



INF 1771 – Inteligência Artificial

Aula 05 – Busca Local

Métodos de Busca

- **Busca Cega ou Exaustiva:**

- Não sabe qual o melhor nó da fronteira a ser expandido. Apenas distingue o estado objetivo dos não objetivos.

- **Busca Heurística:**

- Estima qual o melhor nó da fronteira a ser expandido com base em funções heurísticas.

- **Busca Local:**

- **Operam em um único estado e movem-se para a vizinhança deste estado.**

Estratégias Clássicas de busca

- Busca sistemática
 - um ou mais caminhos na memória
 - Quando chega no objetivo, o caminho para ele constitui a solução do problema

Estratégias Clássicas de busca

- Busca sistemática

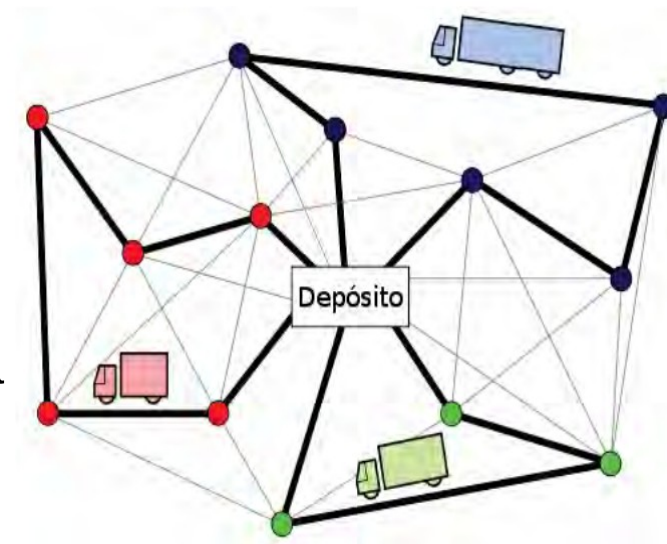
- um ou mais caminhos na memória
- Quando chega no objetivo, o caminho para ele constitui a solução do problema
 - ok para problemas pequenos
 - não-ok para problemas que requerem tempo exponencial para achar uma solução
 - o caminho para a meta pode ser irrelevante

Busca Local

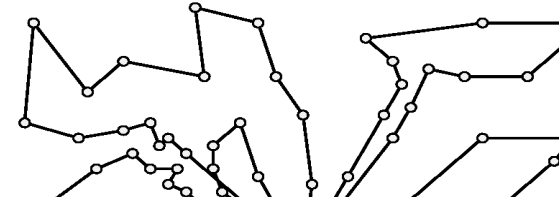
- Adequadas para **problemas de otimização**
- Enumerar os estados é **intratável**
 - algoritmos anteriores são muito caros
- **Não** existe uma estratégia para encontrar a **solução ótima** de forma **eficiente** quando o problema possui instâncias de tamanho grande.

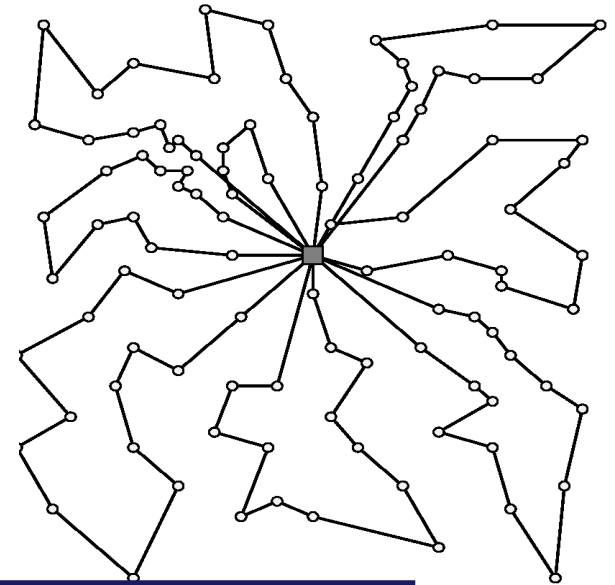
Problemas de Otimização

- Um exemplo: **Problema de Roteamento de Veículos**
- Entrada:
 - n clientes com localização e demandas;
 - Todas as distâncias;
 - Frota homogênea de m veículos localizados no depósito central.
- Saída:
 - Conjunto de Rotas de entrega (cada uma associada a um veículo) minimizando o custo total de viagem.
- **NP-difícil**



Problemas de Otimização

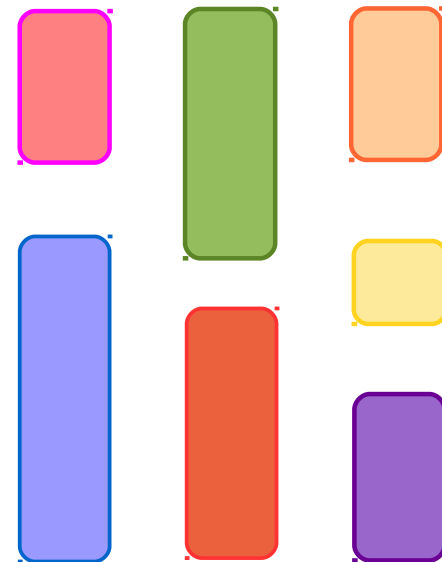
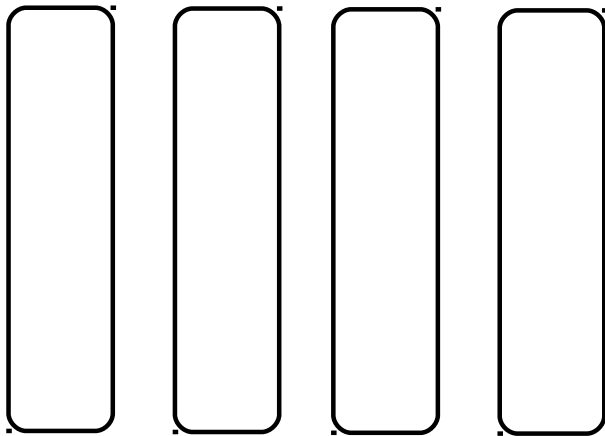
- Um exemplo: **Problema de Roteamento de Veículos**
 - Soluções exatas continuam impraticáveis para a maioria dos problemas de interesse (≥ 200 clientes).
 - Para um problema com $n=100$ clientes e um único veículo, o número de possíveis soluções é:
- 


$$n! = 933262154439441526816992388562667004907159682643816$$
$$2146859296389521759999322991560894146397615651828625369$$
$$792082722375825118521091686400000000000000000000000 \approx$$
$$10^{158}$$

- número de átomos no universo : 10^{80}

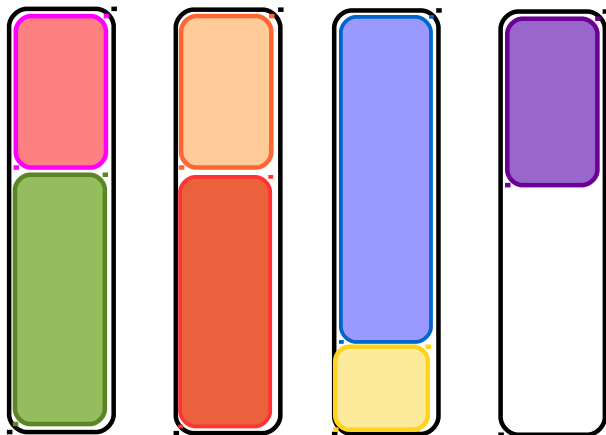
Exemplo – Problema de *Bin Packing*

- Dados: Número finito de *Bins* de capacidade c ; Um Conjunto $V = \{1, \dots, n\}$ de itens de peso w_1, \dots, w_n ;
- Alocar todos os itens minimizando o número de bins.



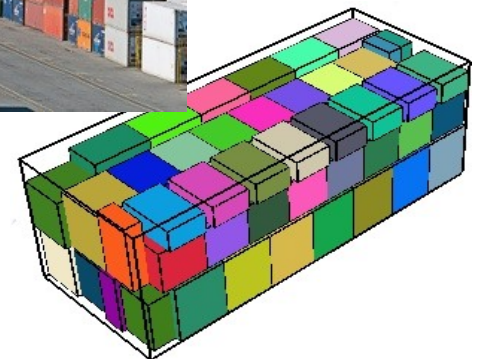
Exemplo – Problema de Bin Packing

- Dados: Número finito de Bins de capacidade c ; Um Conjunto $V = \{1, \dots, n\}$ de itens de peso w_1, \dots, w_n ;
- Alocar todos os itens minimizando o número de bins.



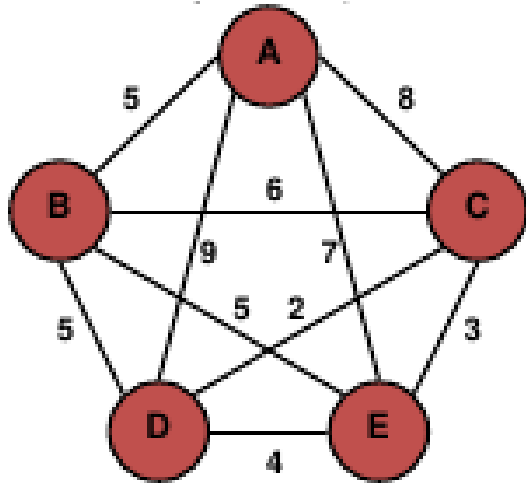
Exemplo – Problema de *Bin Packing*

- Aplicações:
 - Carregamento de caminhões com restrições de peso e capacidade;
 - Carregamento de *containers*;



Exemplo – TSP (Problema do Caixeiro Viajante)

- n cidades representando os clientes; distâncias entre as cidades.
- Percorrer as n cidades apenas uma vez e voltar a cidade de origem minimizando a distância total percorrida.

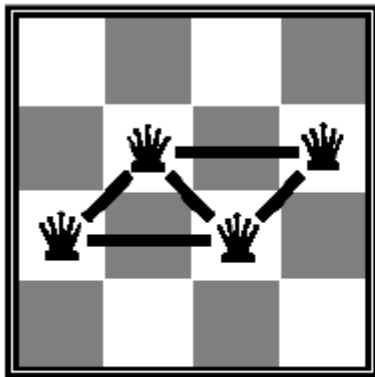


	A	B	C	D	E
A	0	5	8	9	7
B	5	0	6	5	2
C	8	6	0	3	4
D	9	5	3	0	4
E	7	2	4	4	0

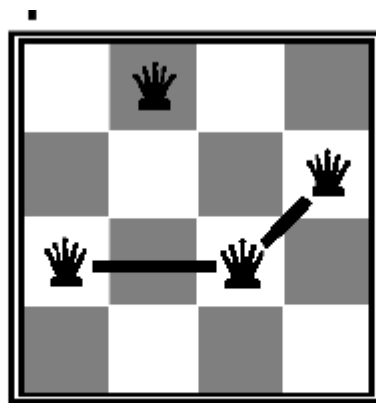


Exemplo - N rainhas

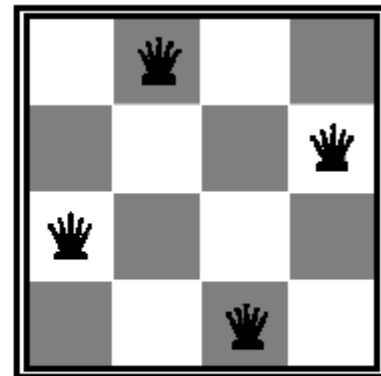
- Posicionar n rainhas em um tabuleiro $n \times n$ de modo que nenhum par de rainhas esteja se atacando.



$$f(x) = 5$$



$$f(x) = 2$$



$$f(x) = 0$$

Algoritmos de busca local

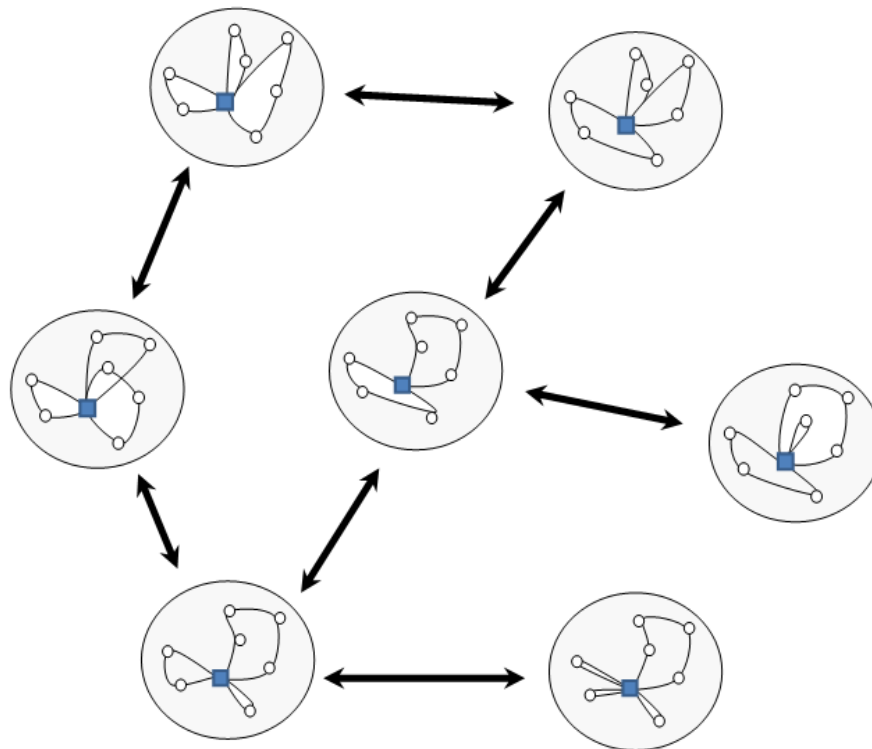
- Usam apenas um nó corrente
 - representam um **estado**, com uma função de **pontuação computável** ($f(x)$)
 - a meta é encontrar o estado com a pontuação **mais alta** (baixa)

Algoritmos de busca local

- Usam apenas um nó corrente
 - representam um **estado**, com uma função de **pontuação computável** ($f(x)$)
 - a meta é encontrar o estado com a pontuação **mais alta** (baixa)
 - Se movem para a **vizinhança** desse nó
- Não necessariamente guardam o caminho da solução

Algoritmos de busca local

- Um grafo direcionado $G=(V,A)$, onde cada nó em V representa uma solução e cada aresta (i,j) em A representa um vizinho j de i .



Vizinhança

- Considere um problema combinatório

$$f(s^*) = \min \{ f(s) : s \in S \}$$

S é o conjunto de soluções.

- Uma vizinhança N é **uma aplicação**, que associa, para cada elemento de S um subconjunto de soluções em S .

Vizinhança - exemplos

- TSP

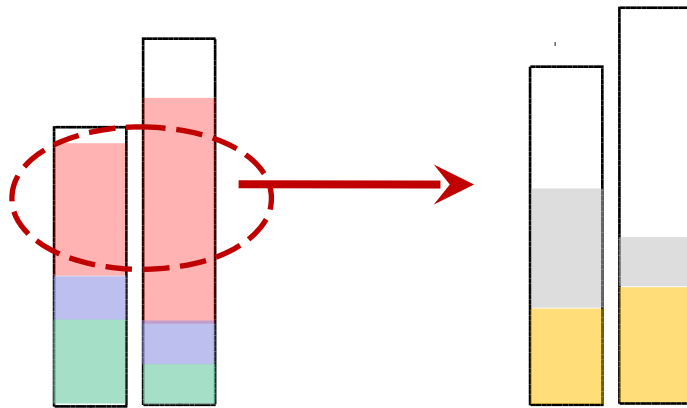
- trocar a posição de duas cidades em um tour

A-B-C-D-E swap(A,D) D-B-C-A-E

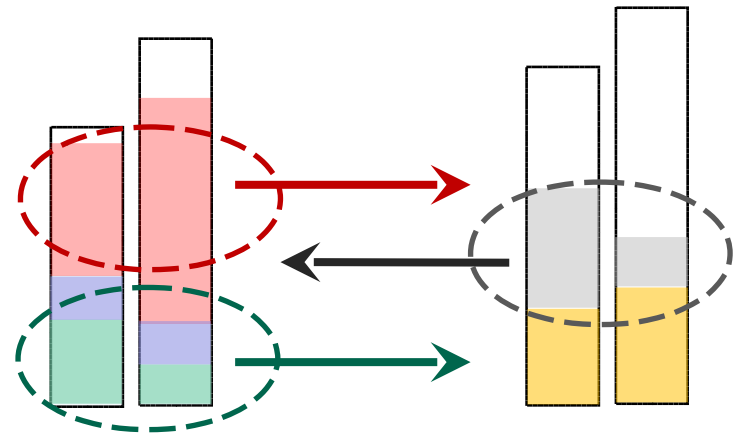
- N rainhas

- mover a rainha na posição mais conflitante mais a direita para uma posição diferente naquela coluna

Vizinhança - exemplos



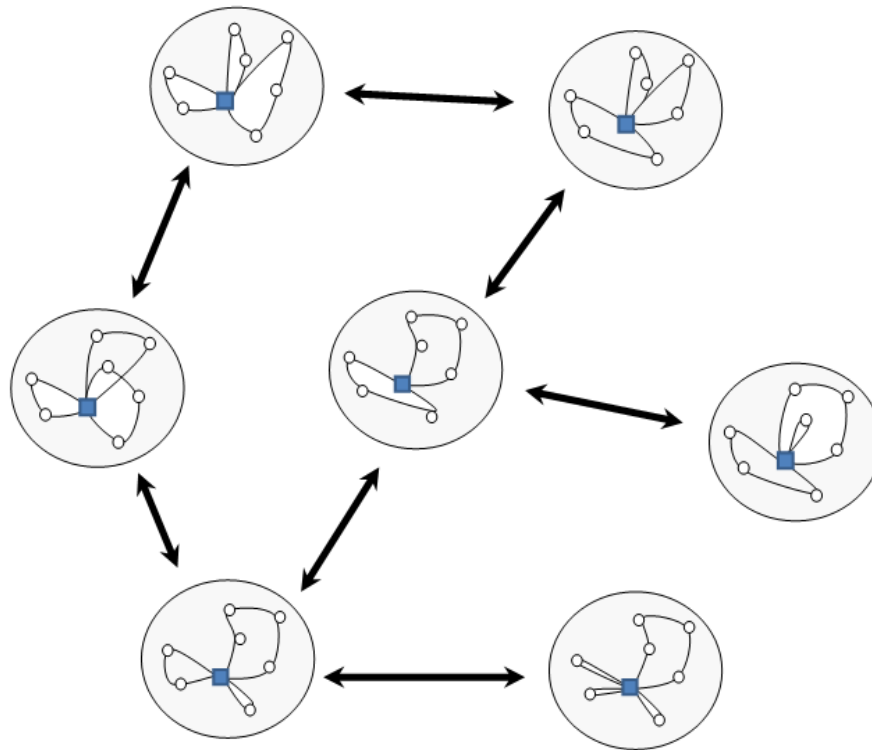
RELOCATE



SWAP (2 vs 1)

Espaço de Busca

- Um grafo direcionado $G=(V,A)$, onde cada nó em V representa uma solução e cada aresta (i,j) em A representa um vizinho de j de i .



Busca Local Genérica

function local-search (problem) returns um estado aceitável

`current` = make_node(problem.initial_state)

loop

if `current.value` possui um valor **aceitável**

return `current.state`

`neighbors` = seleciona e avalia alguns vizinhos de `current`

`neighbor` = um vizinho de `current` em `neighbors`

`current` = `neighbor`

Busca Local Genérica

function local-search (problem) returns um estado aceitável

current = make_node(problem.initial_state)

loop

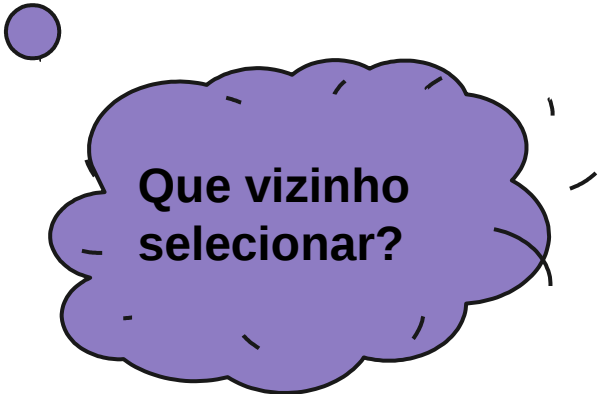
if **current.value** possui um valor **aceitável**

return **current.state**

neighbors = seleciona e avalia alguns vizinhos de **current**

neighbor = um vizinho de **current** em **neighbors**

current = **neighbor**



Que vizinho
selecionar?

Busca Local Genérica

function local-search (problem) returns um estado aceitável

current = make_node(problem.initial_state)

loop

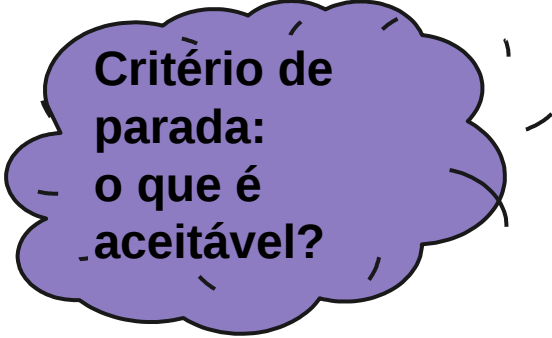
if **current.value** possui um valor **aceitável**

return **current.state** •

neighbors = seleciona e avalia alguns vizinhos de **current** •

neighbor = um vizinho de **current** em **neighbors** •

current = **neighbor** •



Critério de
parada:
o que é
aceitável?

Busca Local mais básica: Hill-climbing

function Hill-climbing (problem) returns um estado que é um máximo local

`current` = make_node(problem.initial_state)

loop

`neighbor` = um sucessor de `current` que tenha o maior valor de pontuação

 if `neighbor.value` <= `current.value`

 return `current.state`

`current` = `neighbor`

Busca Local mais básica: Hill-climbing

function Hill-climbing (problem) returns um estado que é um máximo local

`current` = make_node(problem.initial_state)

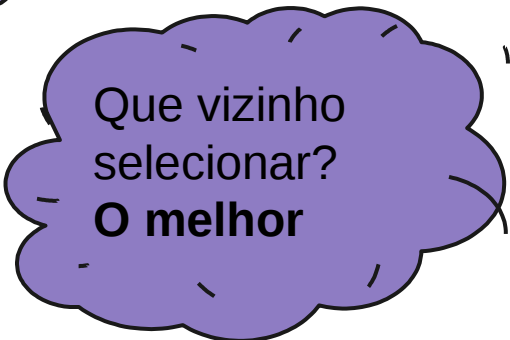
loop

`neighbor` = um sucessor de `current` que tenha o maior valor de pontuação

if `neighbor.value` <= `current.value`

return `current.state`

`current` = `neighbor`



Que vizinho
selecionar?
O melhor

Busca Local mais básica: Hill-climbing

function Hill-climbing (problem) returns um estado que é um máximo local

`current` = make_node(problem.initial_state)

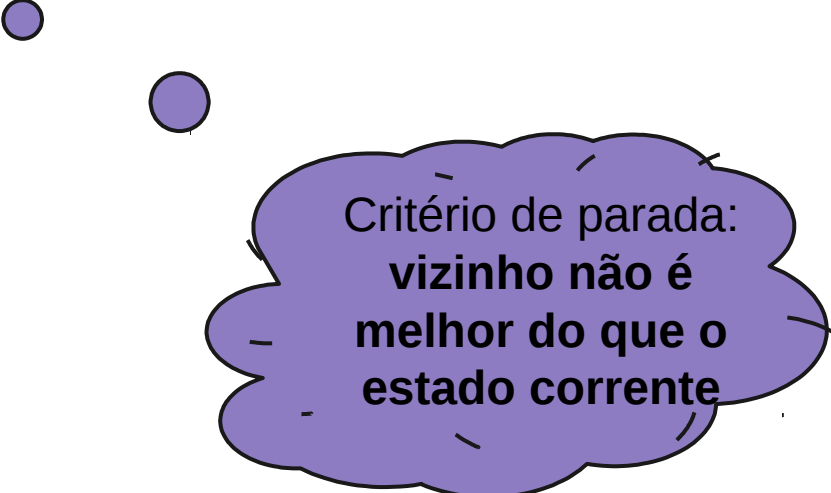
loop

`neighbor` = um sucessor de `current` que tenha o maior valor de pontuação

if `neighbor.value` <= `current.value`

return `current.state`

`current` = `neighbor`



Critério de parada:
**vizinho não é
melhor do que o
estado corrente**

Busca Local

- Algoritmo:

- Heurística que iterativamente move de uma solução S para o melhor vizinho em $N(s)$ até alcançar um máximo local/mínimo local.

```
S = S0 ; /* Initial Solution */
```

```
While not Terminated
```

```
    Explore( N(S) ) ;
```

```
    If there is no better neighbor in  $N(s)$  Then Stop ;
```

```
    S = Select( N(S) ) ;
```

```
End While
```

```
Return Final solution found (local optima).
```

Busca Hill-climbing

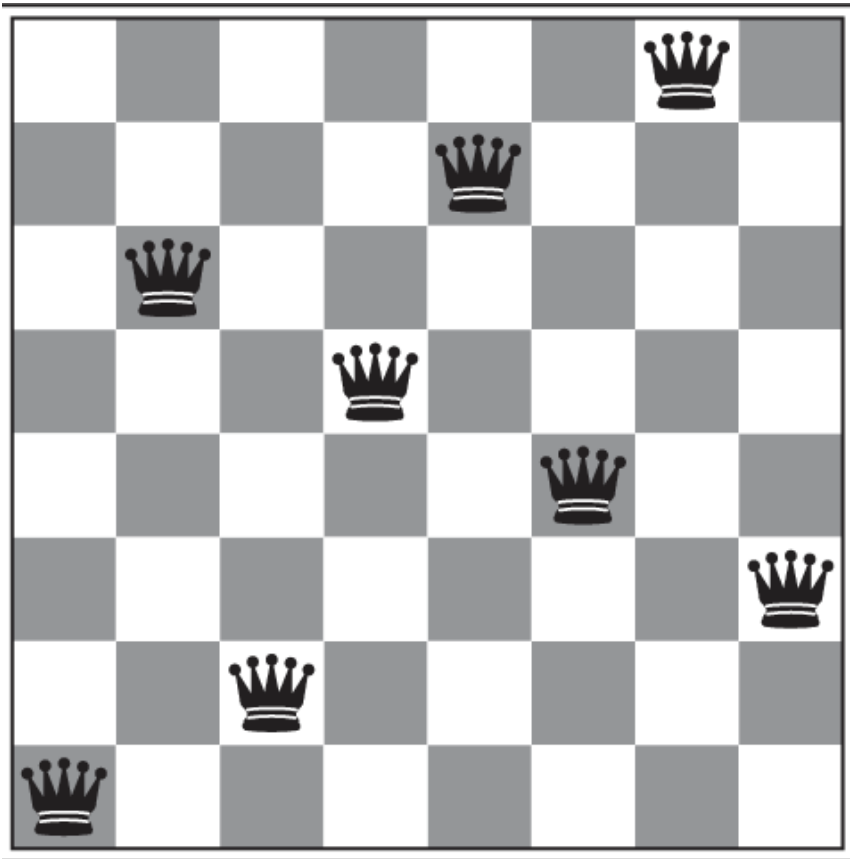
- $f(x)$: pares de rainhas se atacando

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

$$f(x) = 17$$

Busca Hill-climbing básica é gulosa

- Depois de 5 passos...

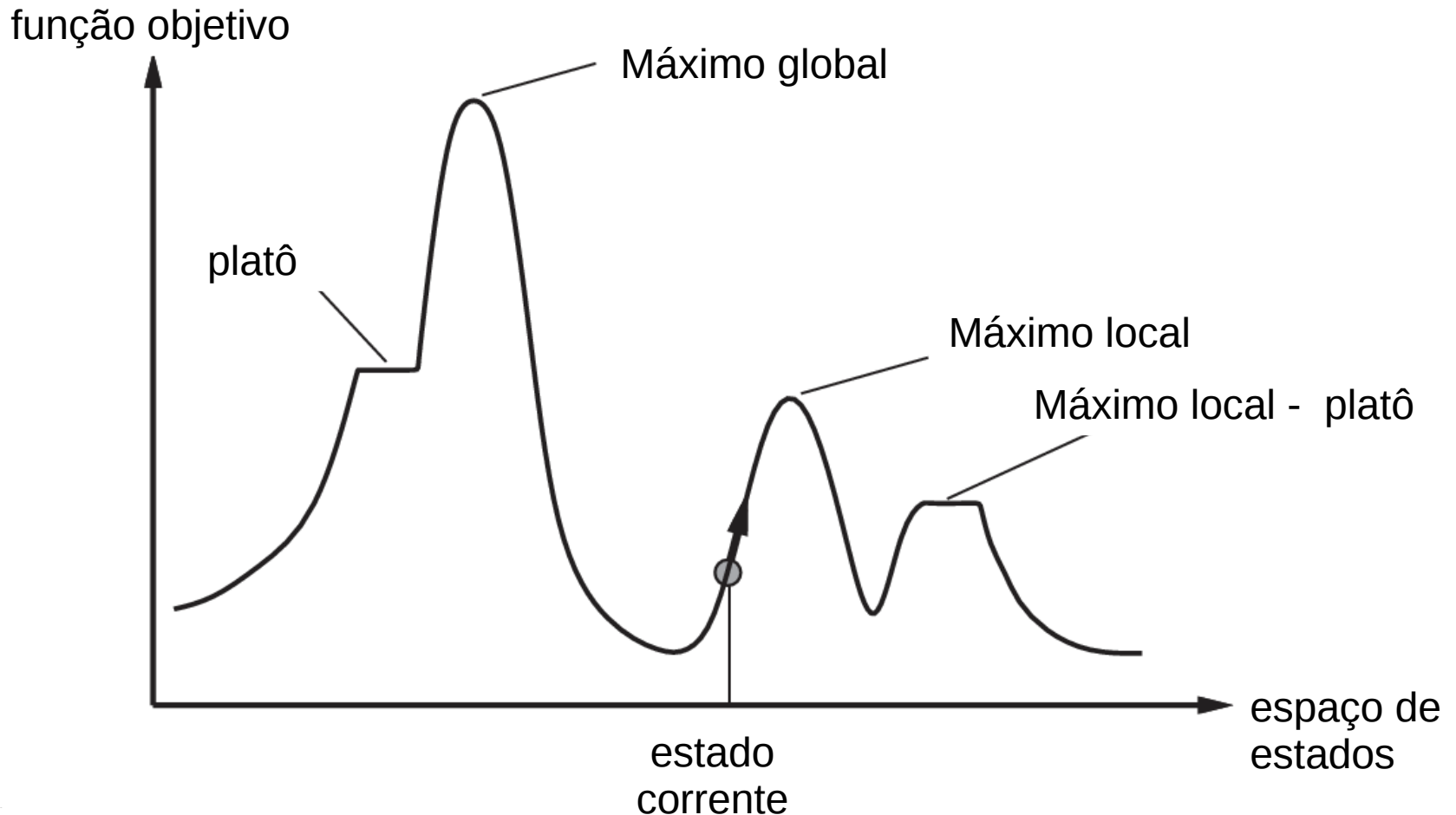


$$f(x) = 1$$

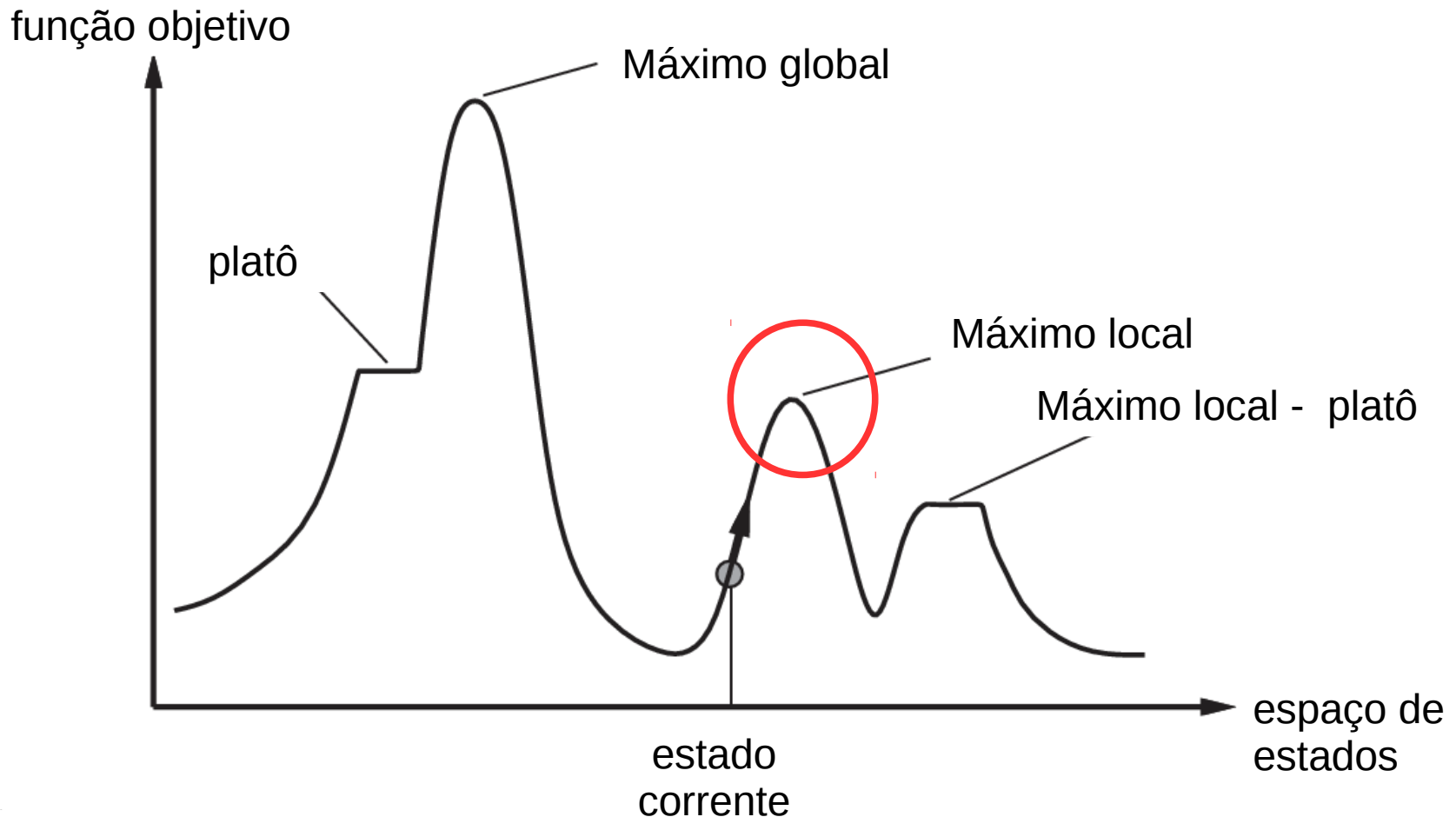
Hill Climbing

- É um **algoritmo guloso** – escolhe sempre o primeiro melhor vizinho para progredir na busca.
- Essa abordagem pode ter **bons resultados em alguns problemas**. Sendo capaz de progredir rapidamente para a solução problema.
- Mas, sofre de sérios **problemas**:
 - Máximos locais
 - Platôs

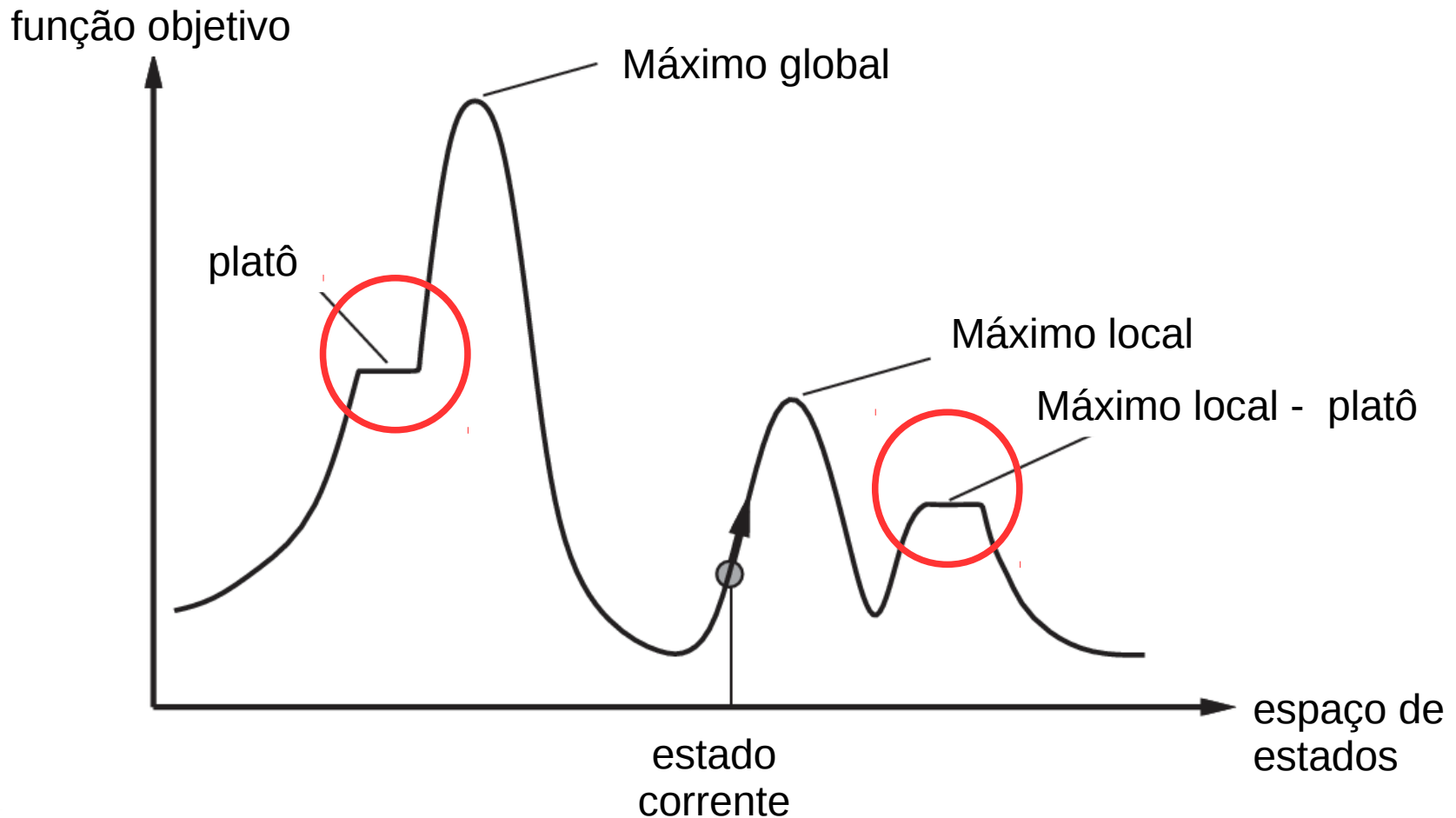
Topologia do espaço de estados



Busca Hill climbing - parado em máximos locais



Busca Hill climbing - platôs



Hill climbing: variações (e lições de vida)

1

Dê um passo a frente
e você não estará
mais no mesmo lugar

Busca Hill-climbing : sideway moves

function Hill-climbing-sw (problem) returns um estado que é um máximo local

`current` = make_node(problem.initial_state)

loop

`neighbor` = um sucessor de `current` que tenha o maior ou igual valor de pontuação

 if `neighbor.value` < `current.value`

 return `current.state`

`current` = `neighbor`

Busca Hill-climbing : sideway moves

function Hill-climbing-sw (problem) returns um estado que é um máximo local

current = make_node(problem.initial_state)

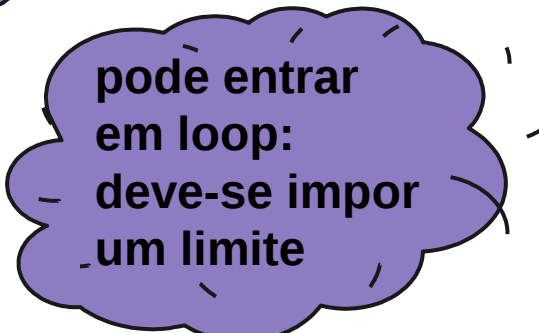
loop

neighbor = um sucessor de **current** que tenha o maior ou igual valor de pontuação

if **neighbor.value** < **current.value**

return **current.state**

current = **neighbor**



podê entrar
em loop:
deve-se impor
um limite

Hill climbing

2

Se você ainda não
atingiu o sucesso,
tente de novo,
mas repense suas
escolhas

Random-restart hill climbing

- Estados iniciais diferentes vão "subir na encosta" por caminhos diferentes

Random-restart hill climbing

Ao ficar preso em um máximo local, reinicie a busca a partir de um ponto aleatório

Random-restart hill climbing

- Estados iniciais diferentes vão "subir na encosta" por caminhos diferentes

Random-restart hill climbing

Ao ficar preso em um máximo local, reinicie a busca a partir de um ponto aleatório

Repita esse processo K vezes

Random-restart hill climbing

- Estados iniciais diferentes vão "subir na encosta" por caminhos diferentes

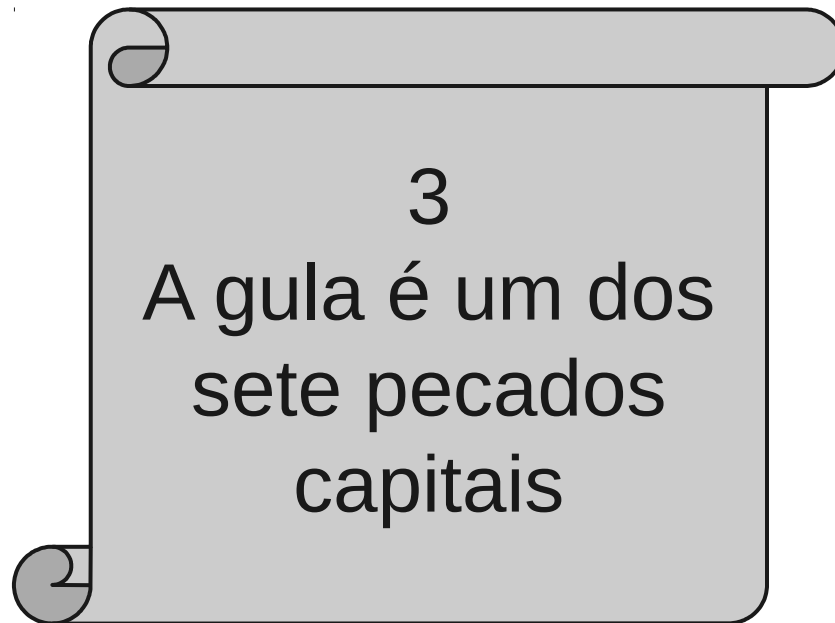
Random-restart hill climbing

Ao ficar preso em um máximo local, reinicie a busca a partir de um ponto aleatório

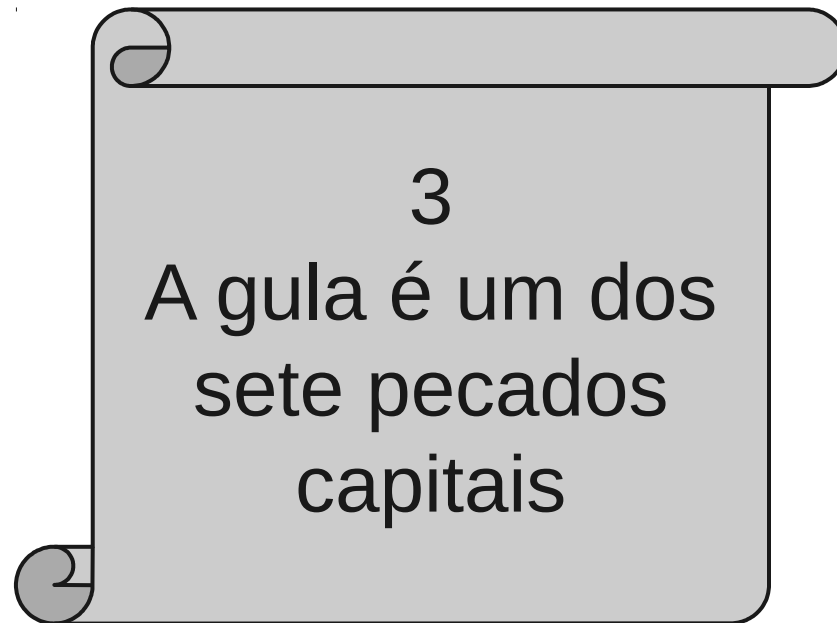
Repita esse processo K vezes

Retorne o k-ésimo melhor máximo local

Hill climbing: variações



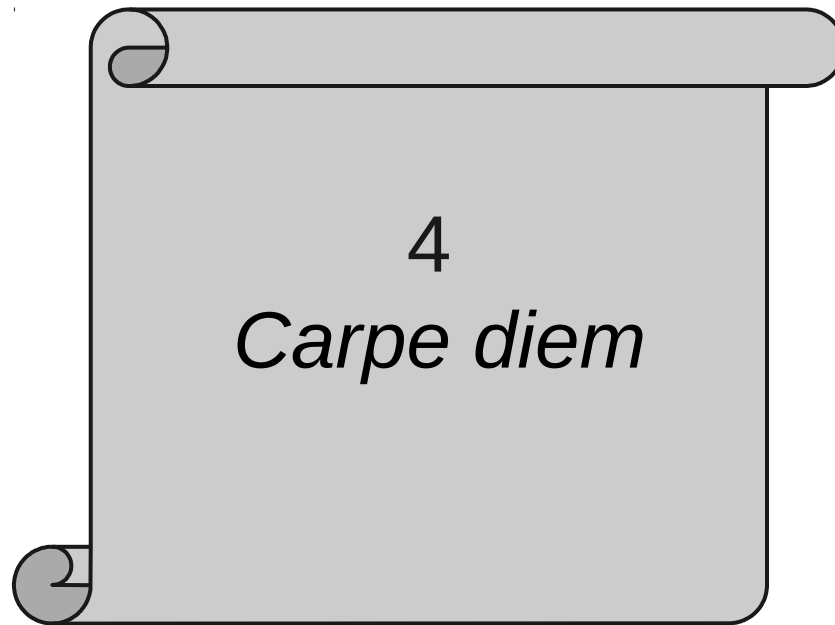
Hill climbing: variações



Hill climbing estocástico

- Seleciona um vizinho de forma **aleatória**
o e não necessariamente o melhor vizinho de todos
- Ainda **exige** uma **melhora** na pontuação

Hill climbing: variações



Hill climbing com primeira escolha

- Vizinhança é muito populosa
 - gasta **tempo** para enumerar
- Escolhe **primeiro** vizinho **que melhora** a pontuação (First-improvement)

Hill climbing com primeira escolha

- Vizinhança é muito populosa
 - gasta **tempo** para enumerar
- Escolhe **primeiro** vizinho **que melhora** a pontuação (First-improvement)
- First improvement X Best improvement

Hill climbing: variações

5

As vezes, para
melhorar é
preciso dar um
passo para trás

Hill climbing com caminhadas aleatórias

- Com uma **probabilidade p** :
 - escolha um **vizinho qualquer**
 - na maioria das implementações, não exige melhora na pontuação nesse passo
 - *Movimento estocástico: random walk*
- Caso contrário:
 - escolha o vizinho com a **maior pontuação**
 - *Movimento guloso*

Busca Hill climbing

- Hill climbing clássico
 - nunca "desce a encosta"
 - nunca permite uma piora na pontuação do novo estado
 -
 - preso em máximos locais
- Busca aleatória
 - move para um sucessor escolhido aleatoriamente e uniformemente
 - ineficiente

Busca Hill climbing

- Hill climbing clássico

- nunca "desce a encosta"

- nunca permite uma piora na avaliação do novo estado

- incompleto

- preso em locais

- Busca aleatória

- move para um sucessor escolhido aleatoriamente e uniformemente

- completo, mas ineficiente



**E se
combinarmos
as
estratégias?**

Simulated Annealing, ideia geral

1. Escolha um estado inicial, s
2. Escolha aleatoriamente um estado t , a partir dos vizinhos de s
3. se $f(t)$ for melhor do que $f(s)$, então
 $s = t$
 senão, com uma probabilidade baixa, faça
 $s = t$
4. Vá para o passo 2 até cansar

Simulated Annealing, ideia geral

1. Escolha um estado inicial, s
2. Escolha aleatoriamente um estado t , a partir dos vizinhos de s

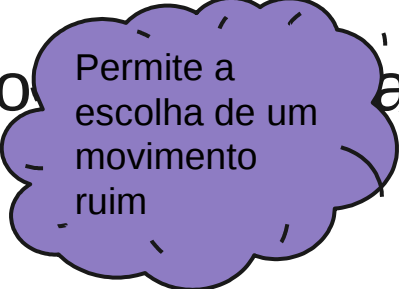
3. se $f(t)$ for melhor do que $f(s)$, então

$$s = t$$

senão, com uma probabilidade baixa, faça

$$s = t$$

4. Vá para o passo 2



Permite a
escolha de um
movimento
ruim

Simulated Annealing

- Probabilidade de aceitar um movimento ruim
 - **varia** de acordo com o **quão ruim** ele é
 - **diminui** conforme o **tempo** passa

Controle do processo de "annealing"

- considere a mudança no desempenho da pontuação
 - $\Delta E = f(\text{novoEstado}) - f(\text{estadoCorrente})$
- se $\Delta E > 0$
 - aceite o novo estado (subida)
- se $\Delta E \leq 0$, o novo estado deve passar em um teste

Controle do processo de "annealing"

- O teste usa uma probabilidade definida pela equação de Boltzmann

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

- $\Delta E \rightarrow -\infty$, $P \rightarrow 0$

◦ se o movimento for muito ruim, probabilidade de aceitá-lo, **diminui exponencialmente**

- $T \rightarrow 0$, $P \rightarrow 0$

◦ conforme temperatura diminui, probabilidade de aceitar um movimento ruim, **diminui exponencialmente**

■ temperatura diminui com o número de passos

Simulated Annealing

Select an initial solution $\omega \in \Omega$

Select the temperature change counter $k = 0$

Select a temperature cooling schedule, t_k

Select an initial temperature $T = t_0 \geq 0$

Select a repetition schedule, M_k , that defines the number of iterations executed at each temperature, t_k

Repeat

Set repetition counter $m = 0$

Repeat

Generate a solution $\omega' \in N(\omega)$

Calculate $\Delta_{\omega, \omega'} = f(\omega') - f(\omega)$

If $\Delta_{\omega, \omega'} \leq 0$, then $\omega \leftarrow \omega'$

If $\Delta_{\omega, \omega'} > 0$, then $\omega \leftarrow \omega'$ with probability $\exp(-\Delta_{\omega, \omega'} / t_k)$

$m \leftarrow m + 1$

Until $m = M_k$

$k \leftarrow k + 1$

Until stopping criterion is met

Pseudo-código: A.G. Nikolaev and S. H. Jacobson, Simulated annealing. in Handbook of Metaheuristics, 2nd Ed.

Simulated annealing - pros e contras

- + Pode chegar em um **máximo global**, se T diminui bem lentamente
- + **Escapa** de máximos locais

Simulated annealing - pros e contras

- + Pode chegar em um **máximo global**, se T diminui bem lentamente
- + **Escapa** de máximos **locais**
- dependente dos parâmetros
- **saída** pode ser completamente **diferente** em diferentes execuções

Simulated annealing

Ilustração do funcionamento do Algoritmo de Simulated Annealing procurando o máximo de uma função

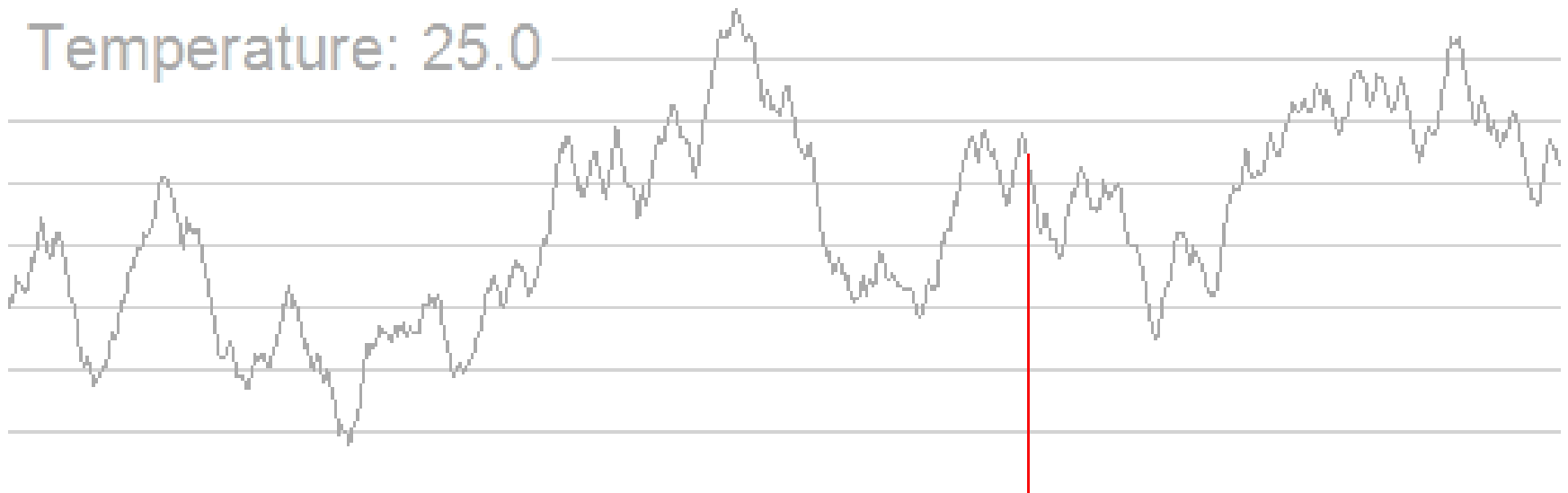


Image GIF: Wikipedia