

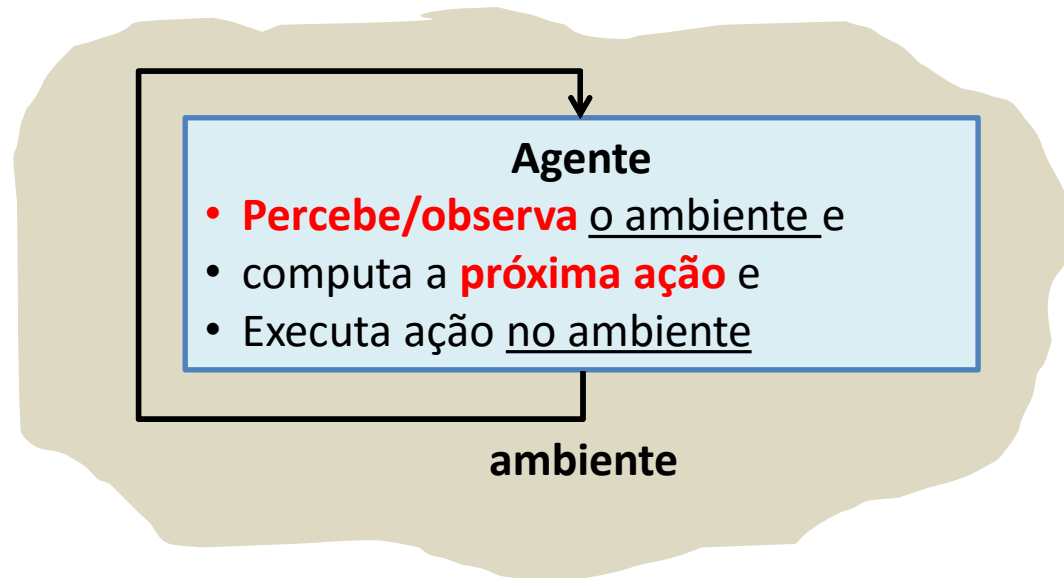
# BUSCA ON-LINE

busca por soluções de forma dinâmica  
(pg. 123 R&N, 2ed.)

# Off-line vs. On-line

Na **busca off-line** o agente primeiro computa toda a sequência de ações para então executá-la no ambiente.

Na **busca on-line**, o agente intercala computação e ação:



Malha-fechada: o agente utiliza as percepções para saber em qual estado se encontra <sub>2</sub>

# Aplicações

aplicável em **ambientes dinâmicos e semi-dinâmicos**

(nestes últimos, há penalidade quando se toma muito tempo para deliberar)

interessante para **ambientes não-determinísticos** pois o agente pode focar em contingências que realmente acontecem ao invés de considerar todas que *podem* acontecer (como no caso da busca em grafo and-or)

aplicáveis **em problemas de exploração**:

- agente não conhece a topografia do ambiente (ou um mapa do ambiente)
- Agente não conhece o resultado de suas ações no ambiente

# Expansão de nós vizinhos

A\* pode expandir um nó em uma região do espaço de estados e, em seguida, em outra região completamente diferente.

Já um algoritmo online somente consegue descobrir sucessores de **um nó no qual ele está fisicamente** – *um robô não pode desaparecer de um lugar e aparecer em outro* - então, a ideia é expandir nós próximos para evitar retornos físicos.



A busca em profundidade expande sempre um nó filho do anterior (exceto quando faz *backtracking*)

BUSCA ON-LINE

# **BUSCA EM PROFUNDIDADE ON-LINE (ON-LINE DFS)**

# Busca em Profundidade on-line

Funções principais do algoritmo:

## **ACTIONS(s)**

retorna as ações possíveis de serem executadas no estado  $s$ ; em princípio, todas as ações

## **RESULT(s, a)**

armazena a função sucessora aprendida durante a exploração e retorna o estado sucessor de  $s$  pela execução de  $a$  (se conhecido)

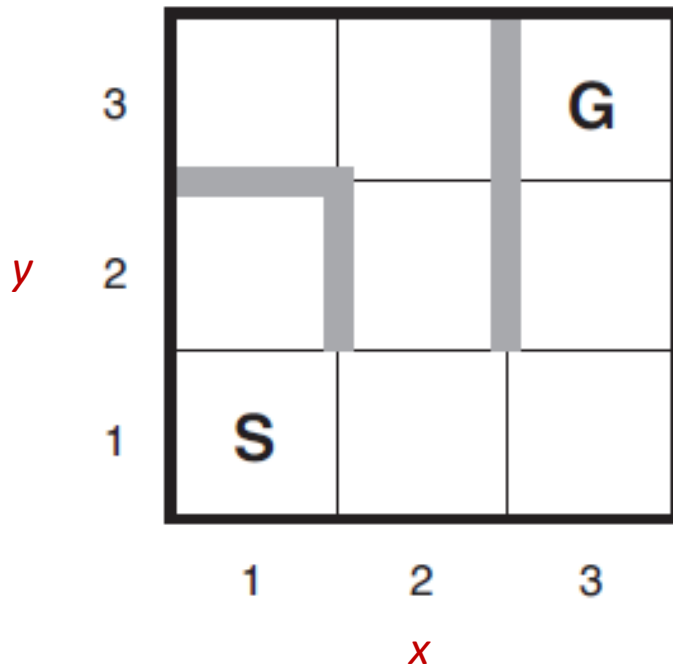
## **UNTRIED(s)**

Lista de ações ainda não tentadas no estado  $s$

## **UNBACKTRACKED(s)**

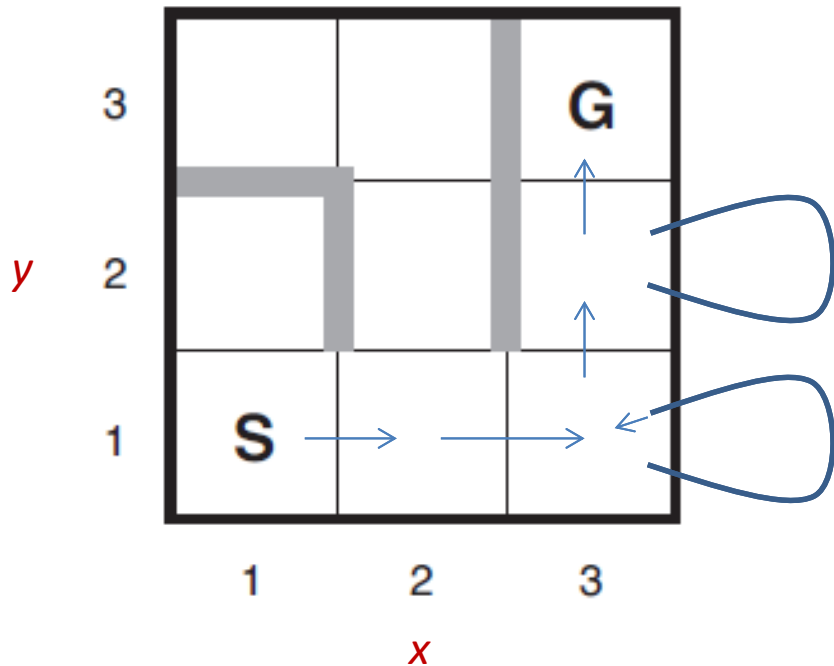
Lista de ações ainda não desfeitas no estado  $s$

# Exemplo 1: on-line DFS

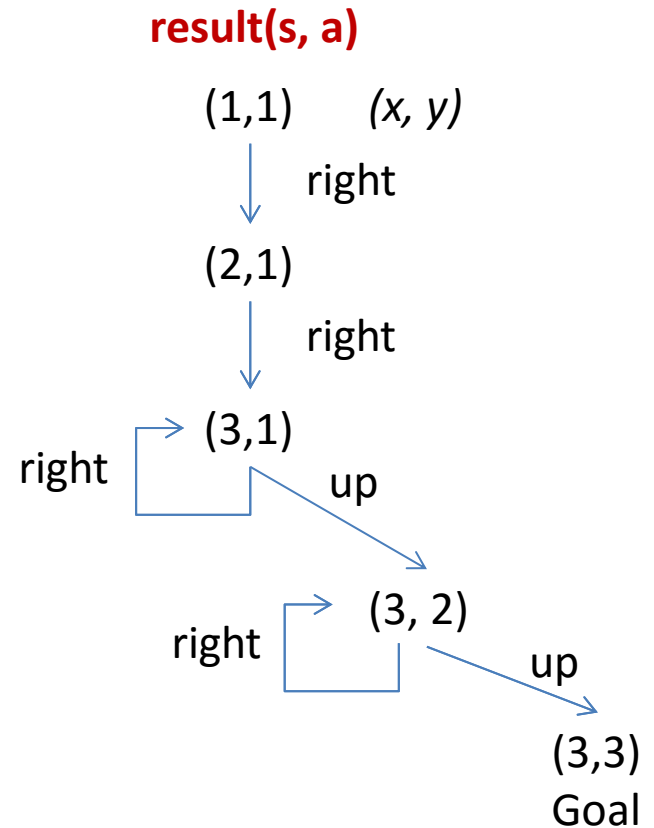


Robô inicia em S e deve atingir G  
Não conhece a topografia do ambiente  
Não conhece o efeito de suas ações  
Tem um sensor que permite goal-test  
**Ações = {Right, Up, Down, Left}**

# Exemplo 1: on-line DFS



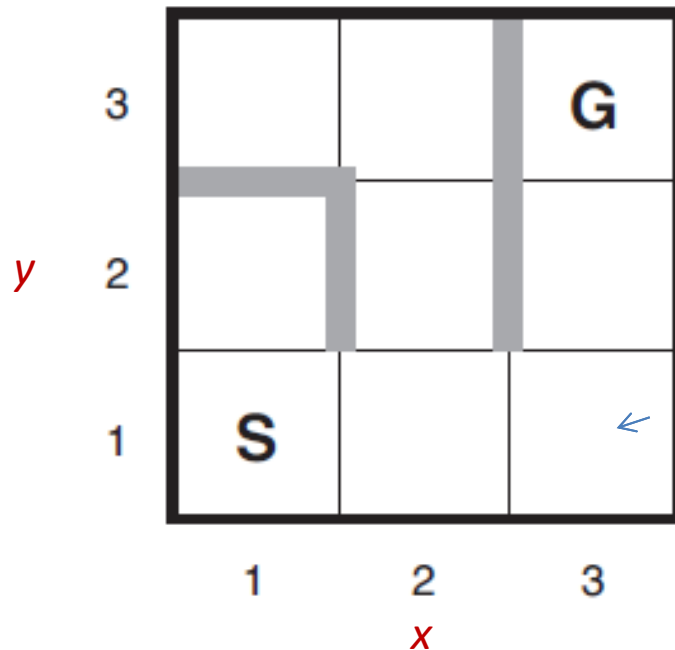
Ações = Right > Up > Down > Left



Função sucessora aprendida durante a execução



# Exercício 1a: on-line DFS

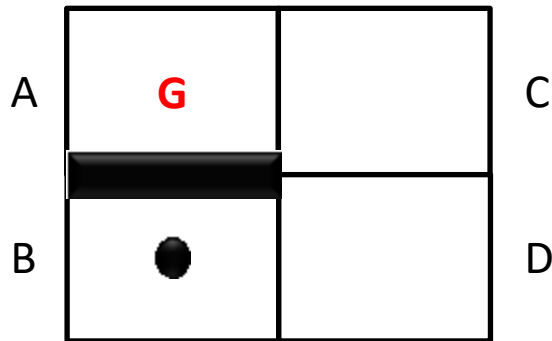


Qual seria a função  $\text{RESULT}(S,a)$  após a execução para esta ordenação de ações?

Qual seria a função  $\text{UNTRIED}(s)$  após a execução para esta ordenação de ações?

**Ações = UP > RIGHT > Down > Left**

## Exemplo 2: on-line DFS com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = U>D>R>L

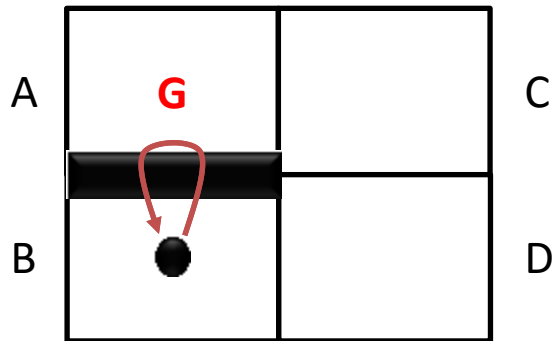
C = U>D>R>L

D = U>D>R>L

Result(s, a) -> s'

	U	D	R	L
A				
B				
C				
D				

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = D>R>L

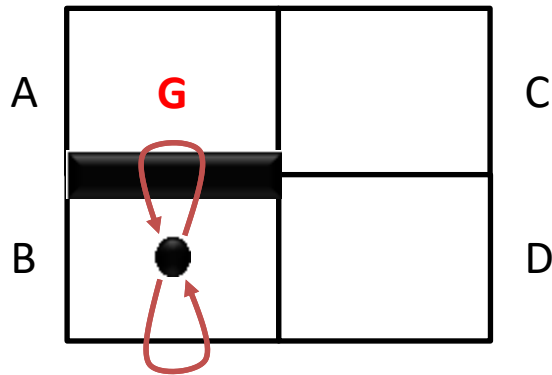
C = U>D>R>L

D = U>D>R>L

Result( $s, a$ ) ->  $s'$

	U	D	R	L
estados	A			
	B	<b>B</b>		
	C			
	D			

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = R>L

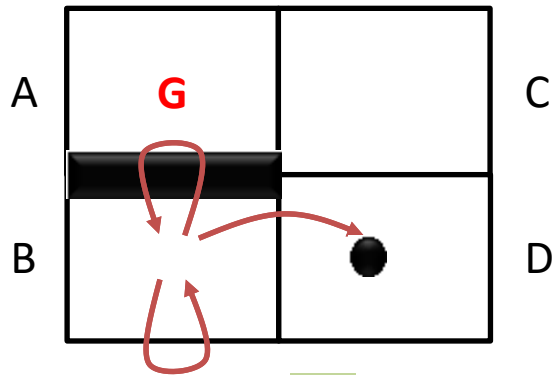
C = U>D>R>L

D = U>D>R>L

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>		
C				
D				

## Exemplo 2 on-line DFS: com backtracking



B

**Unbacktracked** associado  
ao estado **em(D)**

Estados  $S = \{A, B, C, D\}$

$$S_0 = B, s_g = A$$

Ações = {Up, Down, Right, Left}

## Untried

A = U>D>R>L

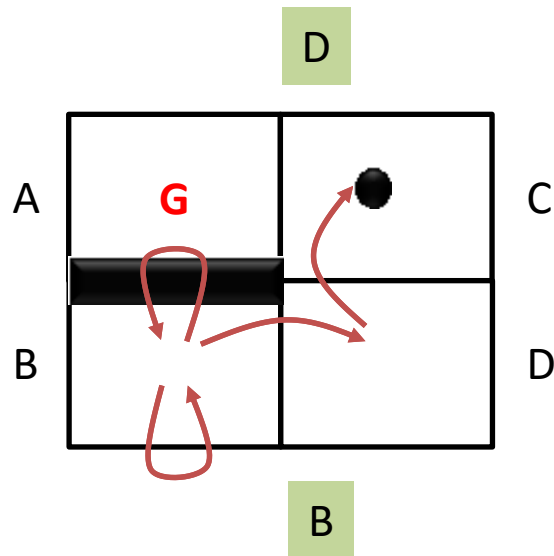
$$B = \quad > L$$
$$C = U > D > R > L$$

D = U>D>R>L

**Result(s, a)  $\rightarrow$  s'**

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	
C				
D				

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = >L

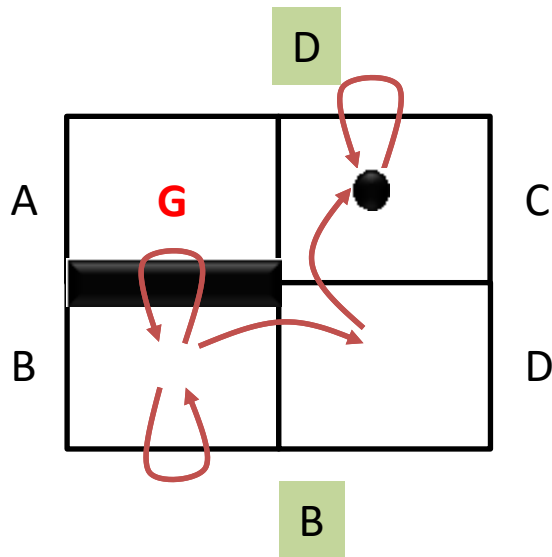
C = U>D>R>L

D = >D>R>L

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	
C				
D	<b>C</b>			

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = >L

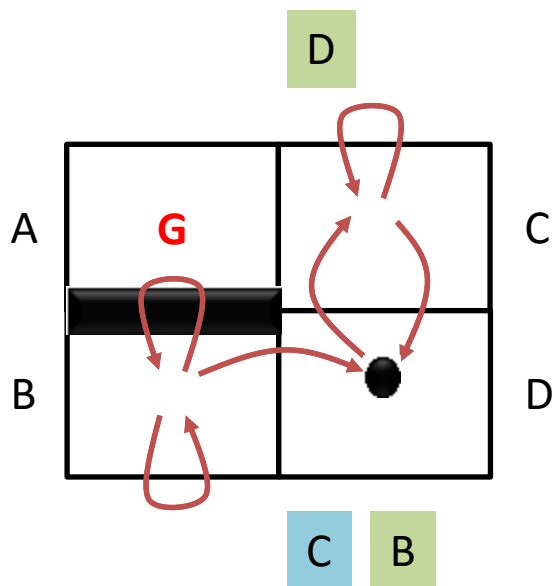
C = >D>R>L

D = >D>R>L

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	
C	<b>C</b>			
D	<b>C</b>			

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = >L

C = >R>L

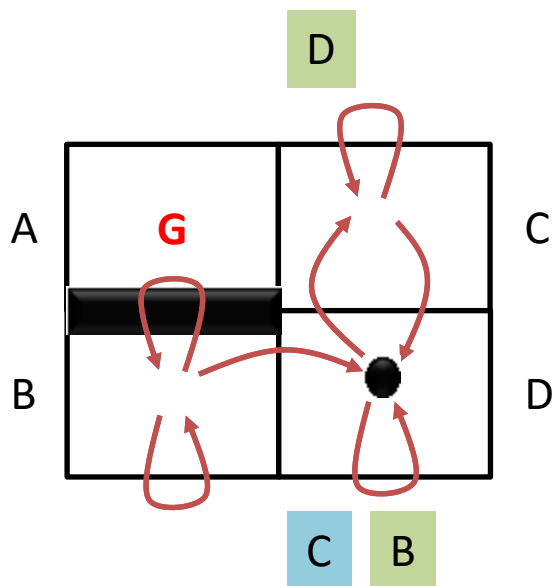
D = >D>R>L

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	
C	<b>C</b>	<b>D</b>		
D	<b>C</b>			



## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = >L

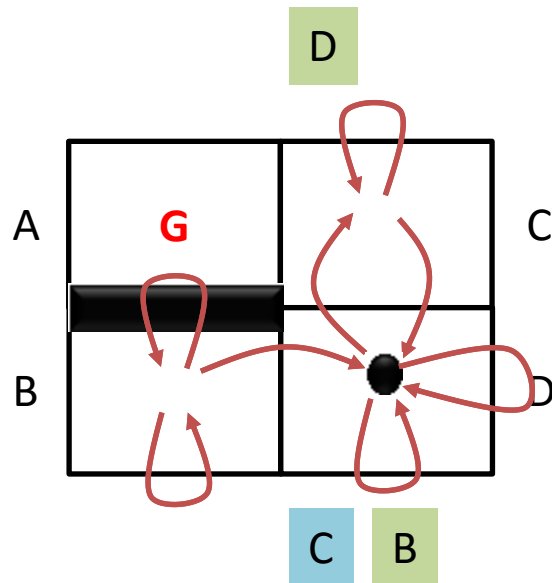
C = >R>L

D = >R>L

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>		

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = >L

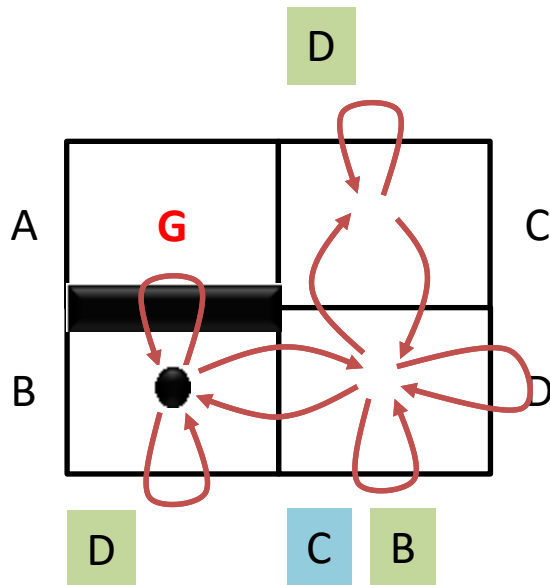
C = >R>L

D = >L

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>	<b>D</b>	

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B = >L

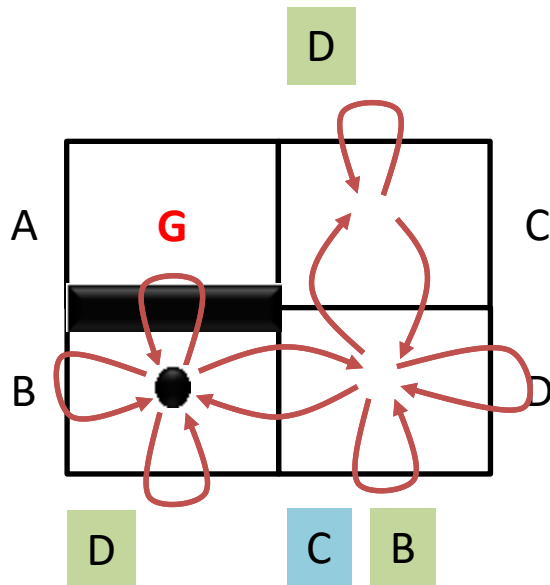
C = >R>L

D =

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B =

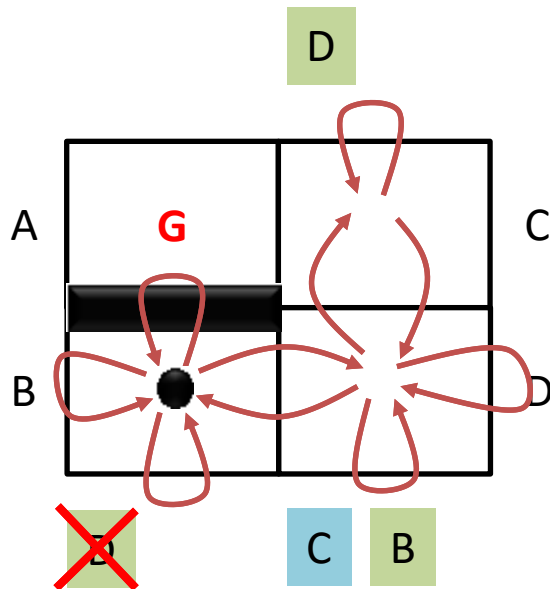
C = >R>L

D =

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	<b>B</b>
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B =

C = >R>L

D =

Result(s, a) -> s'

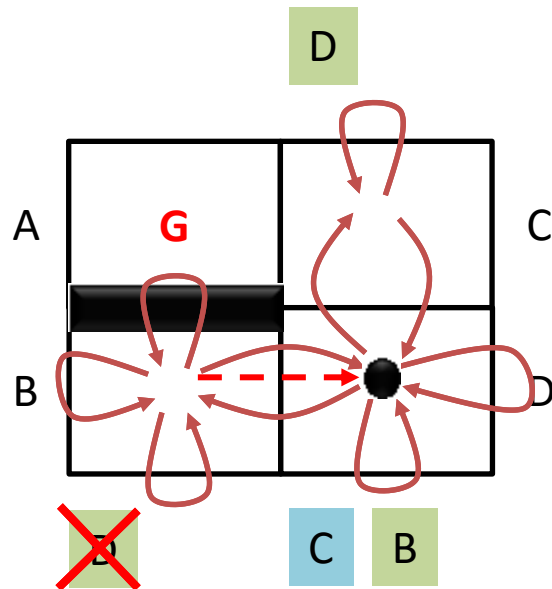
	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	<b>B</b>
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

Não há mais ações a testar no estado **em(B)**.

**Pop (unbacktracked (B)) → vai para estado D**

(i.e. executa a ação que leva o agente ao estado D  
= right)

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B =

C = >R>L

D =

Result(s, a) -> s'

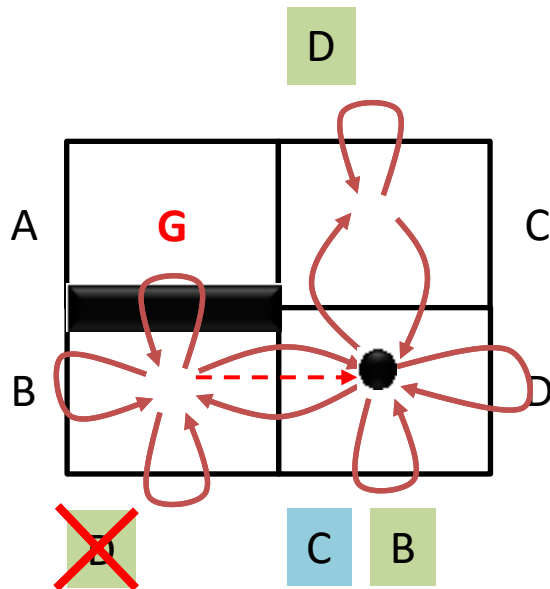
	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	<b>B</b>
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

Não há mais ações a testar no estado **em(B)**.

**Pop (unbacktracked (B)) → vai para estado D**

(i.e. executa a ação que leva o agente ao estado D  
= right)

## Exemplo 2 on-line DFS: com backtracking



Não há mais ações a testar no estado **em(D)**.

Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B =

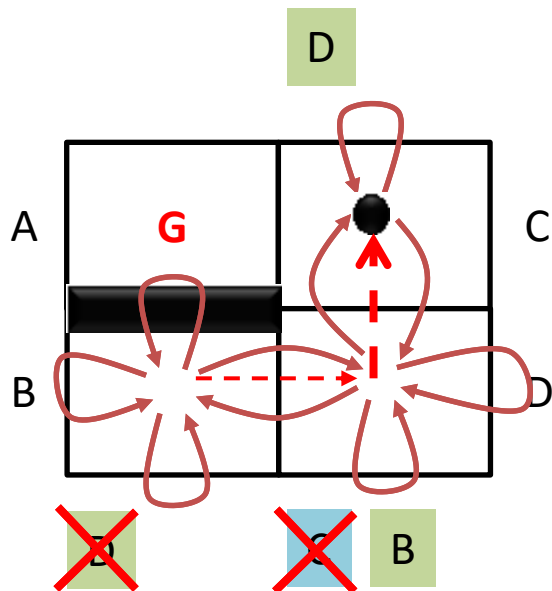
C = >R>L

D =

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	<b>B</b>
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B =

C = >R>L

D =

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	<b>B</b>
C	<b>C</b>	<b>D</b>		
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

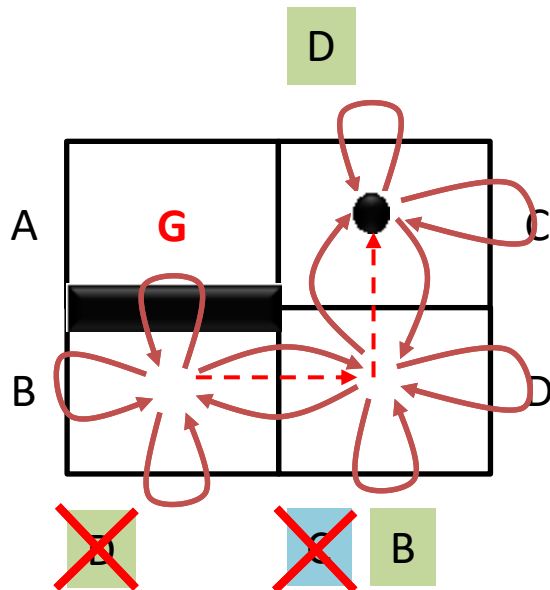
Não há mais ações a testar no estado **em(D)**.

**Pop(unbacktracked(D)) → VAI para estado C**

(i.e. executa a ação que leva o agente ao estado C = UP)



## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B =

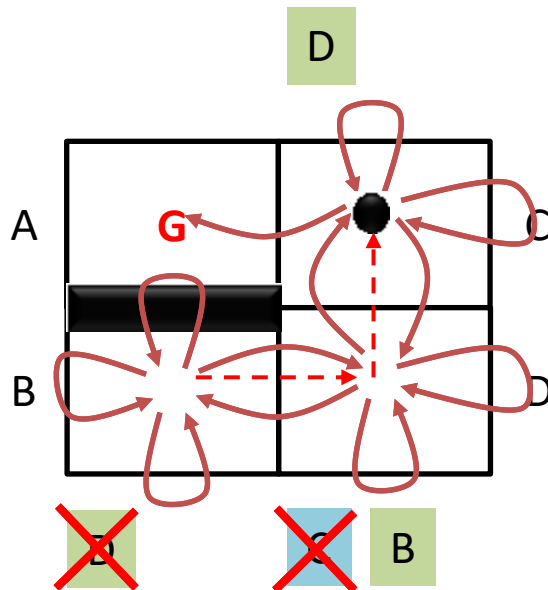
C = >L

D =

Result(s, a) -> s'

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	<b>B</b>
C	<b>C</b>	<b>D</b>	<b>C</b>	
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

## Exemplo 2 on-line DFS: com backtracking



Estados  $S = \{A, B, C, D\}$

$S_0 = B, s_g = A$

Ações = {Up, Down, Right, Left}

Untried

A = U>D>R>L

B =

C =

D =

Result(s, a)  $\rightarrow s'$

	U	D	R	L
A				
B	<b>B</b>	<b>B</b>	<b>D</b>	<b>B</b>
C	<b>C</b>	<b>D</b>	<b>C</b>	<b>A</b>
D	<b>C</b>	<b>D</b>	<b>D</b>	<b>B</b>

# Busca em Profundidade (DFS) online

```
function ONLINE-DFS-AGENT( $s'$ ) returns an action
inputs:  $s'$ , a percept that identifies the current state
persistent: result, a table indexed by state and action, initially empty
               untried, a table that lists, for each state, the actions not yet tried
               unbacktracked, a table that lists, for each state, the backtracks not yet tried
                $s$ ,  $a$ , the previous state and action, initially null

if GOAL-TEST( $s'$ ) then return stop
if  $s'$  is a new state (not in untried) then untried[ $s'$ ]  $\leftarrow$  ACTIONS( $s'$ )
if  $s$  is not null then
    result[ $s$ ,  $a$ ]  $\leftarrow$   $s'$ 
    add  $s$  to the front of unbacktracked[ $s'$ ]
if untried[ $s'$ ] is empty then
    if unbacktracked[ $s'$ ] is empty then return stop
    else  $a \leftarrow$  an action  $b$  such that result[ $s'$ ,  $b$ ] = POP(unbacktracked[ $s'$ ])
else  $a \leftarrow$  POP(untried[ $s'$ ])
 $s \leftarrow s'$ 
return  $a$ 
```

Função sucessora aprendida durante execução

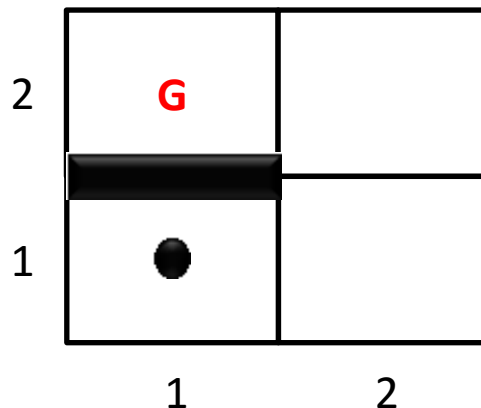
If I've already seen the result of this [ $s$ ,  $a$ ] then don't put it back on the **unbacktracked** list otherwise you can keep oscillating between the same states endlessly.

# Online DFS

O online DFS funciona bem para ambientes estáticos onde o agente não conhece o efeito de suas ações no ambiente.

Este algoritmo só funciona em espaços de estados onde as **ações são reversíveis**.

**Irreversibilidade das ações** é provocada pela própria incapacidade do agente (ex. agente que somente executa UP e RIGHT) ou por ambiente dinâmico (ex. paredes que se fecham atrás do agente)



BUSCA ON-LINE

**LRTA\***

# LRTA\* ou ATRA\*

Learning real-time A\* (Korf, 1990)  
chamado de ATRA\* em português

Recorda-se que lidamos com situações onde é mais importante agir razoavelmente **em tempo hábil** (intercalando raciocínio e ação) do que minimizar o custo de execução tendo que passar muito tempo para encontrar uma solução (planejando).

# LRTA\*

Learning real-time A\* (Korf, 1990)

**ideia:** equilibrar o tempo de **planejamento** e **execução**

Exemplos:

**personagem de games:** inverossímil se ficar sentado numa pedra calculando o caminho mínimo e depois executá-lo rapidamente.

**carro automático:** calcular a trajetória ótima em um curva – antes de acabar o cálculo terá batido!

*Korf, 1990. Real-time heuristic search. Artificial Intelligence, 42(3), 189-212*

# LRTA\* Learning real-time A\* (Korf, 1990)

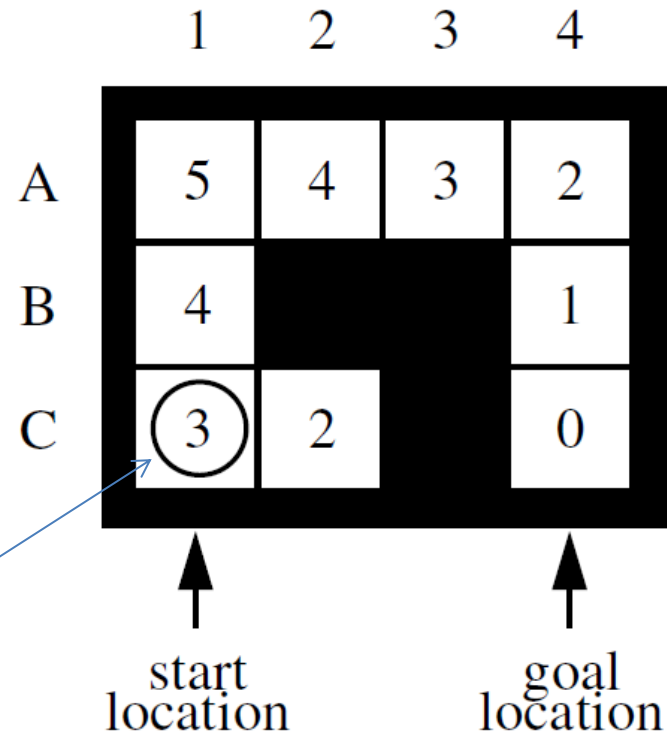
**Cenário:** problema de navegação **em um terreno conhecido**; o agente sabe onde está inicialmente; ambiente semi-dinâmico = penalidade por demora na deliberação.

**Ações do robô:**

Norte, Leste, Sul ou Oeste

**Custo das ações:** 1

**Inicialização da  $h(n)$ :** distância de Manhattan (e.g.  $f(C1) = 3$ )

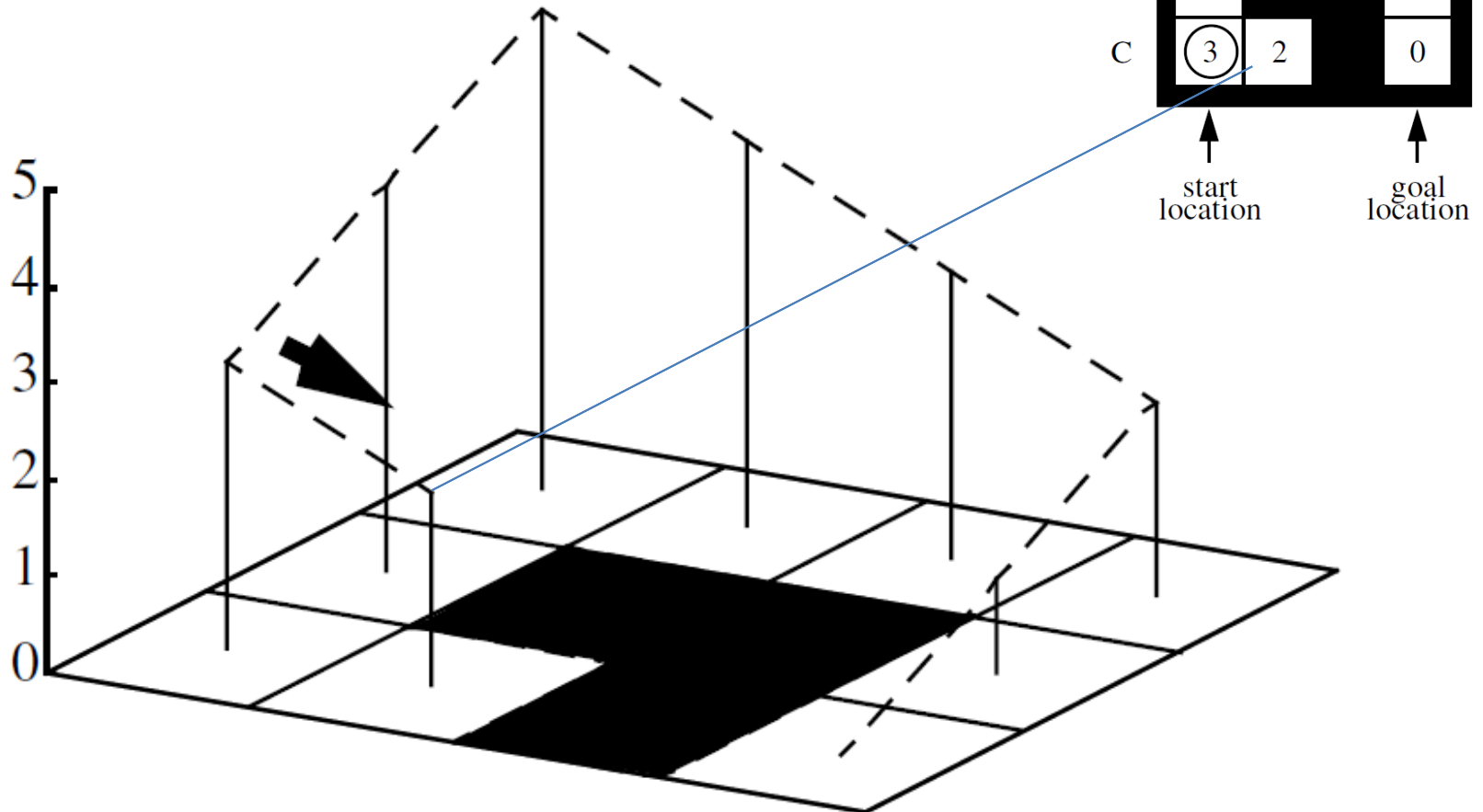


*(Koenig, 2001. Agent Search, AI Magazine, 109-132)*



# LRTA\*

$f(n)$  inicial para todos os estados

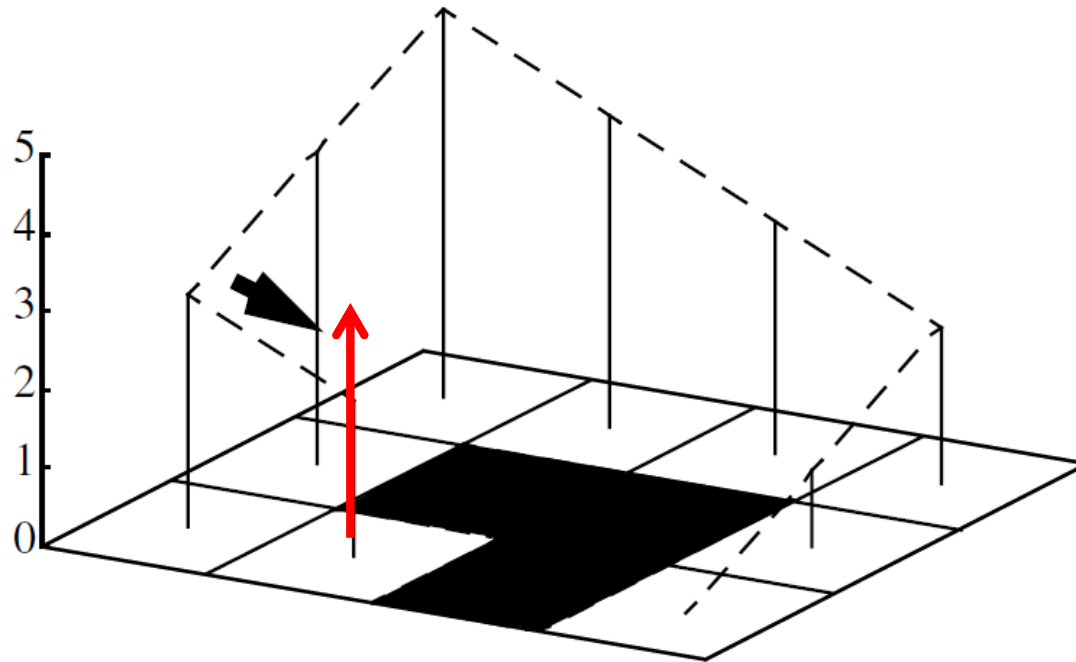


Numa busca gulosa, o robô fica preso em C1 e C2

(Koenig, 2001)

# LRTA\*

LRTA\* escapa dos mínimos locais aumentando o valor de  $f(n)$  'enchendo' os mínimos locais na superfície de valores



# LRTA\*: intuição

$n$  = estado corrente

$n'$  = representa um estado vizinho qualquer

Para cada ação  $\langle a \rangle$  no estado corrente  $\langle n \rangle$ ,  
calcule para todo vizinho de  $\langle n \rangle$  chamado  $n'$ :

$$f(n') = c(n, a, n') + h(n')$$

**Atualizar** custo estimado nó  $n$ :  $h(n) \leftarrow \min_{n'} f(n')$

**Escolher a** ação  $\langle a \rangle$  que leva ao vizinho  $n'$  de menor  $f(n')$ ; em caso de empate, escolha randomicamente.

# LRTA\*: EXEMPLO

**Ações:** NORTE, SUL, LESTE, OESTE

**Custo das ações é uniforme = 1**

**Amarelo:** estado atual = n

**Vermelho:** estado objetivo

**Azul:** avaliação dos nós vizinhos = n'

valores nas casas azuis são temporários

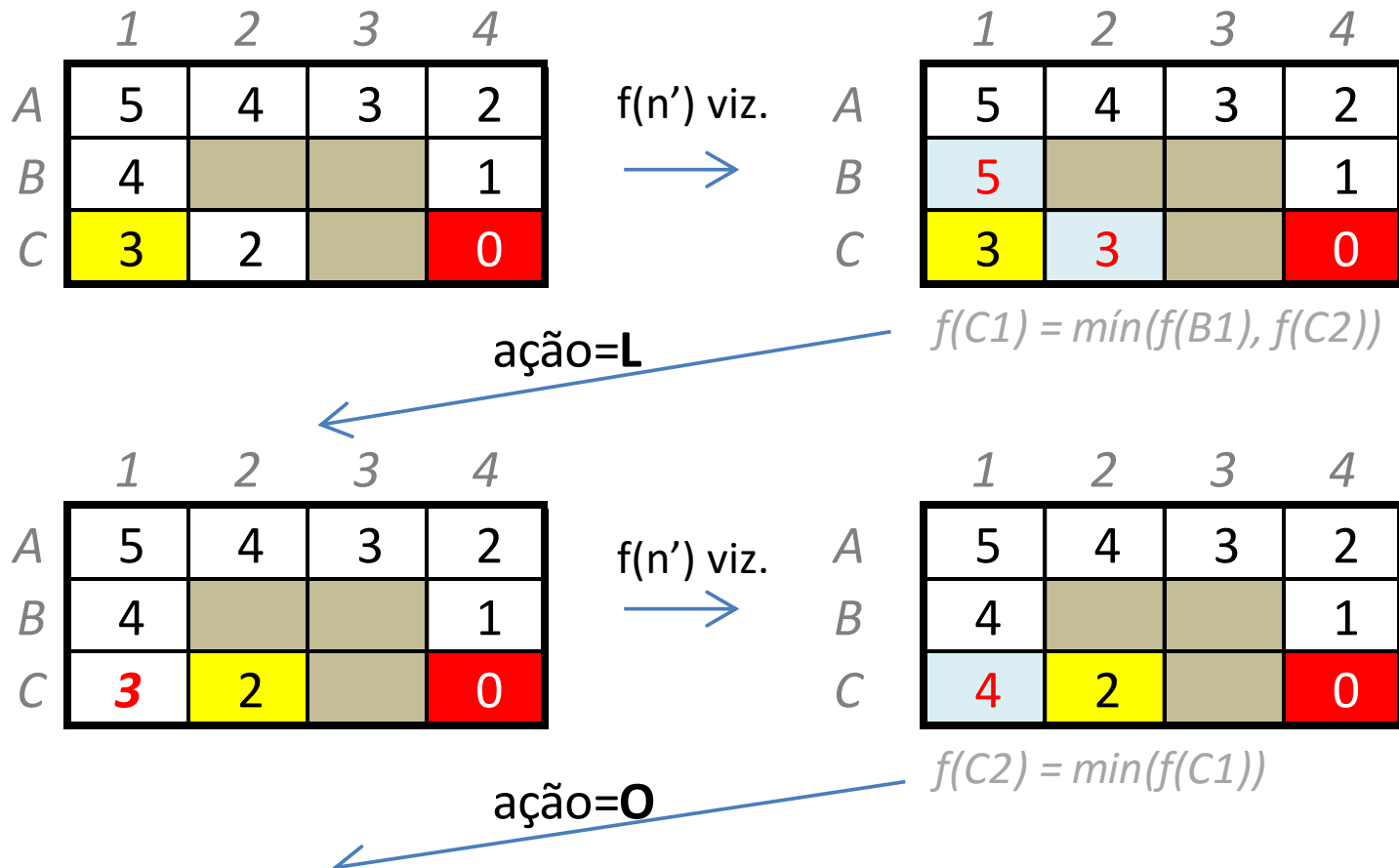
$$f(n') = c(n, a, n') + h(n')$$

**Empates decididos randomicamente**

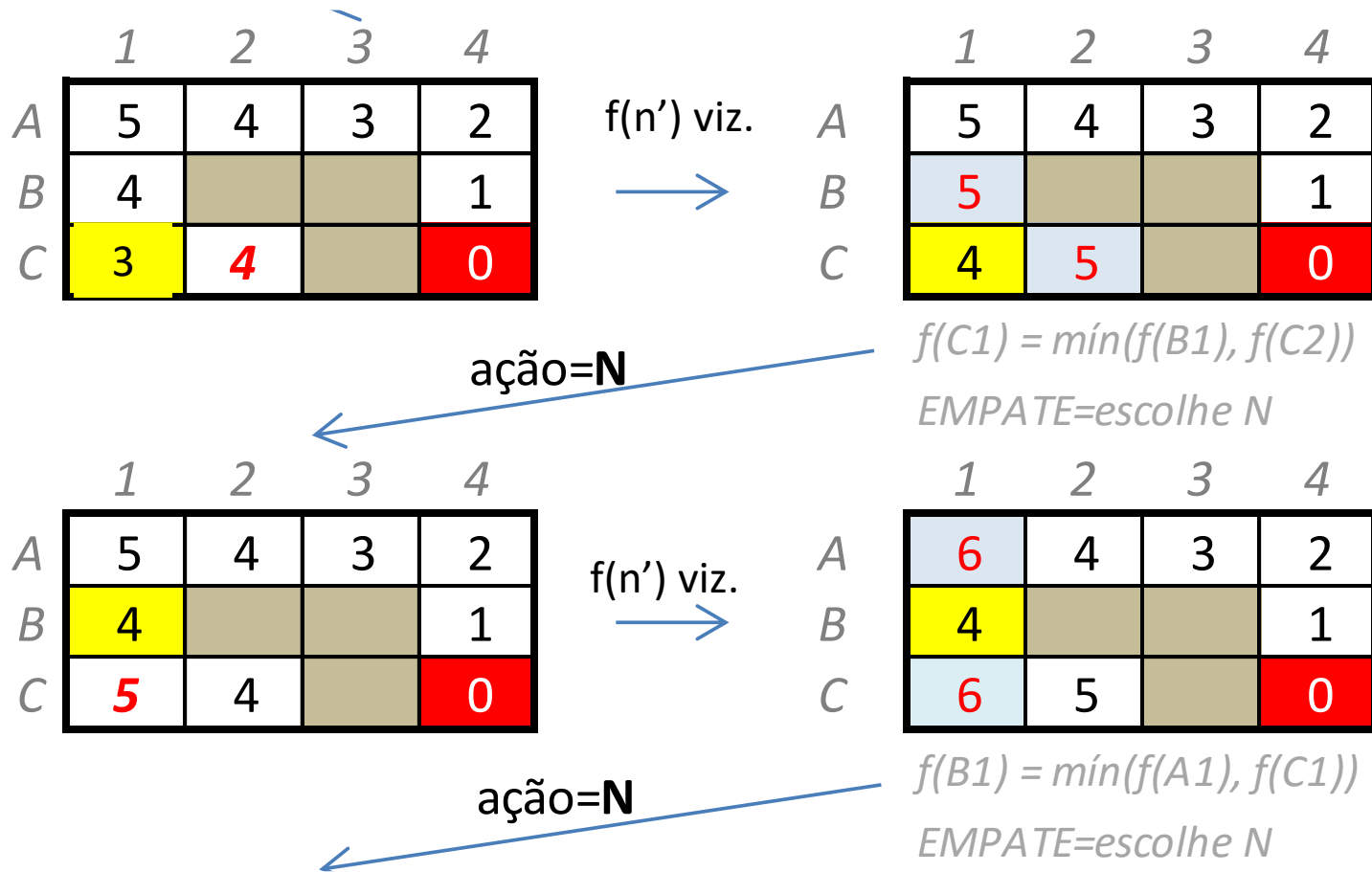
Estados são inicializados com o valor de  $h(n)$  (valores na matriz)

Ao longo da execução,  $h(n)$  é atualizado.

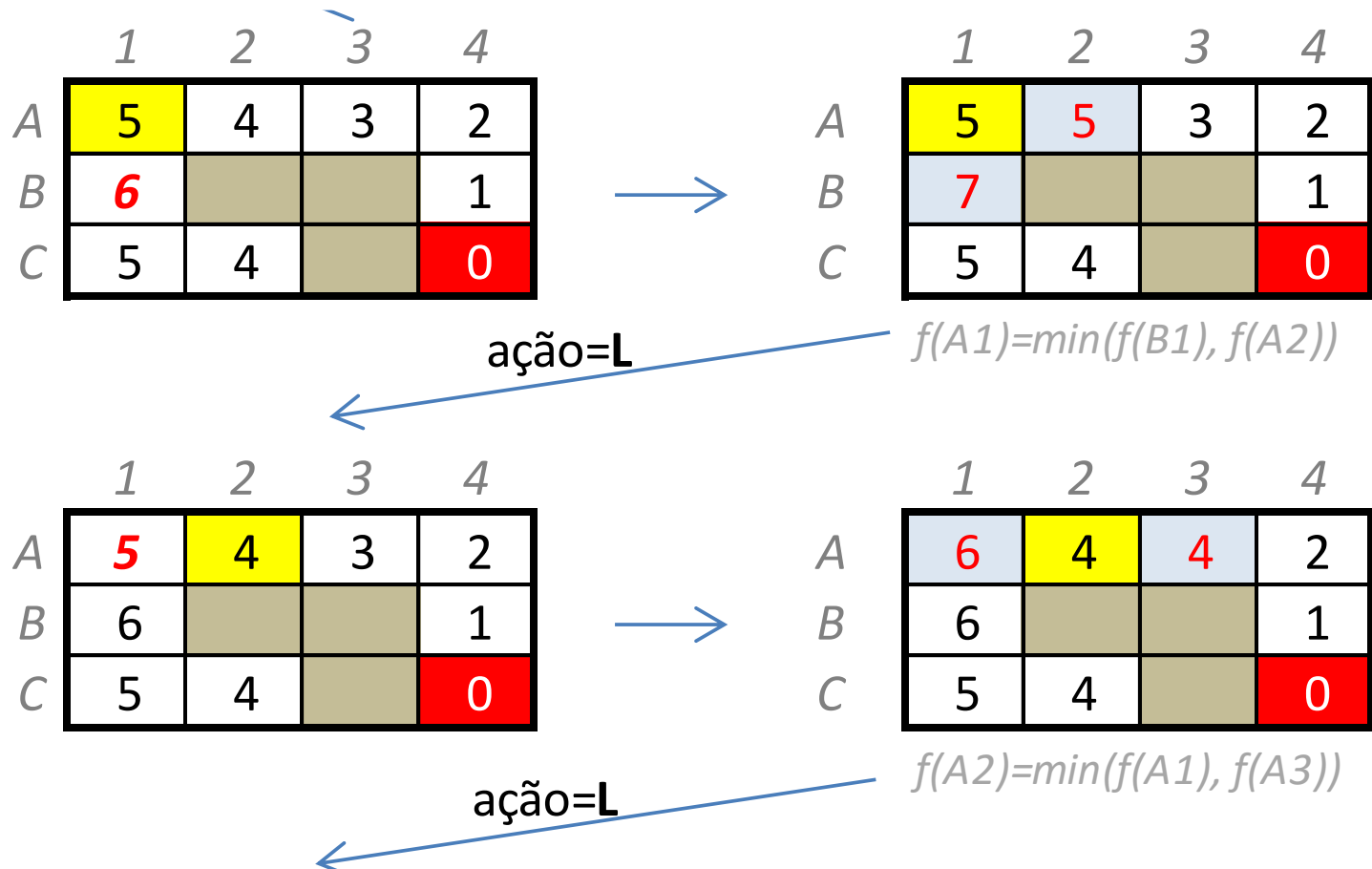
# LRTA\*: EXECUÇÃO 1



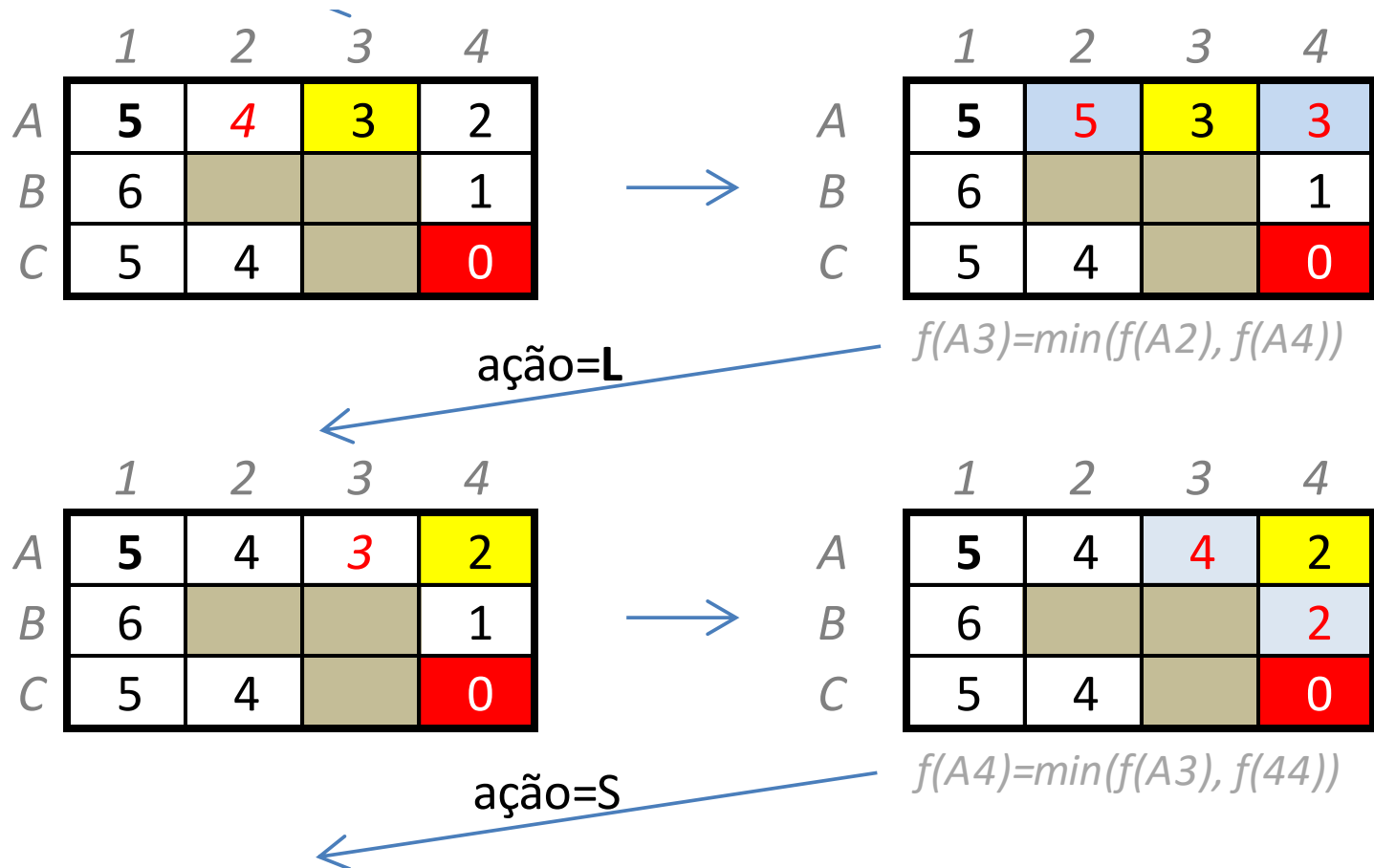
# LRTA\*: EXECUÇÃO 1



# LRTA\*: EXECUÇÃO 1



# LRTA\*: EXECUÇÃO 1





# LRTA\*: EXECUÇÃO 1

	1	2	3	4
A	5	4	3	2
B	6			1
C	5	4		0



	1	2	3	4
A	5	4	3	3
B	6			1
C	5	4		1

$$f(B4) = \min(f(A4), f(C4))$$

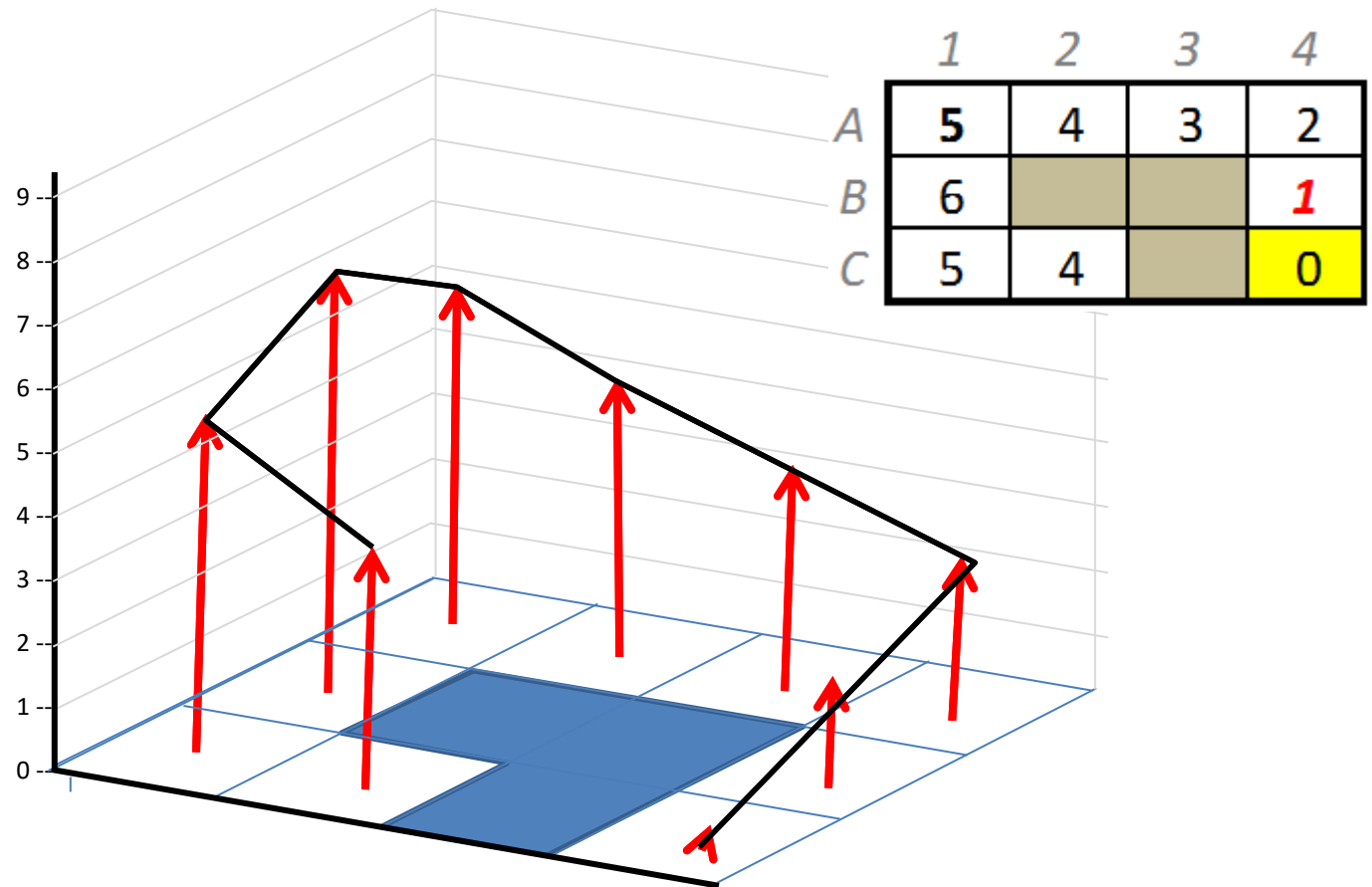
ação=S



	1	2	3	4
A	5	4	3	2
B	6			1
C	5	4		0

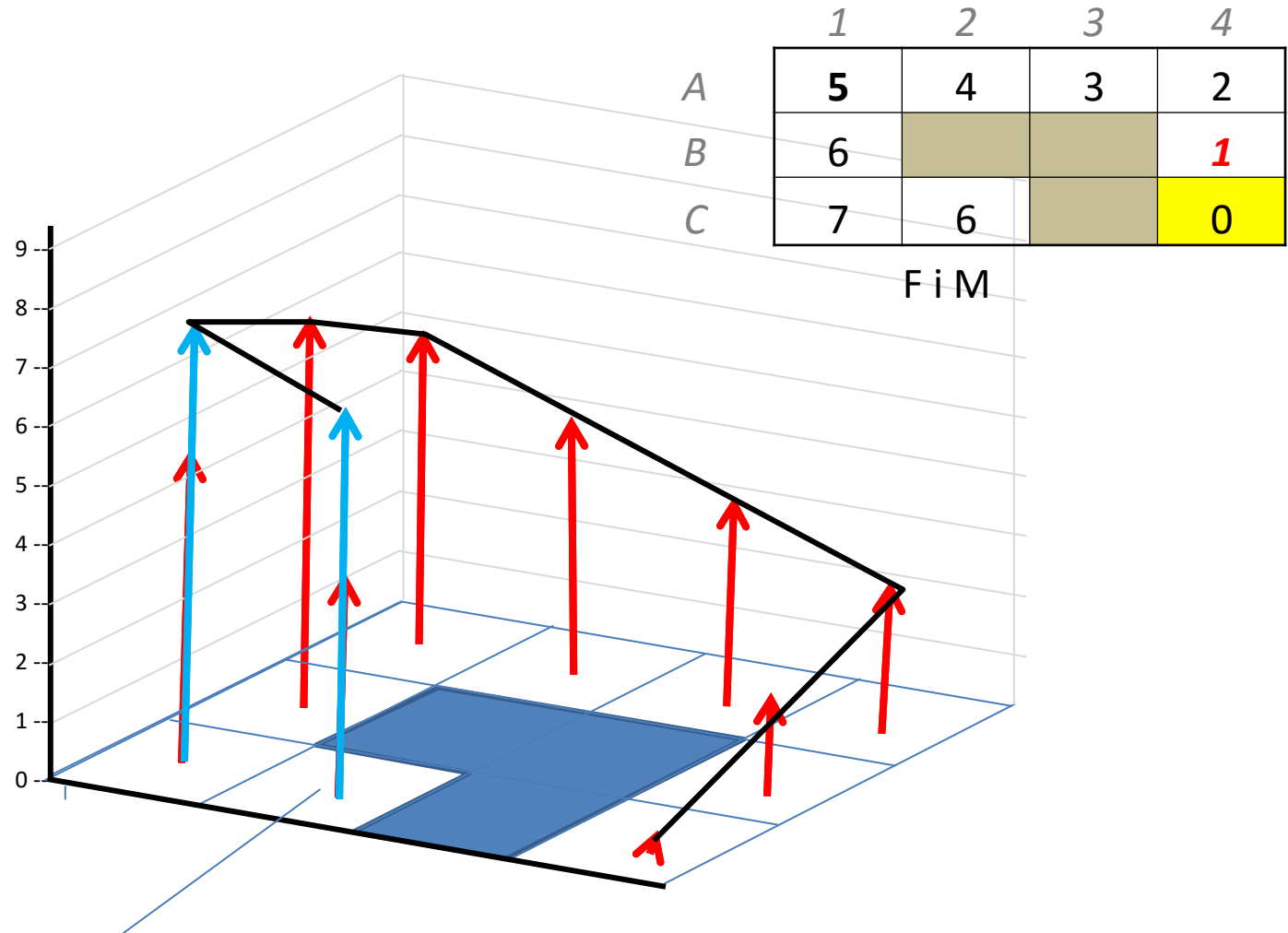
F i M

# LRTA\*: EXECUÇÃO 1



(Koenig, 2001)

# LRTA\*: EXECUÇÃO 2



com mais execuções, o valor da posição C2 atingiria 8 que é o custo real

(Koenig, 2001)

# LRTA\*

## Requisito

Valor inicial de  $h$  deve ser otimista, i.e.  
$$h(n) \leq h^*(n)$$

# LRTA\*

O algoritmo é completo?

**Sim,**

se há um número finito de nós com arestas (ações) de custos positivos, e

se existem caminhos de cada nó ao nó objetivo (*não há nó isolado*) e

se  $h(n)$  é não-negativo (pode ser zero)

então LRTA\* encontrará o nó objetivo

# LRTA\*

## É ótimo?

Requer várias execuções para alcançar otimalidade.

Se as estimativas iniciais forem admissíveis, então, executando-o várias vezes no mesmo problema, os valores aprendidos pelo LRTA\* convergirão para as distâncias reais em todos os caminhos ótimos até o nó objetivo.



quando há empate nos  $f(n')$ , a escolha do vizinho deve ser randômica, caso contrário, o algoritmo, ao encontrar uma solução ótima, sempre a seguirá