PROCURA EM ESPAÇOS DE ESTADOS

Luís Morgado

2022

RACIOCÍNO ATRAVÉS DE PROCURA

EXEMPLO

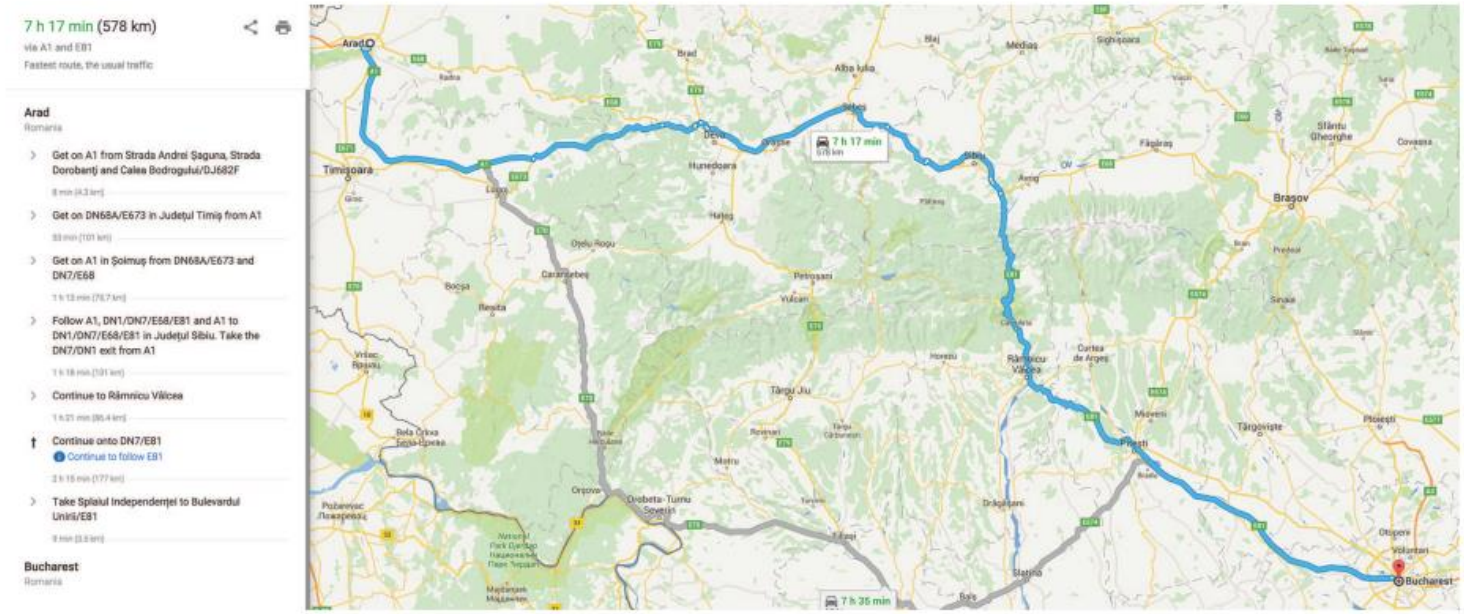
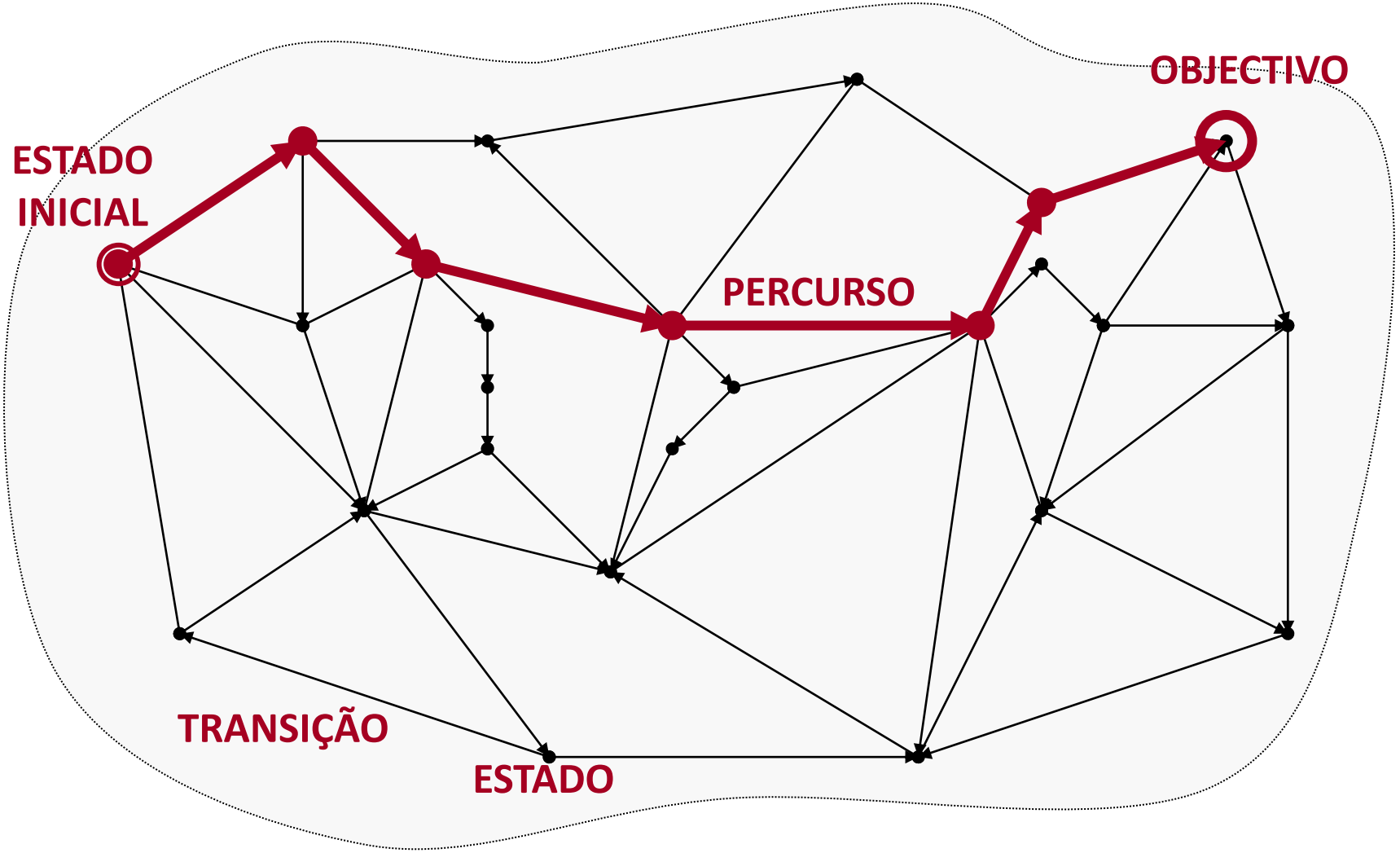


Figure 3.28 A Web service providing driving directions, computed by a search algorithm.

[Russel & Norvig, 2010]

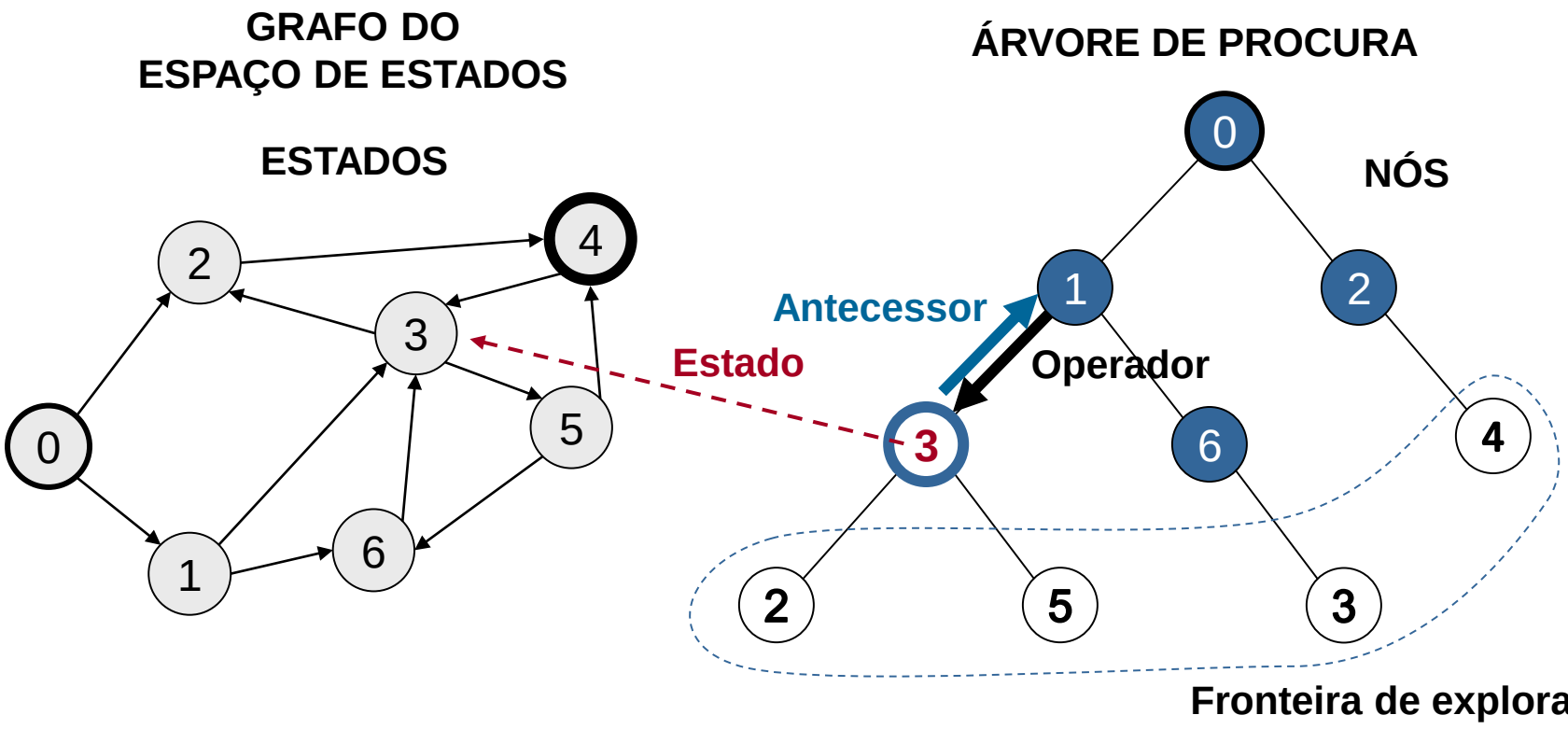
RACIOCÍNO ATRAVÉS DE PROCURA

PROBLEMAS DE PLANEAMENTO



PROCESSO DE PROCURA

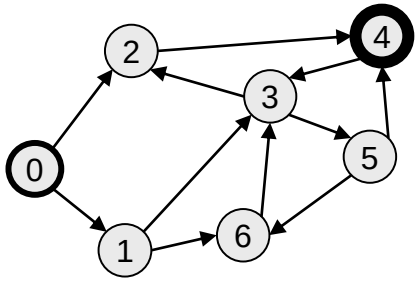
- Exploração sucessiva do espaço de estados
- Etapa de procura: **Nó**
- **Árvore de procura**
  - Raiz: Nó correspondente ao e*stado inicial*
- **Fronteira de exploração** (estrutura de dados com relação de ordem)
  - Critério de ordenação determina estratégia de controlo da procura



MÉTODOS DE PROCURA

Procura em profundidade (*Depth-First Search*)

- Estratégia de controlo
  - Explorar primeiro os nós mais **recentes**



Grafo do
Espaço de Estados

Fronteira de exploração [ ]

[0]

0 [ ]

[**1, 2**]

1 [2]

[**3, 6**, 2]

3 [6,2]

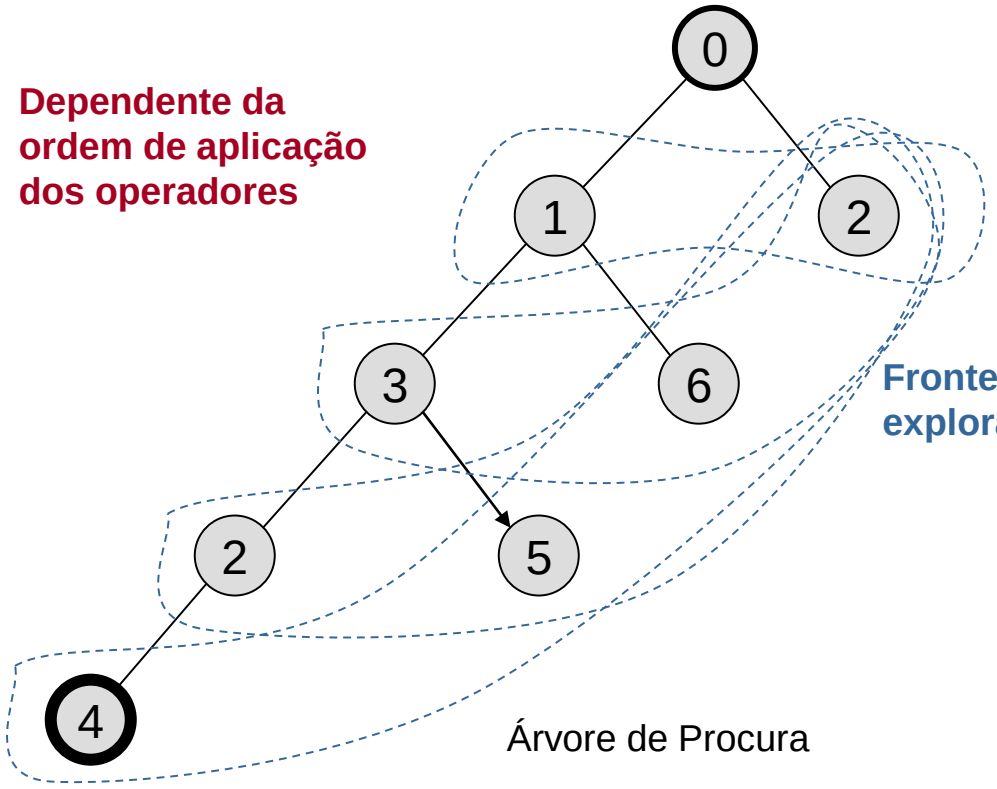
[**2, 5**, 6, 2]

2 [5,6,2]

[**4**, 5, 6, 2]

4 [5,6,2]

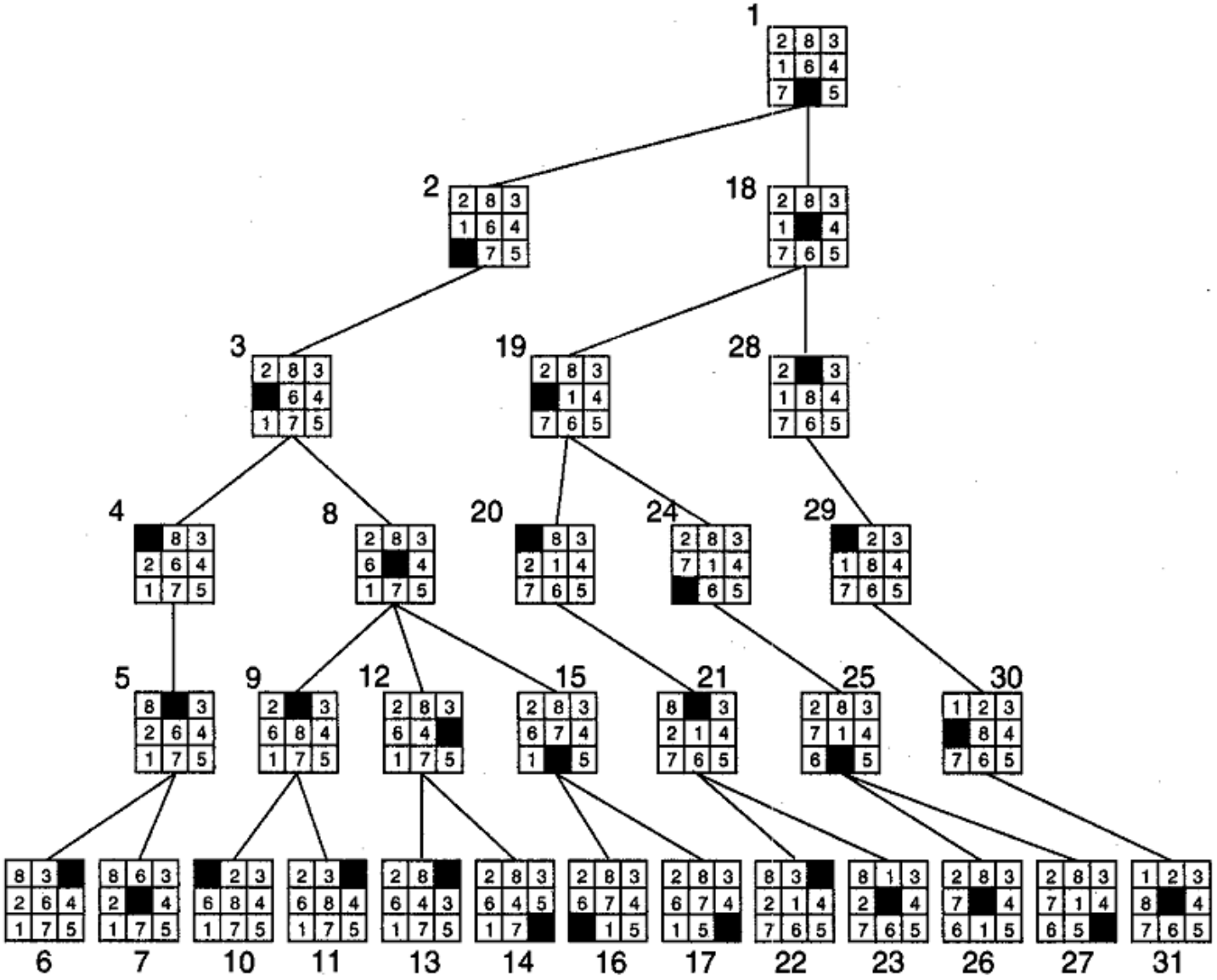
Dependente da ordem de aplicação dos operadores



Fronteira de exploração

Árvore de Procura

PROCURA EM PROFUNDIDADE

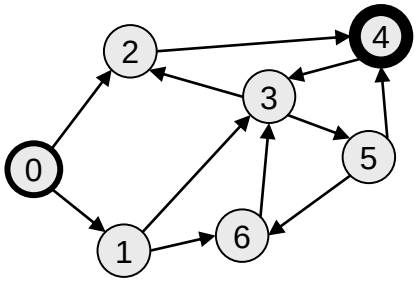


[Luger, 2009]

MÉTODOS DE PROCURA

Procura em largura (*Breadth-First Search*)

- Estratégia de controlo
  - Explorar primeiro os nós mais **antigos**



Grafo do
Espaço de Estados

Fronteira de exploração [ ]

[0]

0 [ ]

[**1, 2**]

1 [2]

[2, **3, 6**]

2 [3,6]

[3, 6, **4**]

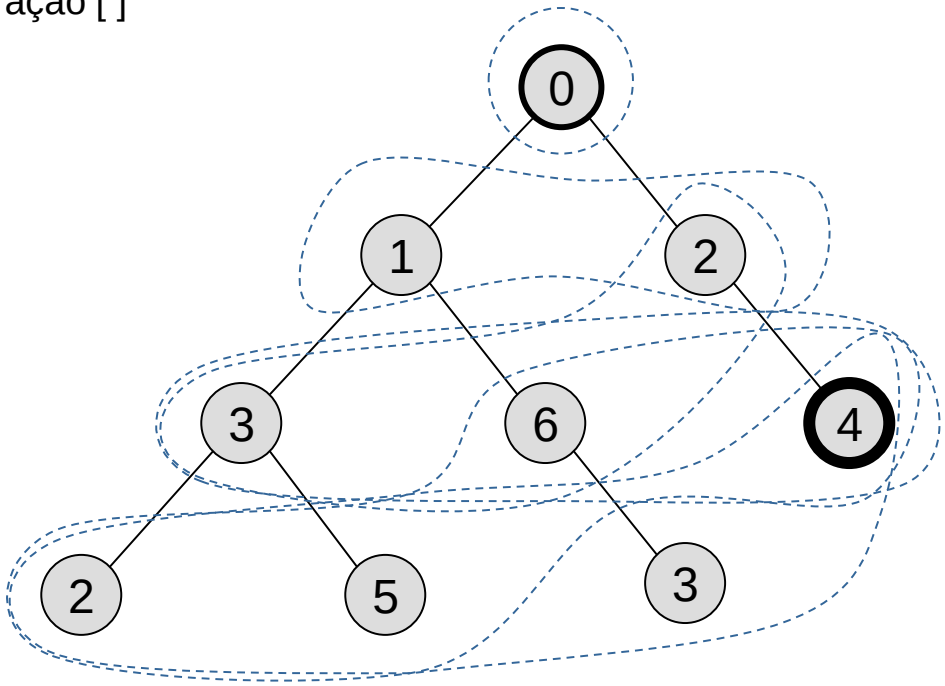
3 [6,4]

[6,4,**2,5**]

6 [4,2,5]

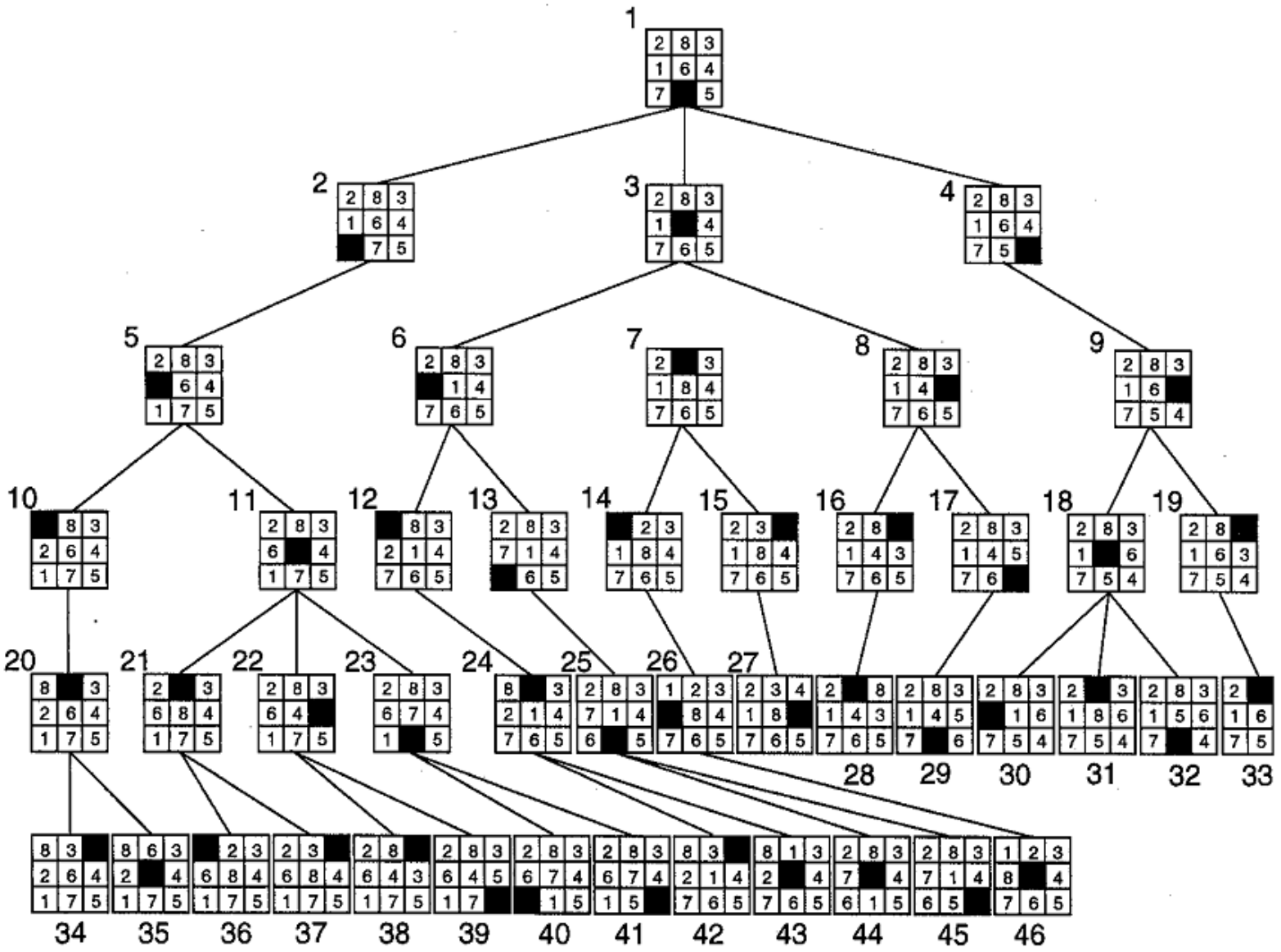
[4,2,5,**3**]

4 [2,5,3]



Árvore de Procura

PROCURA EM LARGURA



[Luger, 2009]

PROCURA EM GRAFOS COM CICLOS

ESTADOS REPETIDOS NA ÁRVORE DE PROCURA

- Grafo do espaço de estados apresenta ciclos
- Múltiplas transições para o mesmo estado
- Acções correspondentes às transições de estado são reversíveis

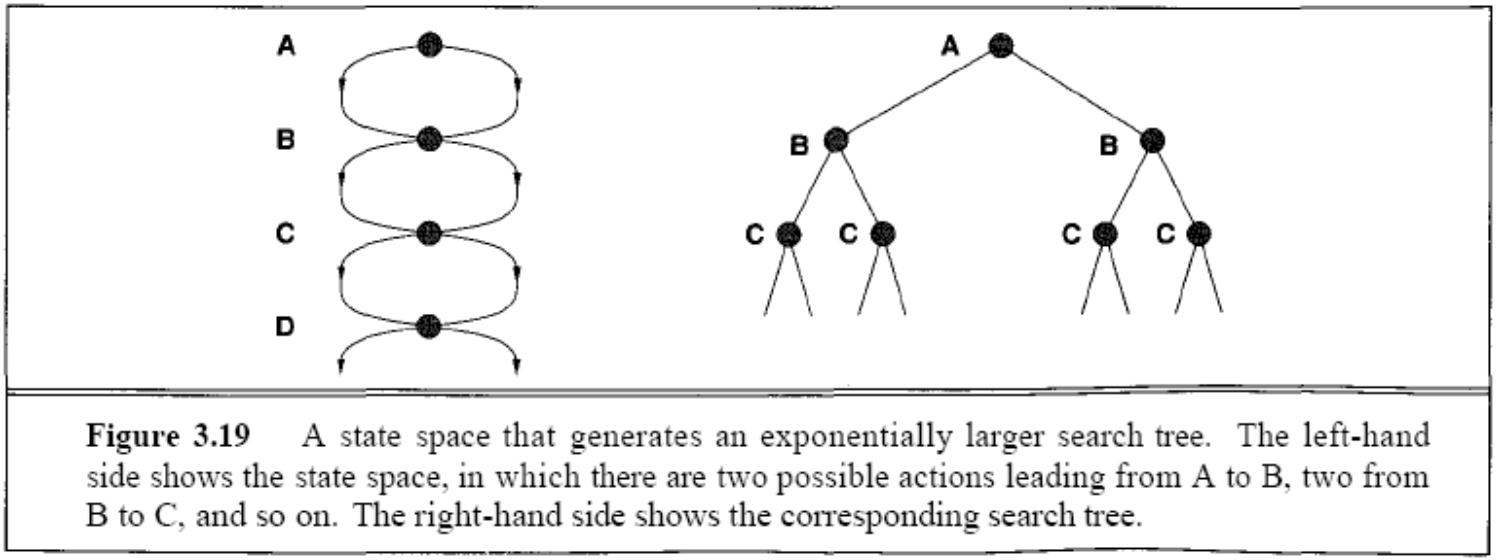


Figure 3.19 A state space that generates an exponentially larger search tree. The left-hand side shows the state space, in which there are two possible actions leading from A to B, two from B to C, and so on. The right-hand side shows the corresponding search tree.

[Russel & Norvig, 2003]

EXPANSÃO DE ESTADOS JÁ ANTERIORMENTE ANALISADOS

- **Desperdício** de recursos (**tempo**, **memória**)

PROCURA GERAL EM GRAFOS

- Nós ***Abertos***: nós gerados mas **não expandidos**
  - Fronteira de exploração
- Nós ***Fechados***: nós **expandidos**
- Ao gerar novo nó sucessor *noSuc* é necessário considerar:
  - $noSuc \notin Abertos \wedge noSuc \notin Fechados$
    - Inserir *noSuc* em *Abertos*
  - $noSuc \in Abertos$
    - Se *noSuc* foi atingido através de um caminho mais curto (com menor custo)
      - Remover nó anterior de *Abertos*
      - inserir *noSuc* em *Abertos*
  - $noSuc \in Fechados$
    - Se *noSuc* foi atingido através de um caminho mais curto (com menor custo)
      - Remover nó anterior de *Fechados*
      - inserir *noSuc* em *Abertos*

PROCURA EM GRAFOS COM CICLOS

MEMÓRIA DE NÓS PROCESSADOS

- Nós gerados mas **não expandidos** (**fronteira** de exploração)
  - **ABERTOS**
- Nós **expandidos**
  - **FECHADOS**

EXPLORADOS

PROCURA EM ESPAÇOS DE ESTADOS

Procura em largura (*Breadth-First Search*)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

```
function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

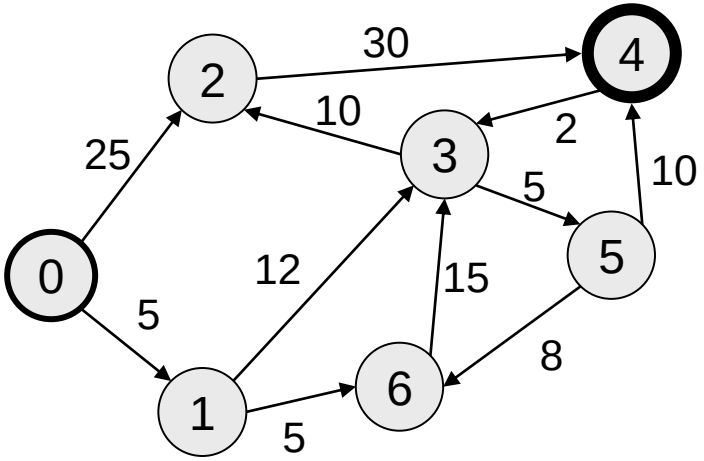
[Russel & Norvig, 2022]

PROCURA MELHOR-PRIMEIRO (*BEST-FIRST*)

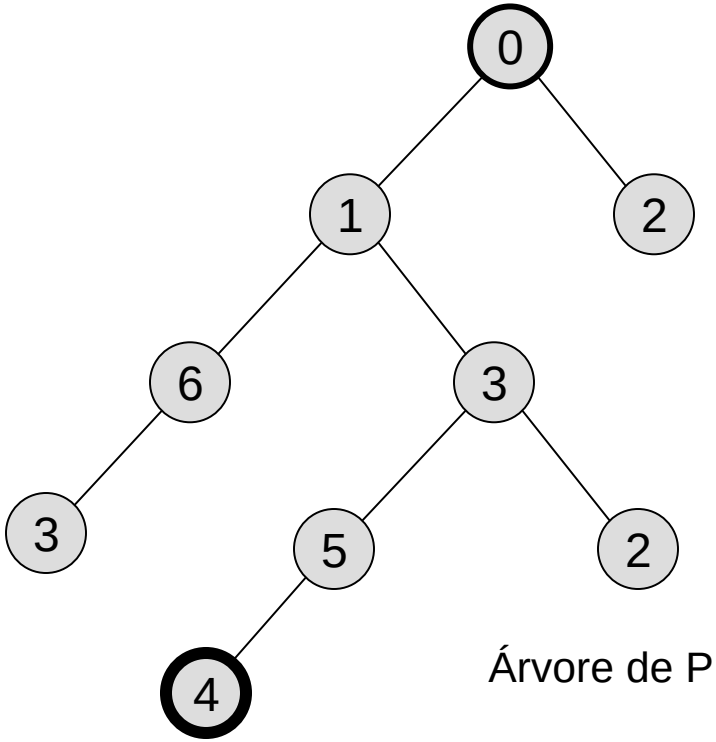
- Tira partido de uma **avaliação do estado**
- Utiliza uma função ***f*** para **avaliação** de cada nó *n* gerado
  - $\boldsymbol{f(n)} \geq 0$
  - Tipicamente $f(n)$ representa uma estimativa do custo da solução através do nó *n*
    - Quanto menor o valor de $f(n)$ mais promissor é o nó *n*
- A **fronteira de exploração** (*Fringe* / *Abertos*) é **ordenada por ordem crescente de *f*(*n*)**

PROCURA DE CUSTO UNIFORME

- Estratégia de controlo
  - Explorar primeiro caminhos com menor custo
  - Custo de transição $\geq \varepsilon > 0$



Grafo do
Espaço de Estados



Árvore de Procura

PROCURA EM ESPAÇOS DE ESTADOS

Procura melhor-primeiro (*Best-First Search*)

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

Procura de custo uniforme

```
function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

[Russel & Norvig, 2022]

MÉTODOS DE PROCURA

QUAL O MELHOR MÉTODO DE PROCURA ?

- Aspectos a considerar num método de procura
  - **COMPLETO**
    - O método de procura garante que, caso exista solução, esta será encontrada
  - **ÓPTIMO**
    - O método de procura garante que, existindo várias soluções, a solução encontrada é a melhor
  - **COMPLEXIDADE**
    - **TEMPO** (complexidade temporal)
      - **Tempo necessário** para encontrar uma solução
    - **ESPAÇO** (complexidade espacial)
      - **Memória necessária** para encontrar uma solução

MÉTODOS DE PROCURA

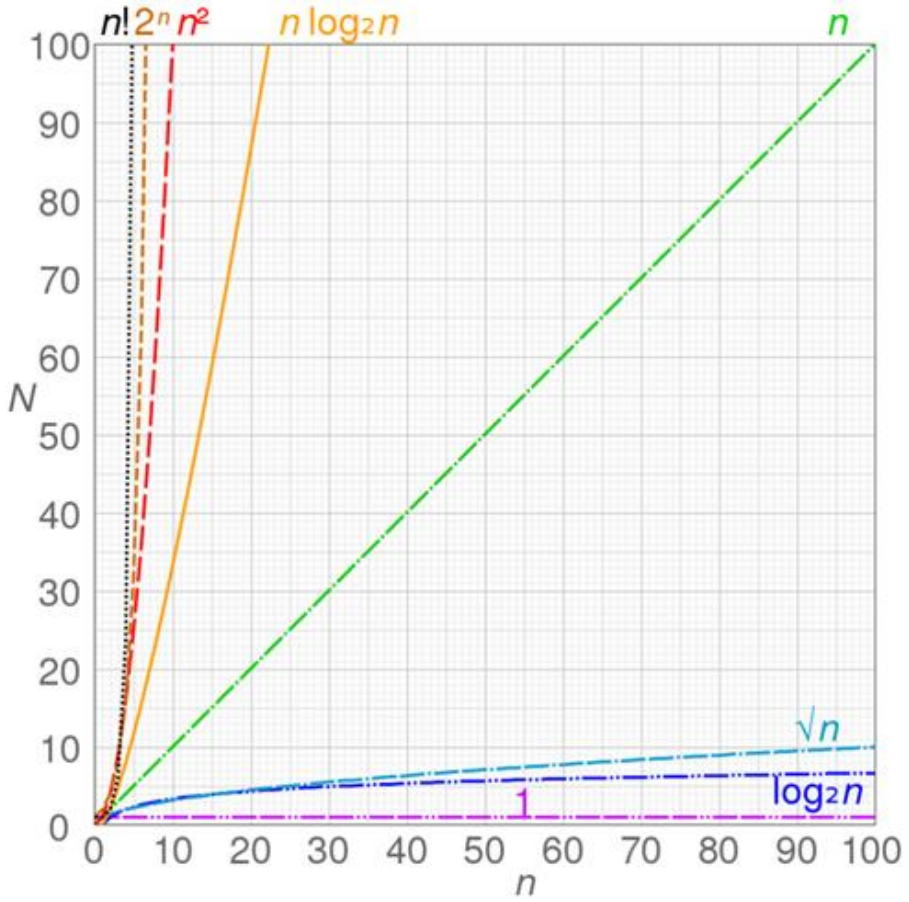
- Parâmetros de caracterização de um método de procura:
  - **FACTOR DE RAMIFICAÇÃO** - ***b***
    - Número máximo de sucessores para um qualquer estado
  - **PROFUNDIDADE DA PROCURA** - ***d***
    - Profundidade do nó objectivo menos profundo na árvore de procura
    - Dimensão do percurso entre o estado inicial e o estado objectivo

COMPLEXIDADE COMPUTACIONAL

Notação $f = O(g)$

$f(x)$ é de ordem $O(g(x))$ se existirem duas constantes positivas x_0 e c tal que:
$(x > x_0) : f(x) \leq cg(x)$

Exemplo:
Funções de referência de complexidade computacional



COMPLEXIDADE COMPUTACIONAL

PROCURA EM LARGURA

Factor de ramificação *(branching factor)*: ***b***

Número de nós a expandir para encontrar uma solução de dimensão ***d***

$1 + b + b^2 + b^3 + \ldots + b^d$ ⟶ Complexidade espacial: $O(b^d)$

Complexidade temporal: $O(b^d)$

COMPLEXIDADE ESPACIAL (MEMÓRIA) EXPONENCIAL

PROCURA EM PROFUNDIDADE

Número de nós a expandir para explorar até uma profundidade ***m***

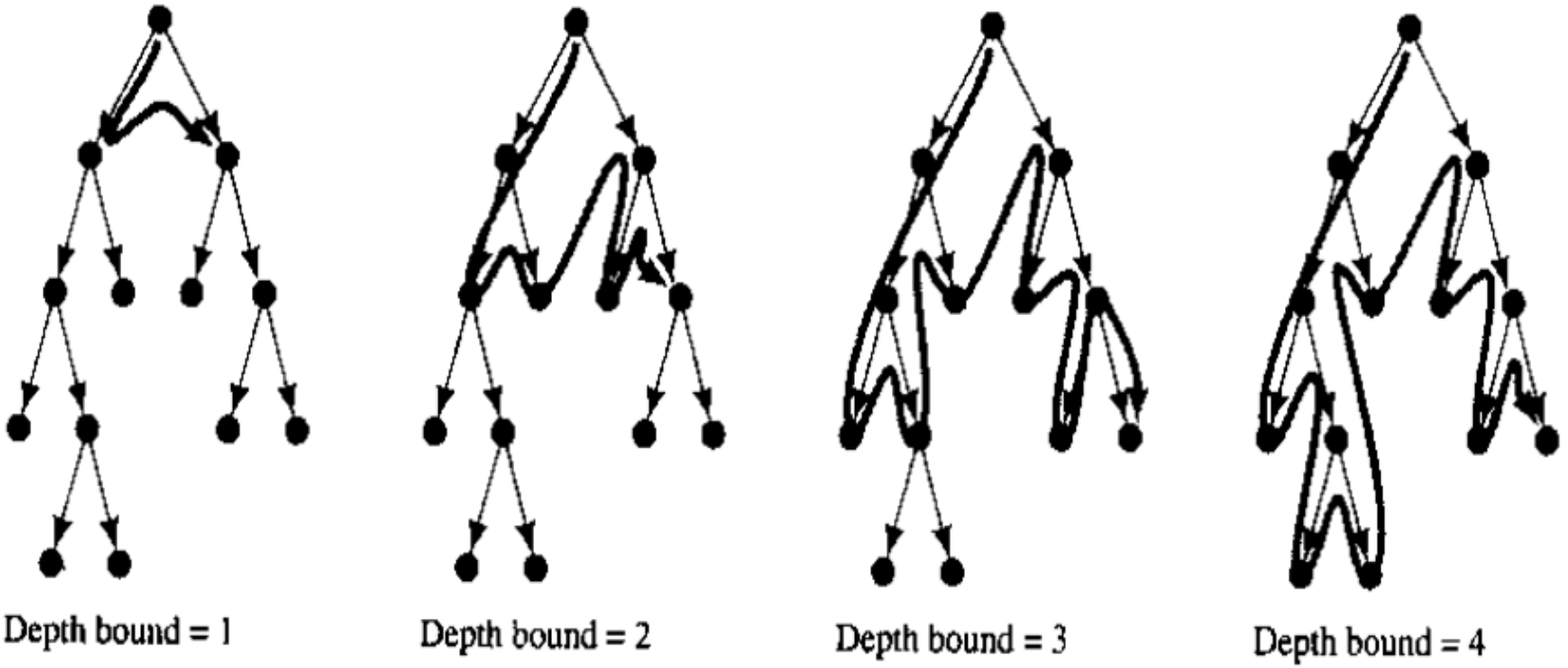
Complexidade espacial: $O(bm)$

Complexidade temporal: $O(b^m)$

PODE NÃO ENCONTRAR SOLUÇÃO

COMPLEXIDADE ESPACIAL (MEMÓRIA) LINEAR

PROCURA EM PROFUNDIDADE ITERATIVA



[Nilsson, 1998]

Número de nós a expandir para encontrar uma solução de dimensão ***d***

$(d+1) + (d)b + (d-1)b^2 + \ldots + 2b^{d-1} + 1b^d$

Complexidade espacial: $O(bd)$

Complexidade temporal: $O(b^d)$

PROCURA EM PROFUNDIDADE

PROCURA EM PROFUNDIDADE LIMITADA (*Depth-Limited Search*)

- Limita a procura a uma profundidade máxima

```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

PROCURA EM PROFUNDIDADE ITERATIVA (*Iterative Deepening Search*)

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result
```

Figure 3.12 Iterative deepening and depth-limited tree-like search. Iterative deepening repeatedly applies depth-limited search with increasing limits. It returns one of three different types of values: either a solution node; or *failure*, when it has exhausted all nodes and proved there is no solution at any depth; or *cutoff*, to mean there might be a solution at a deeper depth than ℓ. This is a tree-like search algorithm that does not keep track of *reached* states, and thus uses much less memory than best-first search, but runs the risk of visiting the same state multiple times on different paths. Also, if the IS-CYCLE check does not check *all* cycles, then the algorithm may get caught in a loop.

[Russel & Norvig, 2022]

COMPLEXIDADE COMPUTACIONAL

Método de Procura	Tempo	Espaço	Óptimo	Completo
Profundidade	$O(b^m)$	$O(bm)$	Não	Não
Largura	$O(b^d)$	$O(b^d)$	Sim	Sim
Custo Uniforme	$O(b^{[C^*/\varepsilon]})$	$O(b^{[C^*/\varepsilon]})$	Sim	Sim
Profundidade Limitada	$O(b^l)$	$O(bl)$	Não	Não
Profundidade Iterativa	$O(b^d)$	$O(bd)$	Sim	Sim

b – factor de ramificação
d – dimensão da solução
m – profundidade da árvore de procura
l – limite de profundidade

C^* – Custo da solução óptima
ε – Custo mínimo de uma transição de estado ($\varepsilon > 0$)

MÉTODOS DE PROCURA NÃO INFORMADA

PROCURA EM PROFUNDIDADE

- Critério de exploração: maior profundidade
- Variantes
  - **PROCURA EM PROFUNDIDADE LIMITADA**
  - **PROCURA EM PROFUNDIDADE ITERATIVA**

PROCURA EM LARGURA

- Critério de exploração: menor profundidade

PROCURA DE CUSTO UNIFORME

- Critério de exploração: custo da solução

BIBLIOGRAFIA

[Russel & Norvig, 2009]

S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd Edition, Prentice Hall, 2009

[Russel & Norvig, 2022]

S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th Edition, Pearson, 2022

[Nilsson, 1998]

N. Nilsson , *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann 1998

[Luger, 2009]

G. Luger , *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* , Addison-Wesley, 2009

[Jaeger & Hamprecht, 2010]

M. Jaeger, F. Hamprecht, *Automatic Process Control for Laser Welding*, Heidelberg Collaboratory for Image Processing (HCI) , 2000