# Algoritmos de pesquisa local

## Capítulo 4, Secções 3–4

# Sumário

- ♢ Hill-climbing (trepa a colina)
- ♢ Simulated annealing (Arrefecimento simulado )
- ♢ Genetic algorithms (algoritmos genéticos) (abreviado)
- ♢ Pesquisa local em espaços contínuos (muito abreviado)

# Algoritmos iterativos

## (Iterative improvement algorithms)

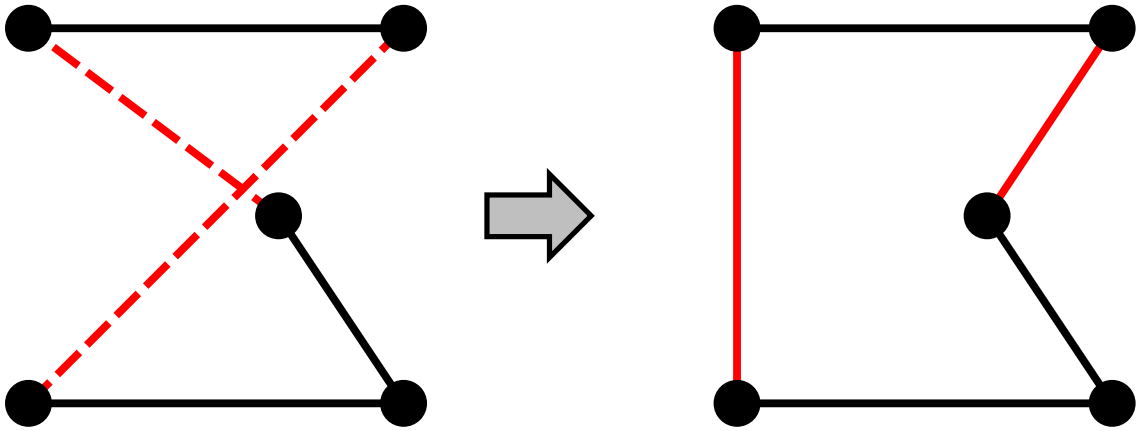
Em muitos problemas de optimização, o **caminho** é irrelevante;
O estado final é a solução.

O espaço de estados = conjunto "completo" de configurações;
encontrar a configuração **óptima**, e.g. TSP (problema do caixeiro viajante)
ou, encontrar a configuração que satisfaz as restrições, e,g, horário

Nesses casos, podemos usar o algoritmo iterativo;
guardar um estado "corrente" e tentar melhorá-lo.

# Exemplo: Problema do caixeiro viajante (TSP)

Fazer o estado inicial igual a uma volta completa pelos pontos, fazer mudanças aos pares

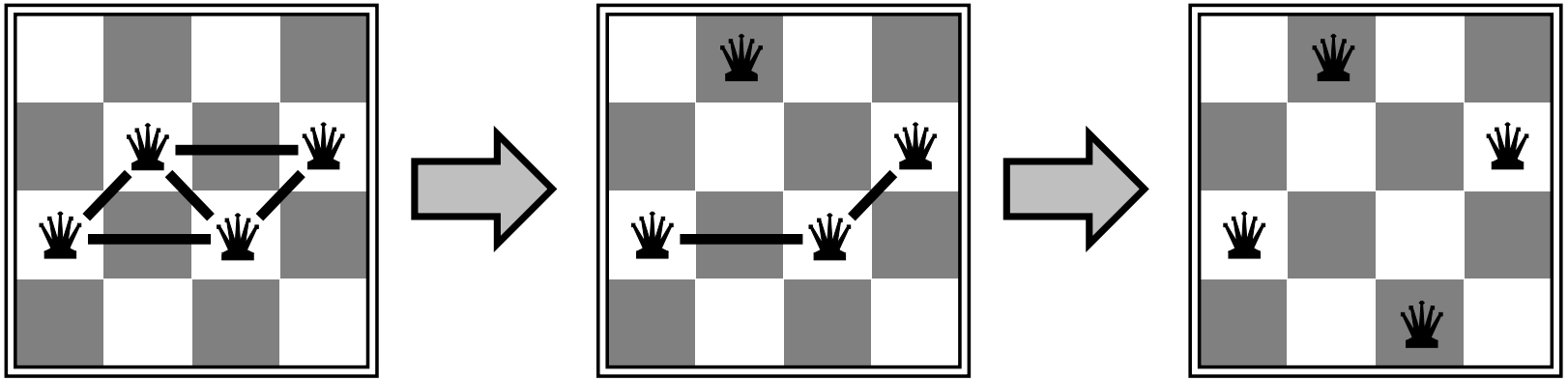


Variantes desta estratégia atingem soluções a 1% da óptima muito rapidamente com centenas de cidades.

# Exemplo: $n$-rainhas

Colocar $n$ rainhas num tabuleiro de $n \times n$ sem que haja duas rainhas na mesma linha, coluna, ou diagonal

Mover a rainha para reduzir o número de conflitos




Consegue-se resolver quase sempre o problema das $n$-rainhas instantaneamente
para $n$ muito grande, e.g., $n = 1$ milhão

# Hill-climbing (trepa a colina)

## (ou gradiante ascendente/descendente)

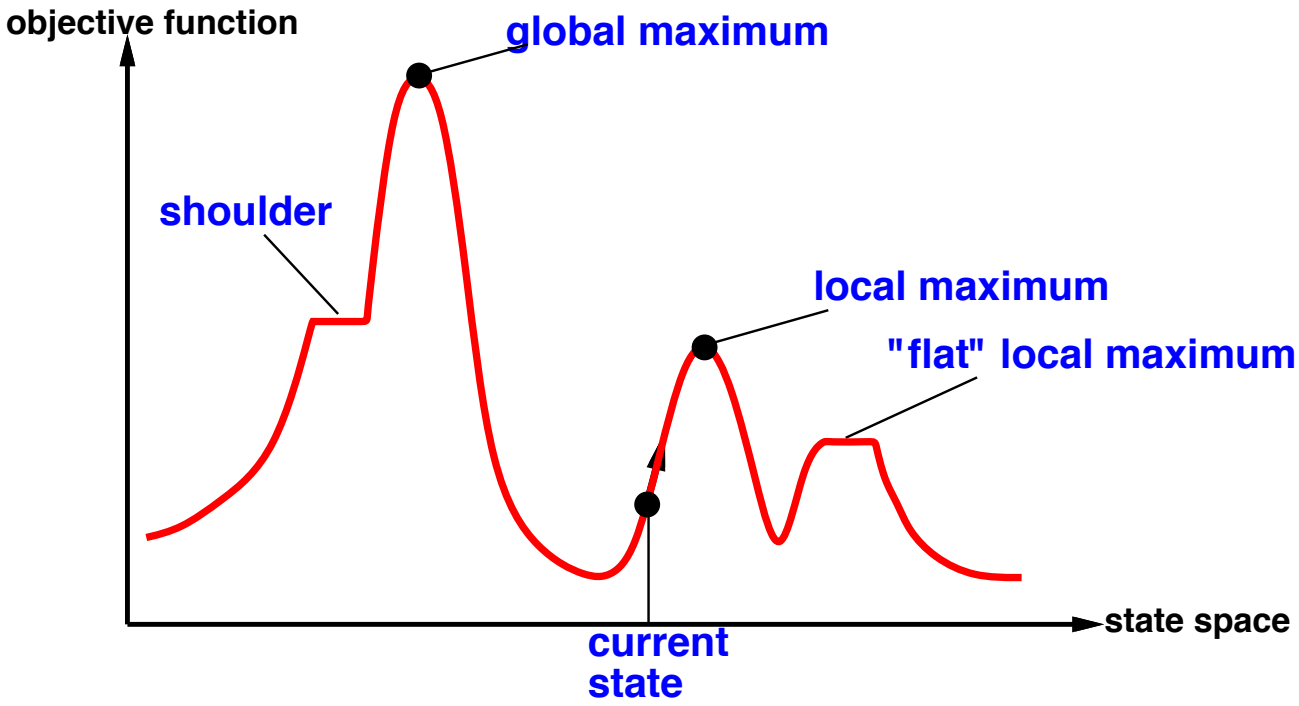
"Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

# Hill-climbing cont.

É útil considerar a paisagem do espaço de estados




Recomeçar aleatoriamente o hill climbing permite ultrapassar os máximos locais — completo

movimentos laterais aleatórios ☺ permite escapar dos planaltos de máximos

# locais

# Arrefecimento simulado - Simulated annealing

Ideia: escapar aos máximos locais permitindo alguns movimentos "maus"

**mas gradualmente desce a frequência o tamanho dos estados maus**

**function** SIMULATED-ANNEALING( *problem*, *schedule*) **returns** a solution state

**inputs**: *problem*, a problem
*schedule*, a mapping from time to "temperature"

**local variables**: *current*, a node
*next*, a node
$T$, a "temperature" controlling prob. of downward steps

*current* $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
**for** $t \leftarrow 1$ **to** $\infty$ **do**
$T \leftarrow$ *schedule*[$t$]
**if** $T = 0$ **then return** *current*
*next* $\leftarrow$ a randomly selected successor of *current*
$\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
**if** $\Delta E > 0$ **then** *current* $\leftarrow$ *next*
**else** *current* $\leftarrow$ *next* only with probability $e^{\Delta\ E/T}$

# Propriedades do simulated annealing

Numa dada "temperatura" $T$, a probabilidade de ocupação dos estados atinge a distribuição Boltzman

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

$T$ decresce lentamente $\Longrightarrow$ atinge sempre o melhor estado $x^*$
porque $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ para pequenos $T$

Isto é uma garantia interessante??

Desenvolvido por Metropolis et al., 1953, modelação de processos físicos.

Usado em desenho VLSI, escalonamento de companhias aéreas, etc.

# Pesquisa local em feixe - Local beam search

Ideia: guardar $k$ estados em vez de 1; escolher os melhores $k$ de todos os sucessores

Não é o mesmo que $k$ pesquisas a correrem em paralelo!
As pesquisas que encontram estados bons recrutam outras pesquisas para se juntarem

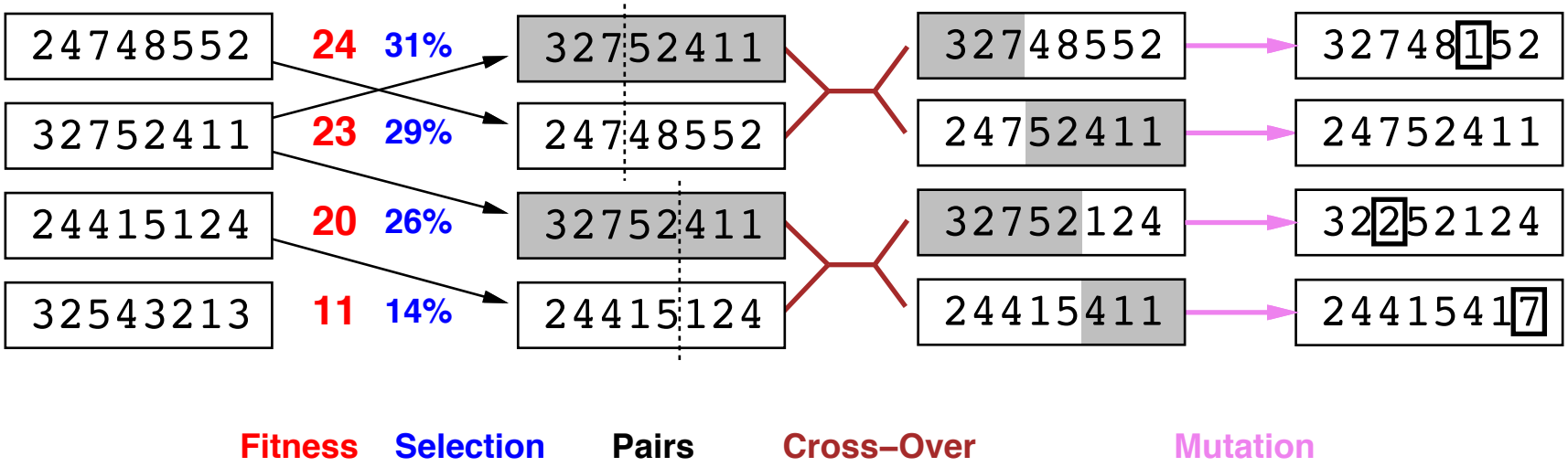
Problema: com alguma frequência, todos os $k$ estados acabam num máximo local

Ideia: escolher $k$ sucessores aleatoriamente

Há uma analogia com a selecção natural!

# Algoritmos Genéticos

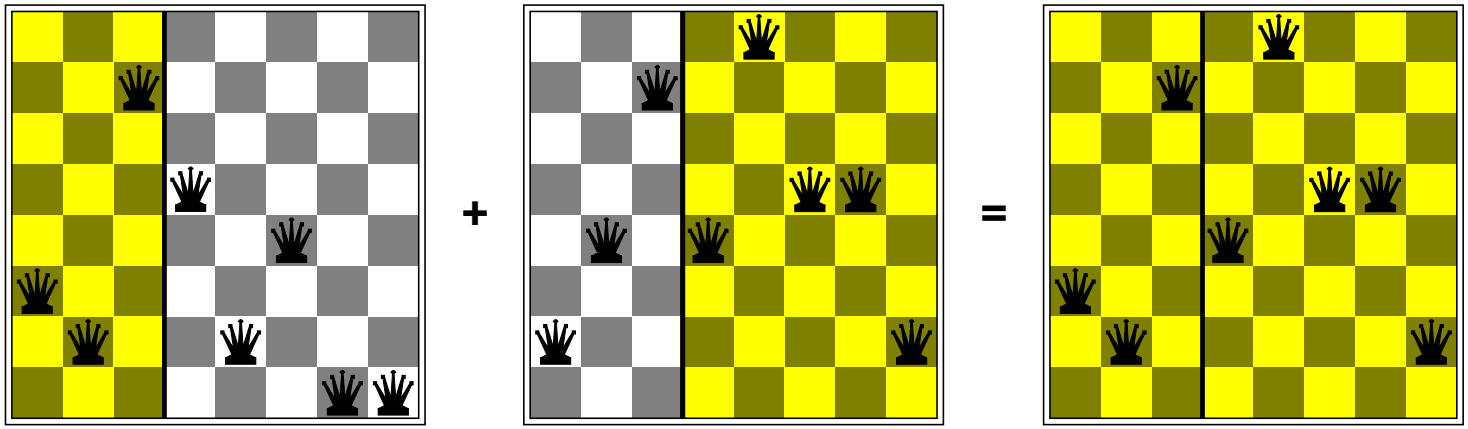
= pesquisa local em feixe estocástica + gerar sucessores a partir de **pares** de estados



# Algoritmos Genéticos cont.

AGs requerem que os estados sejam codificados em strings que são programas genéticos (GPs

O cruzamento ajuda **sse as substrings são componentes com significado**




AGs $\neq$ evolução: e.g., os genes reais codificam a maquina de replicação

# Espaços de estados contínuos - Continuous state spaces

Suponha que queremos situar 3 aeroportos na Roménia:

– 6-D espaços de estados definidos por $(x_1, y_2)$, $(x_2, y_2)$, $(x_3, y_3)$
  – função objectivo $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
    soma das distancias ao quadrado de uma cidade ao aeroporto mais mais próximo

métodos paradiscretização transformam um espaço continuo num espaço discreto,

e.g., o gradiante considera as $\pm\delta$ mudanças em cada coordenada

Métodos deGradiante calculam

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

para aumentar/reduzir $f$, e.g., com $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

Algumas vezes pode resolver para $\nabla f(\mathbf{x}) = 0$ exactamente (e.g., com uma cidade).
Newton–Raphson (1664, 1690) itera $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$
para resolver $\nabla f(\mathbf{x}) = 0$, onde $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$