



Inteligência Artificial Ciência da Computação

Prof. Aline Paes / alinepaes@ic.uff.br

Estratégias de Busca não Clássicas - RN
4.1; ER 3.b



Universidade Federal Fluminense



Estratégias clássicas de busca

- Busca sistemática

- um ou mais caminhos na memória
- alternativas que podem ser exploradas a partir de cada caminho
- Quando chega no objetivo, o caminho para ele constitui a solução do problema

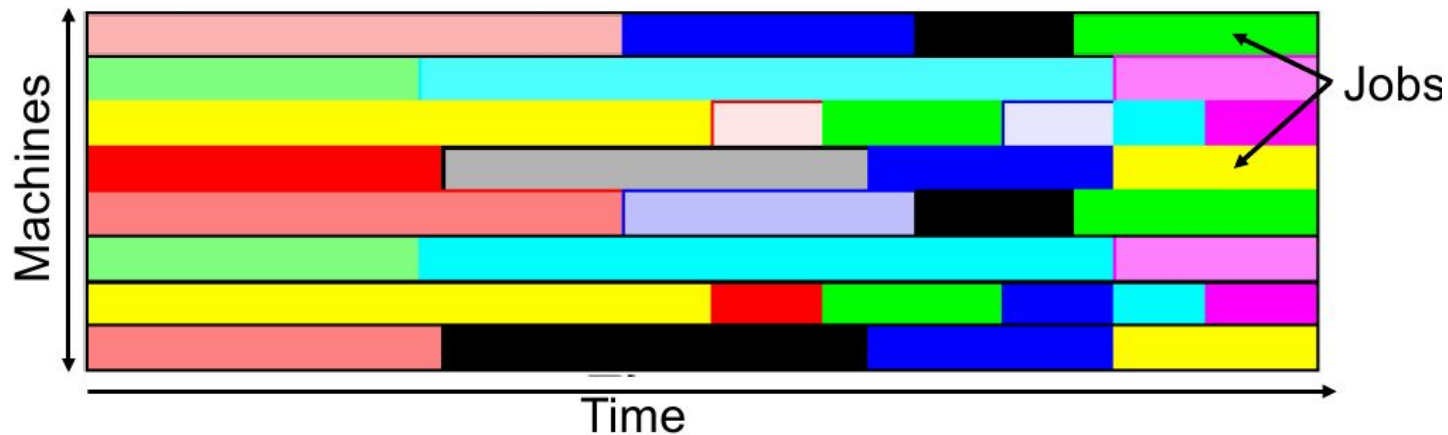
Estratégias clássicas de busca

- Busca sistemática

- um ou mais caminhos na memória
- alternativas que podem ser exploradas a partir de cada caminho
- 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 (ótima)
 - o caminho para a meta pode ser irrelevante

Exemplo - escalonamento

- Objetivo: minimizar o tempo para completar n jobs em m máquinas



Busca Local

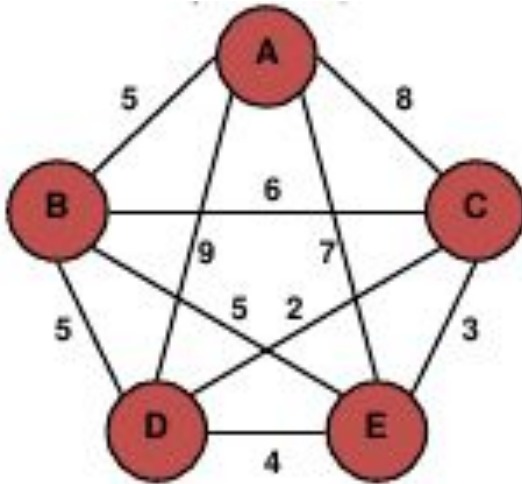
- Adequadas para **problemas de otimização**
- Enumerar os estados é **intratável**
 - algoritmos anteriores são muito caros
- **Não** existe estratégia para encontrar a **solução ótima** de forma **eficiente**

Algoritmos de busca local

- Usam apenas um nó corrente
 - representa um **estado**, com uma função de **pontuação computável (eval)**
 - a meta é encontrar o estado com a pontuação **mais alta** (baixa) ou ao menos um estado com uma pontuação **razoavelmente alta** (baixa)

Exemplo - TSP

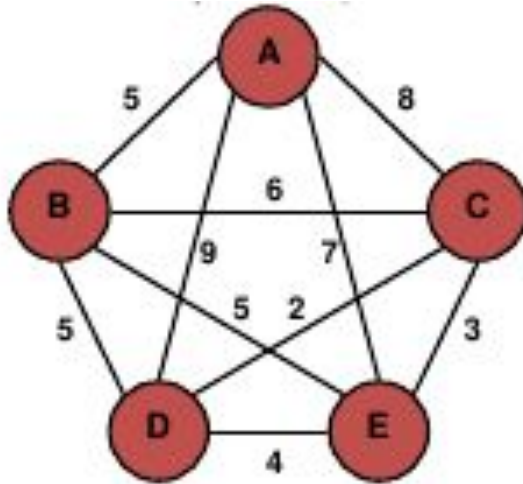
- Existem $(n-1)!/2$ possíveis soluções



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

Exemplo - TSP

- Estados: permutações de cidades
- Eval = distância 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 - SAT

- Espaço de estados: 2^N
- Estado: vetor de associação para N variáveis booleanas
- Eval: #|cláusulas satisfeitas|

$$A \vee \neg B \vee C$$

$$\neg A \vee C \vee D$$

$$B \vee D \vee \neg E$$

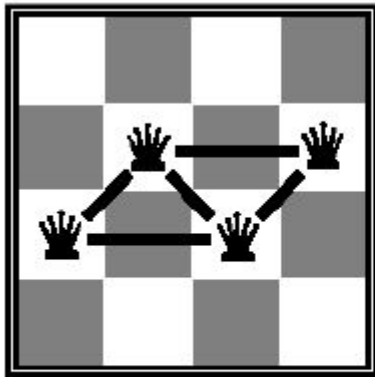
$$\neg C \vee \neg D \vee \neg E$$

$$\neg A \vee \neg C \vee E$$

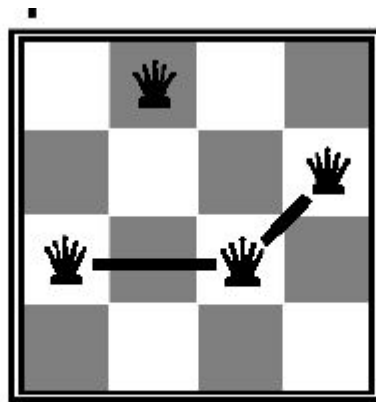
	A	B	C	D	E	Eval
X_1	true	true	false	true	false	5
X_2	true	true	true	true	true	4

Exemplo - N rainhas

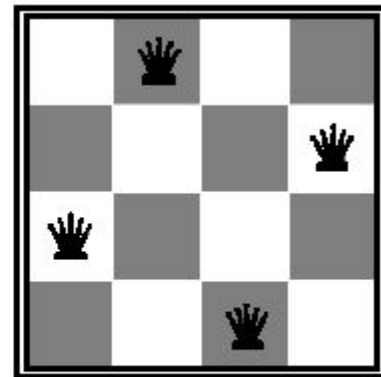
- Espaço de estados: N^N
- Estado: posição de N rainhas em N colunas
- Eval: pares de rainhas que estão se atacando



$$Eval(X) = 5$$



$$Eval(X) = 2$$



$$Eval(X) = 0$$

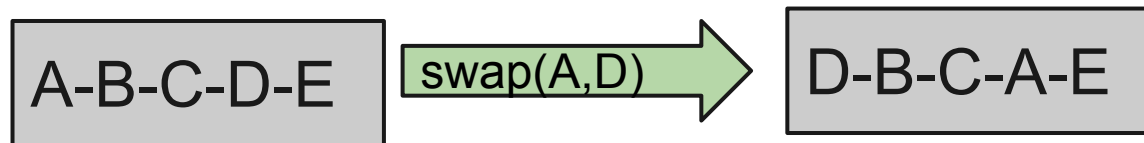
Algoritmos de busca local

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

Vizinhança - exemplos

- TSP

- trocar a posição de duas cidades em um tour



- N rainhas

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

- SAT

- inverter a associação de uma variável

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

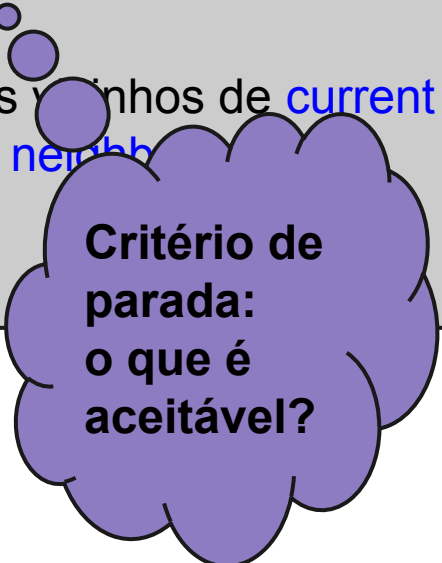
if **current.value** possui um valor **aceitável**

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 Hill-climbing

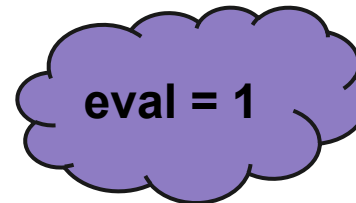
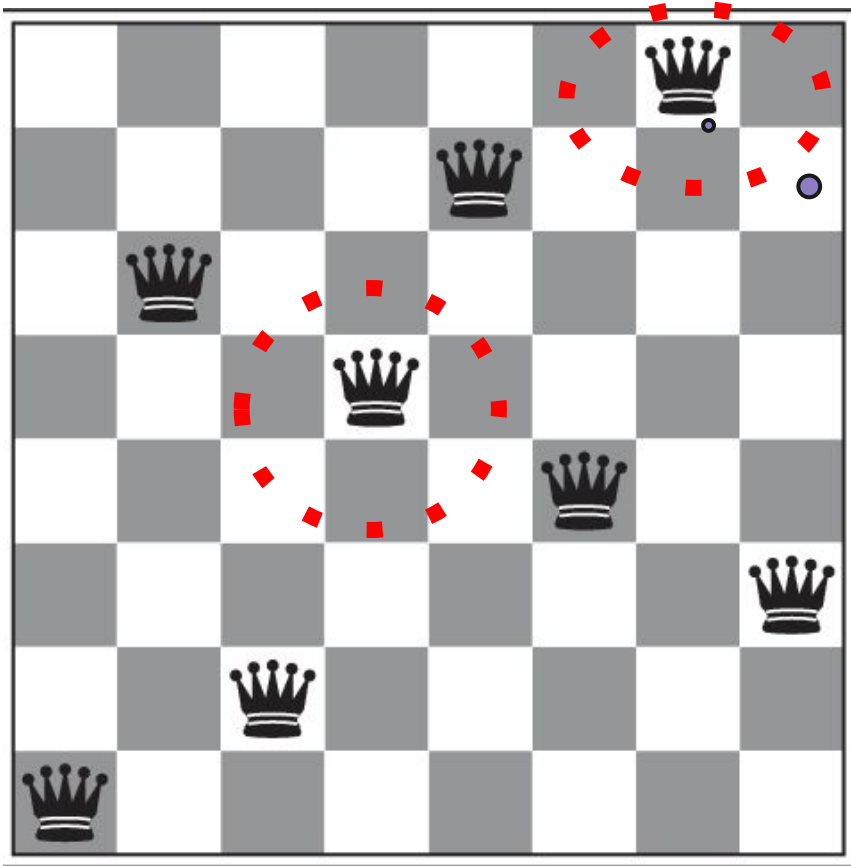
- eval: pares de rainhas se atacando

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

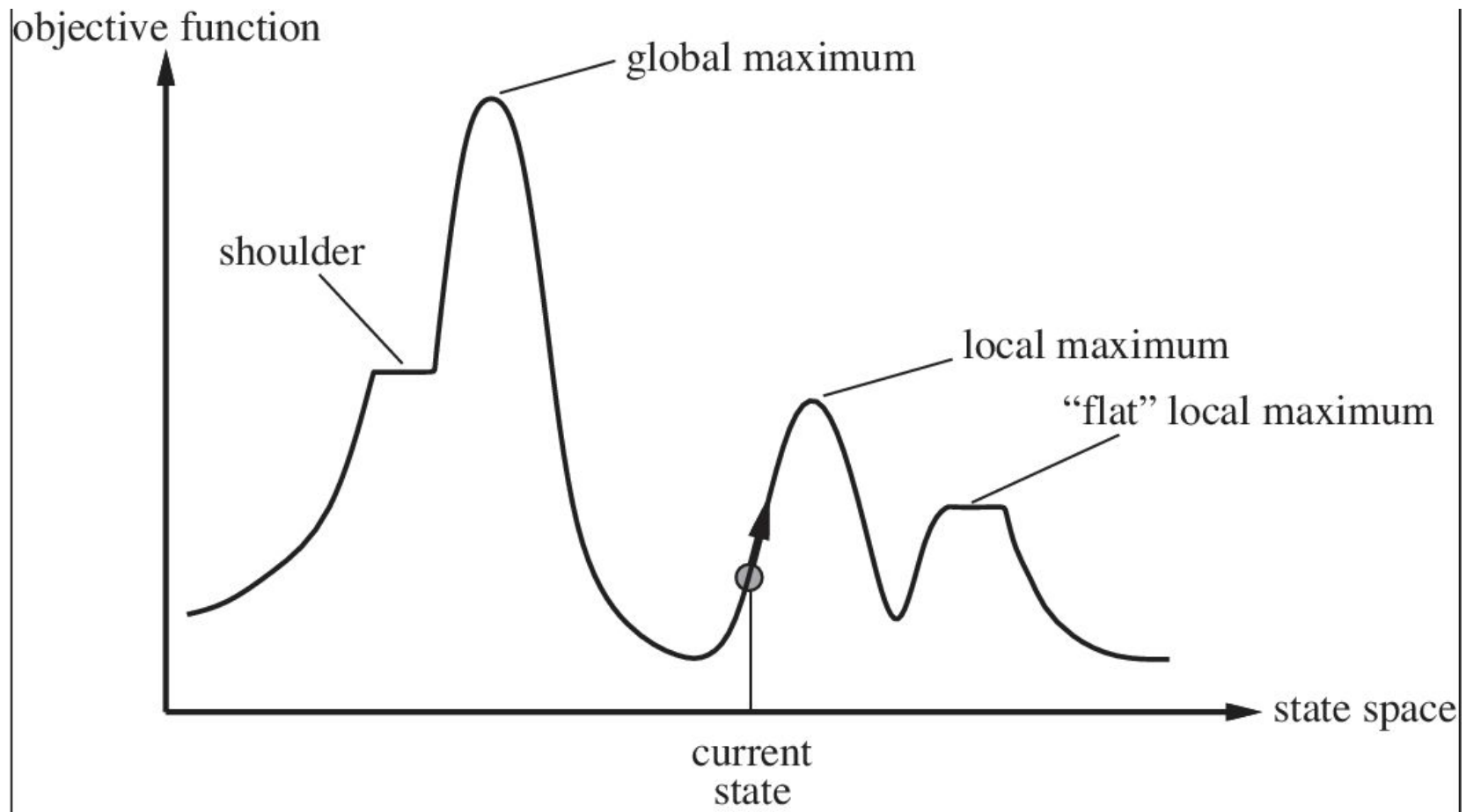
● ● eval= 17

Busca Hill-climbing básica é gulosa

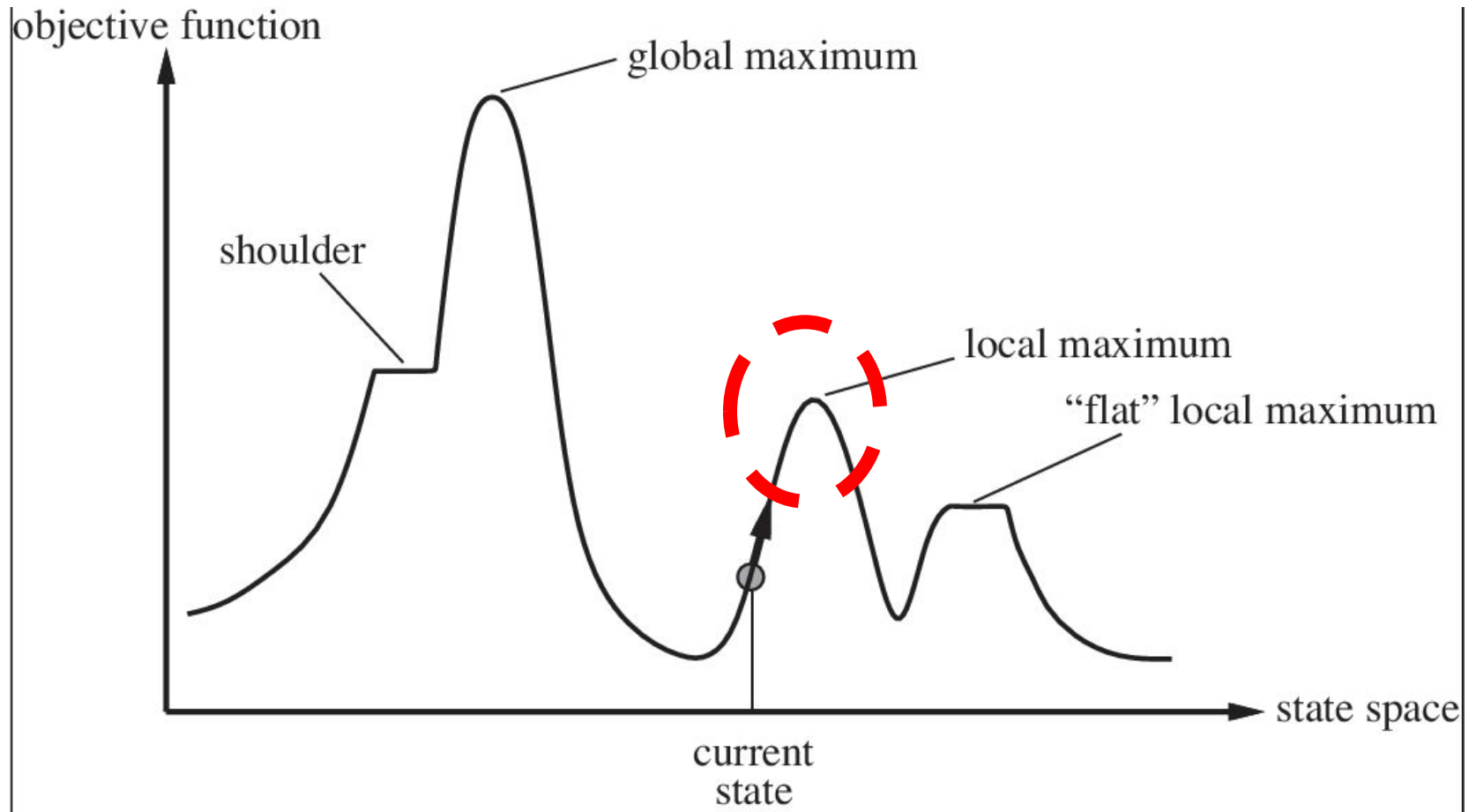
- Depois de 5 passos...



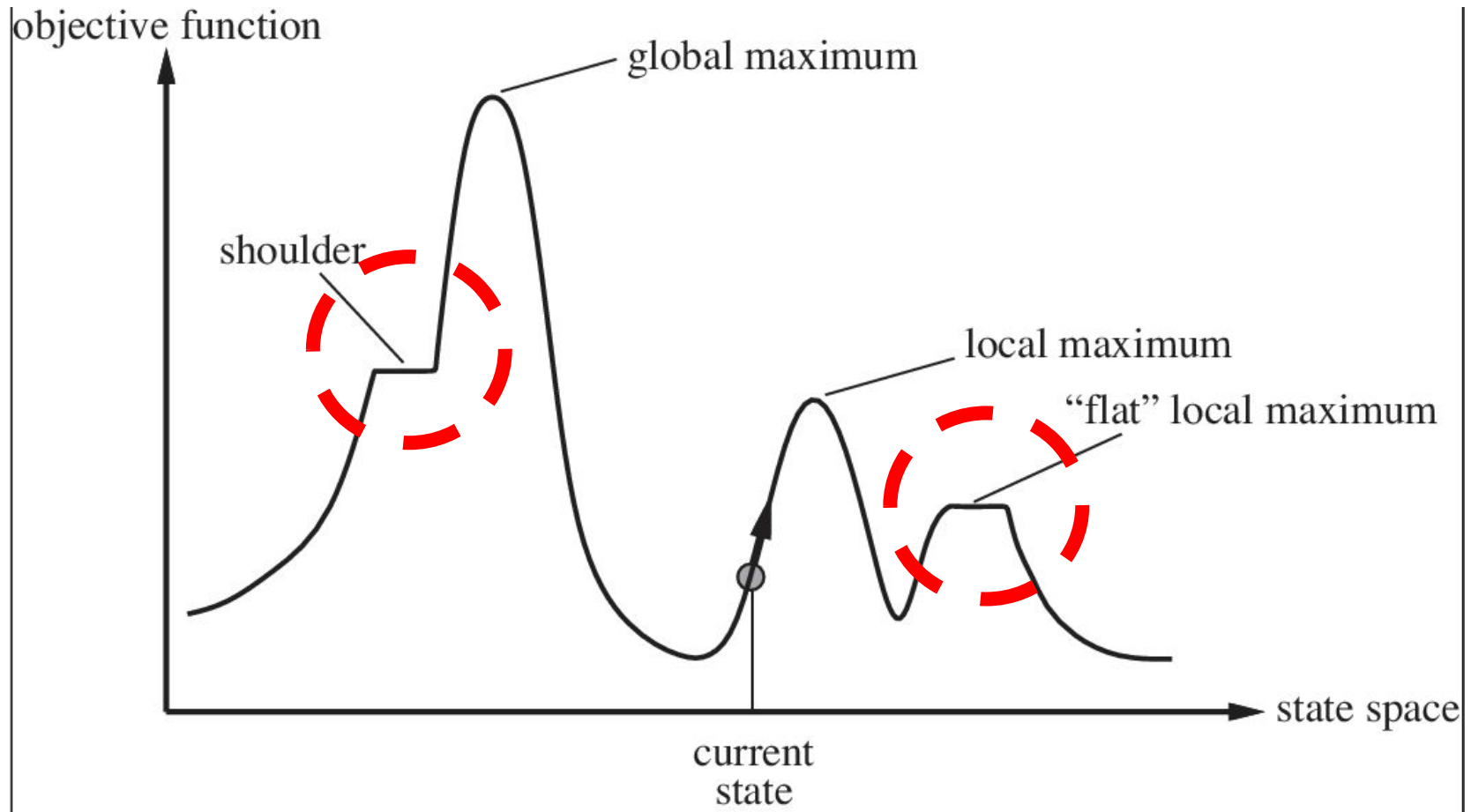
Landscape do espaço de estados



Busca Hill climbing - presa em máximos locais



Busca Hill climbing - plateaux



Hill climbing: variações (e lições de vida)

1

Dê um passo para
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 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 valor de pontuação

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

return `current.state`

`current` = `neighbor`

pode entrar
em loop:
deve-se impor
um limite

Hill climbing: variações

2

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

2

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

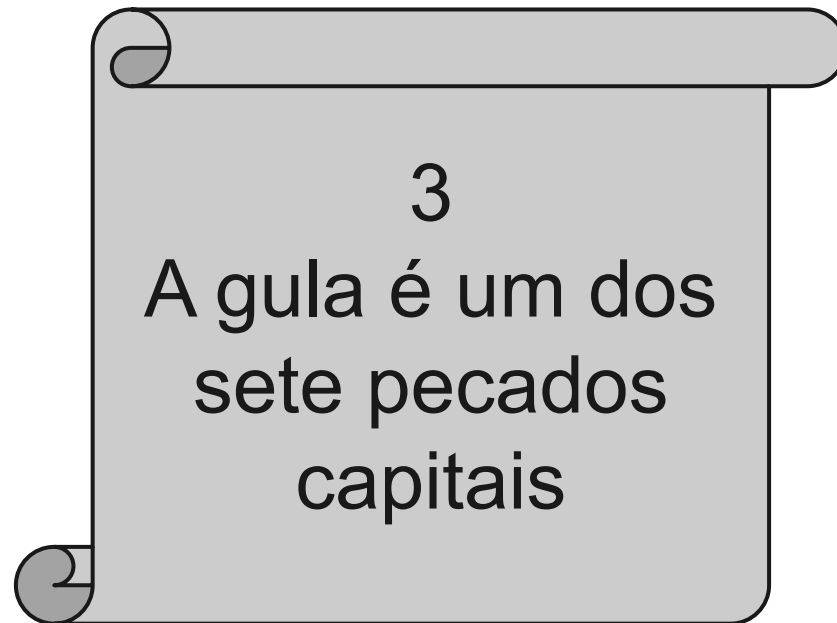
Random-restart hill climbing

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

3

A gula é um dos
sete pecados
capitais



**KEEP
CALM
DON'T
BE
GREEDY**

Hill climbing estocástico

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

Hill climbing: variações

4

*Carpe diem
quam minimum
credula postero*

Hill climbing com primeira escolha

- Vizinhança é muito populosa
 - gasta **tempo** para enumerar
- Escolhe **primeiro** vizinho **que melhora** a pontuação
- Pode ser implementado em conjunto com Hill climbing estocástico

Hill climbing: variações

5

Pode ser que
para melhorar
seja 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*

WalkSAT

- Algoritmo para resolver SAT
 - enquanto não encontrou uma solução ou não atingiu um número máximo de passos
 - escolha uma cláusula não satisfeita
 - selecione e "flip" uma variável da cláusula:
 - com probabilidade p , pegue uma variável aleatória
 - com probabilidade $1-p$, pegue a variável que maximize o número de cláusulas satisfeitas

Busca Hill climbing

- Hill climbing clássico
 - nunca "desce a encosta"
 - nunca permite uma piora na pontuação do novo estado
 - incompleto
 - preso em locais
- Hill climbing puramente com busca aleatória
 - move para um sucessor escolhido aleatoriamente e uniformemente
 - completo, mas ineficiente

Busca Hill climbing

- Hill climbing clássico

- nunca "desce a encosta"
 - nunca permite uma piora na avaliação do novo estado
- incompleto
- preso em locais

- Hill climbing

- move para um sucessor escolhido aleatoriamente e uniformemente
- completo, mas ineficiente

**E se
combinarmos
as
estratégias?**

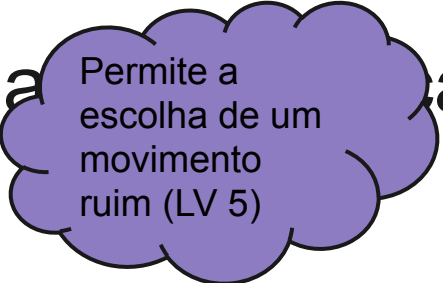
Busca aleatória

Simulated Annealing, ideia geral

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



Permite a
escolha de um
movimento
ruim (LV 5)

Simulated Annealing

- SA básico pode escapar de locais, mas
 - a escolha de um movimento ruim não leva em consideração o **quão ruim** ele é
 - a chance de escolher um movimento ruim é a mesma, independente do **momento** da busca

Simulated Annealing

- SA básico pode escapar de locais, mas
 - a escolha de um movimento ruim não leva em consideração o quanto ruim ele é
 - a chance de aceitar um movimento pior é a mesma, independentemente de quão ruim ele é

5'

Pode ser que para melhorar seja preciso dar um passo para trás, mas

1 - cuidado para onde você vai

2 - Não passe sua vida dando passos para trás

é a busca

Simulated Annealing

- SA básico pode escapar de locais, mas
 - a escolha de um movimento ruim não leva em consideração o **quão ruim** ele é
 - a chance de escolher um movimento ruim é a mesma, independente do **momento** da busca
- 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
 - $\Delta E = \text{eval}(\text{novoEstado}) - \text{eval}(\text{estadoCorrente})$
 - novoEstado é escolhido aleatoriamente
- 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 Boltzman
 - $p = e^{\text{DeltaE} / T}$
- $\text{DeltaE} \rightarrow -\infty$, $P \rightarrow 0$
 - 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

function **simulated-annealing**(problem,temp_function,Kmax) returns um estado de solução

current = make_node(problem.initialState)

k = 0

enquanto k < Kmax

 T = temp_function(k)

 if T = 0

 return *current*

next = um sucessor aleatório de *current*

DeltaE = *next.value* - *current.value*

 if *DeltaE* > 0

current = *next*

 else

current = *next* somente com probabilidade $e^{\text{DeltaE}/T}$

 k++

mapeia tempo
para
"temperatura"

pode ser
aleatório

Simulated annealing - pros e contras

- + Pode ser provado que ele chega em um **máximo global**, se T **diminui bem lentamente**
- + Rápido - apenas **um vizinho** gerado a cada iteração
- + **Escapa** de máximos **locais**

Simulated annealing - pros e contras

- + Pode ser provado que ele chega em um **máximo global**, se T **diminui bem lentamente**
- + Rápido - apenas **um vizinho** gerado a cada iteração
- + **Escapa** de máximos **locais**
- dependentes dos parâmetros **Kmax** e **função de mapeamento**
- **saída** pode ser completamente **diferente** em diferentes execuções

Exercício

Suponha que temos uma configuração do problema das N rainhas em um tabuleiro 5X5. Suponha que todas as 5 rainhas estão inicialmente na primeira linha. Assuma que a função de avaliação é o número de pares de damas que estão na mesma linha, coluna ou diagonal (inicialmente a função de avaliação é 10).

Execute o algoritmo Hill climbing básico e cada uma de suas variações.

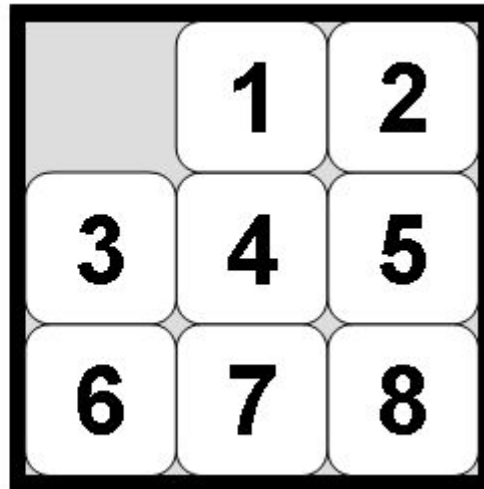
Exercício

Execute o algoritmo simulated annealing para a configuração de um tabuleiro com N rainhas, assumindo uma temperatura de 20k. O tabuleiro está inicialmente com a configuração abaixo. Ignore as setas e assuma que a função de avaliação é a mesma do slide anterior.

				D	
	D				
					D
D					
		D			
			D		

Exercício

Usando as buscas vistas nessa aula, resolva o problema do quebra cabeça de 8, considerando que a pontuação é dada pela heurística da distância de Manhattan. O objetivo é chegar no estado abaixo:



Exercícios

Praticar a busca Hill climbing em

<http://files.bookboon.com/ai/Hill-Climbing-with-Wall-Following.html>