

# Busca Local e Problemas de Otimização

**Inteligência Artificial**  
**PCS3438**

*Anna Helena Reali Costa*  
*Escola Politécnica da USP*  
*Engenharia de Computação (PCS)*

# Classe de problemas de interesse

- Em vários problemas a própria descrição de estado contém toda informação relevante para a solução e o caminho ao estado-objetivo não interessa:
  - Ex: problema das 8 rainhas, projeto de circuitos integrados, escalonamento, problemas de roteamento, de otimização de redes de telecomunicação, etc.

## **Problemas de otimização**

- Buscas Locais (ou de melhorias iterativas) operam num único estado e movem-se para a vizinhança deste estado.

# Busca Local

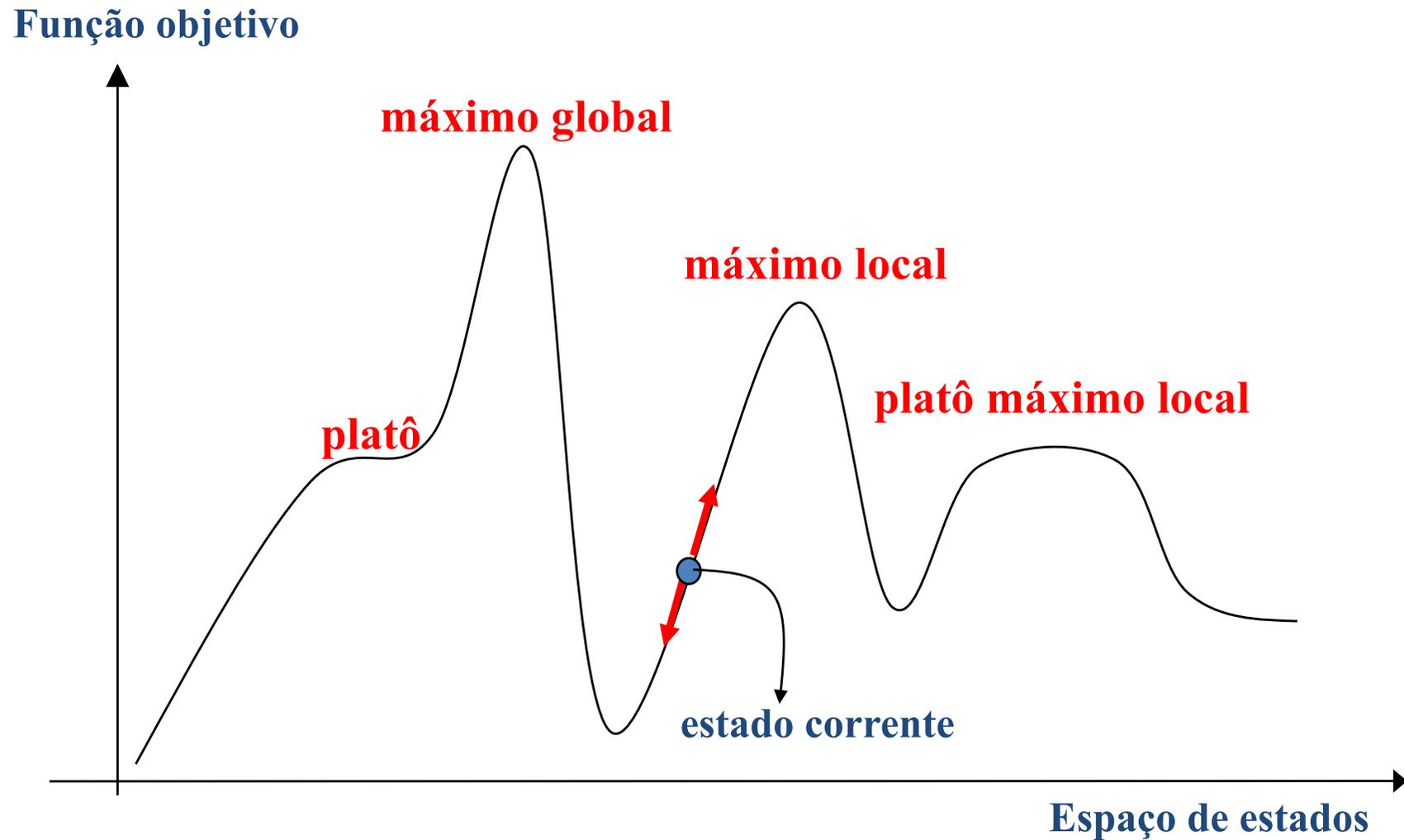
- A ideia é começar com o estado inicial (configuração completa, solução aceitável) e melhorá-lo iterativamente.
- Visualização:
  - Os estados (solução) estão representados sobre uma **superfície** (gráfico);
  - A **altura** de qualquer ponto na superfície corresponde à **função de avaliação** do estado naquele ponto;
  - O algoritmo se “**move**” pela superfície em busca de **pontos mais altos** (melhor avaliação do estado/solução);
  - O ponto mais alto (**máximo global**) corresponde à **solução ótima**.

# Exemplo de Espaço de Estados

avaliação



# Espaço de estados unidimensional



# Busca Local

- Os algoritmos de busca local armazenam apenas o estado atual (baixo uso de memória), e não veem além dos vizinhos imediatos do estado atual.
- Apesar destas restrições, muitas vezes são os melhores métodos para tratar problemas reais muito complexos (espaço contínuo).

# Tipos de Busca local

- 1. *Hill-Climbing*: Subida pela Encosta mais Íngrime ou Busca Local Gulosa**
  - só faz modificações que melhoram o estado atual.
- 2. *Simulated Annealing*: Têmpera Simulada**
  - pode fazer modificações que pioram o estado no momento, para possivelmente melhorá-lo no futuro.
- 3. *Local beam search*: Busca em feixe local**
  - Mantém k estados em vez de um único.
- 4. Algoritmos genéticos (GA)**
  - é uma busca **subida pela encosta**, estocástica, na qual uma grande população de estados é mantida e novos estados são gerados por mutação ou cruzamento.

# (1) Subida da Encosta (Hill Climbing)

- O algoritmo **não** mantém uma árvore de busca:
  - guarda apenas o estado atual e sua avaliação
- É simplesmente um ciclo que move o estado (solução) na direção crescente da função de avaliação
  - muda o estado para o melhor vizinho).

# Hill Climbing ou Gradient Ascent/Descent

**função** HILL-CLIMBING(*problema*)

**retorna** um estado que é um máximo local

*atual* = CRIAR-NÓ(*problema*.ESTADO\_INICIAL)

**repita**

*vizinho* = um sucessor de *atual* com valor mais alto

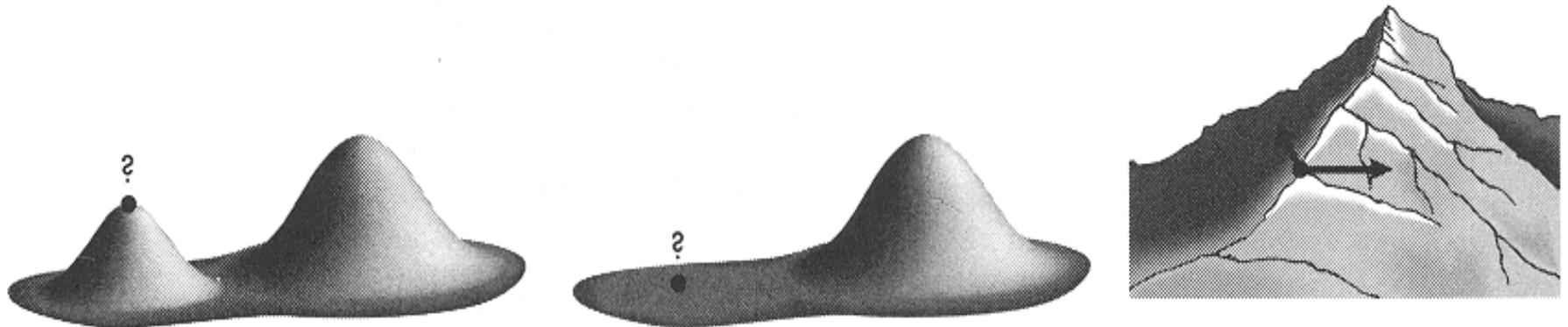
**se** VALOR[*vizinho*] < VALOR[*atual*]

**então retorna** ESTADO[*atual*]

*atual* = *vizinho*

# Subida da Encosta – Problemas

- Isso pode acarretar 3 tipos de problemas:
  1. Máximos locais
  2. Planícies (platôs)
  3. Encostas e picos: somente poucos vizinhos podem melhorar a solução (difícil de encontrá-los)
- Nestes casos, o algoritmo chega a um ponto de onde não faz mais progresso.



# Subida da Encosta – Alternativas

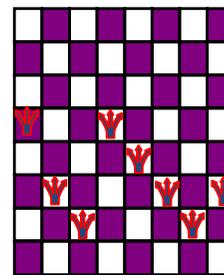
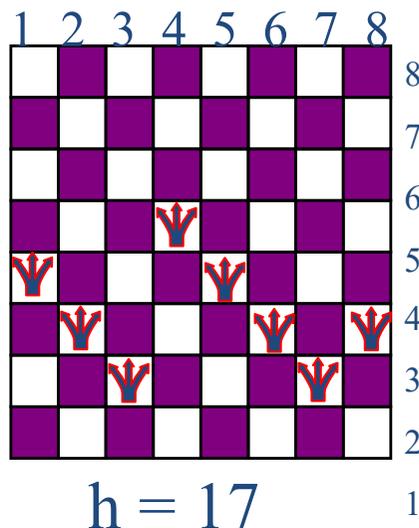
- Solução: **reinício aleatório** (*random restart*)
  - O algoritmo realiza uma série de buscas a partir de estados iniciais gerados aleatoriamente (diferentes reinícios).
  - Cada busca é executada até que:
    - um número máximo estipulado de iterações seja atingido, ou
    - até que os resultados encontrados não apresentem melhora significativa.
  - O algoritmo escolhe o melhor resultado obtido com as diferentes buscas.

# Exemplo

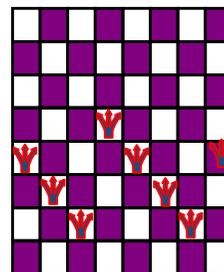
- Heurística h: número de pares de rainhas que se encontram em ataque.

No caso da figura,  $h = 17$  (pares: 1-2, 1-3, 1-5, 2-3, 2-4, 2-6, 2-8, 3-5, 3-7, 4-5, 4-6, 4-7, 5-6, 5-7, 6-7, 6-8, 7-8)

- Movimento: mover na coluna cada possível rainha.



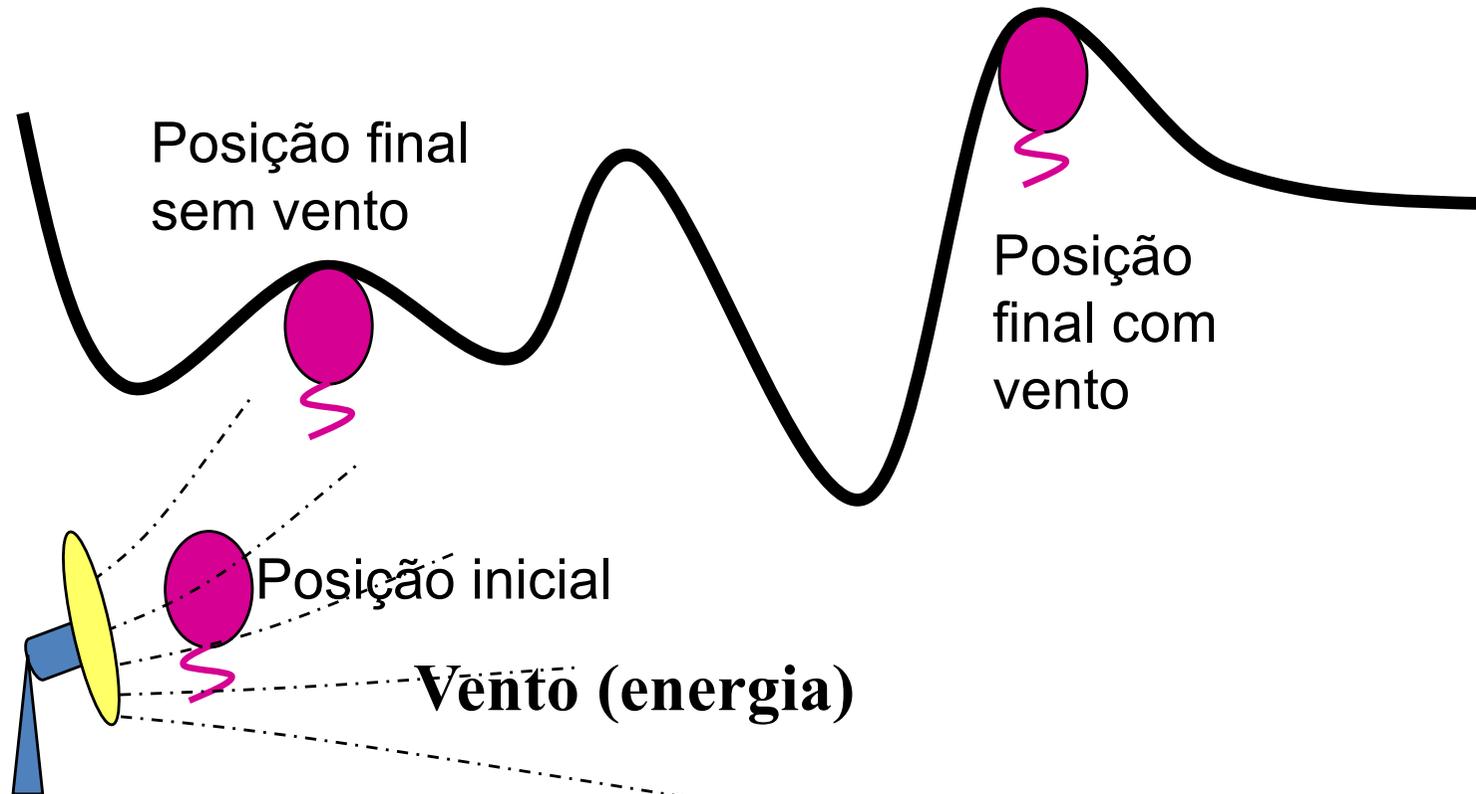
...



O melhor dos sucessores será o próximo estado

## (2) Têmpera simulada

- Semelhante à Subida pela Encosta, porém oferece meios para escapar de máximos locais.



# Têmpera Simulada

**função** TEMPERA-SIMULADA(*problema*, *mapa*)

**retorna** um estado solução

*atual* = CRIAR-NÓ(*problema*.ESTADO\_INICIAL)

**para**  $t = 1$  até  $\infty$  **faça**

$T = \text{mapa}[t]$

**se**  $T=0$  **então retorna** *atual*

*vizinho* = um sucessor aleatório de *atual*

$dE = \text{vizinho.VALOR} - \text{atual.VALOR}$

**se**  $dE > 0$  **então** *atual* = *vizinho*

**senão** *atual* = *vizinho* somente com probabilidade  $e^{dE/T}$

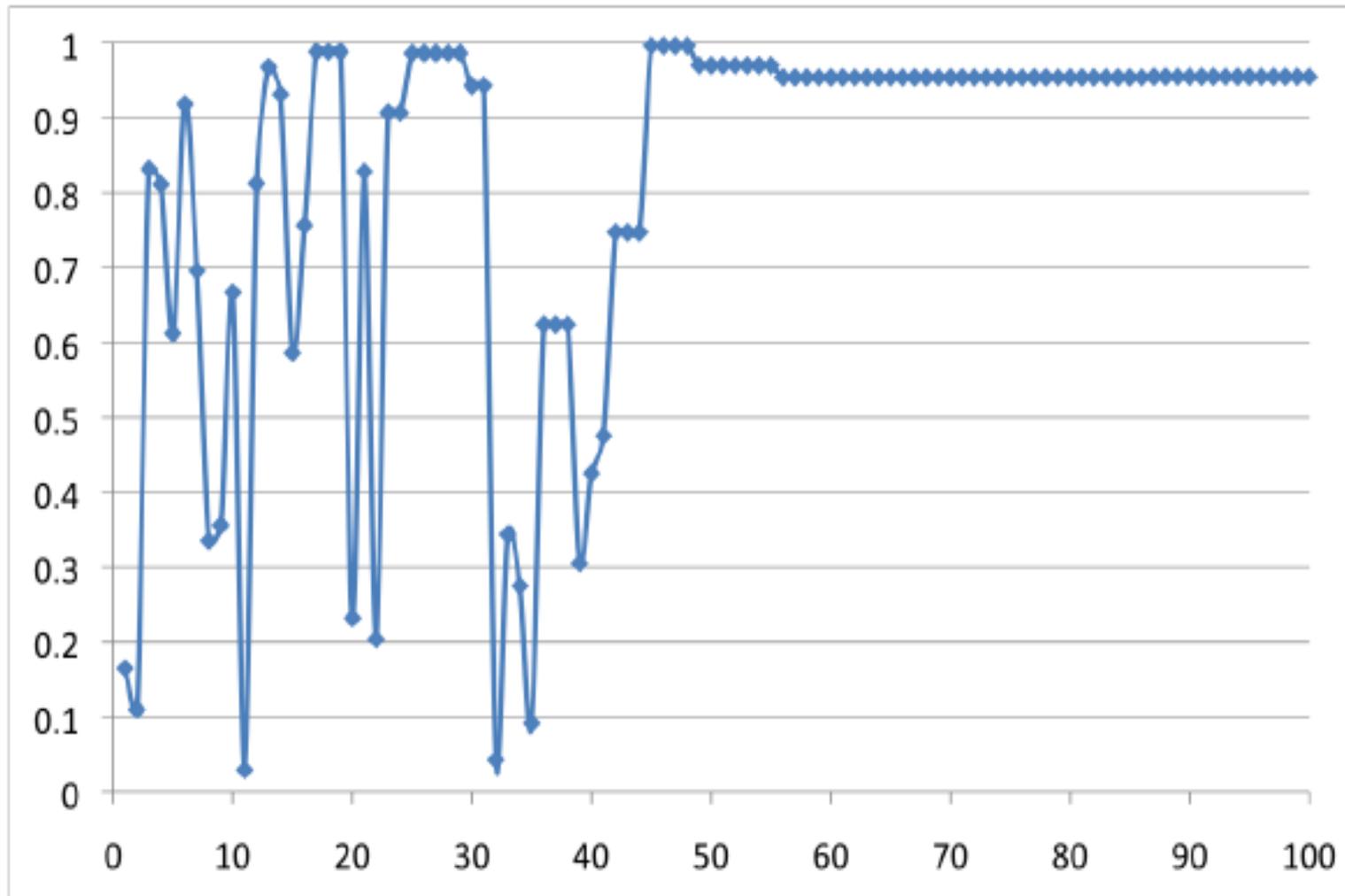
*mapa*: um mapeamento de tempo para "temperatura"

# Têmpera Simulada: considerações

- T é a “temperatura/energia”, reduzida com o tempo de execução
  - com o passar do tempo, Têmpera Simulada fica igual ao Subida da Encosta
- No início, movimentos “ruins” ocorrem com maior frequência.
- Apesar de aumentar o tempo de busca, essa estratégia consegue escapar melhor dos máximos locais.

# Têmpera Simulada: Exemplo

100 iterations, each state is a number  $x \in [0, 1]$ , initial state is  $x = 0$ ,  
 $VALUE(x) = x^2$ , all states are neighbors,  $mapa[t] = 10 \times 0.9^t$



## (3) Busca em Feixe Local

- Começa com  $k$  estados gerados aleatoriamente.
- Em cada passo, são gerados todos os sucessores de todos os  $k$  estados.
- Se um dos sucessores for o objetivo, o algoritmo para; caso contrário, escolhe os  $k$  melhores sucessores a partir da lista completa.
  - Note que isso **NÃO** corresponde à execução de  $k$  reinícios aleatórios em paralelo da busca local Subida da Encosta (*random start*)!
  - Note que sempre somente  $k$  estados são considerados como estados atuais na busca.

# Busca em Feixe Local

**função** BEAM-SEARCH(*problema*, *k*)

**retorna** um estado que é solução

inicia com *k* estados gerados aleatoriamente

**repita**

gera todos sucessores de todos *k* estados

**se** um deles for a solução

**então retorna** solução

**senão** seleciona os *k* melhores sucessores

## (4) Algoritmos Genéticos (AG) ou Computação Evolutiva

- As técnicas de computação evolutiva operam sobre uma população de candidatos em paralelo.
  - Buscam em diferentes áreas do espaço de solução, alocando um número apropriado de membros para a busca em várias regiões.

## (4) Algoritmos Genéticos (AG) ou Computação Evolutiva

- AGs têm as seguintes características:
  - Trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros.
  - Trabalham com uma população e não com um só elemento.
  - Utilizam informações de custo ou recompensa.
  - Utilizam regras de transição não determinísticas.
  - São baseados na técnica gerar-e-testar

# Algoritmos Genéticos

- A modelagem de um problema AG envolve:
  - Função de avaliação
  - Função de aptidão (*fitness*)
  - Codificação da solução

# Função de avaliação

- Provém uma medida de desempenho com respeito a um conjunto particular de parâmetros.
- Deve ser relativamente rápida, uma vez que, em cada iteração, cada membro da população é avaliado e recebe um valor de aptidão.
- A avaliação de um membro (cromossomo) representando um conjunto particular de parâmetros é independente da avaliação de qualquer outro membro

# Função de aptidão (*fitness*)

- Transforma a medida da função de avaliação em alocação de oportunidades reprodutivas.
- É sempre definida de acordo com outros membros da atual população.
- No algoritmo genético canônico, aptidão é definida como

$$\text{fit}(x) = f(x) / f'$$

onde  $f(x)$  é a avaliação associada ao cromossomo  $x$  e  $f'$  é a soma da avaliação (ou média) de todos os membros da população.

- A aptidão pode também ser associada à classificação de um cromossomo na população ou outras medidas.

# Codificação

- O cromossomo contém informação sobre a solução
- Mais usual: sequência binária

Cromossomo 1	1101100100110110
Cromossomo 2	1101111000011110

- Cada bit na sequência (gene) pode representar uma característica da solução ou a série como um todo pode representar um número.

# AG canônico

**[Início]** Gerar uma população aleatória com  $n$  cromossomos

**Repetir até obter a solução (ou terminar)**

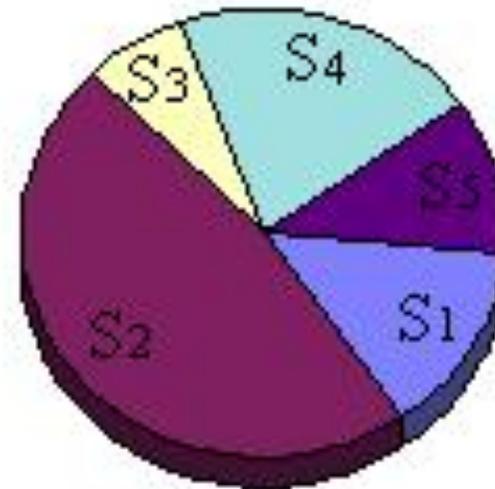
1. **[Avaliação]** Determinar  $f(x)$  e  $fit(x)$  de cada cromossomo  $x$  na população.
2. **[Seleção]** Selecionar elementos para criar uma população intermediária
3. **[Recombinação (Crossover)]** Com uma probabilidade de recombinação, realizar uma recombinação sobre os pais para formar uma nova prole
4. **[Mutação]** Com uma probabilidade de mutação, realizar mutação sobre a nova prole em cada *gene* (posição no cromossomo).
5. **[Atualização da população]** Usar a nova população gerada para repetir os passos 1-5 do algoritmo

**Retornar** a melhor solução na população atual.

# Seleção

- Exemplo: através da Roleta
  - $fit(x)$  define os pais (quanto maior, maior chance de serem selecionados) :

Indivíduo	Aptidão	
$S_i$	$f(S_i)$	Relativa
$S_1$ 10110	2.23	0.14
$S_2$ 11000	7.27	0.47
$S_3$ 11110	1.05	0.07
$S_4$ 01001	3.35	0.21
$S_5$ 00110	1.69	0.11



# Recombinação (Crossover)

- Um método (existem outros):
  1. Escolha aleatoriamente algum ponto (locus entre genes) no cromossomo
  2. Tudo que estiver antes desse ponto será copiado do primeiro pai
  3. Tudo que estiver depois será copiado do segundo pai.

Cromossomo 1	11011   00100110110
Cromossomo 2	11011   11000011110
Prole 1	11011   11000011110
Prole 2	11011   00100110110

# Mutação

- Evita que a população fique presa em um mínimo (máximo) local.
- Altera aleatoriamente a nova prole.
  - Na codificação binária, pode-se mudar alguns bits de 1 para 0 e de 0 para 1:

<b>Prole Original 1</b>	<b>11101111000011110</b>
<b>Prole Original 2</b>	<b>11101100100110110</b>
<b>Prole com Mutação 1</b>	<b>11100111000011110</b>
<b>Prole com Mutação 2</b>	<b>11101101100110110</b>

# AG: parâmetros

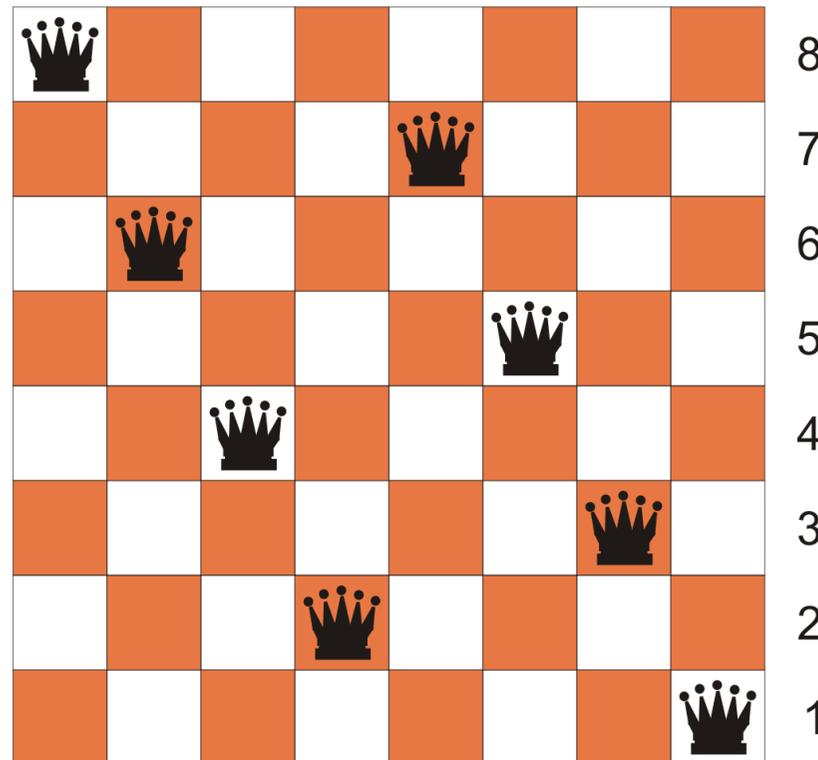
- **Probabilidade de recombinação:** indica o quão frequente a recombinação é executada.
- **Probabilidade de mutação:** indica o quão frequente partes dos cromossomos sofrerão mutações.
- **Tamanho da população:** indica quantos cromossomos existem em uma população.
- **Intervalo de Geração:** controla a porcentagem da população que será substituída durante a próxima geração.

# Exemplo: 8 Rainhas

- **Codificação**

- Cromossomos compostos por 8 números (genes), a posição do gene indica a coluna, o valor do número indica a linha

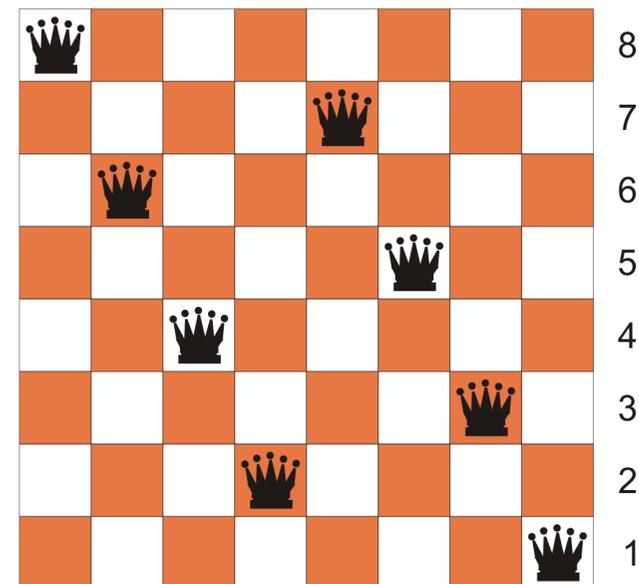
86427531 →



# Exemplo: 8 Rainhas

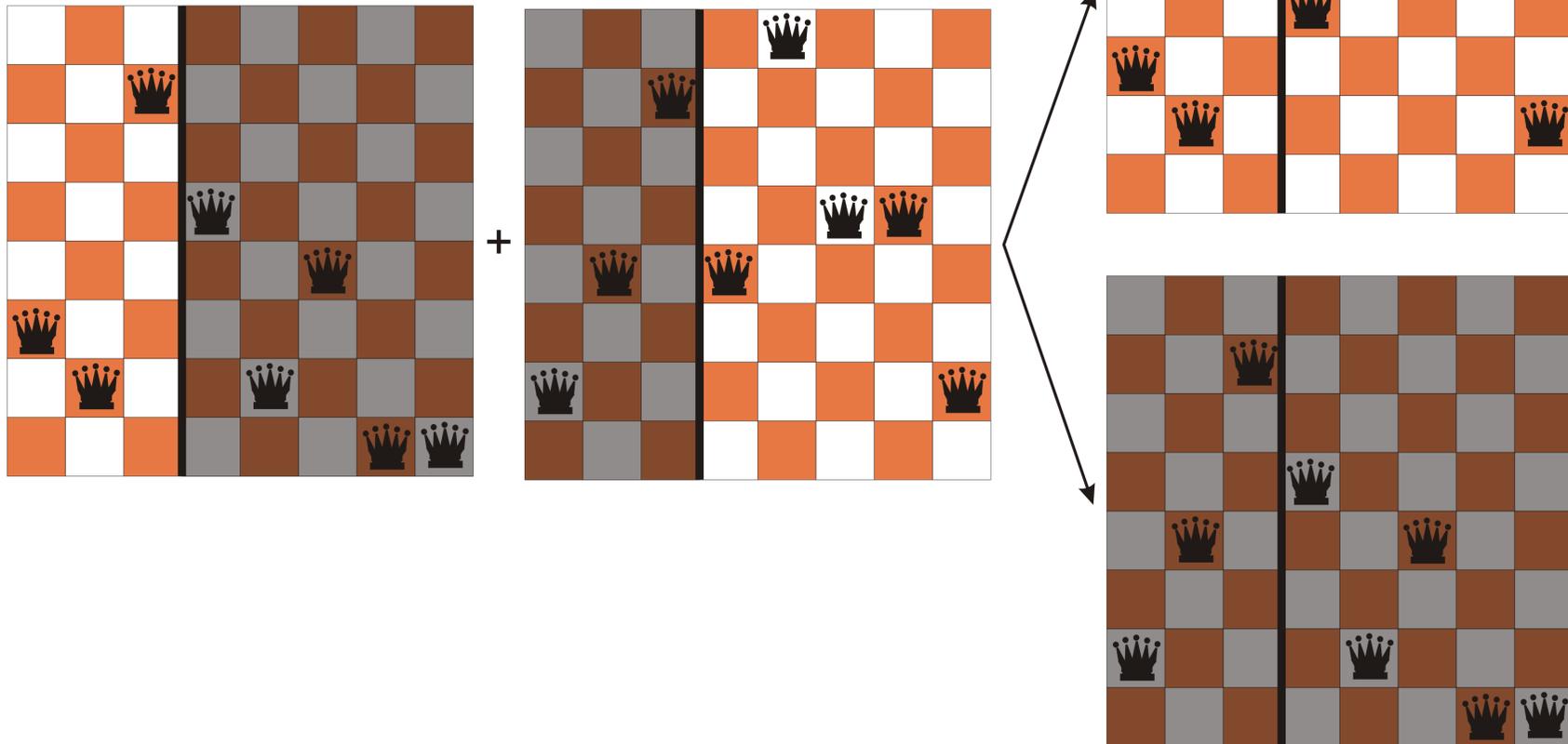
- Função de avaliação:
  - $f(x)$  = número de pares que **não** se atacam
  - Na solução: 1-2, 1-3, ..., 1-8, 2-3, ..., 2-8, 3-4, ..., 3-8, ..., 6-7, 6-8, 7-8 = 28 pares
- Função de aptidão:
  - $fit(x) = f(x) / \Sigma f(x)$  [%]

**x: 1-8 em ataque** →  
→  **$f(x) = 27$**



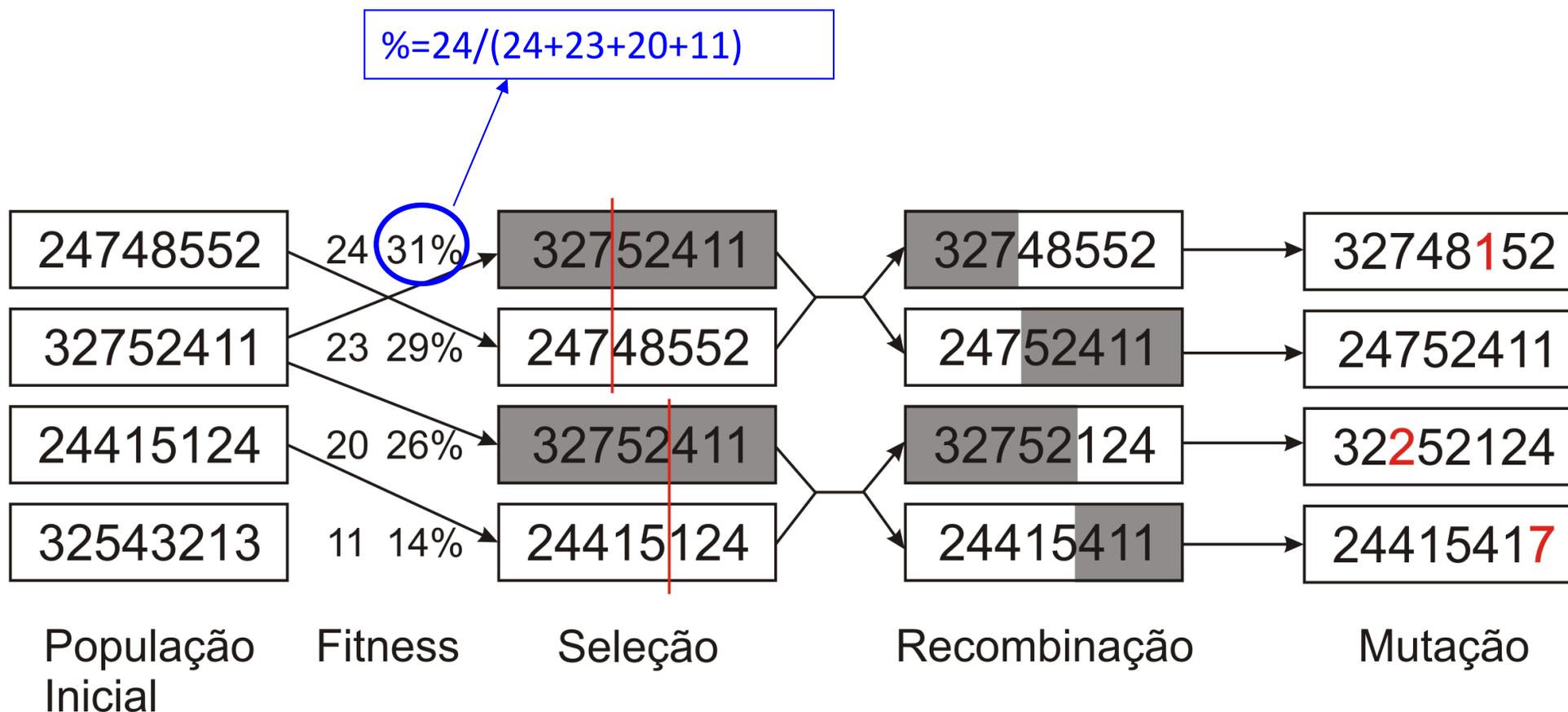
# Exemplo: 8 Rainhas

Recombinação:



# Exemplo: 8 Rainhas

- Indivíduos com maiores aptidões possuem mais chances de serem selecionados



# Bibliografia

- Busca local e problemas de otimização:
  - Capítulo 4 do livro texto (Russel & Norvig, Inteligência Artificial, 3a. Edição)