

BUSCA LOCAL

(PARTE 4 – Resolução de problemas por meio de busca)

Roteiro

- Algoritmos de Busca Local
 - **Subida de encosta** (*Hill-climbing*)
 - **Têmpera Simulada** (*Simulated Annealing*)
 - **Feixe Local** (*Local beam search*)
 - Algoritmos **Genéticos**

Resolução de problemas por meio de busca local

INTRODUÇÃO

Algoritmos de Busca Local

- Em muitos problemas de otimização
 - o caminho ao objetivo é irrelevante (a sequência de ações),
 - só interessa o estado objetivo = solução
- Exemplos
 - problemas de otimização
 - satisfação de restrições (ex. N-Rainhas – configuração final)
- Espaço de estados
 - conjunto de configurações “completas” do mundo
 - Ex. um arranjo das n-rainhas no tabuleiro
- Nestes casos, podem ser usados **algoritmos de busca local**

Vantagens da Busca Local

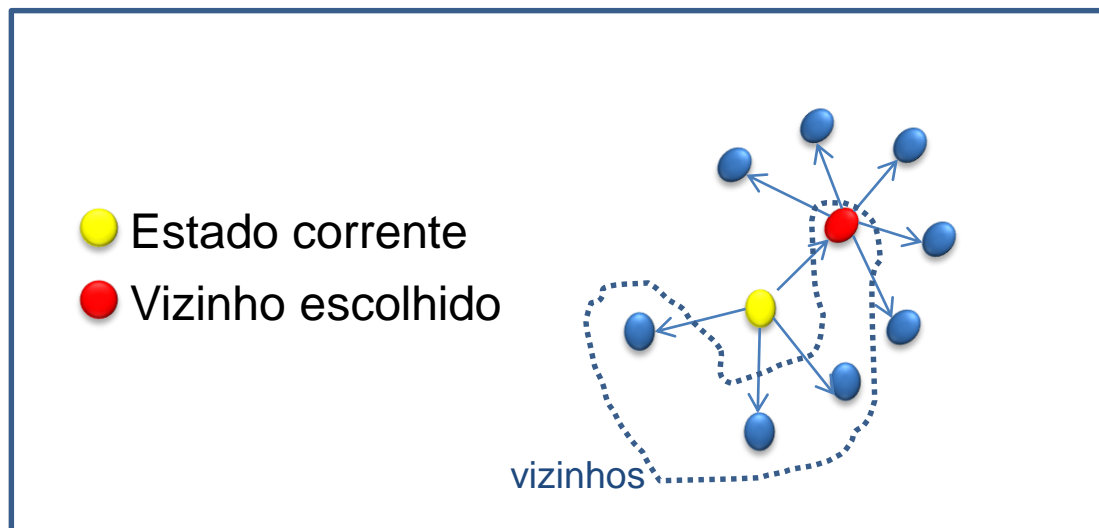
Vantagens em relação às buscas cega e informada:

1. usam pouca memória (normalmente, uma quantidade constante de memória)
2. podem encontrar soluções razoáveis/factíveis em espaços de estados grandes ou infinitos (e também contínuos) para os quais algoritmos sistemáticos como os de busca cega e informada não são adequados.

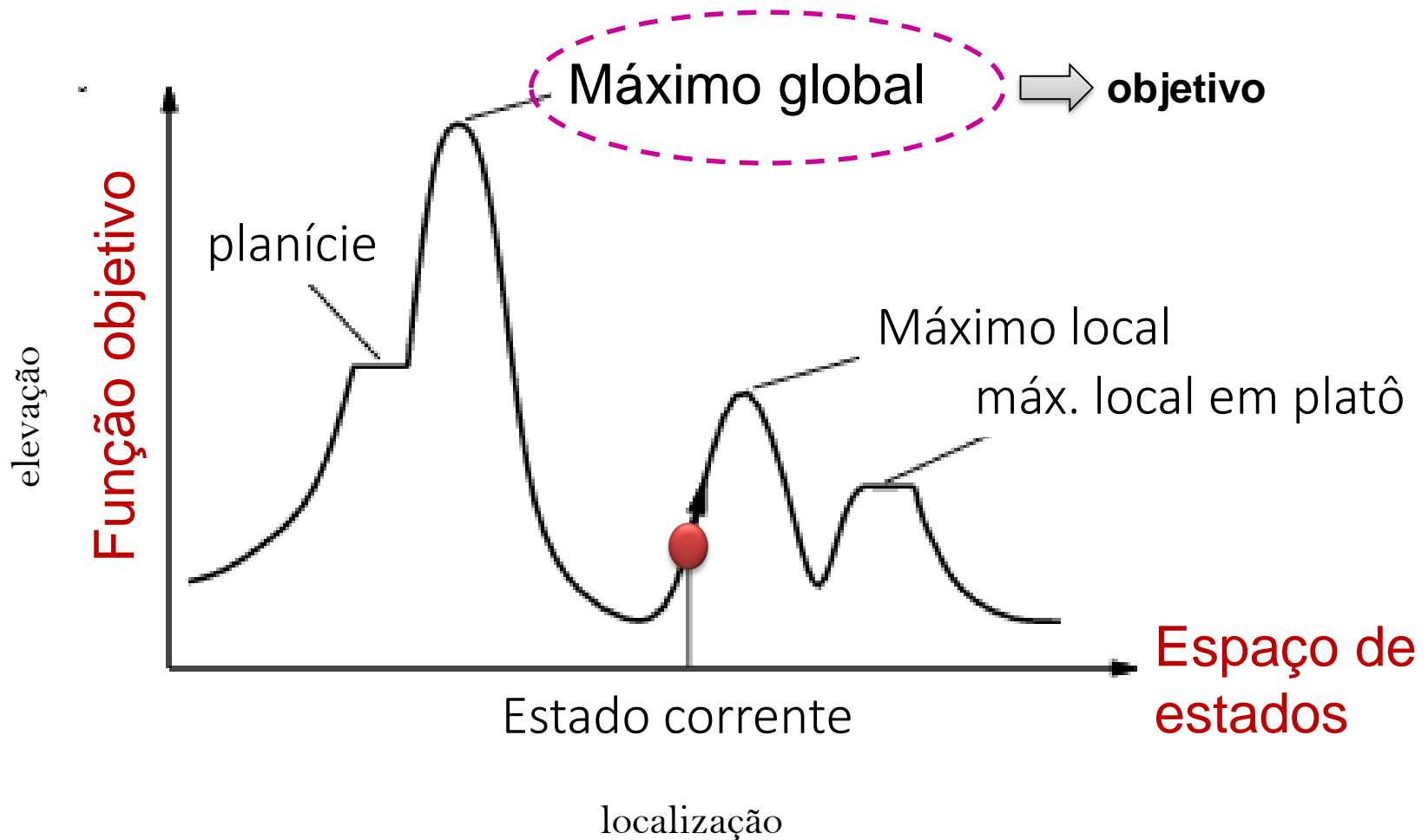
Algoritmos de Busca Local

Estratégia:

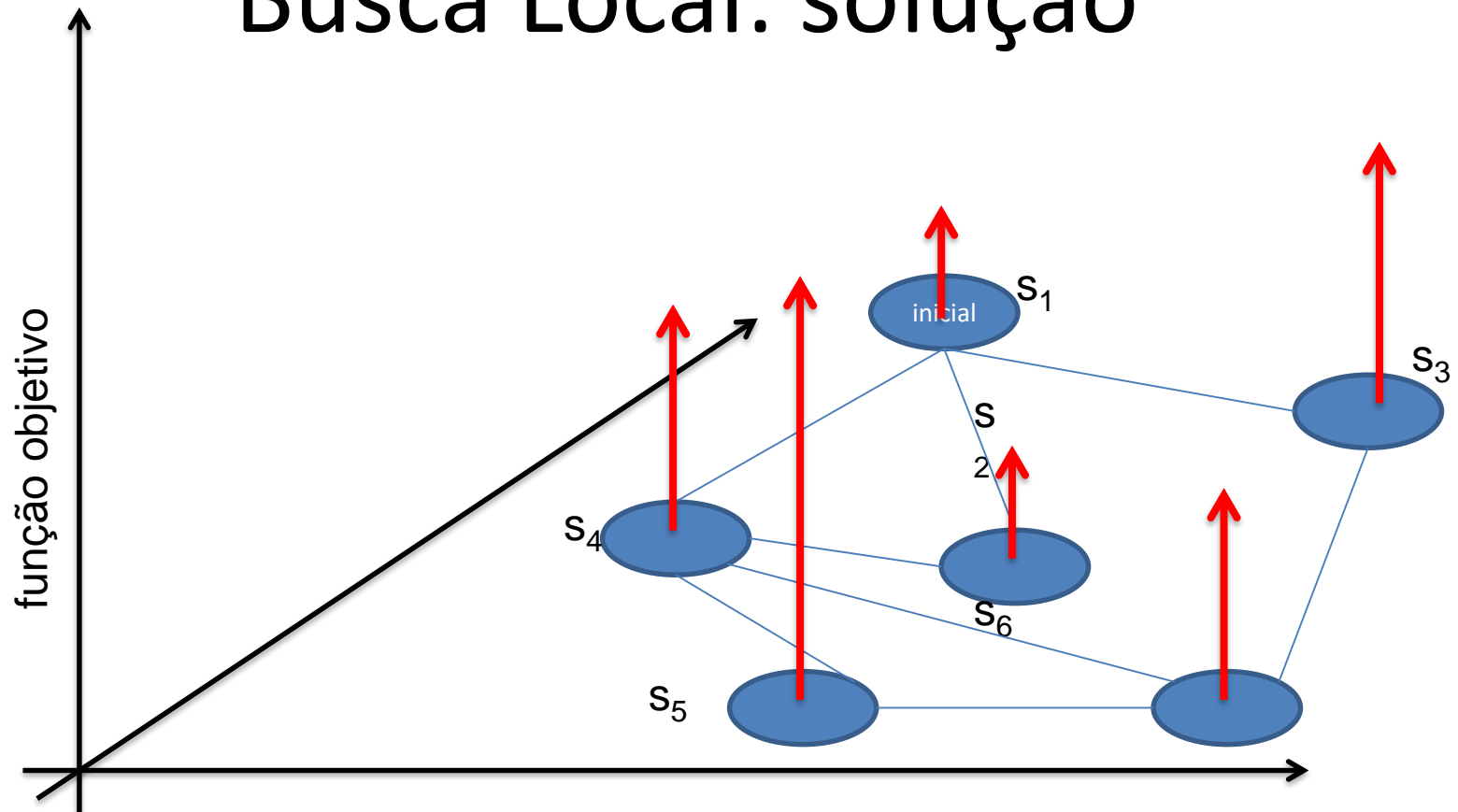
Manter um só estado (ou poucos) como o “atual” e tentar melhorar o mesmo movimentando-se para estados vizinhos imediatos



Algoritmos de Busca Local



Busca Local: solução



Em geral, a **solução** consiste em encontrar um estado (não um caminho) cujo valor da função objetivo seja possivelmente ótimo. Há a valoração da função objetivo (sabe-se que um estado é melhor do que outro) e esta valoração é usada para guiar a busca. Em muitos casos não há objetivo explícito/implícito (ex. Problema da mochila) que permita usar um procedimento de decisão para saber se um estado objetivo foi atingido, então, outros critérios devem ser usados para interromper a busca.

Algoritmos de Busca Local

Se o objetivo é maximizar temos que encontrar o maior pico, se for minimizar, o vale mais profundo.

Completo: se o algoritmo consegue atingir um estado objetivo desde que ele exista.

Ótimo: se consegue encontrar o mínimo/máximo para a função de custo.

Solução de problemas por busca

SUBIDA DE ENCOSTA (HILL CLIMBING)

Subida de Encosta (Hill-climbing)

Metáfora: escalando o Everest no meio de uma tempestade de neve com amnésia”

```
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 // melhor vizinho
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```

Subida de Encosta: 8-rainhas









- **Formulação:** cada estado tem as 8 rainhas no tabuleiro, uma por coluna.
- **Ação:** movimentar **uma rainha** para qualquer posição na sua coluna.
- **Função sucessora:** dado um estado e uma ação, retorna o estado alcançado.
 - Para cada estado, há 56 estados sucessores: 8 rainhas x 7 posições

Função objetivo: é uma heurística **h** = número de pares de rainhas que se atacam direta ou indiretamente

Estado objetivo: configuração com **$h=0$** (mínimo global)
somente para as soluções perfeitas

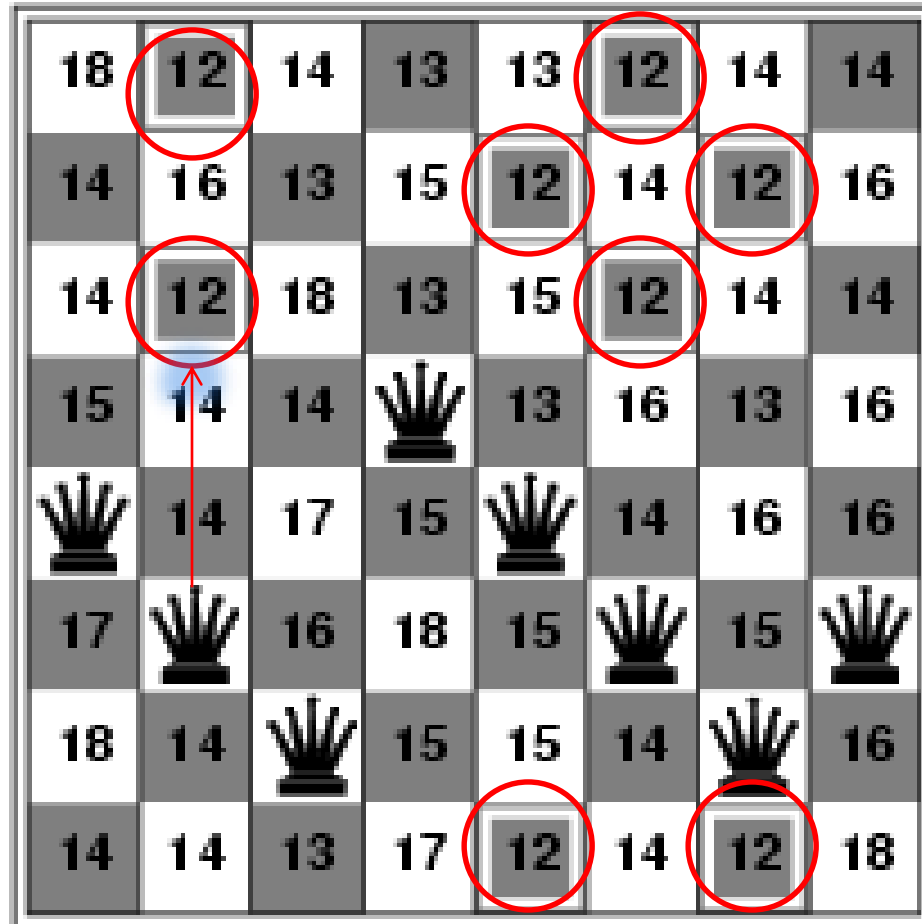
Subida de encosta: 8-rainhas

- Neste estado, **h=17** (há 17 ataques entre rainhas)
- Qual a melhor ação (por uma escolha gulosa)?

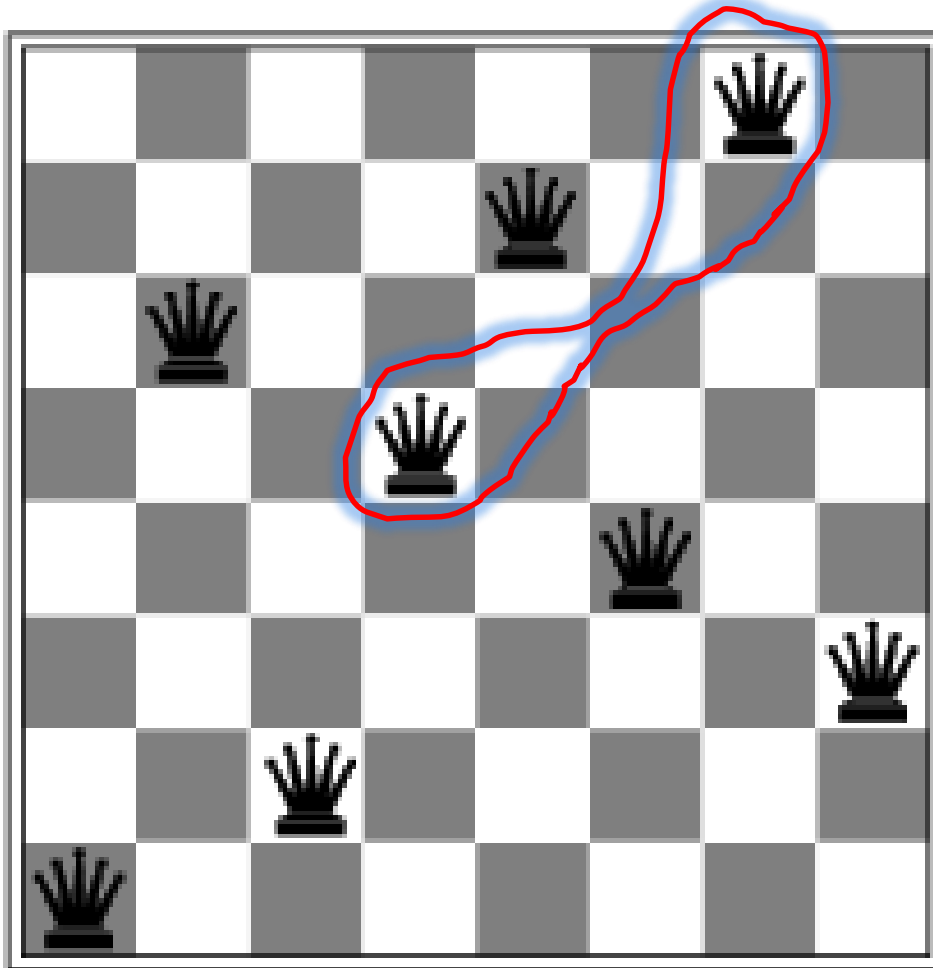
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

Subida de encosta: 8-rainhas

- **Resposta:** qualquer movimento que leve uma das rainhas na sua coluna a uma posição marcada com 12

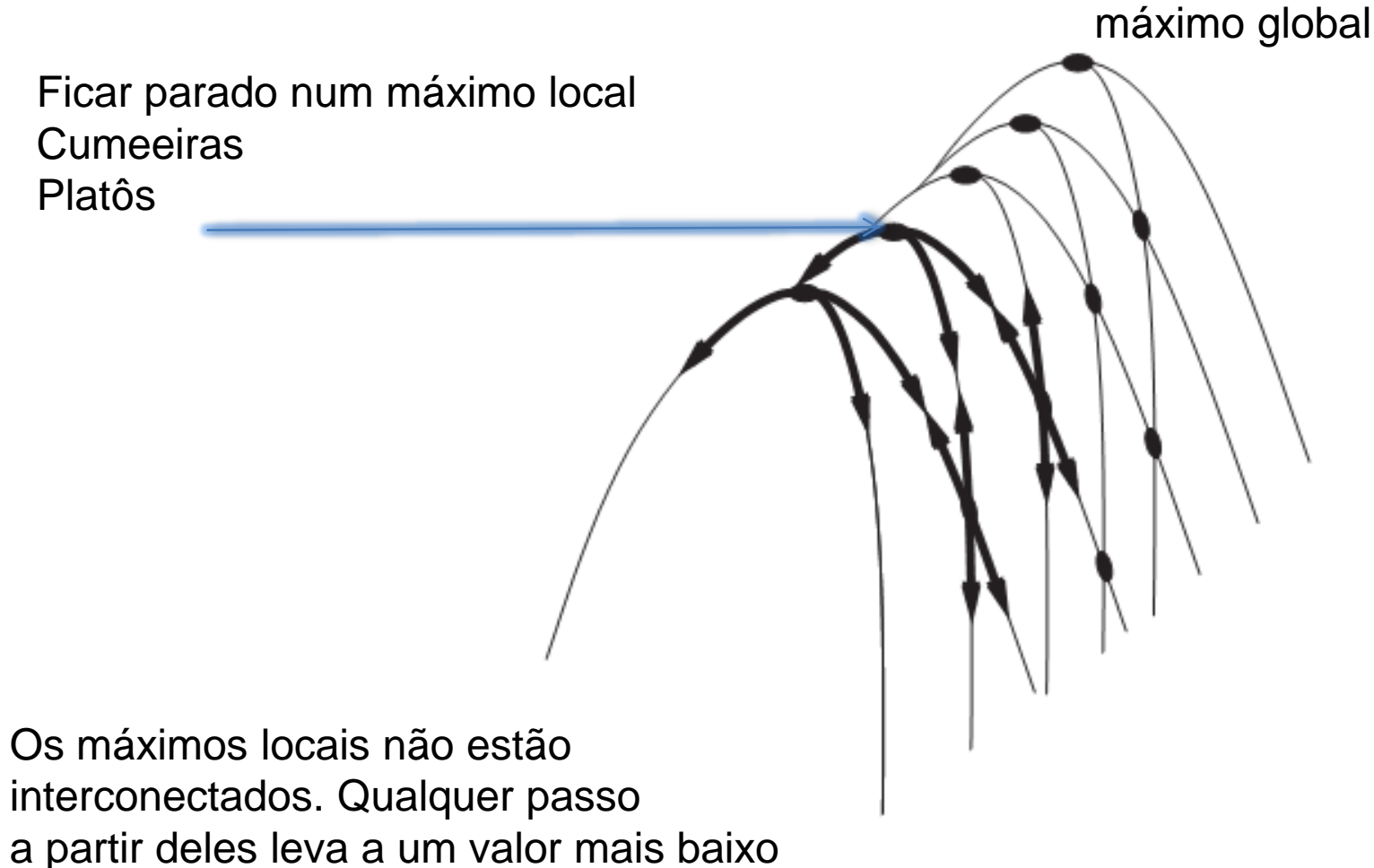


Subida de encosta: 8-rainhas

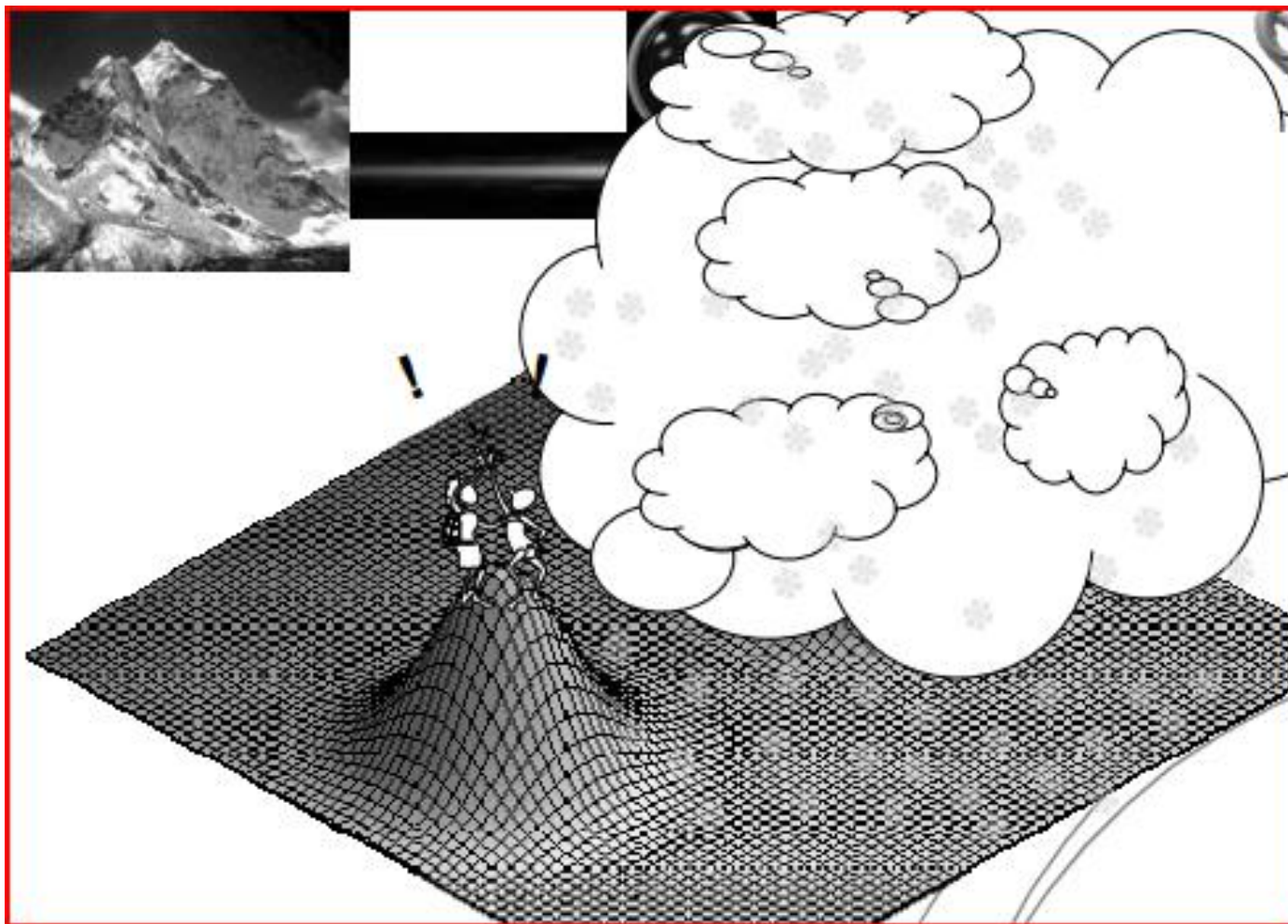


Um mínimo local com $h = 1$

Subida de encosta: problemas

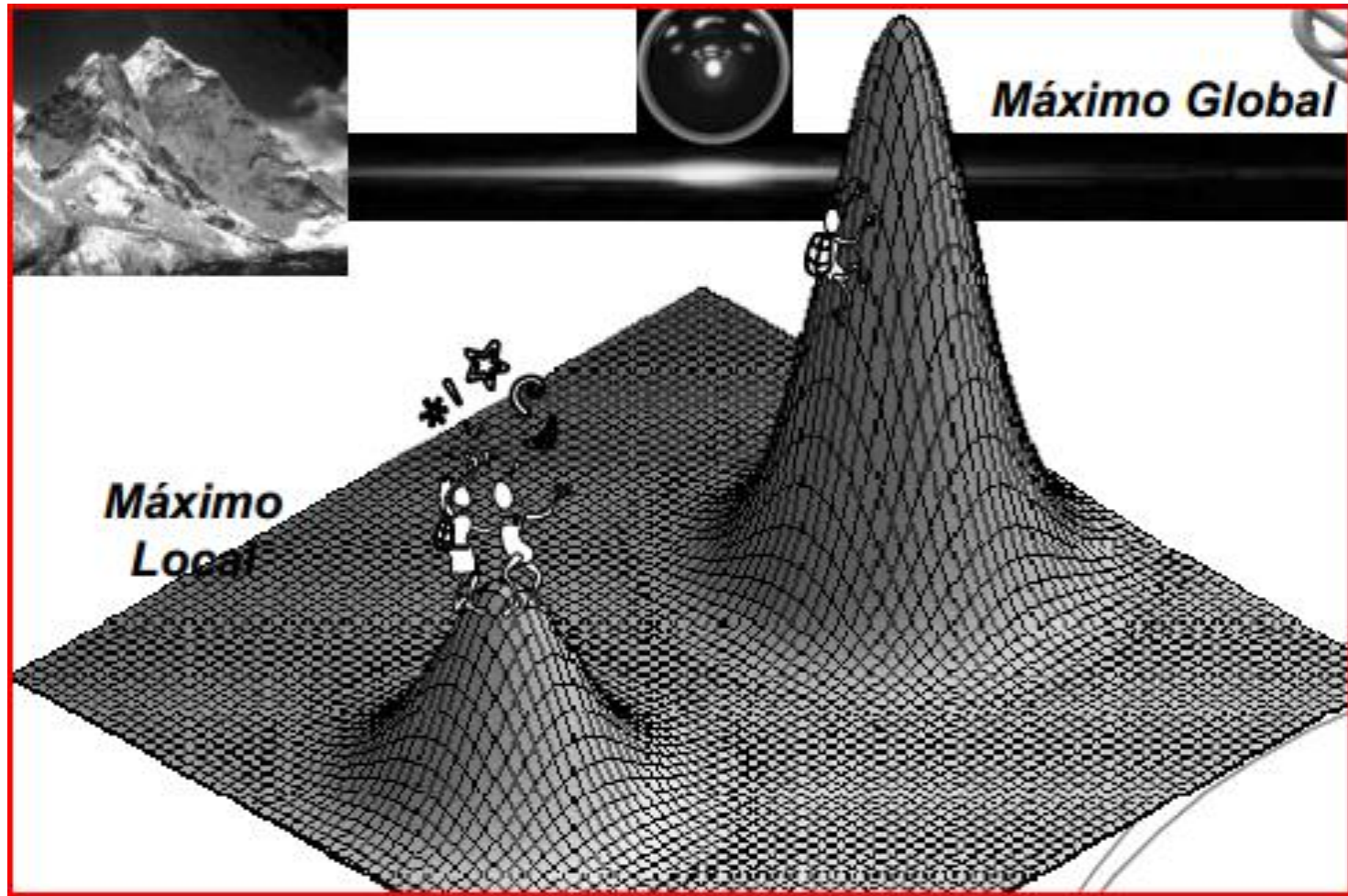


SUBIDA DE ENCOSTAS: PROBLEMAS



fonte: <http://cee.uma.pt/edu/iia/acetatos/iia-Procura%20Informada-PeB.pdf>

subida de encosta: problemas



fonte: <http://cee.uma.pt/edu/iia/acetatos/iia-Procura%20Informada-PeB.pdf>

SUBIDA DE ENCOSTA: *RANDOM RESTART*

- Solução para os problemas
 - fazer reinício aleatório (*random restart*)
 - O algoritmo realiza uma série de buscas a partir de estados iniciais gerados aleatoriamente.
- 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 execuções (diferentes reinícios).
- Cada execução produz apenas uma solução!

Solução de problemas por busca

TÊMPERA SIMULADA (SIMULATED ANNEALING)

Têmpera Simulada

- **Idéia:** fugir de máximo local permitindo alguns movimentos “ruins”, mas diminuindo gradualmente a frequência destas movimentações
 - a temperatura T que fornece energia para estas movimentações decresce ao longo do tempo

Têmpera Simulada: metáfora

Metáfora 1: têmpera de metais e de vidro



Propriedades da Têmpera Simulada

- Pode ser provado que:

Se T decresce de maneira suficientemente lenta, a busca por **têmpera simulada** encontrará um ótimo global com probabilidade próxima de 1 (100%)!!!

- Muito usado para projetar
 - layout de VLSI,
 - escalonamento de vôos
 - Escalonamento de horários em geral

TÊMPERA SIMULADA

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** ∞ **do**

T \leftarrow *schedule*[*t*] // *T schedule é a função que retorna T dado um tempo 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* // se *next* é melhor que o *current*

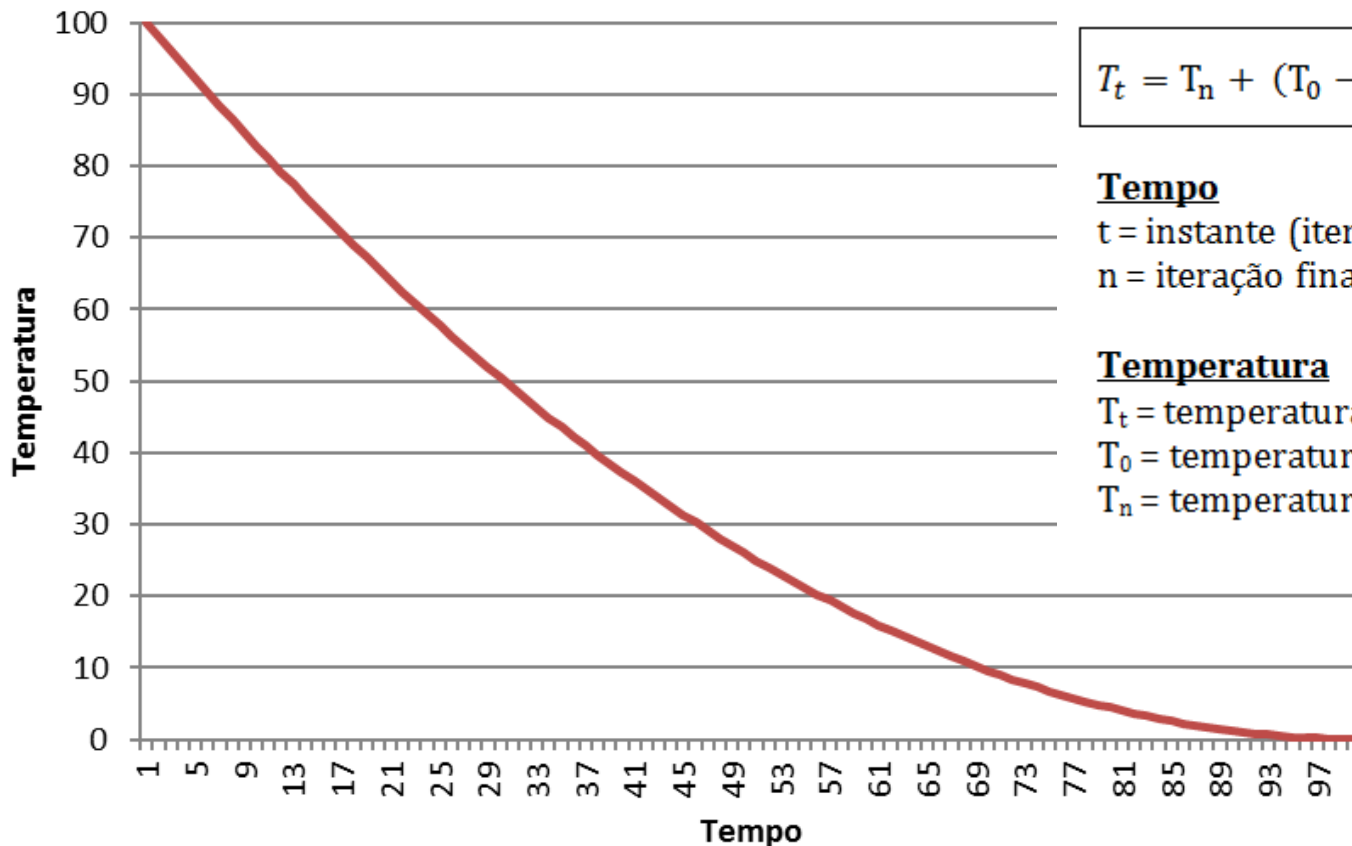
else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$. // faz uma ação ruim de tempos
// em tempos

AGENDAMENTO TÍPICO PARA A FUNÇÃO T

Exponencial decrescente

Temperatura (T) X Tempo (t)

Exemplo



$$T_t = T_n + (T_0 - T_n) * \left(\frac{n - t}{n}\right)^2$$

Tempo

t = instante (iteração) atual = 0

n = iteração final = 100

Temperatura

T_t = temperatura no instante t

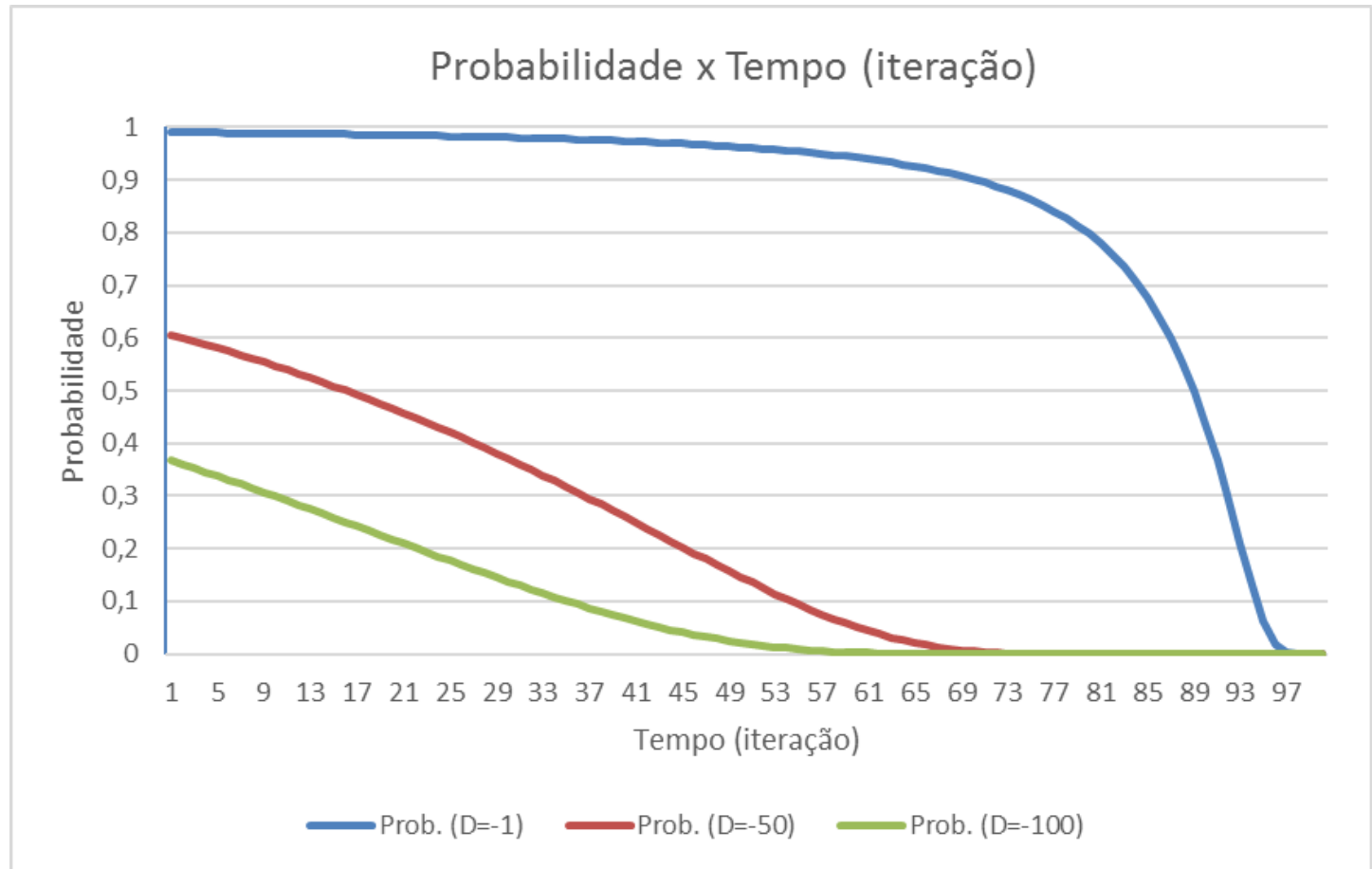
T_0 = temperatura inicial = 100

T_n = temperatura final (máxima) = 0

TÊMPERA SIMULADA

Simulação de probabilidades para T decaindo conforme função exp. decrescente para diferentes Δe ($D=-1$, $D=-50$, $D=-100$)

$$p = e^{\Delta e/T}$$



Soluções de problemas por busca

BUSCA EM FEIXE LOCAL

BUSCA EM FEIXE LOCAL

- Monitora k estados ao invés de um só
- Começa com k estados selecionados aleatoriamente
- Em cada iteração, todos os sucessores de todos os k estados são gerados
- Se qualquer um dos estados é o objetivo então para;
- Se não seleciona os k melhores da lista completa e repete
 - nesta seleção, considerar os estados 'pais'

BUSCA EM FEIXE LOCAL

function Beam-Search(problem, k) returns a solution state

start with **k** randomly generated states

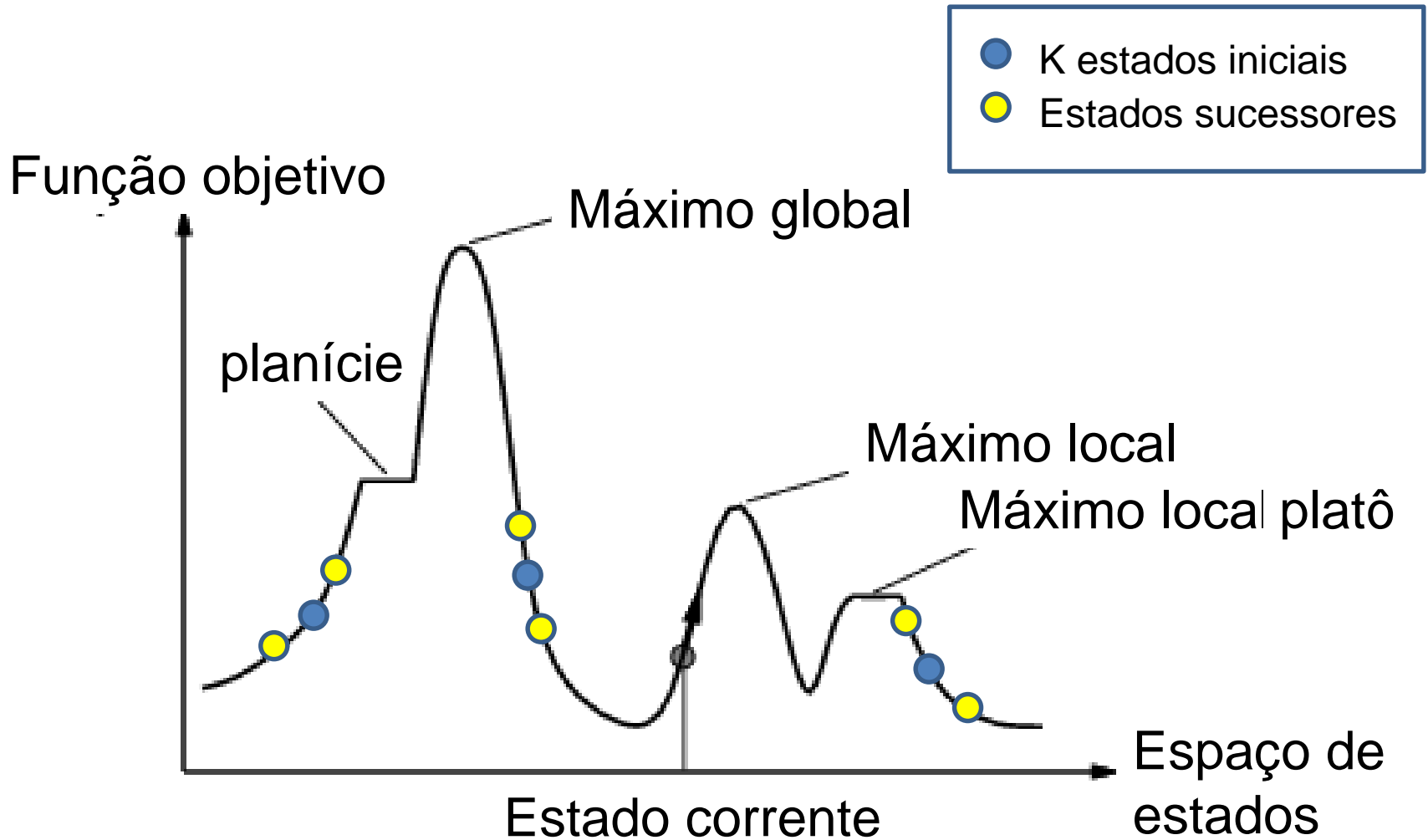
loop

generate all successors of all k states

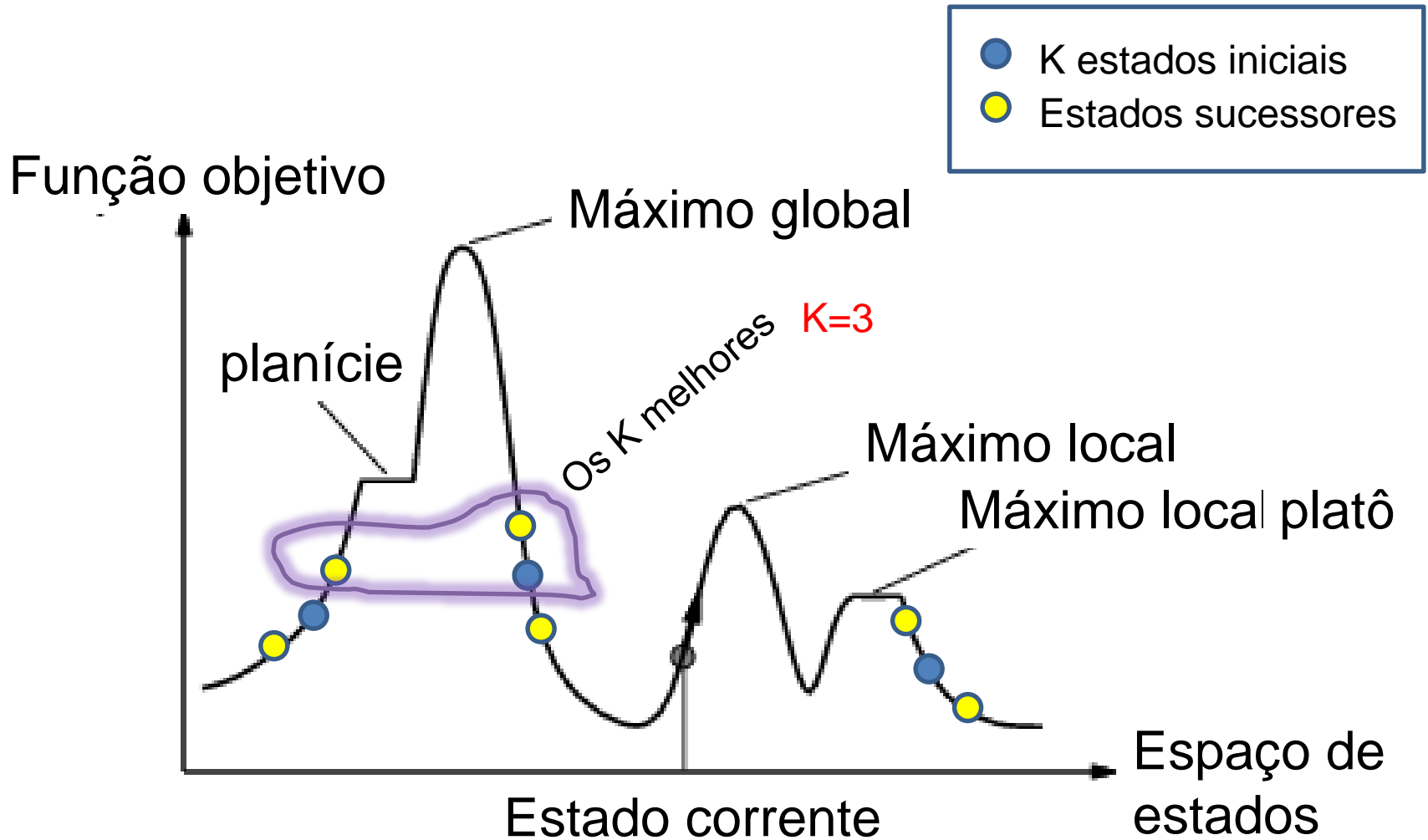
if any of them is a solution then return it

else select the k best successors

BUSCA EM FEIXE LOCAL



BUSCA EM FEIXE LOCAL



Exemplo (Mochila)

Vir da função objetivo para o melhor dos K estados por iteração

Estados iniciais aleatórios,
porém factíveis,
a busca em azul começa
pior!!!

Função objetivo x Iteração

