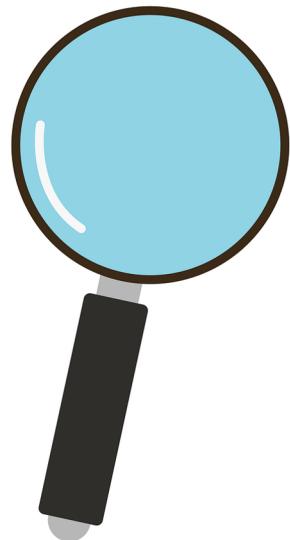


# Métodos de Busca



## Busca Heurística



# Métodos de Busca

## Busca Heurística

- Heurísticas em IA são, basicamente, regras que tentam determinar qual o melhor caminho até a solução.
- Na maioria dos casos, os métodos de busca heurísticos são baseados na maximização ou minimização de alguns aspectos do problema.
- O objetivo é minimizar o custo para se alcançar o objetivo.
- Na maioria dos problemas este custo não pode ser determinado com exatidão.
- A função que estima este custo é denominada de *função heurística* e denotada pela letra  $h$ .

# Métodos de Busca

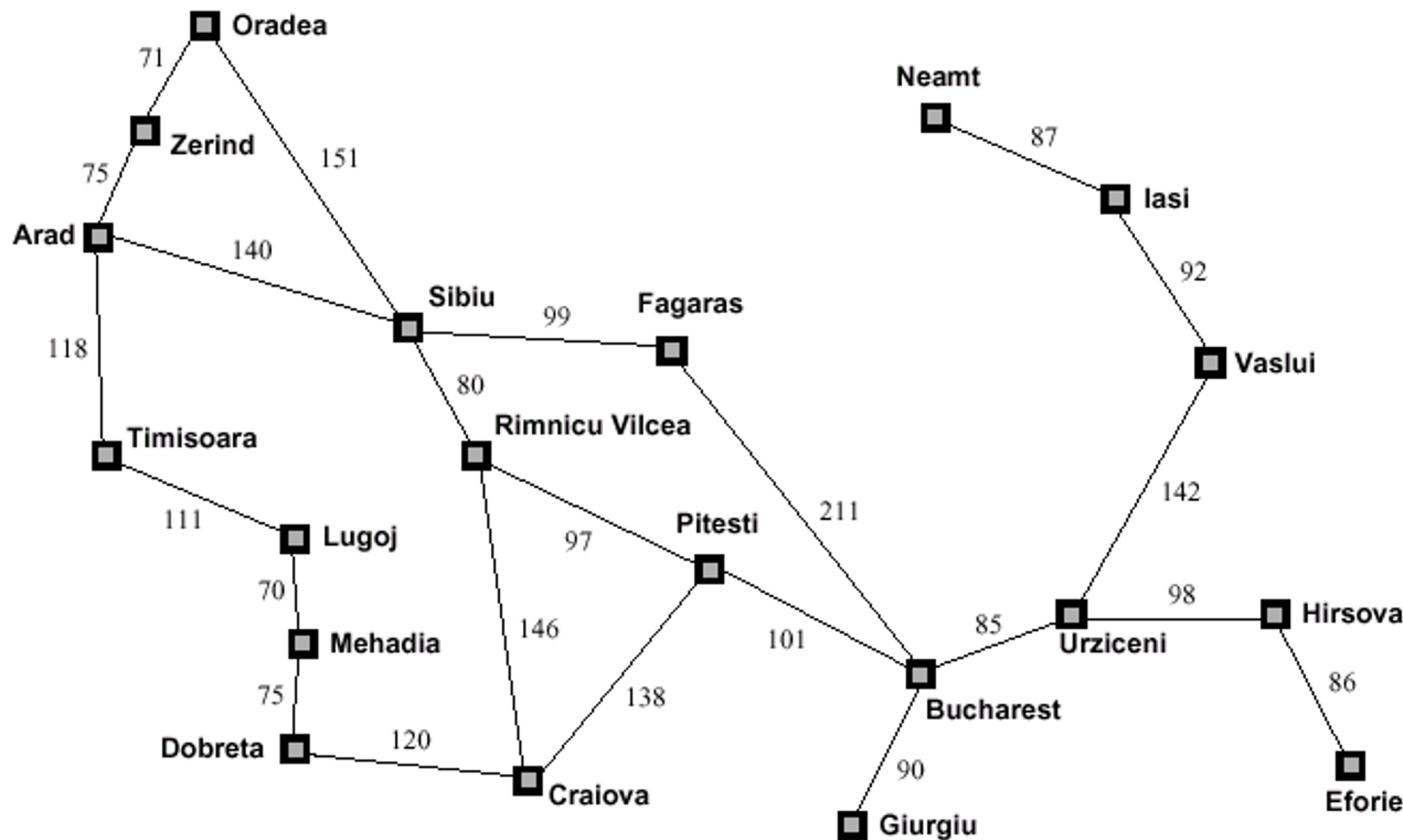
## Busca Heurística: busca gulosa

- Nesta estratégia o próximo nó a ser expandido é o avaliado pela função  $h$  como o que leva à solução por um caminho de menor custo.
- Por exemplo, no caso do problema para se chegar até bucareste, suponha que tenhamos a informação das distância em linha reta de todas as cidades até Bucareste.
- Pode usar esta informação como uma função heurística. Ou seja,

$$h(n) = \text{a distância em linha reta de } n \text{ até Bucareste}$$

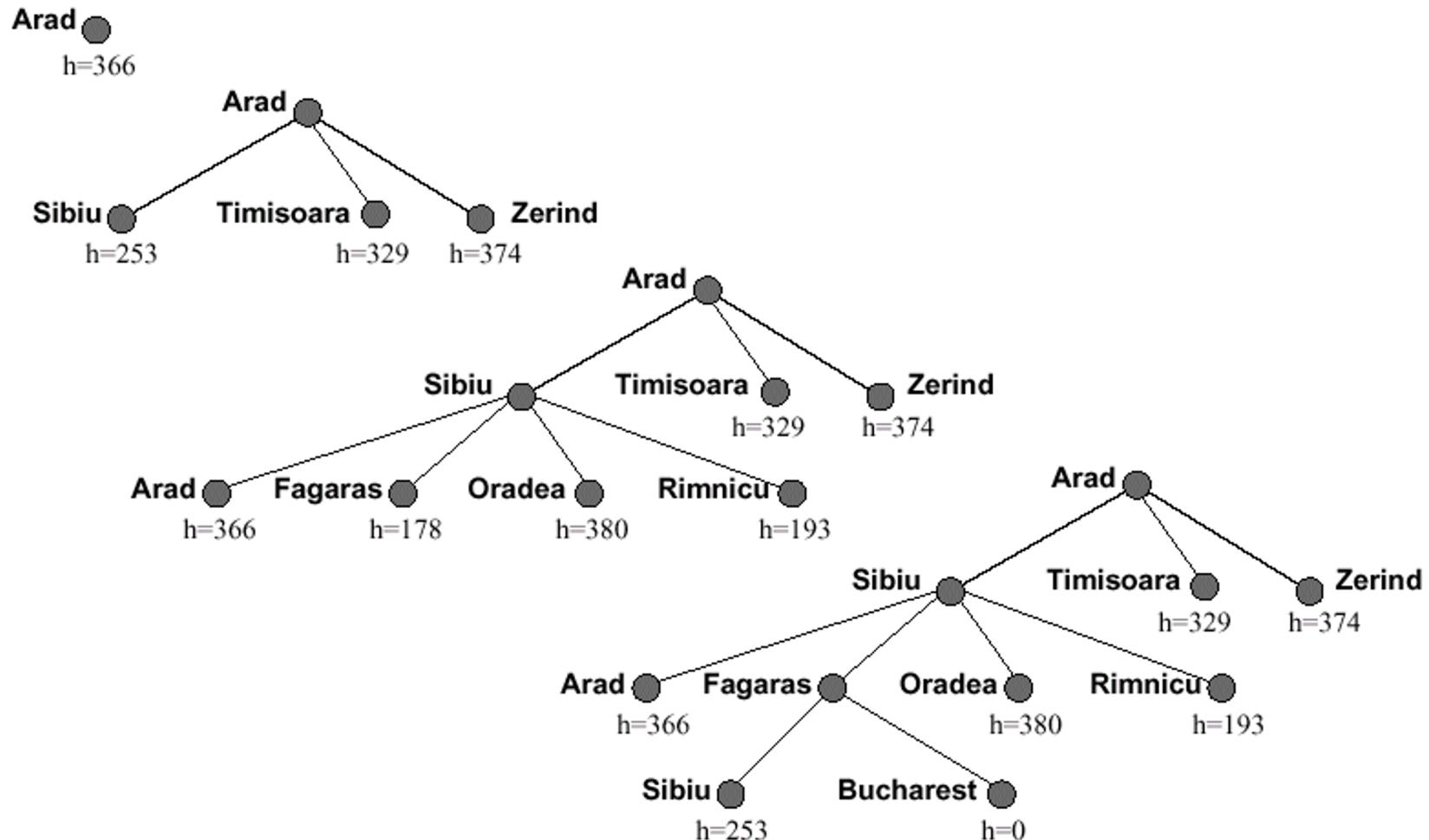
# Métodos de Busca

## Busca Heurística: busca gulosa



# Métodos de Busca

## Busca Heurística: busca gulosa



# Métodos de Busca

## Busca Heurística: busca gulosa

- A busca gulosa possui os mesmos problemas da busca em profundidade: não é ótima e nem completa.
- Além disso, devido à necessidade de reter as informações sobre os nós expandidos, possui requerimentos de memória bem maiores.
- No entanto, pode ser bem eficiente se a função heurística for bem escolhida.

# Métodos de Busca



## Busca Heurística: A\*



Fonte: wikipedia

[http://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm)

# Métodos de Busca

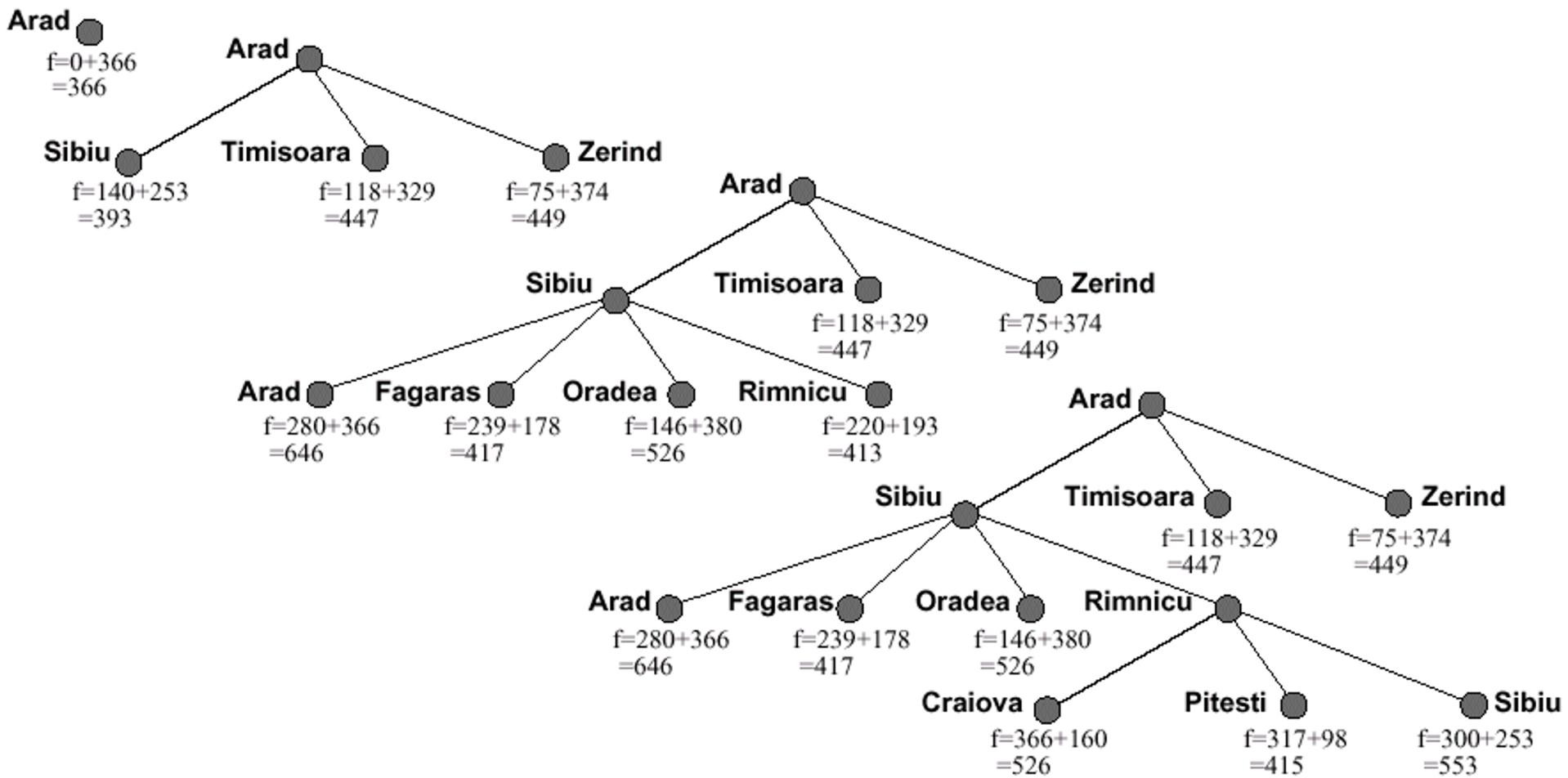
## Busca Heurística: A\*

- A busca gulosa pode ser eficiente mas não é ótima e nem completa.
- A busca de custo uniforme é ótima e completa mas pode ser bastante ineficiente.
- Existe uma forma de combinar as vantagens dos dois métodos?
- **Sim! O método A\*.**
- Ele usa uma função  $f$  que é a combinação de  $g$  e  $h$ :

$$f(n) = g(n) + h(n)$$

# Métodos de Busca

## Busca Heurística: A\*



# Métodos de Busca

## Busca Heurística: A\*

- A estratégia A\* é ótima e completa desde que  $h$  compute uma heurística *admissível*.
- $h$  é admissível se nunca superestima o custo para alcançar o objetivo.
- Por exemplo, a heurística utilizada para alcançar Bucareste é admissível, pois a distância será maior ou igual à estimada.
- Qual seria uma heurística admissível para o quebra-cabeça?

# Métodos de Busca

## Busca Heurística: A\* (comparações Quebra-cabeça)

d	Search Cost			Effective Branching Factor		
	IDS	A*(h <sub>1</sub> )	A*(h <sub>2</sub> )	IDS	A*(h <sub>1</sub> )	A*(h <sub>2</sub> )
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

IDS — Busca em profundidade iterativa.

h<sub>1</sub> — número de elementos fora da posição.

h<sub>2</sub> — soma das distâncias “quarteirões” de cada elemento.

# Métodos de Busca

## Busca Heurística: A\* (prova da otimalidade)

Seja  $f^*$  o custo do caminho até o estado ótimo  $G$ . Se A\* não é ótimo então ela selecionou um nó objetivo  $G_2$  tal que:

$$g(G_2) > f^*$$

Seja  $n$  um nó no caminho ótimo. Uma vez que  $h$  é admissível temos:

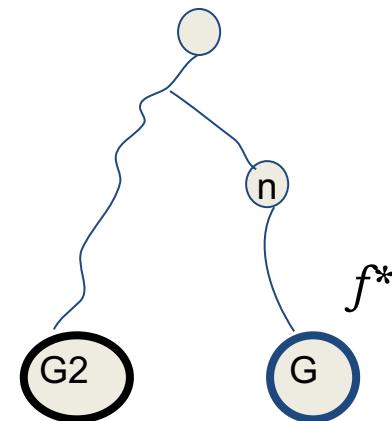
$$f^* \geq f(n)$$

Como  $n$  não foi escolhido temos

$$f(n) \geq f(G_2)$$

Combinando temos

$$f^* \geq f(G_2)$$



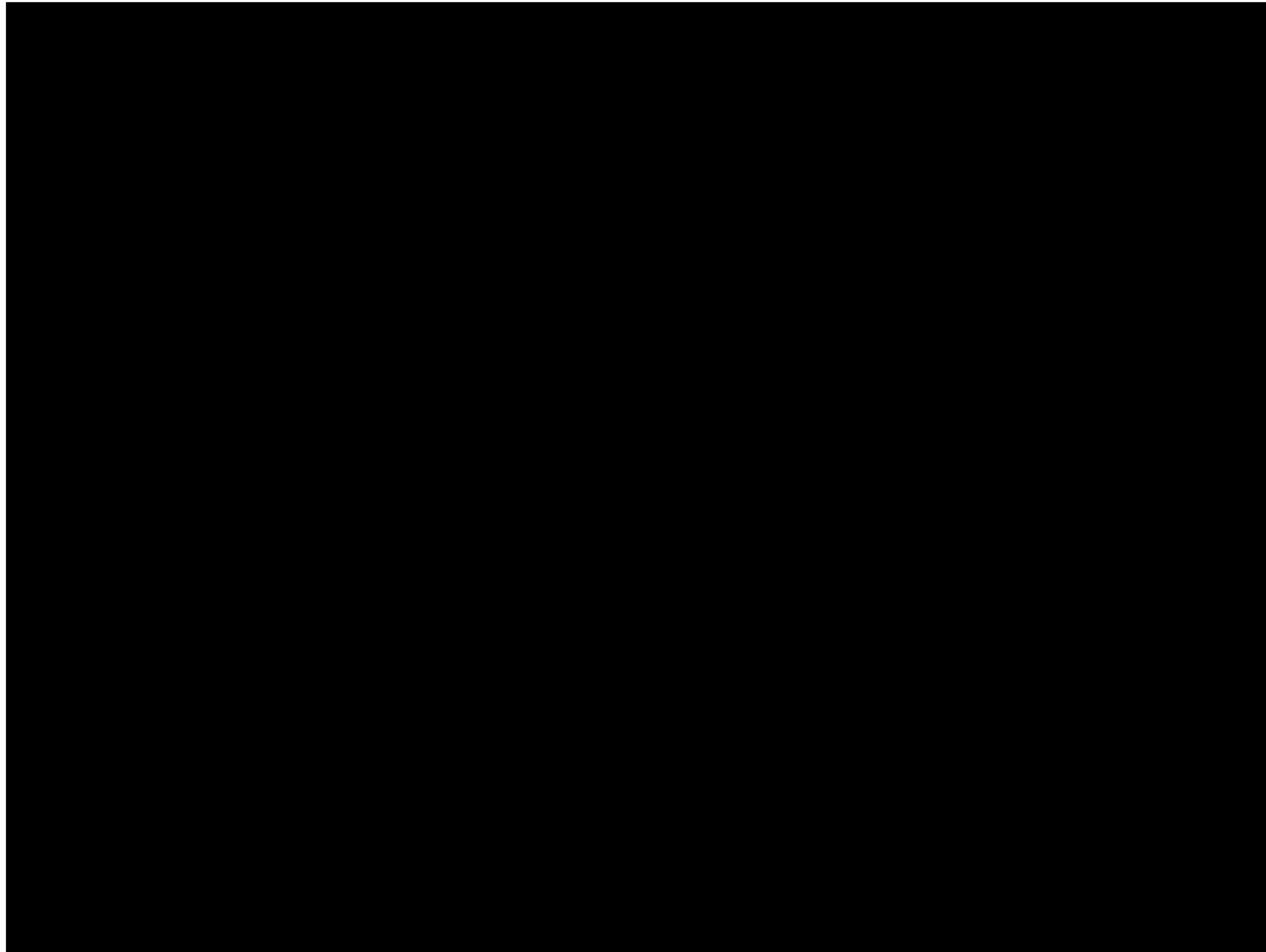
Como é um estado objetivo temos  $h(G_2) = 0$ , logo  $f(G_2) = g(G_2)$ .

$$f^* \geq g(G_2)$$

Portanto, contradição.

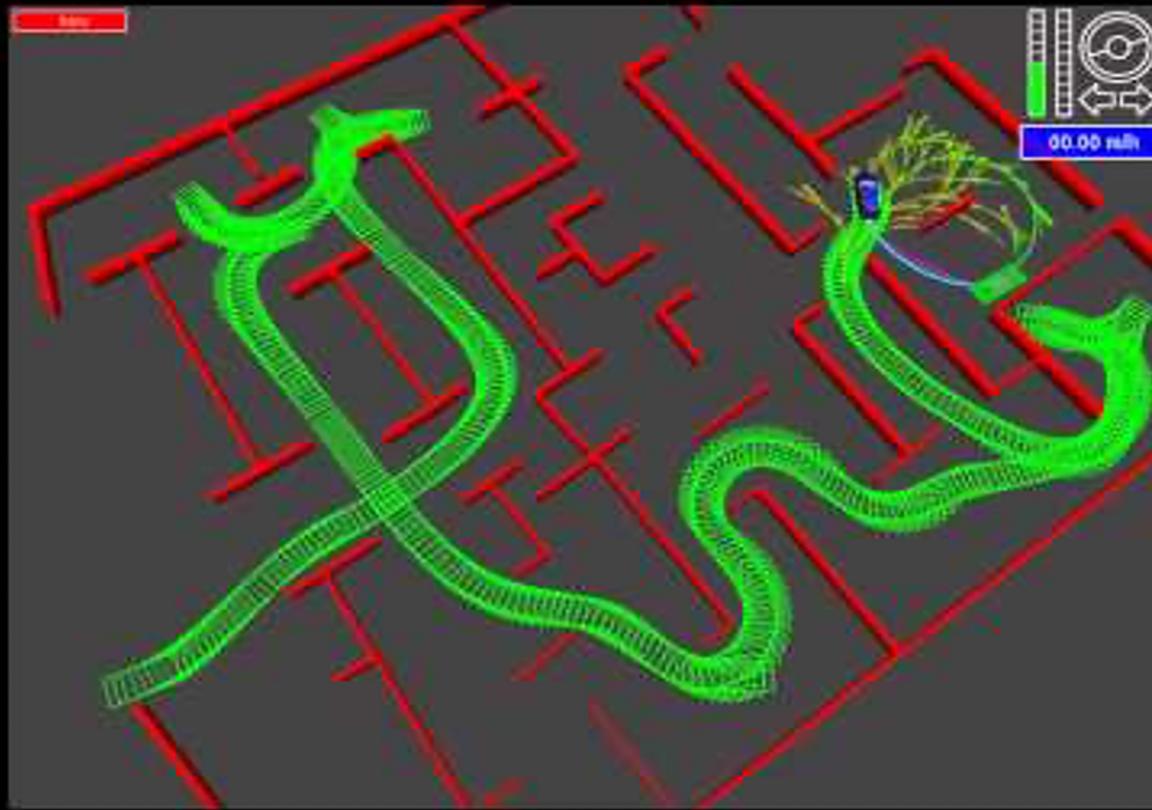
# Métodos de Busca

## Busca Heurística: A\* em jogos



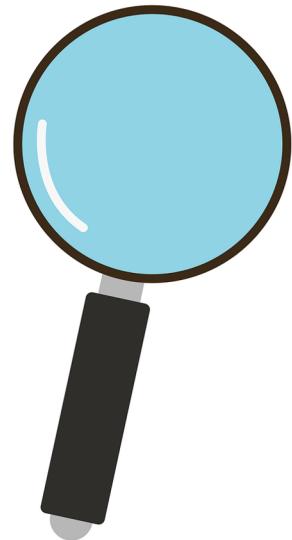
# Métodos de Busca

## Busca Heurística: A\* em jogos





# Busca Local



# Busca Local



- O caminho da solução não é importante.
- A solução é importante.

# Métodos de Busca

## **Busca Heurística: Subida da encosta**



# Busca Local

## **Subida da encosta**

- Na busca gulosa, a cada momento escolhemos o próximo estado que aparenta levar a uma solução maximizando (ou minimizando) algum valor.
- No entanto, é mantida uma lista de estados não visitados que podem, em determinado passo, se tornarem mais interessantes heuristicamente.
- Além disso, é preciso guardar os caminhos até cada nó visitado.
- Na subida da encosta, apenas o estado atual é analisado. Não é mantida uma lista de nós abertos, tampouco os caminhos até cada nó visitado.

# Busca Local

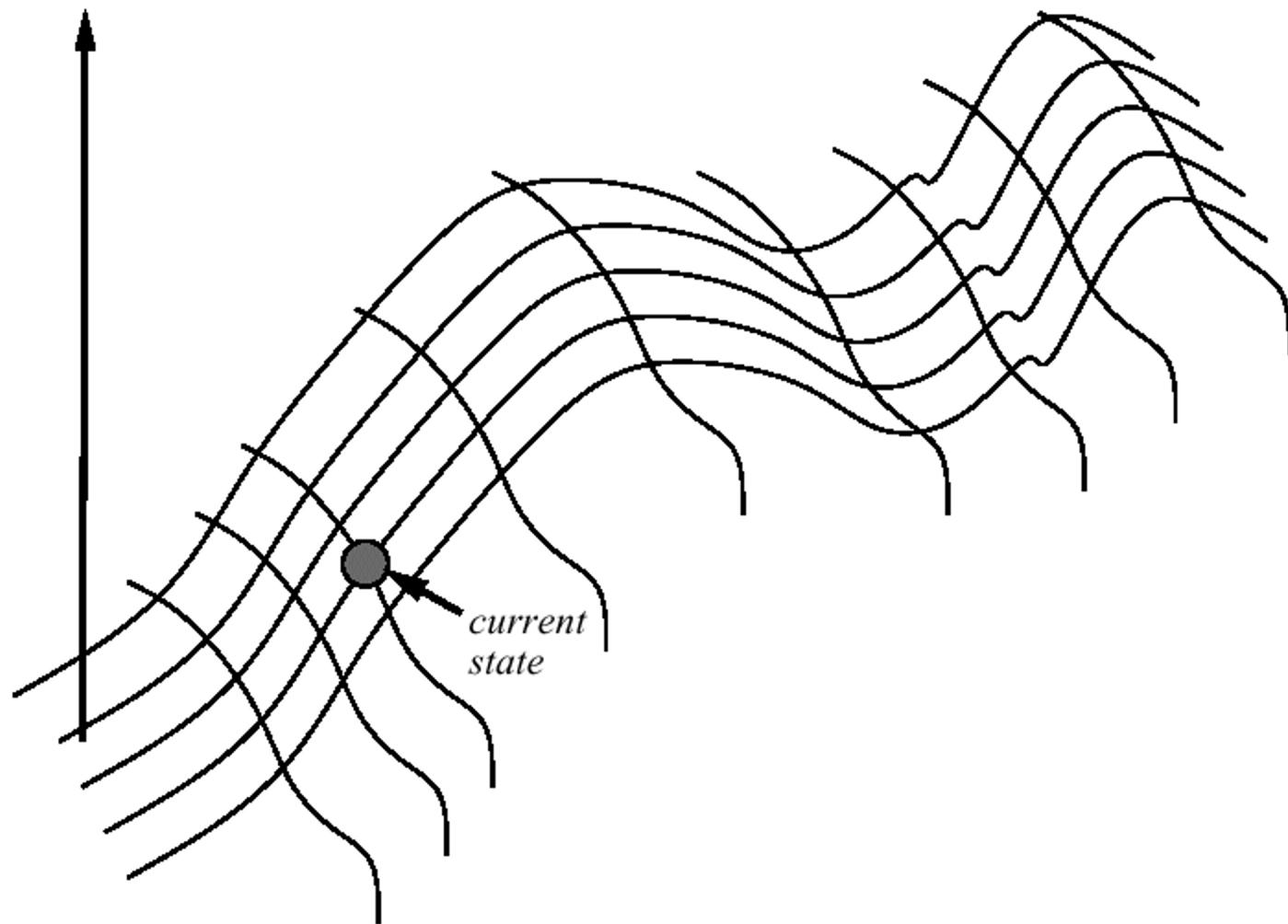
## Subida da encosta

- Formalmente, o algoritmo da subida da encosta escolhe, como próximo passo, o nó que parece estar mais próximo do objetivo.
- O nome do algoritmo deriva da analogia com o andarilho que está perdido no escuro em uma montanha e deseja chegar até o topo da montanha. Mesmo no escuro, o andarilho sabe que cada passo para cima é um passo na direção certa.
- No caso da seleção de uma rota para viagem, podemos incorporar a seguinte heurística: Escolha a cidade que seja a mais distante possível da posição corrente

# Busca Local

## Subida da encosta

*evaluation*



# Busca Local

## Subida da encosta

**function** HILL-CLIMBING(*problem*) **returns** a solution state

**inputs:** *problem*, a problem

**static:** *current*, a node

*next*, a node

*current*  $\leftarrow$  MAKE-NODE(INITIAL-STATE[*problem*])

**loop do**

*next*  $\leftarrow$  a highest-valued successor of *current*

**if** VALUE[next] < VALUE[current] **then return** *current*

*current*  $\leftarrow$  *next*

**end**

# Busca Local

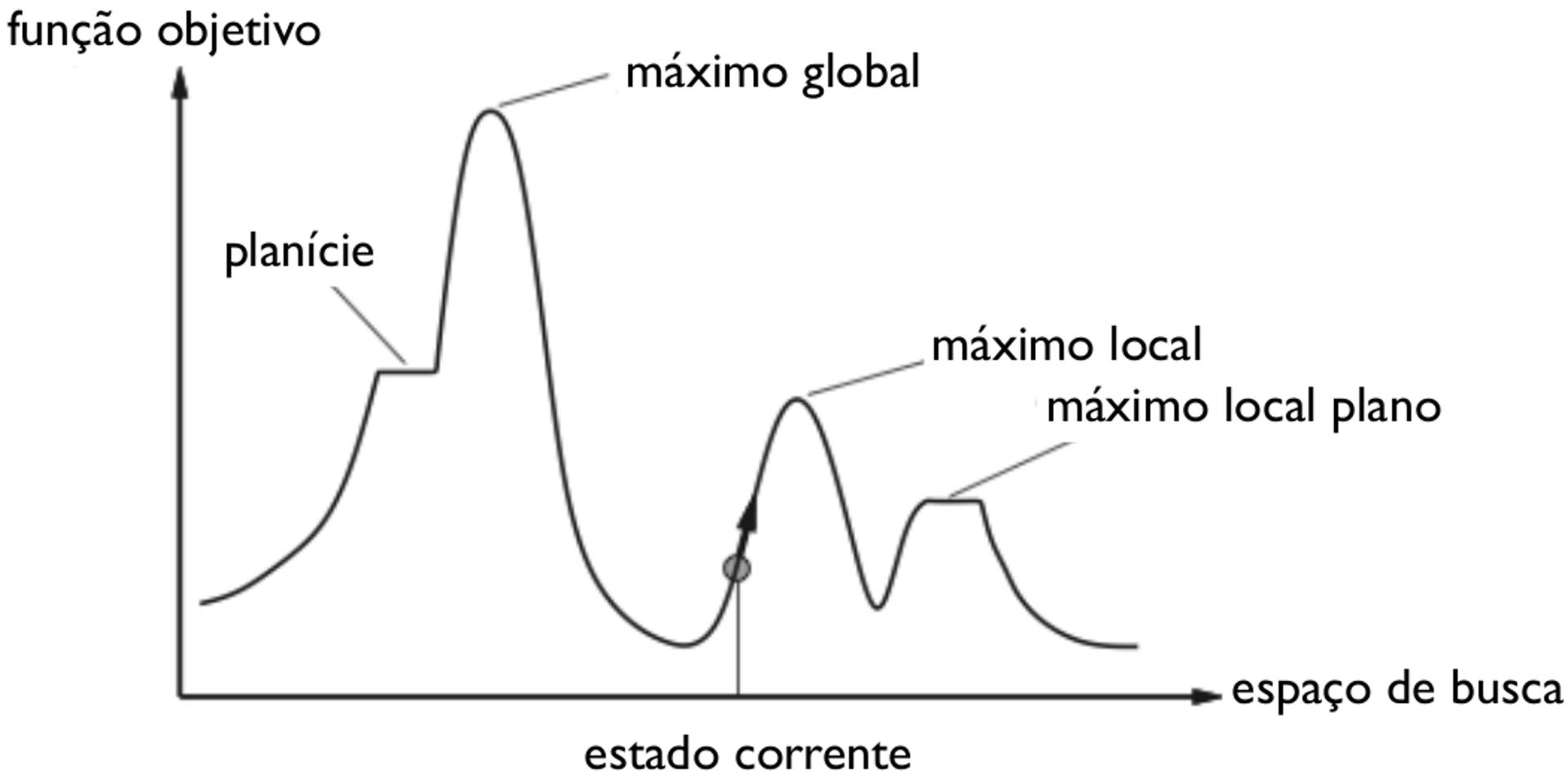


## **Subida da encosta (problemas)**

- Falsas montanhas.
- Planaltos — no qual todos os passos possíveis, a partir do nó corrente, parecem igualmente bons (ou ruins). Nesse caso, a subida da encosta não é melhor do que a busca em profundidade.
- Crista — ocorre quando existe uma área no espaço de busca mais alta do que as áreas circunjacentes, mas que não pode ser atravessada por movimentos singulares numa direção qualquer.

# Busca Local

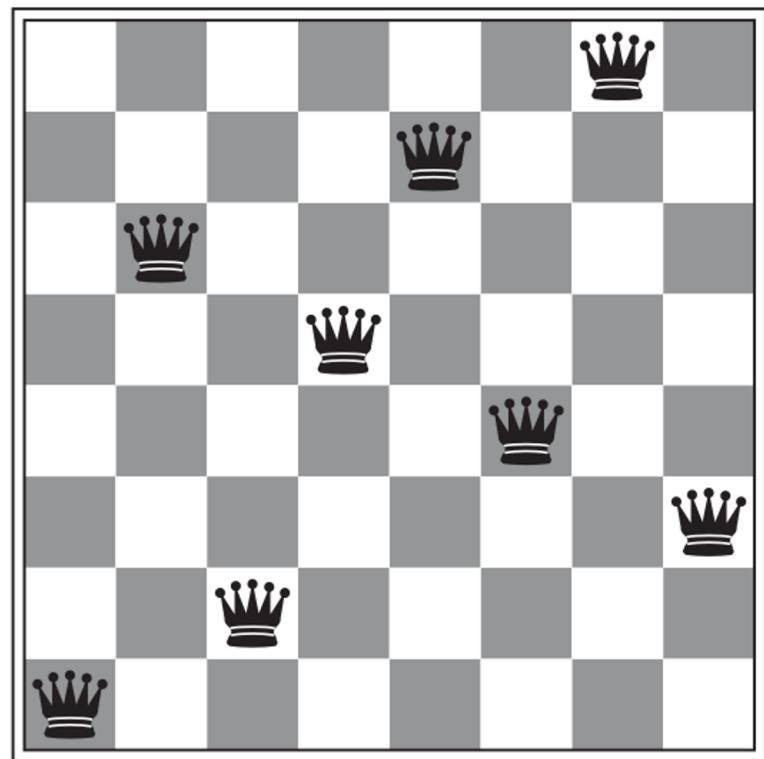
## Subida da encosta (problemas)



# Busca Local

## Subida da encosta (problemas)

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



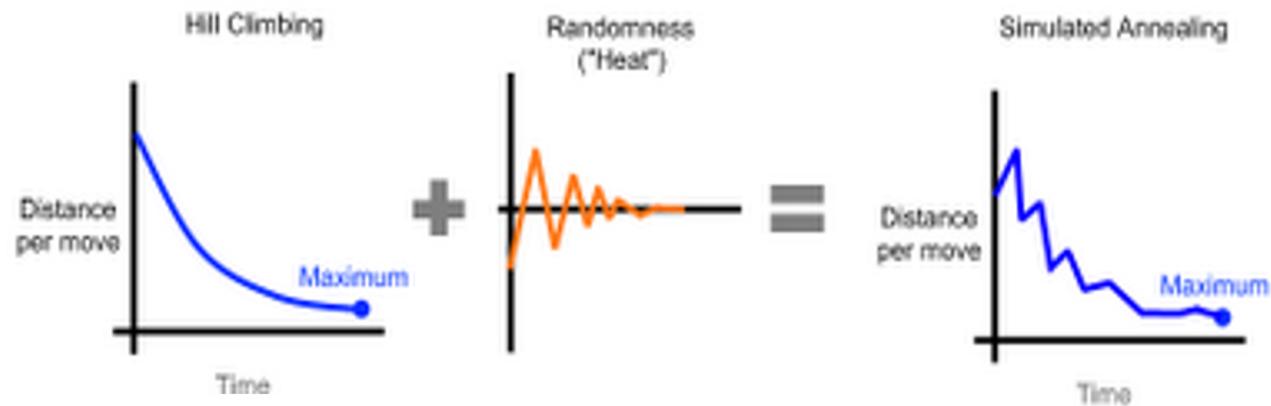
# Busca Local



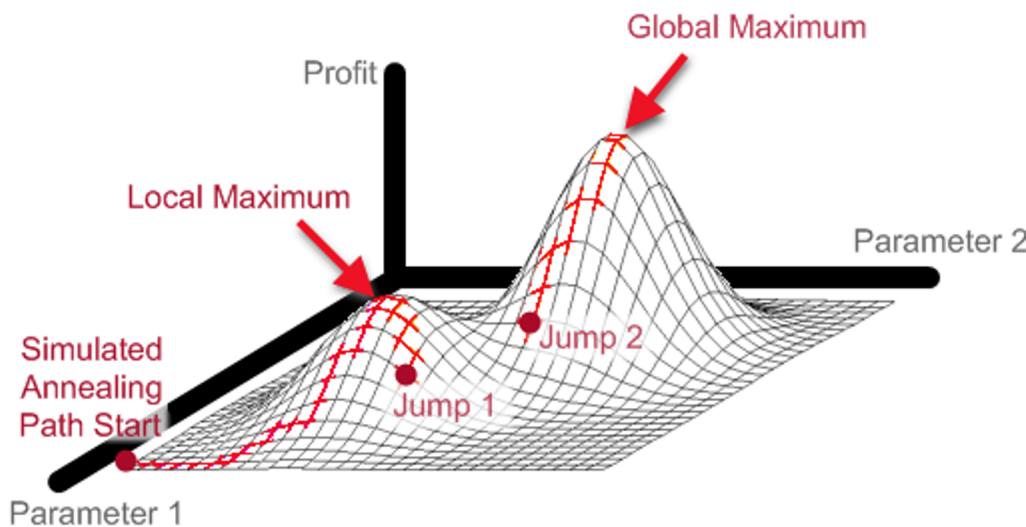
**Têmpera simulada**

# Busca Local

## Têmpera simulada



Simulated Annealing can escape local minima with chaotic jumps



# Busca Local

## **Busca Heurística: Têmpera simulada**

- Ao invés de ficar preso em um máximo local podemos permitir que a pesquisa faça alguns movimentos para baixo para escapar dos máximos locais.
- Esta é a idéia da têmpera simulada. Ao invés de sempre pegar o melhor caminho o algoritmo escolhe um caminho aleatório.
- Se o caminho melhorar a situação ele é escolhido. Caso contrário, existe probabilidade do caminho ser escolhido.
- A probabilidade de escolher um caminho ruim tende a decrescer com o tempo.

# Busca Local

## **Busca Heurística: Têmpera simulada**

- O nome do método vem da técnica de se esfriar um material gradualmente de modo a ter um melhor arranjo das moléculas.
- Esta técnica tem sido usada com sucesso em vários problemas, como projeto de circuitos VLSI e otimização de linhas de produção.

# Busca Local

## Busca Heurística: Têmpera simulada

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

**inputs:** *problem*, a problem

*schedule*, a mapping from time to “temperature”

**static:** *current*, a node

*next*, a node

*T*, a “temperature” controlling the probability of downward steps

*current*  $\leftarrow$  MAKE-NODE(INITIAL-STATE[*problem*])

**for** *t*  $\leftarrow$  1 **to**  $\infty$  **do**

*T*  $\leftarrow$  *schedule*[*t*]

**if** *T*=0 **then return** *current*

*next*  $\leftarrow$  a randomly selected successor of *current*

$\Delta E \leftarrow$  VALUE[*next*] – VALUE[*current*]

**if**  $\Delta E > 0$  **then** *current*  $\leftarrow$  *next*

**else** *current*  $\leftarrow$  *next* only with probability  $e^{\Delta E/T}$



**FIM**