

# Inteligência Artificial – ACH2016

## Aula 03 – Buscas Informadas e Heurísticas de Busca

Norton Trevisan Roman  
(norton@usp.br)

26 de fevereiro de 2019

## Heurística

## Heurística

- Conjunto de regras que aumentam a probabilidade de atingirmos o objetivo mais rapidamente

## Heurística

- Conjunto de regras que aumentam a probabilidade de atingirmos o objetivo mais rapidamente
- São a forma mais comum de trazer conhecimento adicional ao problema

## Heurística

- Conjunto de regras que aumentam a probabilidade de atingirmos o objetivo mais rapidamente
  - São a forma mais comum de trazer conhecimento adicional ao problema

## Busca com Heurística

- Buscas nessa categoria fazem uso de regras práticas

## Heurística

- Conjunto de regras que aumentam a probabilidade de atingirmos o objetivo mais rapidamente
  - São a forma mais comum de trazer conhecimento adicional ao problema

## Busca com Heurística

- Buscas nessa categoria fazem uso de regras práticas
  - Algo que pode ser útil em alguns casos, mas não sempre

## Heurística

- Conjunto de regras que aumentam a probabilidade de atingirmos o objetivo mais rapidamente
  - São a forma mais comum de trazer conhecimento adicional ao problema

## Busca com Heurística

- Buscas nessa categoria fazem uso de regras práticas
  - Algo que pode ser útil em alguns casos, mas não sempre
- Não necessariamente acham a melhor solução, mas podem encontrar a solução de forma mais eficiente

## Estratégia Melhor Primeiro



## Estratégia Melhor Primeiro

- A ideia é seleccionar um nó para visita com base em uma **função de avaliação**  $f(n)$

## Estratégia Melhor Primeiro

- A ideia é seleccionar um nó para visita com base em uma **função de avaliação**  $f(n)$ 
  - Construída como uma estimativa de custo

## Estratégia Melhor Primeiro

- A ideia é seleccionar um nó para visita com base em uma **função de avaliação**  $f(n)$ 
  - Construída como uma estimativa de custo
  - Assim, o nó com a menor avaliação é expandido primeiro

## Estratégia Melhor Primeiro

- A ideia é seleccionar um nó para visita com base em uma **função de avaliação**  $f(n)$ 
  - Construída como uma estimativa de custo
  - Assim, o nó com a menor avaliação é expandido primeiro
- A escolha de  $f$  determina a estratégia de busca

## Estratégia Melhor Primeiro

- A ideia é selecionar um nó para visita com base em uma **função de avaliação**  $f(n)$ 
  - Construída como uma estimativa de custo
  - Assim, o nó com a menor avaliação é expandido primeiro
- A escolha de  $f$  determina a estratégia de busca
  - A maioria dos algoritmos assim incluem uma **função heurística**  $h(n)$  como componente de  $f$

## Estratégia Melhor Primeiro

- A ideia é selecionar um nó para visita com base em uma **função de avaliação**  $f(n)$ 
  - Construída como uma estimativa de custo
  - Assim, o nó com a menor avaliação é expandido primeiro
- A escolha de  $f$  determina a estratégia de busca
  - A maioria dos algoritmos assim incluem uma **função heurística**  $h(n)$  como componente de  $f$
  - Nesse caso,  $h(n)$  é uma estimativa do custo do caminho mais barato do nó  $n$  até o objetivo

## Melhor Primeiro (Gulosa)

## Melhor Primeiro (Gulosa)

- Tenta expandir o nó que parece estar mais próximo do objetivo



## Melhor Primeiro (Gulosa)

- Tenta expandir o nó que parece estar mais próximo do objetivo
- Avalia o nó usando tão somente a função heurística

## Melhor Primeiro (Gulosa)

- Tenta expandir o nó que parece estar mais próximo do objetivo
- Avalia o nó usando tão somente a função heurística
  - Ou seja,  $f(n) = h(n)$

## Melhor Primeiro (Gulosa)

- Tenta expandir o nó que parece estar mais próximo do objetivo
- Avalia o nó usando tão somente a função heurística
  - Ou seja,  $f(n) = h(n)$
- Algoritmo idêntico ao da Busca de Custo Uniforme

## Melhor Primeiro (Gulosa)

- Tenta expandir o nó que parece estar mais próximo do objetivo
- Avalia o nó usando tão somente a função heurística
  - Ou seja,  $f(n) = h(n)$
- Algoritmo idêntico ao da Busca de Custo Uniforme
  - Exceto que, dessa vez, usamos  $h(n)$  para escolher o caminho com menor custo

## Melhor Primeiro (Gulosa)

- Tenta expandir o nó que parece estar mais próximo do objetivo
- Avalia o nó usando tão somente a função heurística
  - Ou seja,  $f(n) = h(n)$
- Algoritmo idêntico ao da Busca de Custo Uniforme
  - Exceto que, dessa vez, usamos  $h(n)$  para escolher o caminho com menor custo
  - Ou seja, em vez de expandir (visitar) o nó  $n$  com o menor caminho até então, expande aquele com menor distância estimada ao objetivo

## Melhor Primeiro (Gulosa) – Problemas

## Melhor Primeiro (Gulosa) – Problemas

- Dependendo da heurística, nem sempre dá a melhor solução

## Melhor Primeiro (Gulosa) – Problemas

- Dependendo da heurística, nem sempre dá a melhor solução
  - Dependendo da heurística, nem sempre dá uma solução



## Melhor Primeiro (Gulosa) – Problemas

- Dependendo da heurística, nem sempre dá a melhor solução
  - Dependendo da heurística, nem sempre dá uma solução

A\*

## Melhor Primeiro (Gulosa) – Problemas

- Dependendo da heurística, nem sempre dá a melhor solução
  - Dependendo da heurística, nem sempre dá uma solução

## A\*

- Forma mais conhecida de busca do tipo Melhor Primeiro

## Melhor Primeiro (Gulosa) – Problemas

- Dependendo da heurística, nem sempre dá a melhor solução
  - Dependendo da heurística, nem sempre dá uma solução

## A\*

- Forma mais conhecida de busca do tipo Melhor Primeiro
- Avalia o nó combinando o custo do caminho até ele ( $g(n)$ ) e o custo estimado dele ( $h(n)$ ) até o objetivo

## Melhor Primeiro (Gulosa) – Problemas

- Dependendo da heurística, nem sempre dá a melhor solução
  - Dependendo da heurística, nem sempre dá uma solução

## A\*

- Forma mais conhecida de busca do tipo Melhor Primeiro
- Avalia o nó combinando o custo do caminho até ele ( $g(n)$ ) e o custo estimado dele ( $h(n)$ ) até o objetivo
  - Ou seja,  $f(n) = g(n) + h(n)$

## A\*

- $f(n)$  é então o custo estimado da melhor solução passando por  $n$

## A\*

- $f(n)$  é então o custo estimado da melhor solução passando por  $n$

### Custo Uniforme

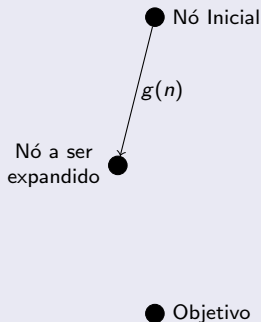


# Buscas Informadas

## A\*

- $f(n)$  é então o custo estimado da melhor solução passando por  $n$

**Custo Uniforme**



**Melhor Primeiro**



# Buscas Informadas

## A\*

- $f(n)$  é então o custo estimado da melhor solução passando por  $n$

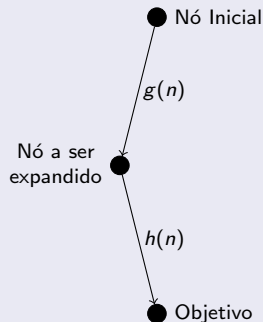
**Custo Uniforme**



**Melhor Primeiro**



**A\***





## A\*

- Se  $h(n)$  for uma heurística admissível e consistente, A\* garantidamente encontra o caminho mais curto ao objetivo

## A\*

- Se  $h(n)$  for uma heurística admissível e consistente, A\* garantidamente encontra o caminho mais curto ao objetivo
- Heurística admissível:

## A\*

- Se  $h(n)$  for uma heurística admissível e consistente, A\* garantidamente encontra o caminho mais curto ao objetivo
- Heurística admissível:
  - Nunca superestima a distância do nó ao objetivo (a distância real é garantidamente maior ou igual à estimada)

## A\*

- Se  $h(n)$  for uma heurística admissível e consistente, A\* garantidamente encontra o caminho mais curto ao objetivo
- Heurística admissível:
  - Nunca superestima a distância do nó ao objetivo (a distância real é garantidamente maior ou igual à estimada)
  - Naturalmente otimista: sempre assume o custo para resolver o problema abaixo do real (ou, no máximo, igual)

## A\*

- Se  $h(n)$  for uma heurística admissível e consistente, A\* garantidamente encontra o caminho mais curto ao objetivo
- Heurística admissível:
  - Nunca superestima a distância do nó ao objetivo (a distância real é garantidamente maior ou igual à estimada)
  - Naturalmente otimista: sempre assume o custo para resolver o problema abaixo do real (ou, no máximo, igual)
  - Ex: Distância euclidiana (linha reta) como forma de distância entre duas cidades  $\rightarrow$  sempre será  $\leq$  que a real

## A\*

- Heurística admissível:
  - Uma vez que  $g(n)$  é o custo real para se chegar a  $n$ , se  $h(n)$  nunca superestimar o custo, então  $f(n) = g(n) + h(n)$  nunca superestimar o custo de uma solução que passe por  $n$

## A\*

- Heurística admissível:
  - Uma vez que  $g(n)$  é o custo real para se chegar a  $n$ , se  $h(n)$  nunca superestimar o custo, então  $f(n) = g(n) + h(n)$  nunca superestimar o custo de uma solução que passe por  $n$
- Heurística consistente:

## A\*

- Heurística admissível:
  - Uma vez que  $g(n)$  é o custo real para se chegar a  $n$ , se  $h(n)$  nunca superestimar o custo, então  $f(n) = g(n) + h(n)$  nunca superestimar o custo de uma solução que passe por  $n$
- Heurística consistente:
  - Se, para cada nó  $n$  e sucessor  $n'$  de  $n$ , o custo estimado de atingir o objetivo a partir de  $n$  não for maior que o custo de ir de  $n$  a  $n'$  mais o custo estimado de se atingir o objetivo a partir de  $n'$



## A\*

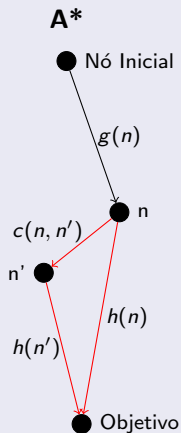
- Heurística admissível:
  - Uma vez que  $g(n)$  é o custo real para se chegar a  $n$ , se  $h(n)$  nunca superestimar o custo, então  $f(n) = g(n) + h(n)$  nunca superestimar o custo de uma solução que passe por  $n$
- Heurística consistente:
  - Se, para cada nó  $n$  e sucessor  $n'$  de  $n$ , o custo estimado de atingir o objetivo a partir de  $n$  não for maior que o custo de ir de  $n$  a  $n'$  mais o custo estimado de se atingir o objetivo a partir de  $n'$
  - Ou seja,  $h(n) \leq c(n, n') + h(n')$

## A\*

- Heurística consistente:
  - Trata-se de uma forma de desigualdade triangular: cada lado do triângulo não pode ser maior que a soma dos outros 2

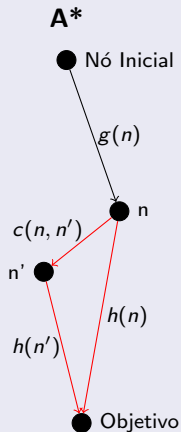
## A\*

- Heurística consistente:
  - Trata-se de uma forma de desigualdade triangular: cada lado do triângulo não pode ser maior que a soma dos outros 2



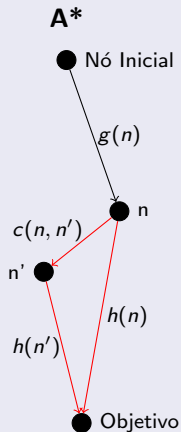
## A\*

- Heurística consistente:
  - Trata-se de uma forma de desigualdade triangular: cada lado do triângulo não pode ser maior que a soma dos outros 2
  - Toda heurística consistente é admissível



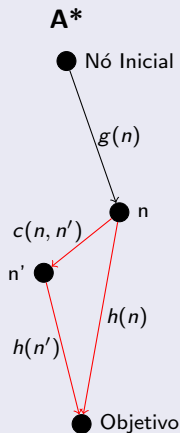
## A\*

- Heurística consistente:
  - Trata-se de uma forma de desigualdade triangular: cada lado do triângulo não pode ser maior que a soma dos outros 2
  - Toda heurística consistente é admissível
- A\* será ótimo quando:



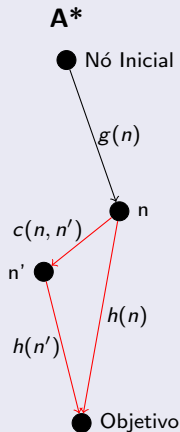
## A\*

- Heurística consistente:
  - Trata-se de uma forma de desigualdade triangular: cada lado do triângulo não pode ser maior que a soma dos outros 2
  - Toda heurística consistente é admissível
- A\* será ótimo quando:
  - Em sua versão para árvores,  $h(n)$  for admissível



## A\*

- Heurística consistente:
  - Trata-se de uma forma de desigualdade triangular: cada lado do triângulo não pode ser maior que a soma dos outros 2
  - Toda heurística consistente é admissível
- A\* será ótimo quando:
  - Em sua versão para árvores,  $h(n)$  for admissível
  - Em sua versão para grafos,  $h(n)$  for consistente



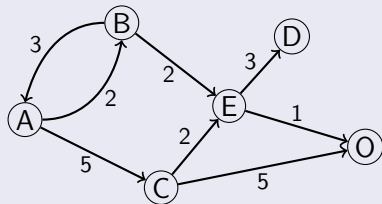
## A\* – Exemplo

- Considere o seguinte grafo



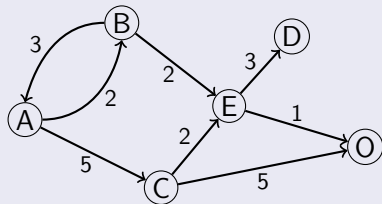
## A\* – Exemplo

- Considere o seguinte grafo



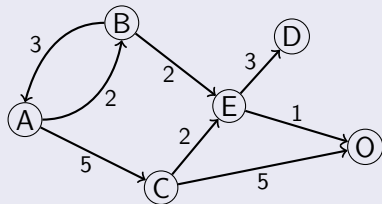
## A\* – Exemplo

- Considere o seguinte grafo
- Imagine que ele representa estradas entre cidades



## A\* – Exemplo

- Considere o seguinte grafo
- Imagine que ele representa estradas entre cidades
- $h(n)$  será então a distância, em linha reta, entre o nó  $n$  e  $O$ , conforme a tabela:



$n$	$h(n)$	$n$	$h(n)$
A	4	D	0.5
B	2.5	E	0.8
C	2		

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com  
    menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

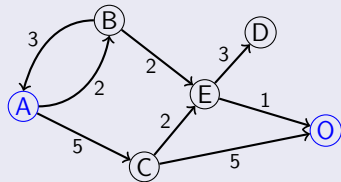
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  **Nó inicial**;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

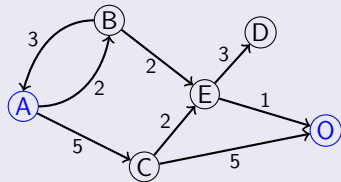
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[A]	0	4	4

C:

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto** *Q não estiver vazia* **faça**

$C \leftarrow$  Retire de Q o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

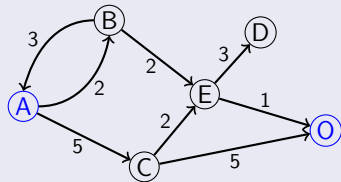
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne C

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a Q

**retorna** *falha*



	Caminho	g	h	f
Q:	[A]	0	4	4

C:

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com  
    menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

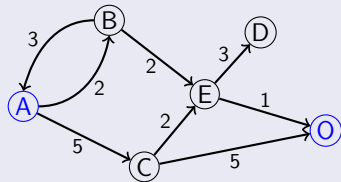
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:				

C: [A] (0,4,4)



# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

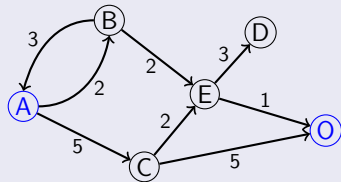
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:				

C: [A] (0,4,4)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com  
    menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

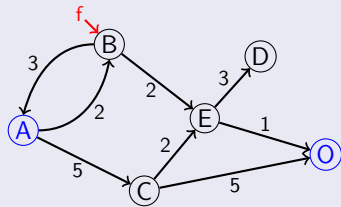
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** *filho  $f$  de  $\text{cabeça}(C)$*  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:				

C: [A] (0,4,4)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

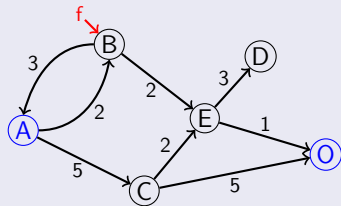
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[B,A]	2	2.5	4.5

C: [A] (0,4,4)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

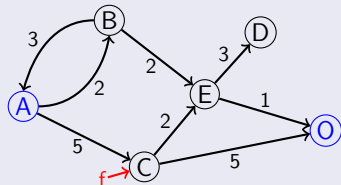
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** *filho  $f$  de  $\text{cabeça}(C)$*  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[B,A]	2	2.5	4.5

C: [A] (0,4,4)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com  
    menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

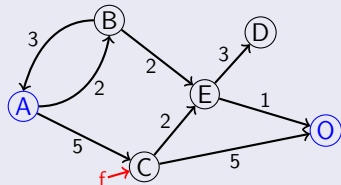
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[B,A]	2	2.5	4.5
	[C,A]	5	2	7

C: [A] (0,4,4)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto** *Q não estiver vazia* **faça**

$C \leftarrow$  Retire de Q o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

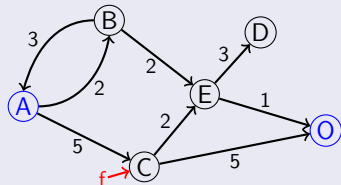
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne C

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a Q

**retorna** *falha*



	Caminho	g	h	f
Q:	[B,A]	2	2.5	4.5
	[C,A]	5	2	7

C: [A] (0,4,4)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com  
    menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

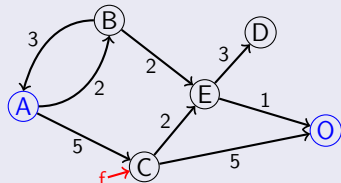
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7

C: [B,A] (2,2.5,4.5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

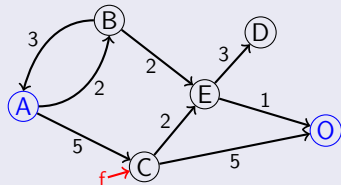
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7

C: [B,A] (2,2.5,4.5)



# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

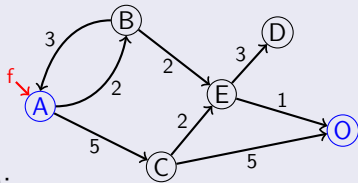
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7

C: [B,A] (2,2.5,4.5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

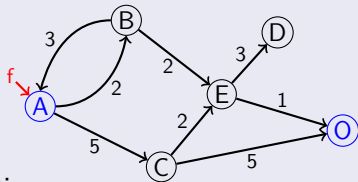
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9

C: [B,A] (2,2.5,4.5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

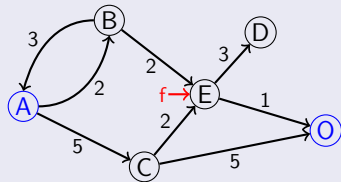
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9

C: [B,A] (2,2.5,4.5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

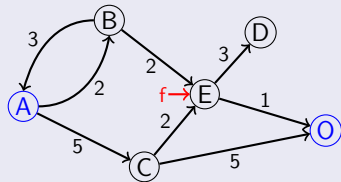
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[E,B,A]	4	0.8	4.8

C: [B,A] (2,2.5,4.5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

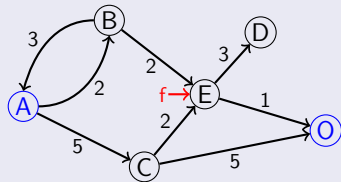
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[E,B,A]	4	0.8	4.8

C: [B,A] (2,2.5,4.5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com  
    menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

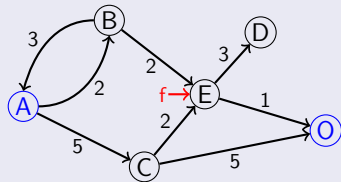
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9

C: [E,B,A] (4,0.8,4.8)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

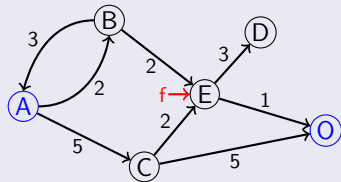
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9

C: [E,B,A] (4,0.8,4.8)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

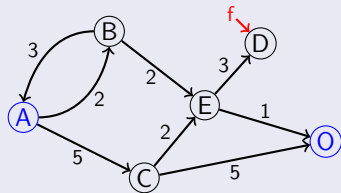
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada**  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9

C: [E,B,A] (4,0.8,4.8)



# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

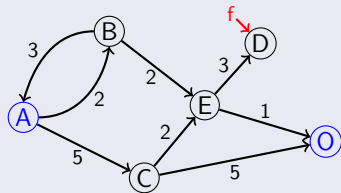
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[D,E,B,A]	7	0.5	7.5

C: [E,B,A] (4,0.8,4.8)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

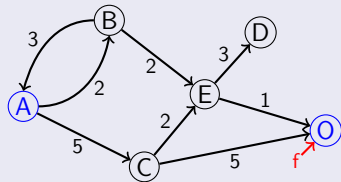
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[D,E,B,A]	7	0.5	7.5

C: [E,B,A] (4,0.8,4.8)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

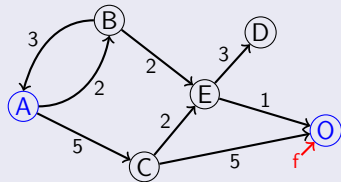
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ **Adicione**  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[D,E,B,A]	7	0.5	7.5
	[O,E,B,A]	5	0	5

C: [E,B,A] (4,0.8,4.8)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

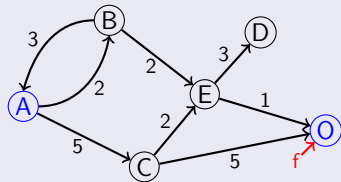
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[D,E,B,A]	7	0.5	7.5
	[O,E,B,A]	5	0	5

C: [E,B,A] (4,0.8,4.8)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com  
    menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

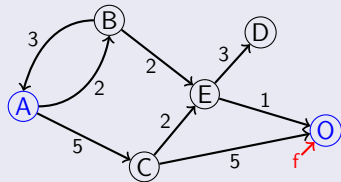
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[D,E,B,A]	7	0.5	7.5

C: [O,E,B,A] (5,0,5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

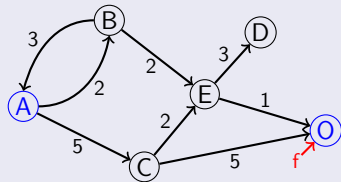
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ Retorne  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*



	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[D,E,B,A]	7	0.5	7.5

C: [O,E,B,A] (5,0,5)

# Buscas Informadas

## A\* – Algoritmo

$Q \leftarrow$  Nó inicial;

**enquanto**  $Q$  não estiver vazia **faça**

$C \leftarrow$  Retire de  $Q$  o caminho com menor  $f(C) = g(C) + h(\text{cabeça}(C))$ ;

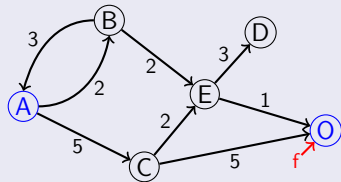
**se**  $\text{cabeça}(C) = \text{objetivo}$  **então**

        └ **Retorne**  $C$

**para cada** filho  $f$  de  $\text{cabeça}(C)$  **faça**

        └ Adicione  $[f, C]$  a  $Q$

**retorna** *falha*

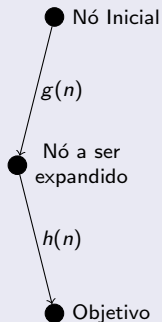


	Caminho	g	h	f
Q:	[C,A]	5	2	7
	[A,B,A]	5	4	9
	[D,E,B,A]	7	0.5	7.5

C: [O,E,B,A] (5,0,5)

## A\* – Observações

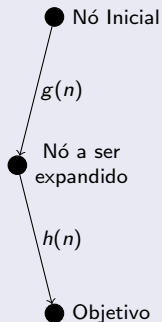
- No início da busca:





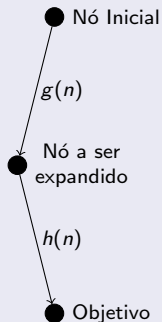
## A\* – Observações

- No início da busca:
  - $g(n)$  influi pouco



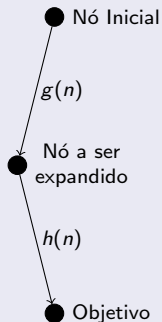
## A\* – Observações

- No início da busca:
  - $g(n)$  influi pouco
  - $h(n)$  domina



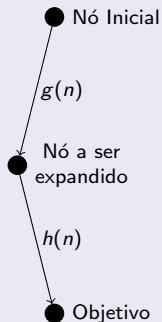
## A\* – Observações

- No início da busca:
  - $g(n)$  influi pouco
  - $h(n)$  domina
  - Escolhe caminho que parece ser o melhor



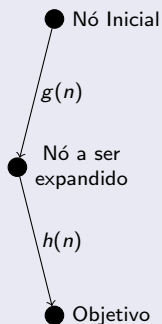
## A\* – Observações

- No início da busca:
  - $g(n)$  influi pouco
  - $h(n)$  domina
  - Escolhe caminho que parece ser o melhor
- Ao final da busca:



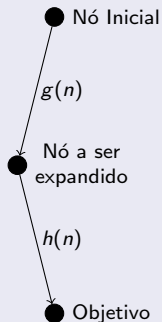
## A\* – Observações

- No início da busca:
  - $g(n)$  influi pouco
  - $h(n)$  domina
  - Escolhe caminho que parece ser o melhor
- Ao final da busca:
  - $h(n)$  influi pouco



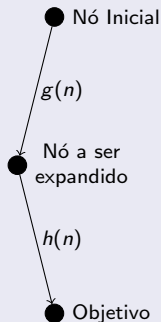
## A\* – Observações

- No início da busca:
  - $g(n)$  influi pouco
  - $h(n)$  domina
  - Escolhe caminho que parece ser o melhor
- Ao final da busca:
  - $h(n)$  influi pouco
  - $g(n)$  domina



## A\* – Observações

- No início da busca:
  - $g(n)$  influi pouco
  - $h(n)$  domina
  - Escolhe caminho que parece ser o melhor
- Ao final da busca:
  - $h(n)$  influi pouco
  - $g(n)$  domina
  - Acaba ficando com o menor caminho



## A\* – Vantagem

- A\* ignora caminhos que, em linha reta, se afastam muito do objetivo



## A\* – Vantagem

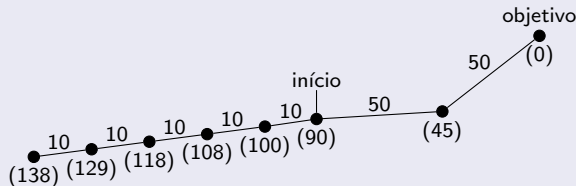
- A\* ignora caminhos que, em linha reta, se afastam muito do objetivo
- Busca de custo uniforme pode demorar a perceber que está no caminho errado

## A\* – Vantagem

- A\* ignora caminhos que, em linha reta, se afastam muito do objetivo
  - Busca de custo uniforme pode demorar a perceber que está no caminho errado
  - A\* decide (bem) antes.

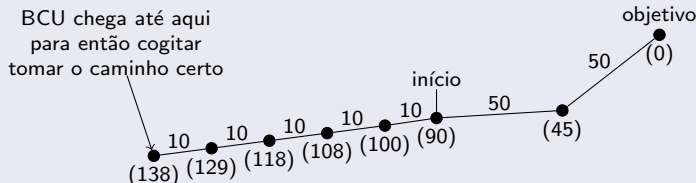
## A\* – Vantagem

- A\* ignora caminhos que, em linha reta, se afastam muito do objetivo
- Busca de custo uniforme pode demorar a perceber que está no caminho errado
- A\* decide (bem) antes. Ex:



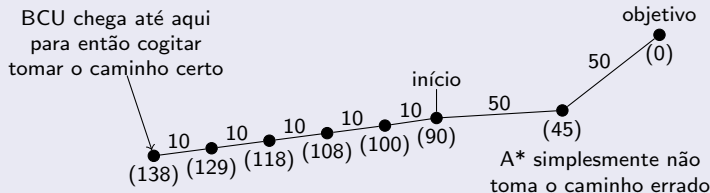
## A\* – Vantagem

- A\* ignora caminhos que, em linha reta, se afastam muito do objetivo
- Busca de custo uniforme pode demorar a perceber que está no caminho errado
- A\* decide (bem) antes. Ex:



## A\* – Vantagem

- A\* ignora caminhos que, em linha reta, se afastam muito do objetivo
- Busca de custo uniforme pode demorar a perceber que está no caminho errado
- A\* decide (bem) antes. Ex:



# Busca Local

Até agora vimos problemas que eram:

Até agora vimos problemas que eram:

- Observáveis: sabemos em que estado estamos

## Até agora vimos problemas que eram:

- Observáveis: sabemos em que estado estamos
  - Se conhecermos todos os estados então o ambiente é **totalmente observável**



## Até agora vimos problemas que eram:

- Observáveis: sabemos em que estado estamos
  - Se conhecermos todos os estados então o ambiente é **totalmente observável**
- Determinísticos: sabemos em que estado estaremos se executarmos determinada ação

## Até agora vimos problemas que eram:

- Observáveis: sabemos em que estado estamos
  - Se conhecermos todos os estados então o ambiente é **totalmente observável**
- Determinísticos: sabemos em que estado estaremos se executarmos determinada ação
- De ambientes conhecidos: conhecemos todas as regras de ação do ambiente

## Até agora vimos problemas que eram:

- Observáveis: sabemos em que estado estamos
  - Se conhecermos todos os estados então o ambiente é **totalmente observável**
- Determinísticos: sabemos em que estado estaremos se executarmos determinada ação
- De ambientes conhecidos: conhecemos todas as regras de ação do ambiente
  - Sabemos a saída (ou probabilidade de cada saída) para todas as ações dadas. Se o ambiente for desconhecido, teremos antes que aprender como ele funciona

## Problema

- E se não for esse o caso?

## Problema

- E se não for esse o caso?
  - E se não tivermos acesso a toda a informação necessária?

## Problema

- E se não for esse o caso?
  - E se não tivermos acesso a toda a informação necessária?
  - E se tivermos restrições de memória, nos proibindo de carregar o grafo todo?

# Busca Local

## Problema

- E se não for esse o caso?
  - E se não tivermos acesso a toda a informação necessária?
  - E se tivermos restrições de memória, nos proibindo de carregar o grafo todo?

## Solução

- Busca local

# Busca Local

## Problema

- E se não for esse o caso?
  - E se não tivermos acesso a toda a informação necessária?
  - E se tivermos restrições de memória, nos proibindo de carregar o grafo todo?

## Solução

- Busca local
  - Avalia e modifica apenas o estado atual, em vez de explorar caminhos a partir de um estado inicial



# Busca Local

## Problema

- E se não for esse o caso?
  - E se não tivermos acesso a toda a informação necessária?
  - E se tivermos restrições de memória, nos proibindo de carregar o grafo todo?

## Solução

- Busca local
  - Avalia e modifica apenas o estado atual, em vez de explorar caminhos a partir de um estado inicial
  - Naturalmente, não se aplica a casos em que queremos o caminho. Queremos apenas o estado final (objetivo)

# Busca Local

## Características

- Algoritmos de busca local operam usando um único estado (em vez de um grafo de estados)

# Busca Local

## Características

- Algoritmos de busca local operam usando um único estado (em vez de um grafo de estados)
  - Em geral movem-se apenas para vizinhos desse estado

# Busca Local

## Características

- Algoritmos de busca local operam usando um único estado (em vez de um grafo de estados)
  - Em geral movem-se apenas para vizinhos desse estado
- Tipicamente, os caminhos são ignorados

# Busca Local

## Características

- Algoritmos de busca local operam usando um único estado (em vez de um grafo de estados)
  - Em geral movem-se apenas para vizinhos desse estado
- Tipicamente, os caminhos são ignorados

## Vantagens

# Busca Local

## Características

- Algoritmos de busca local operam usando um único estado (em vez de um grafo de estados)
  - Em geral movem-se apenas para vizinhos desse estado
- Tipicamente, os caminhos são ignorados

## Vantagens

- Usam pouca memória

# Busca Local

## Características

- Algoritmos de busca local operam usando um único estado (em vez de um grafo de estados)
  - Em geral movem-se apenas para vizinhos desse estado
- Tipicamente, os caminhos são ignorados

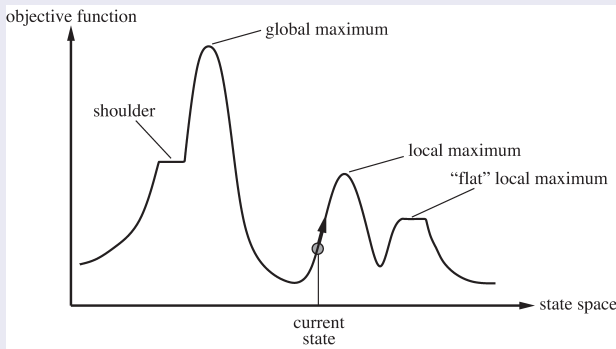
## Vantagens

- Usam pouca memória
- Frequentemente encontram soluções razoáveis em grandes (ou mesmo infinitos) espaços de busca

# Busca Local

## Espaços de busca

- Podemos definir o espaço de busca em termos de seus estados (eixo x) e sua função de avaliação (y):



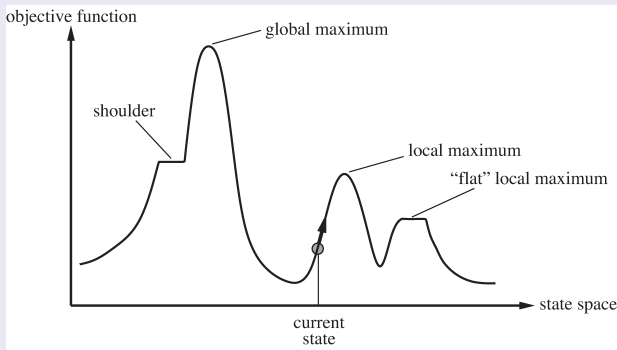
Fonte: AIMA. Russell & Norvig.



# Busca Local

## Espaços de busca

- Se essa função corresponder ao custo, queremos encontrar o vale mais profundo – **mínimo global**

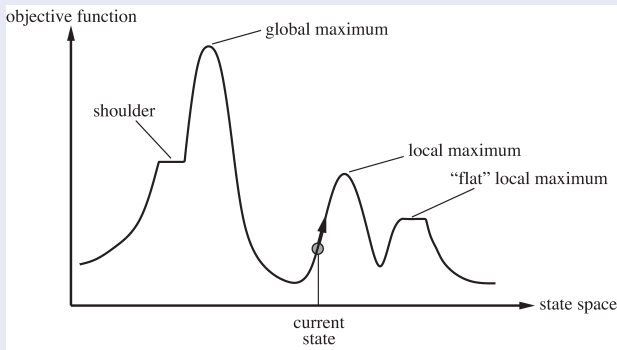


Fonte: AIMA. Russell & Norvig.

# Busca Local

## Espaços de busca

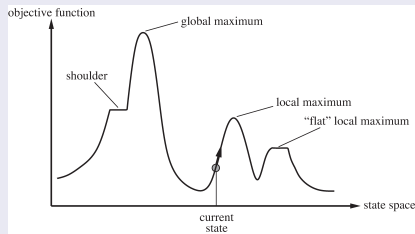
- Se corresponder ao objetivo, queremos encontrar o pico mais alto – **máximo global**



Fonte: AIMA. Russell & Norvig.

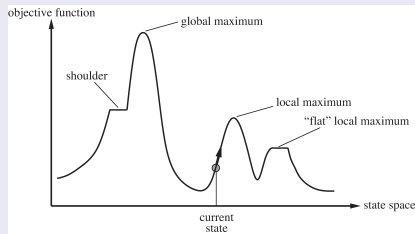
## Definições

- Um algoritmo de busca local será **completo** se sempre encontrar um objetivo, caso exista



## Definições

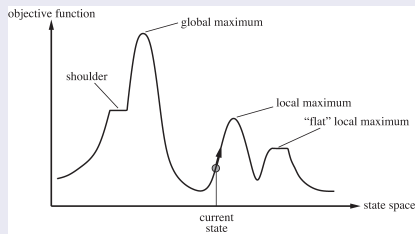
- Um algoritmo de busca local será **completo** se sempre encontrar um objetivo, caso exista
- Um algoritmo de busca local será **ótimo** se sempre encontrar um máximo (ou mínimo) global



## Hill Climbing

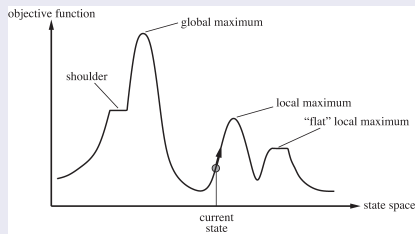
## Hill Climbing (Subida da Encosta)

- Escolhe, a cada passo, o nó que parece levar mais próximo do objetivo



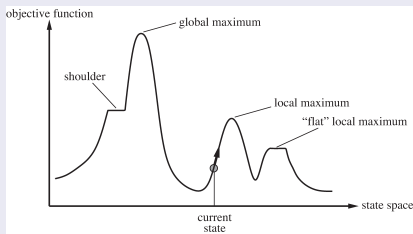
## Hill Climbing (Subida da Encosta)

- Escolhe, a cada passo, o nó que parece levar mais próximo do objetivo
- Move-se continuamente na direção de valores crescentes – encosta acima



## Hill Climbing (Subida da Encosta)

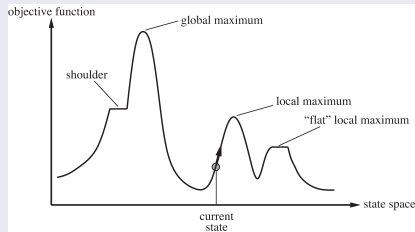
- Escolhe, a cada passo, o nó que parece levar mais próximo do objetivo
- Move-se continuamente na direção de valores crescentes – encosta acima
- Exige uma função de avaliação, para indicar quão próximo da solução está o nó atual





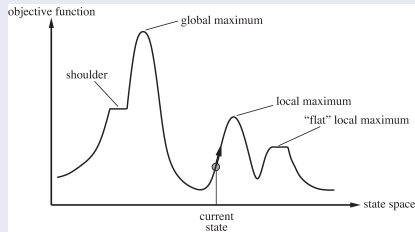
## Hill Climbing (Subida da Encosta)

- Pára quando atinge um pico



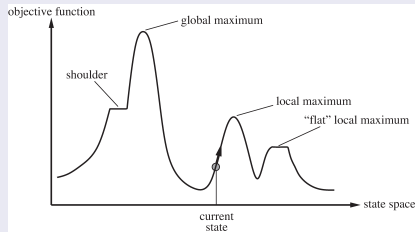
## Hill Climbing (Subida da Encosta)

- Pára quando atinge um pico
- Nenhum vizinho tem valor maior que o estado atual



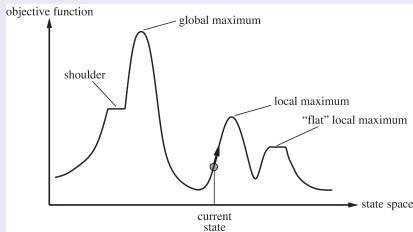
## Hill Climbing (Subida da Encosta)

- Pára quando atinge um pico
- Nenhum vizinho tem valor maior que o estado atual
- Não olha além dos vizinhos imediatos do estado em que está



## Hill Climbing (Subida da Encosta)


- Pára quando atinge um pico
  - Nenhum vizinho tem valor maior que o estado atual
- Não olha além dos vizinhos imediatos do estado em que está
- Considera todos os movimentos a partir do estado atual, selecionando o melhor deles como próximo estado



## Hill Climbing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
enquanto atual  $\neq$  objetivo E possui vizinhos faça  
|   vizinho  $\leftarrow$  vizinho de atual com maior valor de avaliação;  
|   se avaliação(vizinho)  $\leq$  avaliação(atual) então  
|   |   retorna atual;  
|   atual  $\leftarrow$  vizinho;  
retorna atual;
```

## Hill Climbing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
enquanto atual  $\neq$  objetivo E possui vizinhos faça  
  | vizinho  $\leftarrow$  vizinho de atual com maior valor de avaliação;  
  | se avaliação(vizinho)  $\leq$  avaliação(atual) então  
  |   | retorna atual;  
  |   | atual  $\leftarrow$  vizinho;   
retorna atual;
```

A cada passo, o nó atual é substituído pelo seu melhor vizinho

## Hill Climbing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
enquanto atual  $\neq$  objetivo E possui vizinhos faça  
  | vizinho  $\leftarrow$  vizinho de atual com maior valor de avaliação;  
  | se avaliação(vizinho)  $\leq$  avaliação(atual) então  
  |   | retorna atual;  
  |   |  
  | atual  $\leftarrow$  vizinho;  
retorna atual;
```

Nessa versão, o melhor vizinho era aquele com maior valor de avaliação.

## Hill Climbing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
enquanto atual  $\neq$  objetivo E possui vizinhos faça  
  | vizinho  $\leftarrow$  vizinho de atual com maior valor de avaliação;  
  | se avaliação(vizinho)  $\leq$  avaliação(atual) então  
  |   | retorna atual;  
  |   |  
  | atual  $\leftarrow$  vizinho;  
retorna atual;
```

Se usássemos uma heurística de custo, teria que ser o vizinho com o menor custo

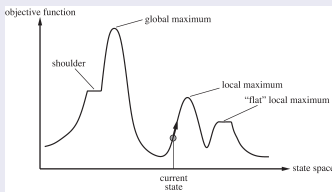


# Busca Local

## Hill Climbing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;  
**enquanto** *atual*  $\neq$  objetivo **E** possui vizinhos **faça**  
    *vizinho*  $\leftarrow$  vizinho de *atual* com maior valor de avaliação;  
    **se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*) **então**  
        **retorna** *atual*;  
    *atual*  $\leftarrow$  *vizinho*;  
**retorna** *atual*;

Hill Climbing modifica  
então o estado atual  
tentando melhorá-lo



## Hill Climbing – Algoritmo


```
atual  $\leftarrow$  Nó inicial;  
enquanto  $atual \neq objetivo$  E possui vizinhos faça  
    | vizinho  $\leftarrow$  vizinho de atual com maior valor de avaliação;  
    | se  $avaliação(vizinho) \leq avaliação(atual)$  então  
    | | retorna atual;  
    | |  
    | atual  $\leftarrow$  vizinho;  
    |  
retorna atual;
```

E se houver um empate  
entre vizinhos de um nó?

## Hill Climbing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
enquanto atual  $\neq$  objetivo E possui vizinhos faça  
  | vizinho  $\leftarrow$  vizinho de atual com maior valor de avaliação;  
  | se avaliação(vizinho)  $\leq$  avaliação(atual) então  
  |   | retorna atual;  
  |   |  
  |   | atual  $\leftarrow$  vizinho;  
  |  
retorna atual;
```

O algoritmo tipicamente  
escolhe um aleatoriamente



## Hill Climbing – Exemplo

- Voar de uma cidade a outra, minimizando o número de conexões

## Hill Climbing – Exemplo

- Voar de uma cidade a outra, minimizando o número de conexões
- Heurística:

## Hill Climbing – Exemplo

- Voar de uma cidade a outra, minimizando o número de conexões
- Heurística:
  - Quanto maior a distância de cada voo, a partir do estado corrente, maior a probabilidade de estar mais perto do destino

## Hill Climbing – Exemplo

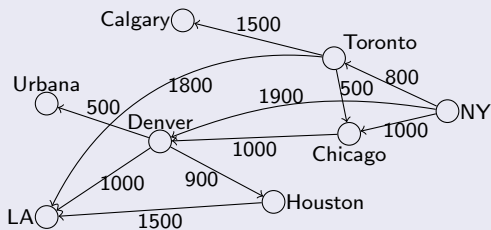
- Voar de uma cidade a outra, minimizando o número de conexões
- Heurística:
  - Quanto maior a distância de cada voo, a partir do estado corrente, maior a probabilidade de estar mais perto do destino
- Função de Avaliação:

## Hill Climbing – Exemplo

- Voar de uma cidade a outra, minimizando o número de conexões
- Heurística:
  - Quanto maior a distância de cada voo, a partir do estado corrente, maior a probabilidade de estar mais perto do destino
- Função de Avaliação:
  - Maior aresta saindo do nó corrente  $\Rightarrow$  maior valor do nó ao final dessa aresta



## Hill Climbing – Exemplo

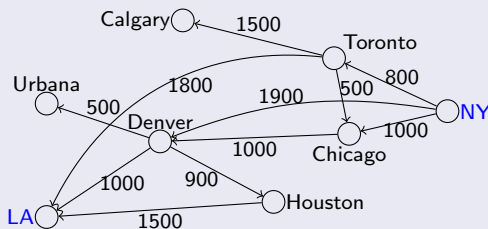


Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior

    valor de avaliação;

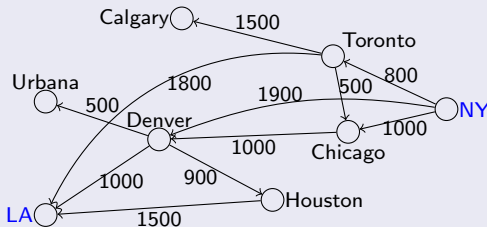
**se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*)

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  **Nó inicial**;

**enquanto** *atual*  $\neq$  *objetivo* **E possui vizinhos** **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

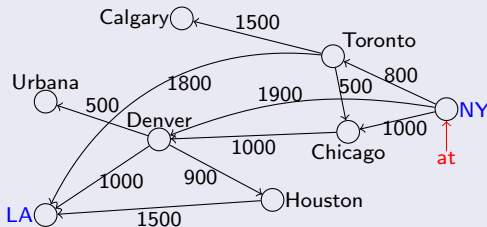
**se** *avaliação(vizinho)*  $\leq$  *avaliação(atual)*

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  *objetivo* **E possui vizinhos** **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

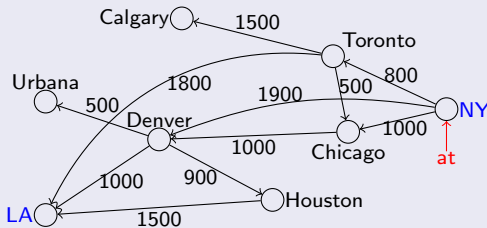
**se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*)

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior

    valor de avaliação;

**se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*)

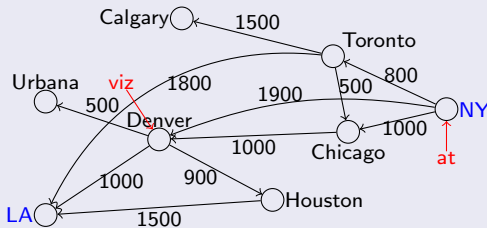
**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;

Quanto mais longe do  
nó atual melhor será o  
valor de um nó vizinho



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

**se** *avaliação(vizinho)*  $\leq$  *avaliação(atual)*

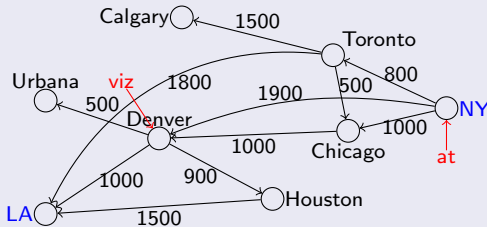
**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;

Por essa heurística, o valor do nó atual será sempre menor que o do vizinho, se houver vizinho



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

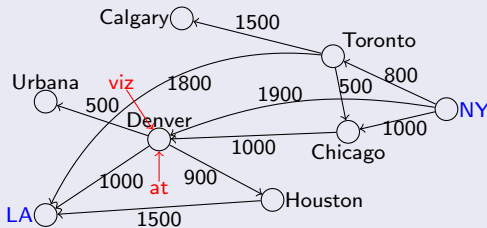
**se** *avaliação(vizinho)*  $\leq$  *avaliação(atual)*

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.



# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  *objetivo* **E possui vizinhos** **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

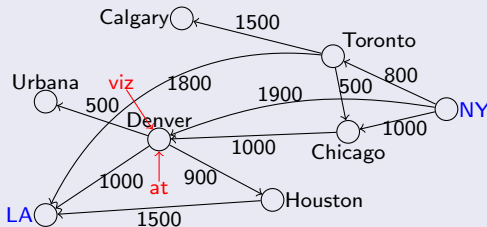
**se** *avaliação(vizinho)*  $\leq$  *avaliação(atual)*

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior

    valor de avaliação;

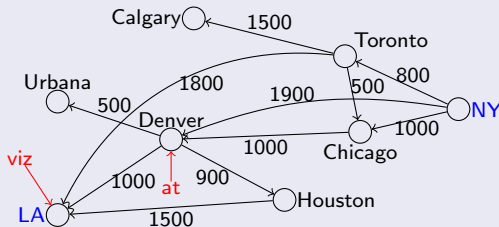
**se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*)

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

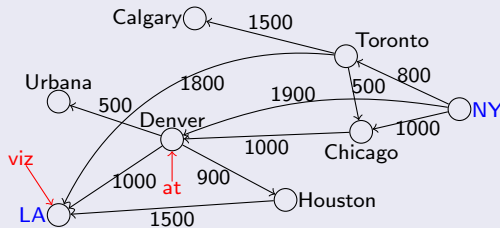
**se** *avaliação(vizinho)*  $\leq$  *avaliação(atual)*

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

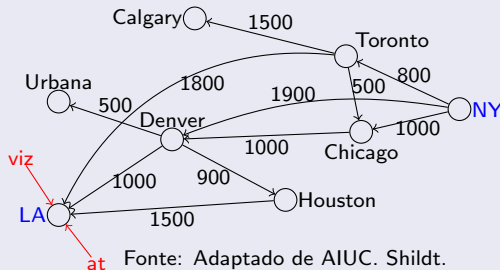
**se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*)

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



Fonte: Adaptado de AIUC. Shildt.

# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  *objetivo* **E possui vizinhos** **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

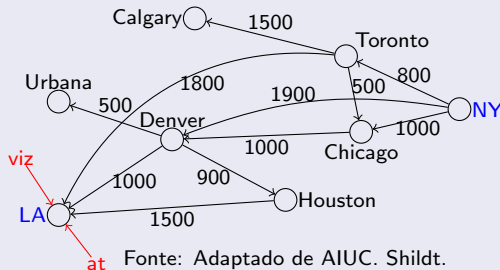
**se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*)

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



# Busca Local

## Hill Climbing – Exemplo

NY para LA:

*atual*  $\leftarrow$  Nó inicial;

**enquanto** *atual*  $\neq$  objetivo *E* possui vizinhos **faça**

*vizinho*  $\leftarrow$  vizinho de *atual* com maior  
    valor de avaliação;

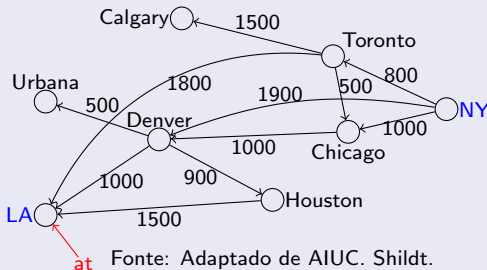
**se** *avaliação*(*vizinho*)  $\leq$  *avaliação*(*atual*)

**então**

**retorna** *atual*;

*atual*  $\leftarrow$  *vizinho*;

**retorna** *atual*;



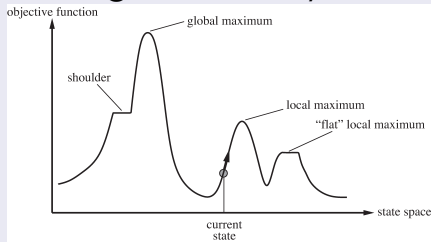
Fonte: Adaptado de AIUC. Shildt.

## Hill Climbing – Problemas

- Pode não encontrar uma solução (mesmo havendo uma)

## Hill Climbing – Problemas

- Pode não encontrar uma solução (mesmo havendo uma)
- Não consegue ir adiante nas seguintes situações:

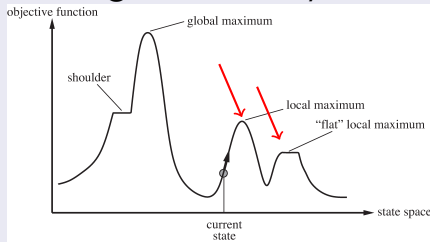


Fonte: AIMA. Russell & Norvig.



## Hill Climbing – Problemas

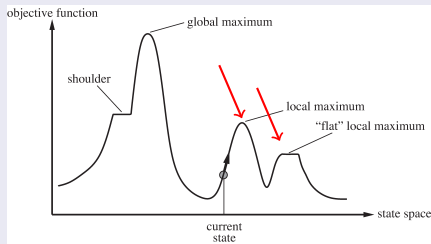
- Pode não encontrar uma solução (mesmo havendo uma)
- Não consegue ir adiante nas seguintes situações:
  - Máximos locais: quando não há vizinho melhor, mas o estado atual não é a melhor solução (ou nem mesmo é solução)



Fonte: AIMA. Russell & Norvig.

## Hill Climbing – Problemas

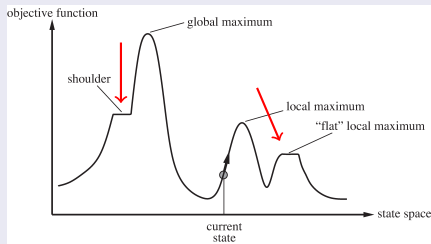
- Não consegue ir adiante nas seguintes situações:
  - Cordilheiras: sequência de máximos locais



Fonte: AIMA. Russell & Norvig.

## Hill Climbing – Problemas

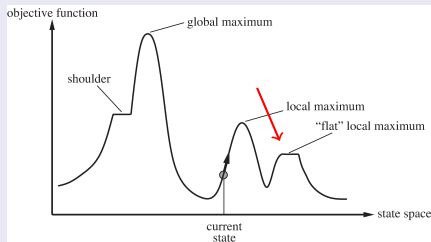
- Não consegue ir adiante nas seguintes situações:
  - Cordilheiras: sequência de máximos locais
  - Platôs: quando os vizinhos são iguais ao estado atual, e este não é solução



Fonte: AIMA. Russell & Norvig.

## Hill Climbing – Problemas

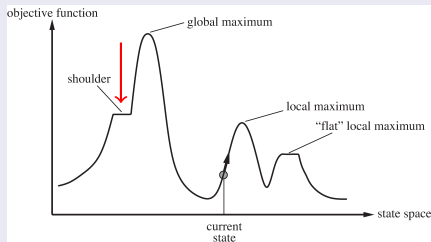
- Não consegue ir adiante nas seguintes situações:
  - Cordilheiras: sequência de máximos locais
  - Platôs: quando os vizinhos são iguais ao estado atual, e este não é solução
  - Podem ser máximos locais planos



Fonte: AIMA. Russell & Norvig.

## Hill Climbing – Problemas

- Não consegue ir adiante nas seguintes situações:
  - Cordilheiras: sequência de máximos locais
  - Platôs: quando os vizinhos são iguais ao estado atual, e este não é solução
  - Podem ser máximos locais planos ou “ombros”

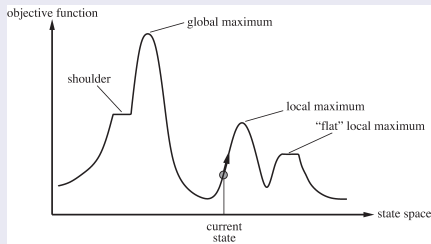


Fonte: AIMA. Russell & Norvig.

## Hill Climbing – Problemas

- Não consegue ir adiante nas seguintes situações:

- Cordilheiras: sequência de máximos locais
- Platôs: quando os vizinhos são iguais ao estado atual, e este não é solução
  - Podem ser máximos locais planos ou “ombros”



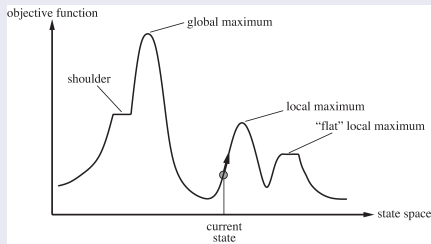
Fonte: AIMA. Russell & Norvig.

- Solução: Subida da Encosta de Recomeço Aleatório

## Hill Climbing – Problemas

- Não consegue ir adiante nas seguintes situações:

- Cordilheiras: sequência de máximos locais
- Platôs: quando os vizinhos são iguais ao estado atual, e este não é solução
  - Podem ser máximos locais planos ou “ombros”



Fonte: AIMA. Russell & Norvig.

- Solução: Subida da Encosta de Recomeço Aleatório
  - Random-Restart Hill Climbing

## Random-Restart Hill Climbing – Funcionamento



## Random-Restart Hill Climbing – Funcionamento

- Se não conseguir da primeira vez, tente de novo

## Random-Restart Hill Climbing – Funcionamento

- Se não conseguir da primeira vez, tente de novo
- Se ficou preso em um estado não muito bom:

## Random-Restart Hill Climbing – Funcionamento

- Se não conseguir da primeira vez, tente de novo
- Se ficou preso em um estado não muito bom:
  - Gere aleatoriamente um novo estado inicial

## Random-Restart Hill Climbing – Funcionamento

- Se não conseguir da primeira vez, tente de novo
- Se ficou preso em um estado não muito bom:
  - Gere aleatoriamente um novo estado inicial
  - Comece a busca novamente

## Random-Restart Hill Climbing – Funcionamento

- Se não conseguir da primeira vez, tente de novo
- Se ficou preso em um estado não muito bom:
  - Gere aleatoriamente um novo estado inicial
  - Comece a busca novamente



Fonte: <https://somosamp.wordpress.com/2017/02/02/toca-raul-tente-outra-vez/>

## Random-Restart Hill Climbing

- O algoritmo executa, então, uma série de hill climbings a partir de estados iniciais aleatórios, até que o objetivo seja atingido

## Random-Restart Hill Climbing

- O algoritmo executa, então, uma série de hill climbings a partir de estados iniciais aleatórios, até que o objetivo seja atingido
- Embora não pareça, é muito eficiente

## Random-Restart Hill Climbing

- O algoritmo executa, então, uma série de hill climbings a partir de estados iniciais aleatórios, até que o objetivo seja atingido
- Embora não pareça, é muito eficiente
- Também é completo, com probabilidade  $\approx 1$



## Random-Restart Hill Climbing

- O algoritmo executa, então, uma série de hill climbings a partir de estados iniciais aleatórios, até que o objetivo seja atingido
- Embora não pareça, é muito eficiente
- Também é completo, com probabilidade  $\approx 1$ 
  - Se cada busca tiver probabilidade  $p$  de sucesso, o número de reinícios esperado é  $\frac{1}{p}$

## Random-Restart Hill Climbing

- O algoritmo executa, então, uma série de hill climbings a partir de estados iniciais aleatórios, até que o objetivo seja atingido
- Embora não pareça, é muito eficiente
- Também é completo, com probabilidade  $\approx 1$ 
  - Se cada busca tiver probabilidade  $p$  de sucesso, o número de reinícios esperado é  $\frac{1}{p}$
  - Ex:  $p = 0.01 \Rightarrow \frac{1}{p} = 100$

## Simulated Annealing

## Simulated Annealing

- Por nunca “descer a encosta”, movendo-se para estados com menor valor (ou maior custo), hill climbing é garantidamente incompleto

## Simulated Annealing

- Por nunca “descer a encosta”, movendo-se para estados com menor valor (ou maior custo), hill climbing é garantidamente incompleto
- Sempre pode ficar preso em máximos locais, platôs etc.

## Simulated Annealing

- Por nunca “descer a encosta”, movendo-se para estados com menor valor (ou maior custo), hill climbing é garantidamente incompleto
  - Sempre pode ficar preso em máximos locais, platôs etc.
- Por outro lado, até uma caminhada aleatória seria completa, embora ineficiente

## Simulated Annealing

- Por nunca “descer a encosta”, movendo-se para estados com menor valor (ou maior custo), hill climbing é garantidamente incompleto
  - Sempre pode ficar preso em máximos locais, platôs etc.
- Por outro lado, até uma caminhada aleatória seria completa, embora ineficiente
  - Quando escolhemos aleatoriamente o vizinho de um nó, a partir de uma distribuição uniforme

## Simulated Annealing

- Por nunca “descer a encosta”, movendo-se para estados com menor valor (ou maior custo), hill climbing é garantidamente incompleto
  - Sempre pode ficar preso em máximos locais, platôs etc.
- Por outro lado, até uma caminhada aleatória seria completa, embora ineficiente
  - Quando escolhemos aleatoriamente o vizinho de um nó, a partir de uma distribuição uniforme
- E se combinássemos os 2?



## Simulated Annealing

- Por nunca “descer a encosta”, movendo-se para estados com menor valor (ou maior custo), hill climbing é garantidamente incompleto
  - Sempre pode ficar preso em máximos locais, platôs etc.
- Por outro lado, até uma caminhada aleatória seria completa, embora ineficiente
  - Quando escolhemos aleatoriamente o vizinho de um nó, a partir de uma distribuição uniforme
- E se combinássemos os 2? Têmpera Simulada

## Simulated Annealing

- Por nunca “descer a encosta”, movendo-se para estados com menor valor (ou maior custo), hill climbing é garantidamente incompleto
  - Sempre pode ficar preso em máximos locais, platôs etc.
- Por outro lado, até uma caminhada aleatória seria completa, embora ineficiente
  - Quando escolhemos aleatoriamente o vizinho de um nó, a partir de uma distribuição uniforme
- E se combinássemos os 2? Têmpera Simulada
  - Simulated Annealing

## Simulated Annealing

- Em metalurgia, têmpera é o processo usado para deixar metais e vidros mais resistentes

## Simulated Annealing

- Em metalurgia, têmpera é o processo usado para deixar metais e vidros mais resistentes
- Estes são aquecidos até uma alta temperatura e então gradualmente esfriados

## Simulated Annealing

- Em metalurgia, têmpera é o processo usado para deixar metais e vidros mais resistentes
  - Estes são aquecidos até uma alta temperatura e então gradualmente esfriados
- E o que isso tem a ver com nosso problema?

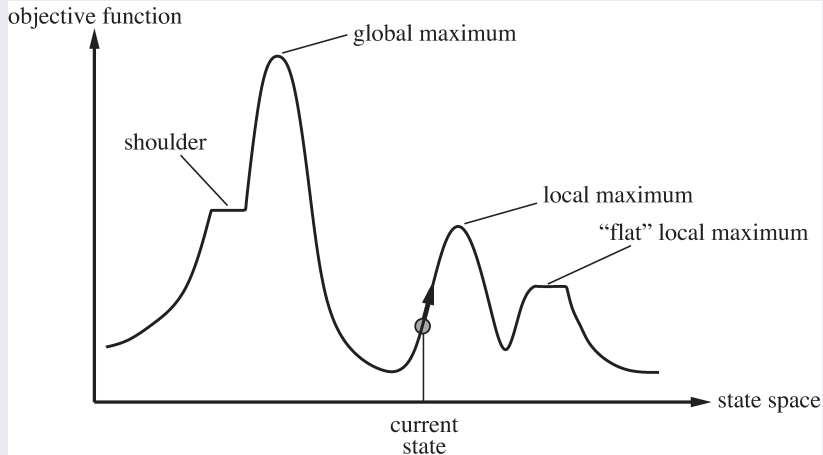
## Simulated Annealing

- Em metalurgia, têmpera é o processo usado para deixar metais e vidros mais resistentes
  - Estes são aquecidos até uma alta temperatura e então gradualmente esfriados
- E o que isso tem a ver com nosso problema?
  - Com Simulated Annealing, começamos adicionando um ruído aleatório alto (alta temperatura), e então gradualmente reduzimos esse ruído (baixamos a temperatura)

## Simulated Annealing

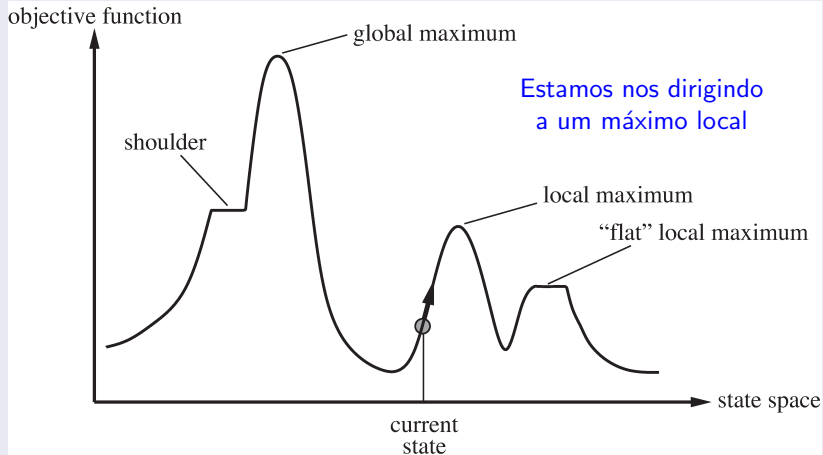
- Em metalurgia, têmpera é o processo usado para deixar metais e vidros mais resistentes
  - Estes são aquecidos até uma alta temperatura e então gradualmente esfriados
- E o que isso tem a ver com nosso problema?
  - Com Simulated Annealing, começamos adicionando um ruído aleatório alto (alta temperatura), e então gradualmente reduzimos esse ruído (baixamos a temperatura)
  - Assim, se estivermos presos em um mínimo local (ou máximo), conseguimos nos livrar dele com um “salto”

## Simulated Annealing



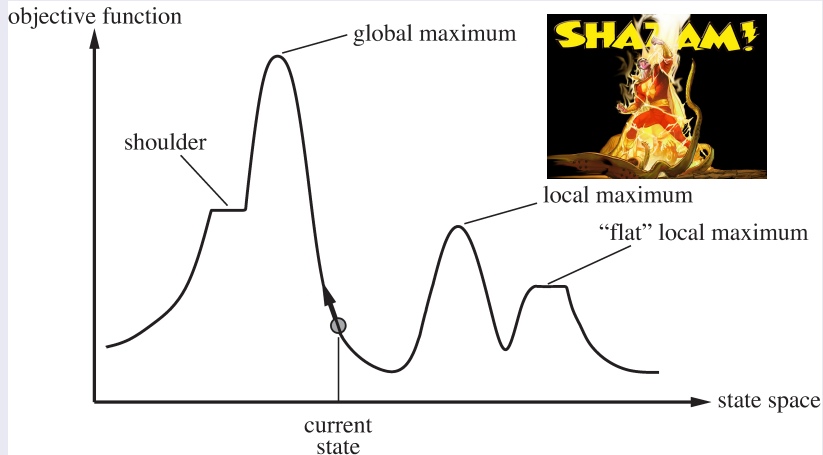


## Simulated Annealing



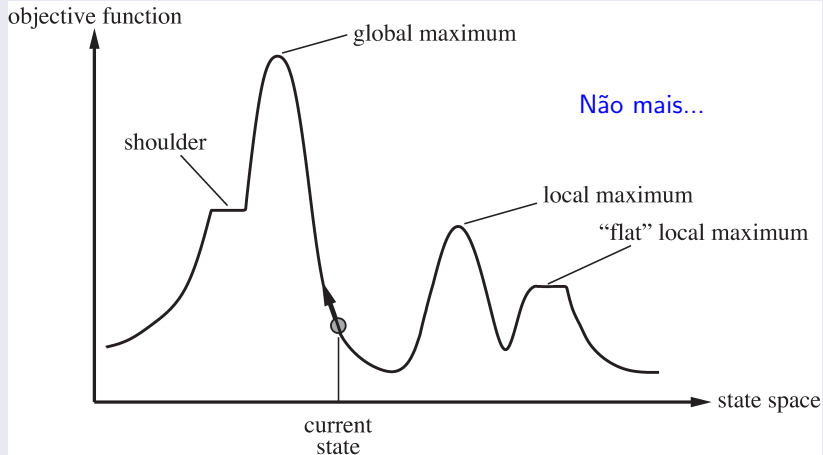
# Busca Local

## Simulated Annealing

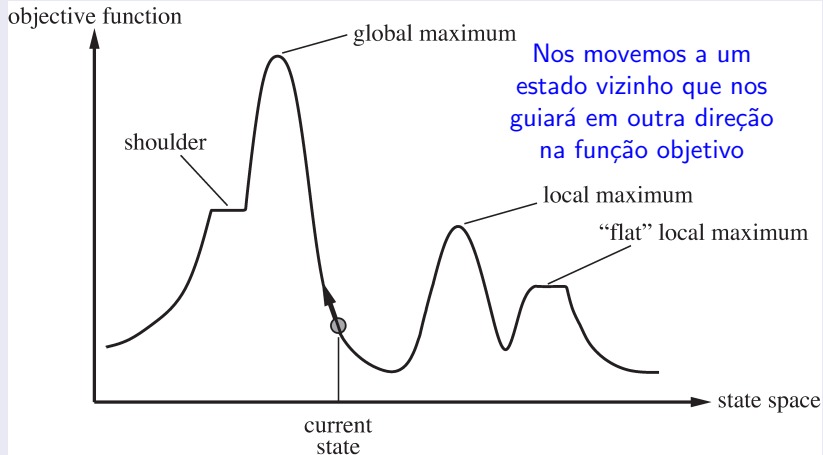


# Busca Local

## Simulated Annealing



## Simulated Annealing



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

        └ faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

## Simulated Annealing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
T  $\leftarrow$  T0;  
repita  
| próximo  $\leftarrow$  seleciona aleatoriamente vizinho de atual;  
|  $\Delta E \leftarrow$  avaliação(próximo) - avaliação(atual);  
| se  $\Delta E > 0$  então  
| | atual  $\leftarrow$  próximo  
| senão  
| | faça atual  $\leftarrow$  próximo com probabilidade  $p = e^{\Delta E/T}$   
| T  $\leftarrow$  próximaTemperatura(T);  
até T < Tfinal;  
retorna atual;
```

Escolhe o próximo movimento aleatoriamente

## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

Se o movimento melhora a  
situação, é sempre aceito

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow \text{avaliação}(\text{próximo}) - \text{avaliação}(\text{atual})$ ;

**se**  $\Delta E > 0$  **então**

        |  $\text{atual} \leftarrow \text{próximo}$

**senão**

        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow \text{próximaTemperatura}(T)$ ;

**até**  $T < T_{\text{final}}$ ;

**retorna** *atual*;

## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

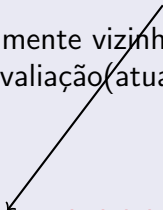
        | **faça** *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Do contrário, ele tem probabilidade  $e^{\Delta E/T}$  de ser aceito

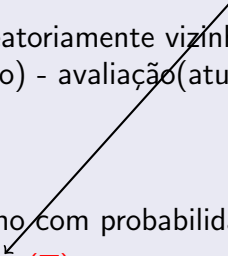




## Simulated Annealing – Algoritmo

```
atual ← Nó inicial;  
T ←  $T_0$ ;  
repita  
| próximo ← seleciona aleatoriamente vizinho de atual;  
|  $\Delta E$  ← avaliação(próximo) - avaliação(atual);  
| se  $\Delta E > 0$  então  
| | atual ← próximo  
| senão  
| | faça atual ← próximo com probabilidade  $p = e^{\Delta E / T}$   
| T ← próximaTemperatura(T);  
até  $T < T_{final}$ ;  
retorna atual;
```

Mapeamento que determina o valor da temperatura a cada iteração



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

Pode ser algo tão simples  
quanto  $T \leftarrow \alpha T, \alpha < 1$

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

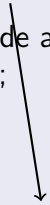
**até**  $T < T_{final}$ ;

**retorna** *atual*;

## Simulated Annealing – Algoritmo

```
atual ← Nó inicial;  
T ← T0;  
repita  
| próximo ← seleciona aleatoriamente vizinho de atual;  
|  $\Delta E \leftarrow \text{avaliação}(\text{próximo}) - \text{avaliação}(\text{atual});$   
| se  $\Delta E > 0$  então  
| | atual ← próximo  
| senão  
| | faça atual ← próximo com probabilidade  $p = e^{\Delta E/T}$   
| T ← próximaTemperatura(T);  
até  $T < T_{\text{final}}$ ;  
retorna atual;
```

A probabilidade decresce exponencialmente com a piora de *próximo* (se  $\Delta(E) \leq 0$ )



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

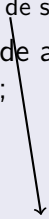
        └ faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Se  $\Delta(E) \approx 0$ , então  
 $p \approx 1$ , e a atribuição tem  
mais chance de ser aceita



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Também decresce exponencialmente com a temperatura  $T$

## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

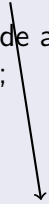
        └ faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Se  $T \rightarrow \infty$ , então  
 $\frac{\Delta(E)}{T} \rightarrow 0$  e  $p \rightarrow 1$



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

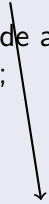
        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Se  $T \rightarrow 0$ , então  $\frac{\Delta(E)}{T} \rightarrow -\infty$   
(lembre que  $\Delta(E) \leq 0$ ) e  $p \rightarrow 0$



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

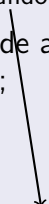
        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Assim, movimentos ruins têm mais chance de serem aceitos no início, quando  $T$  é alta





## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

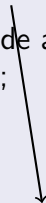
        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Na medida em que  $T$  reduz,  
eles se tornam menos prováveis



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

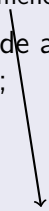
        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Movimentos “colina abaixo” são então mais aceitos no início, ficando cada vez menos frequentes



## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  avaliação(*próximo*) - avaliação(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

Se próximaTemperatura baixar  $T$  de modo suficientemente lento, o algoritmo encontrará um ótimo global com  $p \approx 1$

## Simulated Annealing – Algoritmo

*atual*  $\leftarrow$  Nó inicial;

$T \leftarrow T_0$ ;

Aqui medimos o valor de cada nó

**repita**

*próximo*  $\leftarrow$  seleciona aleatoriamente vizinho de *atual*;

$\Delta E \leftarrow$  **avaliação**(*próximo*) - **avaliação**(*atual*);

**se**  $\Delta E > 0$  **então**

        | *atual*  $\leftarrow$  *próximo*

**senão**

        | faça *atual*  $\leftarrow$  *próximo* com probabilidade  $p = e^{\Delta E/T}$

$T \leftarrow$  próximaTemperatura( $T$ );

**até**  $T < T_{final}$ ;

**retorna** *atual*;

## Simulated Annealing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
T  $\leftarrow T_0$ ;  
repita  
| próximo  $\leftarrow$  seleciona aleatoriamente vizinho de atual;  
|  $\Delta E \leftarrow$  avaliação(próximo) - avaliação(atual);  
| se  $\Delta E > 0$  então  
| | atual  $\leftarrow$  próximo  
| senão  
| |  $\perp$  faça atual  $\leftarrow$  próximo com probabilidade  $p = e^{\Delta E / T}$   
| T  $\leftarrow$  próximaTemperatura(T);  
até  $T < T_{final}$ ;  
retorna atual;
```

Se medíssemos custo,  
teríamos que mudar...

## Simulated Annealing – Algoritmo

```
atual  $\leftarrow$  Nó inicial;  
T  $\leftarrow T_0$ ;  
repita  
| próximo  $\leftarrow$  seleciona aleatoriamente vizinho de atual;  
|  $\Delta E \leftarrow \text{custo}(\text{próximo}) - \text{custo}(\text{atual})$ ;  
| se  $\Delta E \leq 0$  então  
| | atual  $\leftarrow$  próximo  
| senão  
| | faça atual  $\leftarrow$  próximo com probabilidade  $p = e^{-\Delta E/T}$   
| T  $\leftarrow$  próximaTemperatura(T);  
até  $T < T_{final}$ ;  
retorna atual;
```

Se medíssemos custo,  
teríamos que mudar...

# Referências

- Russell, S.; Norvig P. (2010): Artificial Intelligence: A Modern Approach. Prentice Hall. 3a ed.
- Schildt, H. (1987): Artificial Intelligence Using C. McGraw-Hill.
- <https://www.baeldung.com/java-hill-climbing-algorithm>
- [http://conteudo.icmc.usp.br/pessoas/sandra/G9\\_t2/annealing.htm](http://conteudo.icmc.usp.br/pessoas/sandra/G9_t2/annealing.htm)
- [https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16\\_410F10\\_lec14.pdf](https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec14.pdf)
- <http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-034Spring-2005/LectureNotes/index.htm>