **Sub-áreas comuns em otimização**
  - **Otimização Restrita**
    - Com restrições no domínio das variáveis de decisão
  - **Programação Linear**
    - Otimização com função objetivo e restrições lineares
    - Exemplo de otimização (minimização) restrita

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) = \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{sujeito a} \quad & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{A}\mathbf{x} \geq \mathbf{b} \end{aligned}$$

Sendo $\mathbf{x} \in \mathrm{R}^n, \mathbf{c} \in \mathrm{R}^n, \mathbf{b} \in \mathrm{R}^m, \mathbf{A} \in \mathrm{R}^{m \times n}$

    - Podem ser resolvidos por métodos iterativos deterministas polinomiais (ex.: **Simplex**)

1.1.1. Introdução à Otimização

- **Sub-áreas comuns em otimização**
  - Programação Linear
    - Exemplo: Problema da Dieta
      - Assuma que existem *n* tipos de comida
      - O *j*-ésimo tipo de comida custa c_j reais por unidade
      - Existem *m* tipos de nutrientes básicos
      - Em uma dieta básica, você deve receber no mínimo b_i unidades do *i*-ésimo nutriente por dia
      - Considere que cada unidade do tipo de comida *j* contém a_{ij} unidades do *i*-ésimo nutriente por dia
      - Seja x_j o número de unidades do tipo de comida *j* por dia na dieta
      - O objetivo é minimizar o custo total da dieta, ou seja

Minimize $f(\mathbf{x}) = \mathbf{c}^{\mathrm{T}}\mathbf{x}$

Sujeito a $\mathbf{A}\mathbf{x} \geq \mathbf{b}$

$\mathbf{x} \geq \mathbf{0}$

1.1.1. Introdução à Otimização

Exemplo. [Chong & Zak, 2001]

Consider the following linear program (adapted from [88]):

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{c}^T &= [1, 5], \\ \boldsymbol{x} &= [x_1, x_2]^T, \\ \boldsymbol{A} &= \begin{bmatrix} 5 & 6 \\ 3 & 2 \end{bmatrix}, \\ \boldsymbol{b} &= [30, 12]^T. \end{aligned}$$

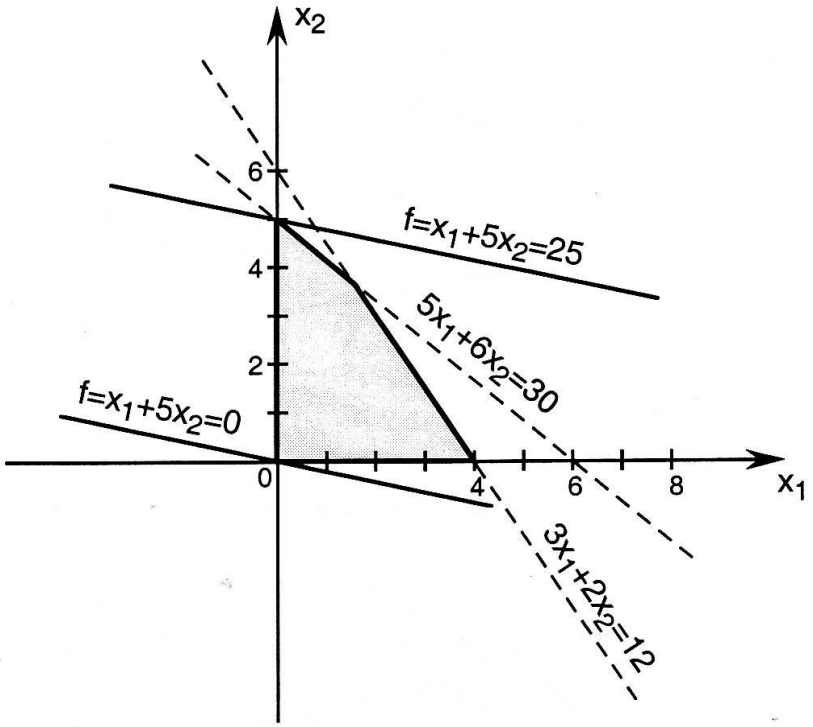


Figure 15.3 Geometric solution of a linear program in $\mathbb{R}^2$

1.1.1. Introdução à Otimização

- **Sub-áreas comuns em otimização**
  - **Otimização Contínua**
    - As variáveis de decisão (**x**) são contínuas
      - Exemplo: problema da dieta é um problema de otimização contínua, classificado ainda na sub-área de programação linear
  - **Otimização Discreta**
    - As variáveis de decisão (**x**) são discretas
    - Inclui o campo da otimização combinatória
      - Soluções são permutações de objetos
        - » Exemplo: TSP

1.1.2. Técnicas Clássicas Utilizadas em Otimização

- **Em geral, a maioria das técnicas clássicas de otimização pode ser classificada em uma (ou mais) das classes seguintes**

1. Métodos baseados em cálculo

- *Indiretos*

  - Em otimização contínua, procuram os ótimos através da resolução de um conjunto de equações resultantes de igualar o gradiente da função objetivo a zero
  - Exemplo: dado *c* real,

    minimize $f(x) = (x-c)^2$

    sujeito a $x \in \mathbb{R}$

1.1.2. Técnicas Clássicas Utilizadas em Otimização

1. Métodos baseados em cálculo

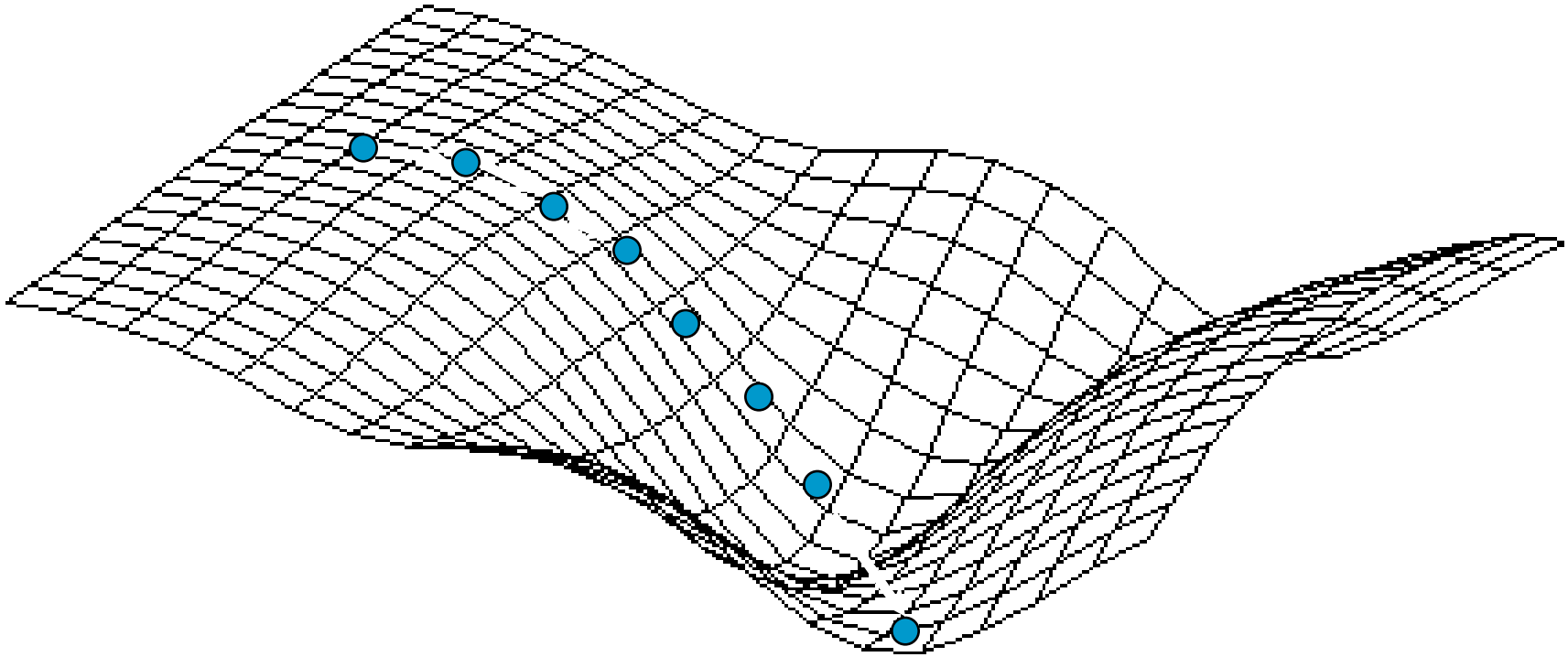
- *Diretos* (*Hill-Climbing*):
  - Considerando que a superfície de função objetivo é um parabolóide, podemos utilizar uma técnica de otimização baseada no cálculo do *gradiente*
  - Exemplo: *Método do Gradiente Descendente*
    - Comece em um determinado ponto e caminhe na direção que minimiza (ou maximiza) a função objetivo (gradiente local descendente)

$$\mathbf{x}(n+1) = \mathbf{x}(n) - \eta \, \nabla f\left(\mathbf{x}(n)\right)$$

na qual $\eta > 0$

1.1.2. Técnicas Clássicas Utilizadas em Otimização

Exemplo: 2 dimensões



1.1.2. Técnicas Clássicas Utilizadas em Otimização

1. Métodos baseados em cálculo

- Diretos:
  - Exemplo: *Método de Newton*
    - Uso da segunda derivada
- Regra

$$\mathbf{F}(\boldsymbol{x}) \cong D^2 f(\boldsymbol{x}) = \begin{bmatrix} \partial^2 f(\boldsymbol{x}) / \partial^2 x_1 & \cdots & \partial^2 f(\boldsymbol{x}) / \partial x_n \partial x_1 \\ \vdots & \ddots & \vdots \\ \partial^2 f(\boldsymbol{x}) / \partial x_1 \partial x_n & \cdots & \partial^2 f(\boldsymbol{x}) / \partial^2 x_n \end{bmatrix}$$

$$\mathbf{x}(n+1) = \mathbf{x}(n) - \mathbf{F}^{-1}(\mathbf{x}(n) \nabla f(\mathbf{x}(n))$$

1.1.2. Técnicas Clássicas Utilizadas em Otimização

1. Métodos baseados em cálculo

- Extremamente interessantes em determinados tipos de problemas

- Algumas dificuldades
  - O que fazer quando existem ótimos locais?
    - **Problemas multimodais** (problemas com vários ótimos)
    - Superfícies ruidosas
  - Necessidade de conhecimento da relação entre a função objetivo e os parâmetros que estão sendo otimizados

1.1.2. Técnicas Clássicas Utilizadas em Otimização

2. Esquemas enumerativos (ou métodos exaustivos)

- Em um espaço de busca finito ou discretizado, procure pelo ponto ótimo em todos os pontos do espaço de busca
- Em alguns problemas, alguns pontos podem ser descartados durante a busca
  - Exemplos:
    - *Branch and bound*
    - *Programação Dinâmica*

1.1.2. Técnicas Clássicas Utilizadas em Otimização

2. Esquemas enumerativos

- Programação Dinâmica
  - Utiliza a estratégia Dividir para Conquistar
    - A programação dinâmica decompõe o problema em sub-problemas. Ela calcula a solução para todos os sub-problemas
      - partindo dos sub-problemas menores para os maiores
      - armazenando os resultados em uma tabela
    - As soluções dos sub-problemas são unidas
  - Vangagem: uma vez que um sub-problema é resolvido, a resposta é armazenada em uma tabela e nunca mais este sub-problema precisa ser resolvido de novo
  - Desvantagem: nem sempre o problema pode ser resolvido através da decomposição em problemas menores

1.1.2. Técnicas Clássicas Utilizadas em Otimização

2. Esquemas enumerativos

- Programação Dinâmica
  - Exemplo: *Problema do Produto de n Matrizes*
    - $M = M_1 \times M_2 \times \cdots \times M_n$
      - » sendo M_i uma matriz com d_{i-1} linhas e d_i colunas
    - Sendo que o produto de uma matriz $p \times q$ por outra matriz $q \times r$ requer $O(p\ q\ r)$ operações
    - A ordem da multiplicação pode ter um efeito enorme no número total de operações de adição e multiplicação necessárias para obter M
    - Considere, por exemplo, o produto $M = M_1\ [10, 20] \times M_2[20, 50] \times M_3\ [50, 1] \times M_4\ [1, 100]$
      - » ordem $M = M_1 \times (M_2 \times (M_3 \times M_4))$ requer 125.000 operações
      - » ordem $M = (M_1 \times (M_2 \times M_3)) \times M_4$ requer 2.200 operações

1.1.2. Técnicas Clássicas Utilizadas em Otimização

2. Esquemas enumerativos

- Programação Dinâmica
  - Exemplo: Problema do Produto de *n* Matrizes
    - Poderíamos tentar todas as ordens possíveis e ver a que tem menor número de operações
      - A complexidade neste caso é exponencial
    - É possível obter um algoritmo O(n^3) usando programação dinâmica

1.1.2. Técnicas Clássicas Utilizadas em Otimização

2. Esquemas enumerativos

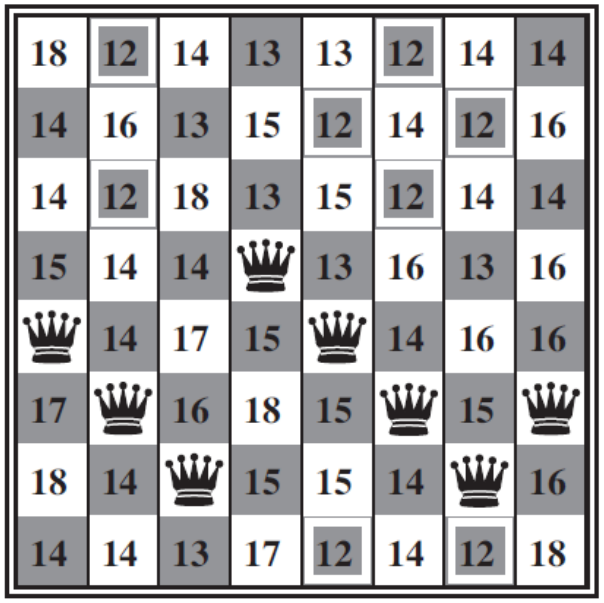
- Dificuldades
  - *Maldição da dimensionalidade*
    - em problemas com muitos parâmetros, não é prático (quando é possível) procurar por todas a soluções
    - Exemplo:
      - na busca em um espaço *n*-dimensional em que os parâmetros assumem valores binários, 2^{n-1} soluções devem ser examinadas
    - Problemas NP: ver Seção 1.1.4.

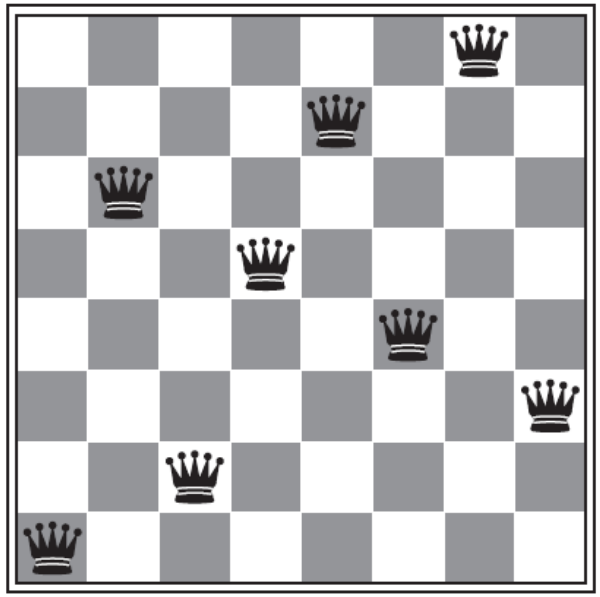
1.1.2. Técnicas Clássicas Utilizadas em Otimização

3. **Métodos iterativos baseados em busca local**
   - Também chamados de busca local gulosa
   - Busca iterativa na vizinhança
     i. busque a melhor solução na vizinhança local da solução atual.
     ii. Repita (até que um critério de parada seja alcançado) considerando a melhor solução vizinha como a solução atual.
   - Repare que o método do gradiente descendente é um caso particular de busca local

1.1.2. Técnicas Clássicas Utilizadas em Otimização

Exemplo: problema das *N*-rainhas





(a) (b)

Figure 4.3 FILES: figures/8queens-successors.eps (Wed Nov 4 16:23:55 2009) figures/8queens-local-minimum.eps (Wed Nov 4 16:14:15 2009). (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

1.1.3. Heurísticas e Metaheurísticas

- **Heurísticas**
  - Utilizadas em estratégias para busca de soluções
  - Derivadas a partir de experiências anteriores em problemas simulares
  - São específicas de acordo com o problema
    - Exemplo: programas inteligentes para jogar Xadrez
  - Muitas vezes, **não garantem necessariamente que o ótimo global será encontrado**
    - Geralmente empregadas quando métodos exatos são lentos e quando está ok “achar uma solução boa, mas não necessariamente ótima”

1.1.3. Heurísticas e Metaheurísticas

- **Metaheurísticas**

  - Procedimento de alto nível usado para encontrar, gerar ou selecionar uma heurística que pode prover uma solução suficientemente boa em problemas de otimização com informação incompleta ou imperfeita e/ou com **limitada capacidade computacional**
    - Podem, nestes problemas, encontrar boas soluções mais eficientemente que métodos tradicionais

  - Amostram soluções do espaço de soluções em problemas em que este é muito grande para ser completamente examinado
    - Assim, quando a capacidade de computação é limitada, **não garantem necessariamente que o ótimo global será encontrado**

1.1.3. Heurísticas e Metaheurísticas

- **Metaheurísticas**
  - Podem utilizar pouca informação sobre o problema a ser resolvido
    - Podem, portanto, serem utilizadas em uma grande variedade de problemas
  - Muitas metaheurísticas utilizam métodos estocásticos para vasculhar o espaço de soluções
    - Exemplos:
      - *Simulated Annealing* (Recozimento Simulado)
      - Computação Evolutiva
        - **Metaheurísticas populacionais** (Exemplo: **Algoritmos Genéticos**)

1.1.4. Classes de Problemas

- **Em geral,**
  - Problemas que podem ser resolvidos por um algoritmo de tempo polinomial (ou seja, com tempo de execução $O(n^k)$, sendo n a dimensão do problema)
    - São considerados tratáveis
  - Problemas que não podem ser resolvidos por um algoritmo de tempo polinomial
    - São considerados intratáveis (ou difíceis)

1.1.4. Classes de Problemas

- **Todos os problemas podem ser resolvidos em tempo polinomial?**
  - Não!!!
    - Exemplos
      - Torre de Hanói: $O(2^n)$
      - Achar todos os caminhos no Problema do Caixeiro Viajante: O(n!)
  - **Problemas Não-Polinomiais**

1.1.4. Classes de Problemas

- **Métodos de Aproximação (ou Não-Deterministas ou Estocáticos)**
  - É capaz de escolher uma dentre várias alternativas possíveis a cada passo
  - Utilizam a função *escolhe(C)*, que escolhe um dos elementos do conjunto *C* de forma não necessariamente determinista (por exemplo, de forma aleatória)
  - Exemplo:

```
Algoritmo buscaAleatoria( a[ ] , l , r  , x )

 i  ← escolhe( a[ ] , l , r )
 se ( a[i] = x )
            retorna sucesso
 senão
            retorna insucesso
fim se
```

1.1.4. Classes de Problemas

- **Consideraremos aqui duas classes de problemas:**
  - Classe **P**
  - Classe **NP**

1.1.4. Classes de Problemas

- **Classe P**
  - Consiste no conjunto de problemas que podem ser resolvidos em tempo polinomial por algoritmos deterministas
    - **Problemas Polinomiais**
    - Ou seja, podem ser resolvidos em tempo $O(n^k)$ para alguma constante k e tamanho da entrada n
    - Exemplos:
      - Busca Linear: $O(n)$
      - Busca Binária: $O(\log_2 n)$
      - Ordenação por Heapsort: $O(n \log_2 n)$

1.1.4. Classes de Problemas

- **Problemas da Classe P são considerados tratáveis**
  - Embora seja razoável considerar um algoritmo com $O(n^{100})$ intratável, poucos problemas práticos exigem tempo de execução tão alto
    - Além disso, a experiência mostra que, uma vez que um algoritmo de tempo polinomial para um problema é descoberto, algoritmos mais eficientes freqüentemente aparecem
  - Um problema que pode ser resolvido em tempo polinomial em um modelo de computação pode ser resolvido em outro
    - Ex.: em um computador paralelo em que o número de processadores cresce polinomialmente com o número de entradas
  - A composição (polinomial) de problemas polinomiais resulta em um problema polinomial

1.1.4. Classes de Problemas

- **Classe NP: Classe dos problemas que**
  - São “verificáveis” em tempo polinomial
    - Ou seja, se tivéssemos de algum modo um “certificado” de uma solução, poderíamos verificar se o certificado é correto em tempo polinomial
      - Exemplo:
        - » É difícil provar que um número natural qualquer *z* (por exemplo, 15.790.107) não é composto (ou seja, que *z* não pode ser escrito como o produto de dois outros números naturais *x* e *y* diferentes de 1 se os valores de *x* e *y* não são conhecidos)
        - » Contudo, é fácil verificar que *z* (por exemplo, 15.790.107) é composto se *x* e *y* (por exemplo 3251 e 4857) são conhecidos
  - Podem ser resolvidos por algoritmos não-deterministas em tempo polinomial

1.1.4. Classes de Problemas

- **Para mostrar que um determinado problema está em NP, basta**
  - Apresentar um algoritmo não-determinista que “pode” resolve-lo em tempo polinomial
  - Apresentar um algoritmo determinista polinomial para verificar que uma dada solução é válida

1.1.4. Classes de Problemas

- **Como os algoritmos deterministas são um caso especial de algoritmos não-deterministas, sabe-se que $P \subseteq NP$**
  - Ou seja, qualquer problema em P também está em NP, tendo em vista que, se um problema está em P, então podemos resolvê-lo em tempo polinomial
- **A questão aberta é se P é ou não um subconjunto próprio de NP**
  - Ou seja, se $P=NP$ ou se $P \neq NP$
  - Esta questão é o problema não-resolvido mais famoso da ciência da computação

1.1.4. Classes de Problemas

- **Será que existem algoritmos polinomiais deterministas para todos os problemas em NP?**
  - Se a resposta é positiva, então P=NP
    - No entanto, não foram encontrados até agora algoritmos deterministas polinomiais para muitos problemas em NP
  - Se a resposta é negativa, então P≠NP
    - No entanto, a prova para tal informação parece exigir técnicas desconhecidas e nenhum limite inferior não-polinomial foi provado para vários destes problemas

1.1.4. Classes de Problemas

- **Tais fatos trazem as seguintes conseqüências:**
  - Existem muitos problemas práticos em NP que podem ou não pertencer a P
    - Não conhecemos algoritmos polinomiais deterministas eficientes para tais problemas
  - Se conseguirmos um indicativo que um problema pertence a NP, então é mais produtivo procurar um algoritmo não-determinista para ele do que ficar tentando achar um algoritmo polinomial determinista
  - Como não existe a prova de que NP≠P, sempre há a esperança de que alguém descubra um algoritmo polinomial eficiente para tais problemas
  - Existe um esforço considerável para provar que NP≠P
    - Mas a questão continua em aberto

Comentários Finais

- **Referências**

  - **Papadimitriou,C. H. & Steinglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Dover Publications Inc.**
    - Capítulo 1
  - **Chong, E. K. P. & Zak, S. H. (2001). *An introduction to optimization.* 2nd Edition, John Wiley & Sons Inc.**
    - Capítulos 8 e 15
  - **Ziviani, N. (2004). "*Projeto de Algoritmos*", Thomson.**
    - Seção 2.6