回顾

南京大学 NANJING UNIVERSITY

- **智能体**
  - 智能体和环境
  - 理性的概念
  - PEAS（性能度量、环境、执行器、传感器）
  - 任务环境的属性
  - 智能体类型

提纲

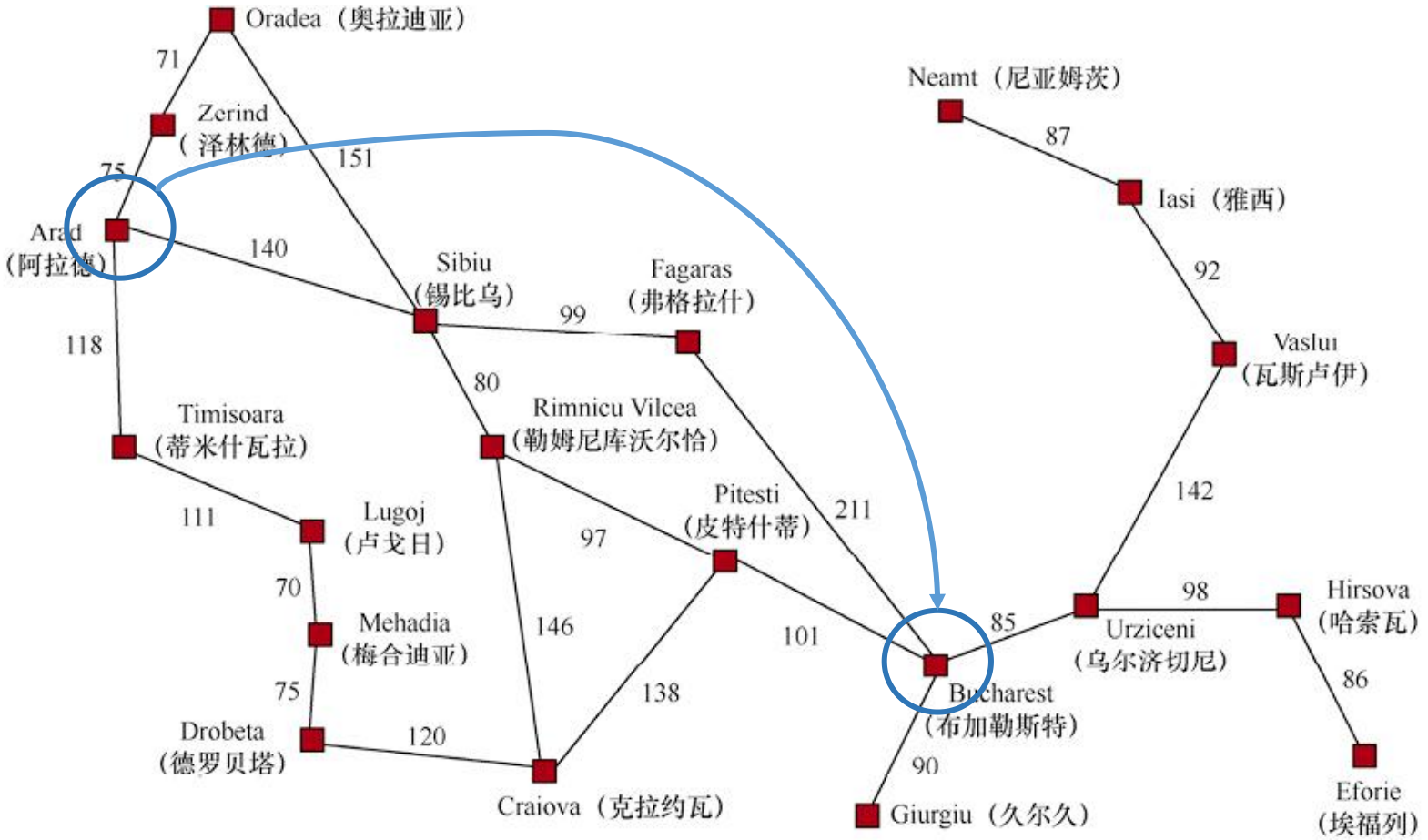
- **通过搜索进行问题求解**
  - 问题求解智能体
  - 搜索算法
  - 无信息搜索策略
  - 有信息（启发式）搜索策略
- **复杂环境中的搜索**
  - 局部搜索和最优化问题
  - 连续空间中的局部搜索

提纲

南京大学 NANJING UNIVERSITY

- **通过搜索进行问题求解**
  - 问题求解智能体
  - 搜索算法
  - 无信息搜索策略
  - 有信息（启发式）搜索策略
- **复杂环境中的搜索**
  - 局部搜索和最优化问题
  - 连续空间中的局部搜索

问题求解智能体



罗马尼亚部分地区的简化道路图，道路距离单位为英里（1英里 = 1.61千米）

问题求解智能体

有了上述地图信息，智能体可以执行以下4个阶段的**问题求解过程**：

- **目标形式化（goal formulation）**：目标为到达布加勒斯特。目标通过限制智能体的目的和需要考虑的动作来组织其行为。

- **问题形式化（problem formulation）**：实现目标所必需的状态和动作分别是不同的城市和在城市间行驶。

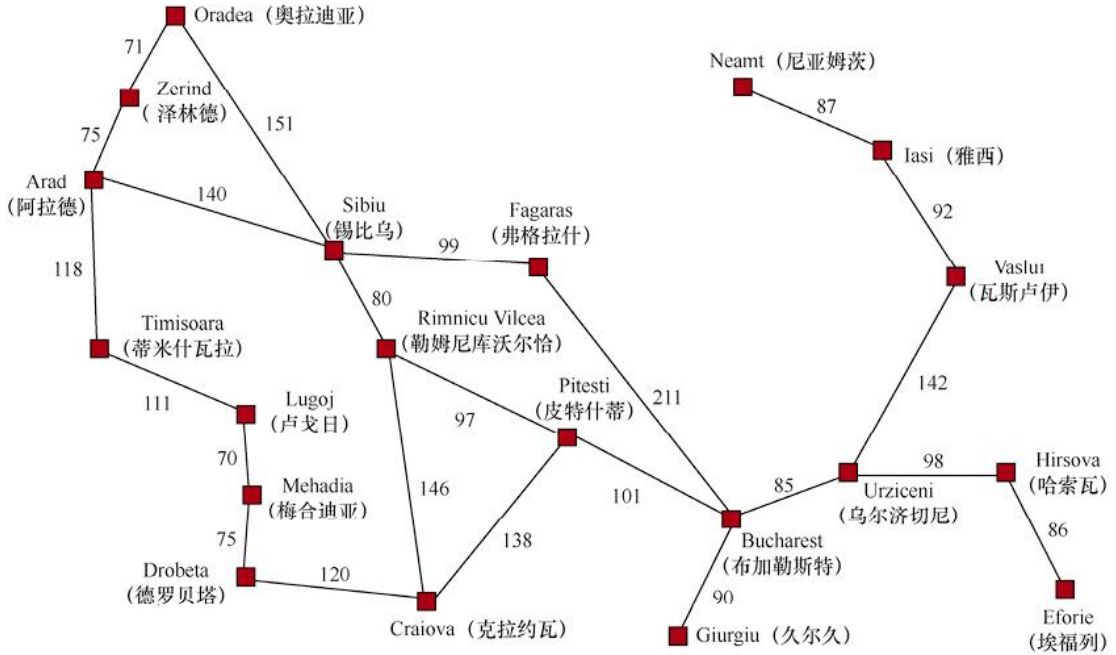
- **搜索（search）**：在真实世界中采取任何动作之前，智能体会在其模型中模拟一系列动作，并进行搜索，直到找到一个能到达目标的动作序列。这样的序列称为**解（solution）**。

- **执行（execution）**：现在智能体可以执行解中的动作，一次执行一个动作。

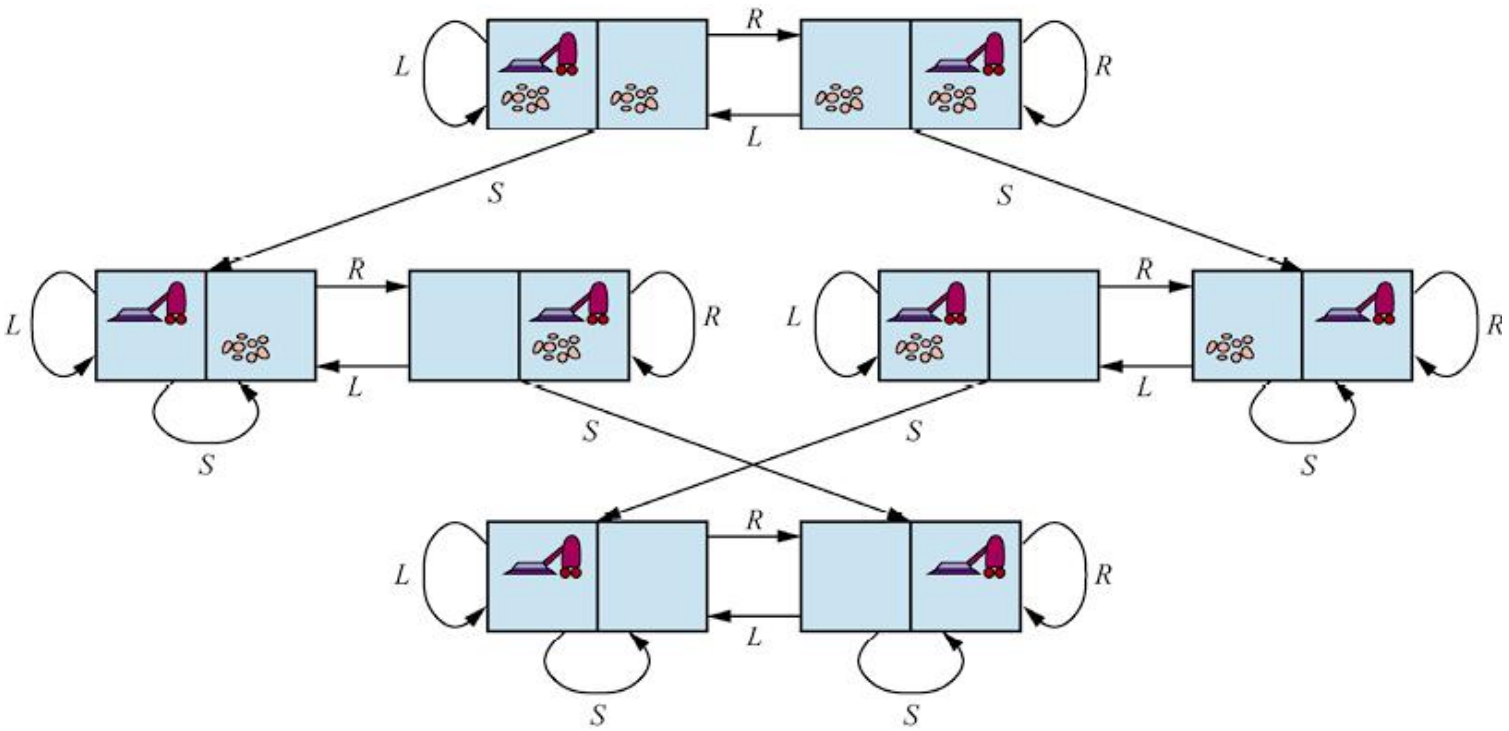
问题求解智能体

搜索问题的形式化定义：

- **状态空间（state space）**：可能的环境状态（state）的集合
- **初始状态（initial state）**
- 一个或多个**目标状态（goal state）**的集合
- **行动（action）**：在某一状态s可以采取的行动
- **转移模型（transition model）**给出在状态s中执行动作a所产生的状态
- **动作代价函数（action cost function）**给出在状态s中执行动作a从而转移到状态s’的数值代价。



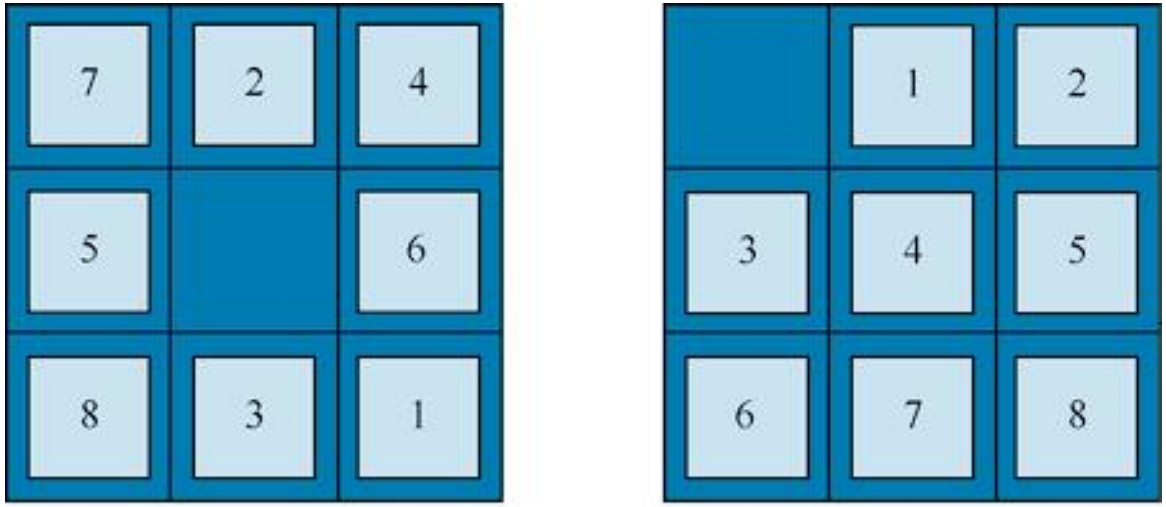
问题示例



两个单元格的真空吸尘器世界的状态空间图

- **状态**：即哪些对象在哪些单元格中。
- **动作**：吸尘（Suck）、向左（Left）移动和向右（Right）移动。
- **转移模型**
- **目标状态**： 每个单元格都保持干净的状态。
- **动作代价**： 每个动作的代价都是1。

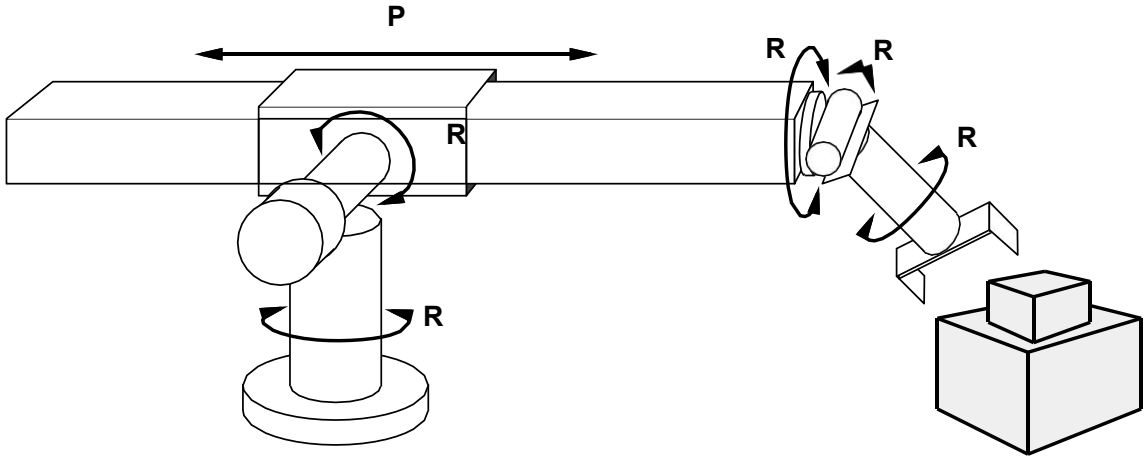
问题实例



8数码问题的一个典型实例

- **状态**：指定每个滑块位置的状态描述。
- **动作**：对空格执行Left、Right、Up或Down。
- **转移模型**
- **目标状态**：通常用有序编号指定目标状态，如右图所示。
- **动作代价**：每个动作的代价都为1。

问题实例



机器人装配

- **状态**：机器人手臂关节角度和要装配物体部件的实值坐标。
- **动作**：机器人关节的连续移动。
- **转移模型**
- **目标状态**：物体完整组装。
- **动作代价**：动作执行的时间。

搜索算法

南京大学 NANJING UNIVERSITY

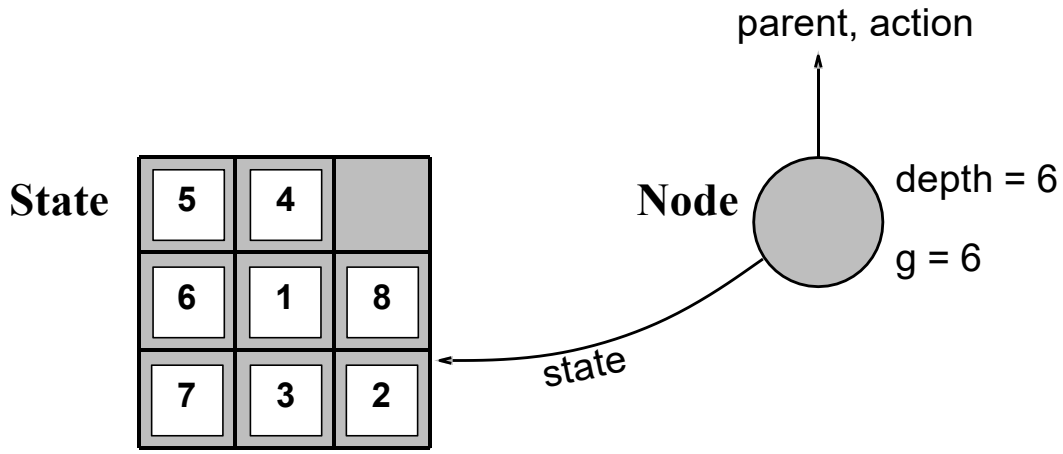
树搜索算法

该算法通过**扩展（expand）**节点，从初始状态形成各条路径，并试图找到一条可以达到某个目标状态的路径。搜索树中的每个**节点（node）**对应于状态空间中的一个**状态（state）**，搜索树中的边对应于动作，树的根对应于问题的初始状态。

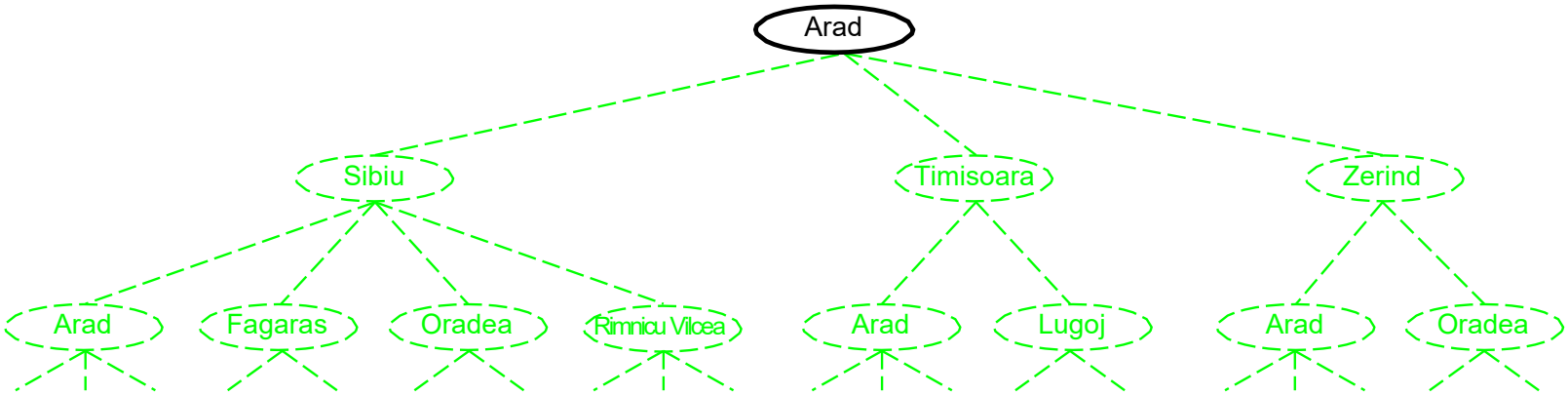
```
function Tree-Search( problem, strategy) returns a solution, or
    failure initialize the search tree using the initial state
    of problem
    loop do
        if there are no candidates for expansion then
        return failure choose a leaf node for expansion
        according to strategy
        if the node contains a goal state then return the corresponding
        solution
        else expand the node and add the resulting nodes to the search
        tree
    end
```

搜索算法

状态（state）： 世界中一个物理配置的表示
节点（node）： 一个数据结构，是搜索树的组成部分，包括父节点，子节点，深度和路径代价

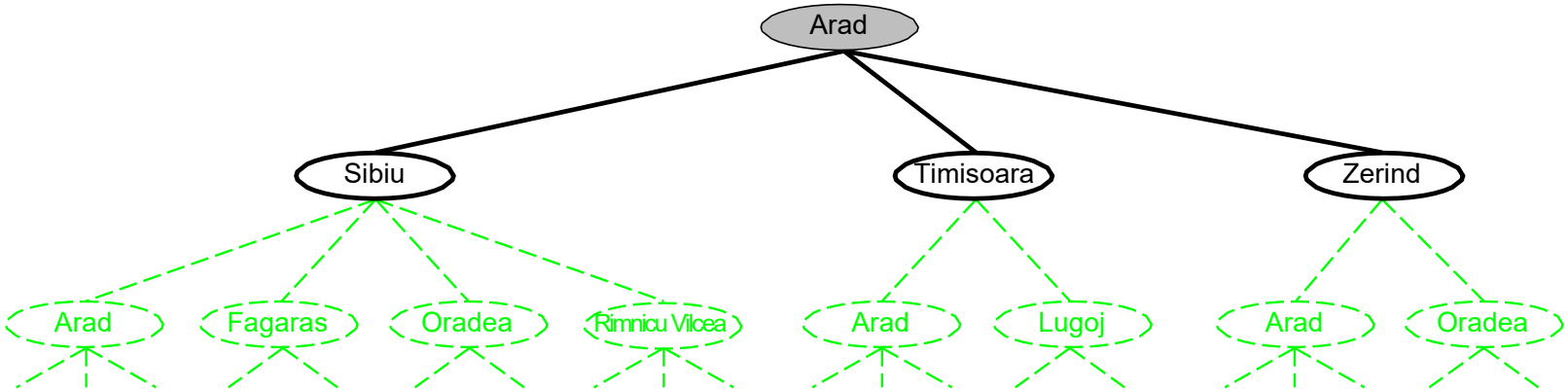


搜索算法



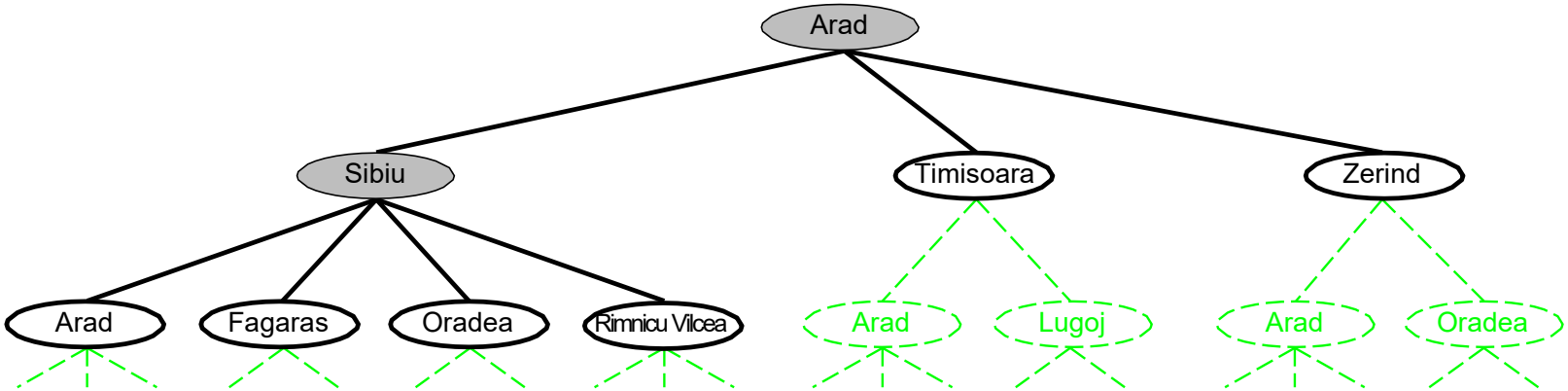
部分搜索树，用于寻找从Arad到Bucharest的路线。

搜索算法



部分搜索树，用于寻找从Arad到Bucharest的路线。

搜索算法

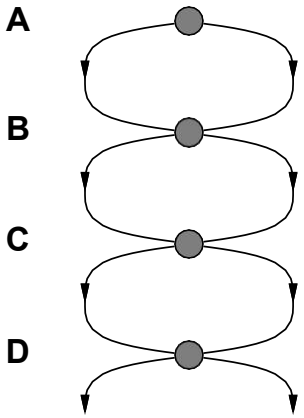


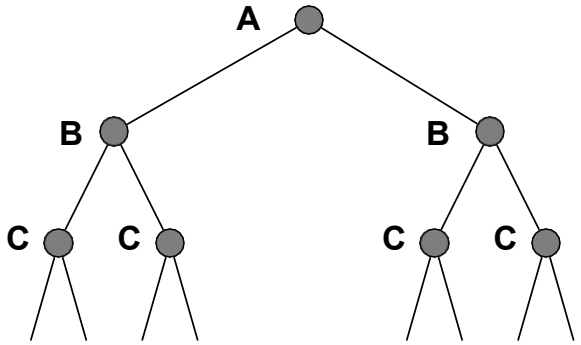
部分搜索树，用于寻找从Arad到Bucharest的路线。

搜索算法

南京大学 NANJING UNIVERSITY

冗余路径





如果搜索算法检查冗余路径，我们称之为**图搜索（graph search）**；
否则，称之为**树搜索（tree-like search）**

树搜索实现

南京大学 NANJING UNIVERSITY

```
function Tree-Search( problem, frontier) returns a solution, or
    failure  frontier ← Insert (Make-Node(Initial-State[problem]),
    frontier)   loop do
        if frontier is empty then return failure
        node ← Remove-Front(frontier)
        if Goal-Test(problem, State(node)) then return node
        frontier ← InsertAll(Expand(node, problem), frontier)
```

```
function Expand( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in Successor-Fn(problem, State[node]) do
        s ← a new Node
        Parent-Node[s] ← node;                    Action[s] ← action;
                                  State[s] ← result
        Path-Cost[s] ← Path-Cost[node] + Step-Cost(node, action, s)
        Depth[s] ← Depth[node] + 1
        add s to successors
    return successors
```

图搜索实现

南京大学 NANJING UNIVERSITY

```
function Graph-Search( problem, frontier ) returns a solution, or
failure

    closed ← an empty set
    frontier ← Insert (Make-Node(Initial-State[problem]), frontier )
    loop do
        if frontier is empty then return failure
        node ← Remove-Front(frontier )
        if Goal-Test(problem, State[node]) then return node
        if State[node] is not in closed then
            add State[node] to closed
            frontier ← InsertAll(Expand(node, problem), frontier )
    end
```

搜索算法

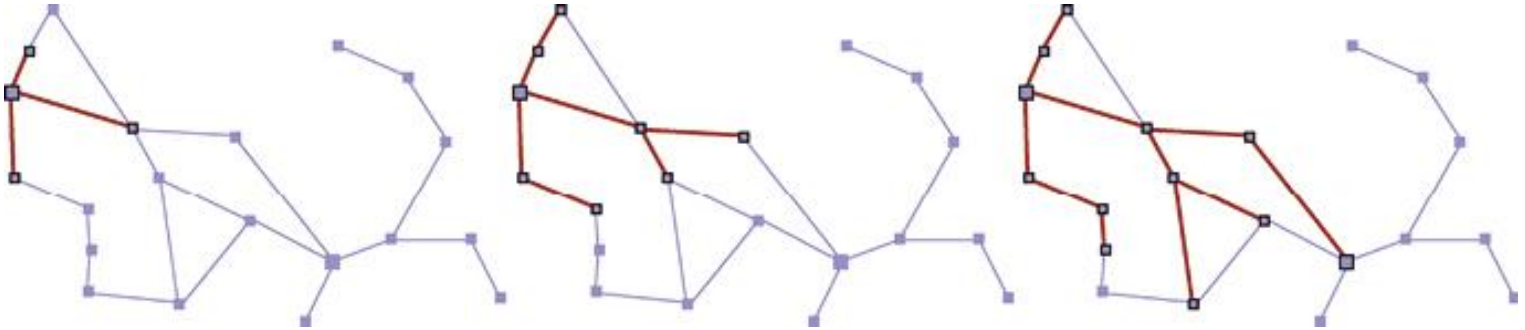
南京大学 NANJING UNIVERSITY

最佳优先搜索（best-first search）

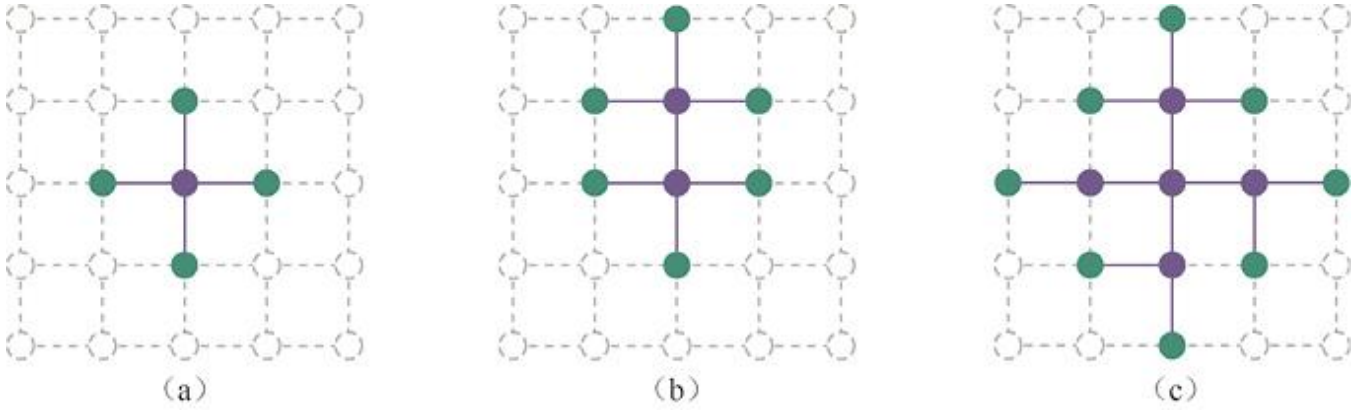
```
function BEST-FIRST-SEARCH(problem, f) returns 一个解节点或 failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← 一个以 f 排序的优先队列，其中一个元素为node
  reached ← 一个查找表，其中一个条目的键为problem.INITIAL，值为node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s不在reached中 or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        将child添加到frontier中
  return failure

function EXPAND(problem, node) yields 节点
  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)
```

搜索算法



罗马尼亚问题的**图搜索**生成的搜索树序列。



以矩形网格问题为例说明**图搜索**的分离性质

搜索算法

搜索数据结构

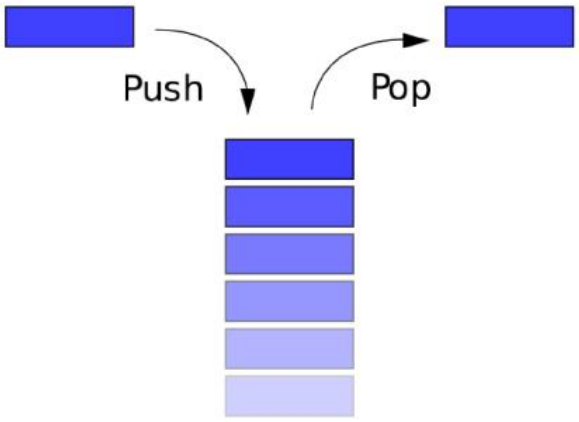
搜索算法需要一个数据结构来跟踪搜索树。树中的**节点（node）**由一个包含4个组成部分的数据结构表示。

- node.State：节点对应的状态。
- node.Parent：父节点，即树中生成该节点的节点。
- node.Action：父节点生成该节点时采取的动作。
- node.Path-Cost：从初始状态到此节点的路径总代价。在数学公式中，一般使用g(node)表示Path-Cost。

我们需要一个数据结构来存储**边界**。一个恰当的选择是某种队列（queue），因为边界上的操作有以下几个。

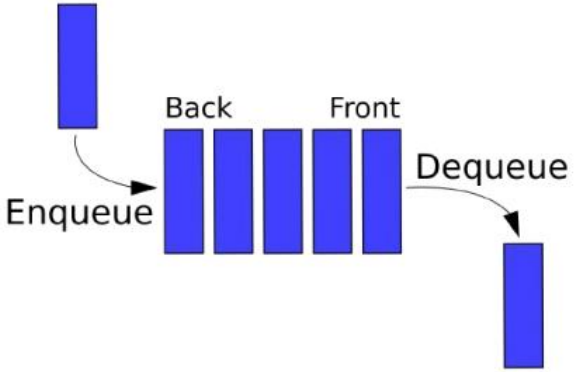
- Is-Empty(frontier)：返回true当且仅当边界中没有节点。
- Pop(frontier)：返回边界中的第一个节点并将它从边界中删除。
- Top(frontier)：返回（但不删除）边界中的第一个节点。
- Add(node, frontier)：将节点插入队列中的适当位置。

不同类型的队列



stack

LIFO队列



queue

FIFO队列

问题求解性能评估

算法性能评价：

- **完备性（completeness）**：当存在解时，算法是否能保证找到解，当不存在解时，是否能保证报告失败？
- **代价最优性（cost optimality）**：它是否找到了所有解中路径代价最小的解？
- **时间复杂性（time complexity）**：找到解需要多长时间？可以用秒数来衡量，或者更抽象地用状态和动作的数量来衡量。
- **空间复杂性（space complexity）**：执行搜索需要多少内存？

时间和空间复杂性则由如下3个指标衡量：

b—搜索树的最大分支因子

d—最小代价解的深度或动作数

m—状态空间的最大深度

无信息搜索策略

无信息搜索算法不提供有关某个状态与目标状态的接近程度的任何线索：

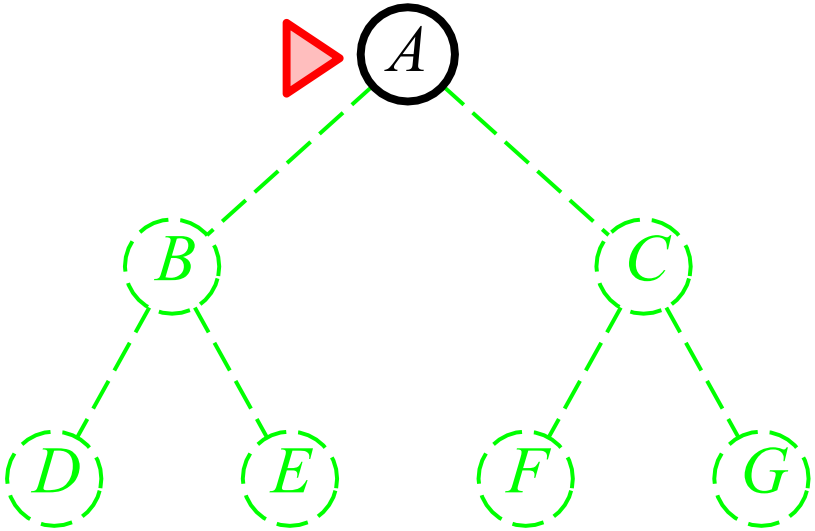
- **广度优先搜索 Breadth-first search**
- **一致代价搜索 Uniform-cost search**
- **深度优先搜索 Depth-first search**
- **深度受限搜索 Depth-limited search**
- **迭代加深搜索 Iterative deepening search**

无信息搜索策略

南京大学 NANJING UNIVERSITY

广度优先搜索

- 优先扩展最浅的未被扩展的节点
- 边界（frontier）可以实现为一个FIFO队列，即, 新节点（总是比其父节点更深）进入队列的队尾，而旧节点，即比新节点浅的节点，首先被扩展。



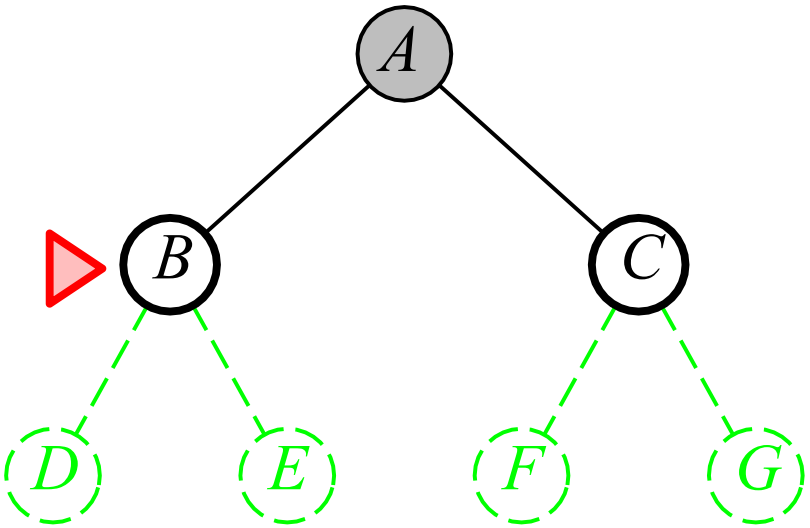
简单二叉树上的广度优先搜索。每个阶段接下来要扩展的节点用三角形标记表示。

无信息搜索策略

南京大学 NANJING UNIVERSITY

广度优先搜索

- 优先扩展最浅的未被扩展的节点
- 边界（frontier）可以实现为一个FIFO队列，即, 新节点（总是比其父节点更深）进入队列的队尾，而旧节点，即比新节点浅的节点，首先被扩展。



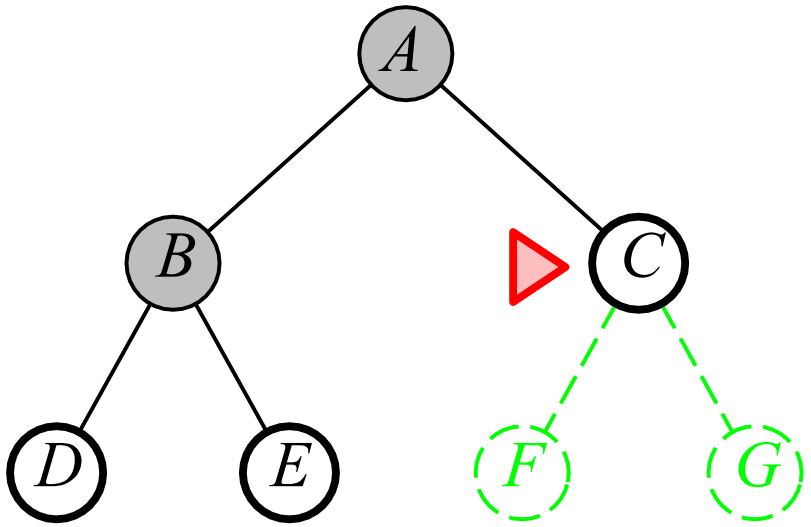
简单二叉树上的广度优先搜索。每个阶段接下来要扩展的节点用三角形标记表示。

无信息搜索策略

南京大学 NANJING UNIVERSITY

广度优先搜索

- 优先扩展最浅的未被扩展的节点
- 边界（frontier）可以实现为一个FIFO队列，即, 新节点（总是比其父节点更深）进入队列的队尾，而旧节点，即比新节点浅的节点，首先被扩展。



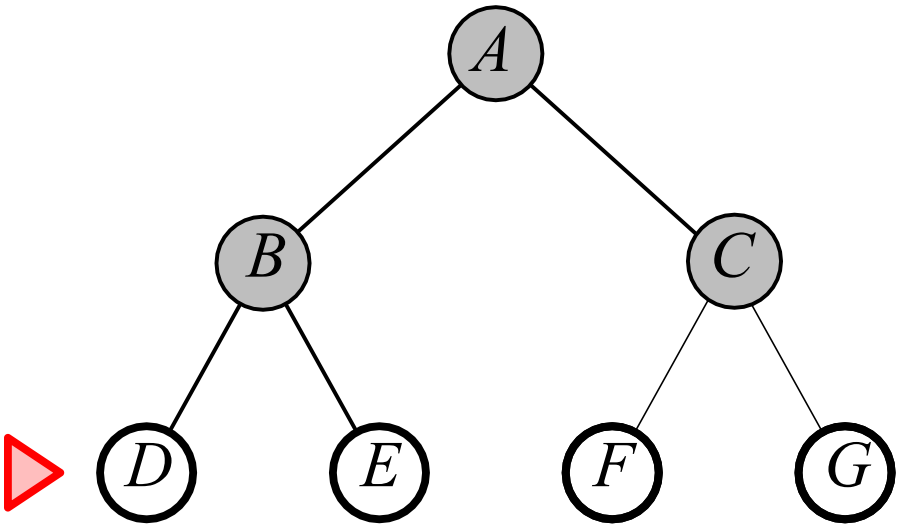
简单二叉树上的广度优先搜索。每个阶段接下来要扩展的节点用三角形标记表示。

无信息搜索策略

南京大学 NANJING UNIVERSITY

广度优先搜索

- 优先扩展最浅的未被扩展的节点
- 边界（frontier）可以实现为一个FIFO队列，即, 新节点（总是比其父节点更深）进入队列的队尾，而旧节点，即比新节点浅的节点，首先被扩展。



简单二叉树上的广度优先搜索。每个阶段接下来要扩展的节点用三角形标记表示。

无信息搜索策略

南京大学 NANJING UNIVERSITY

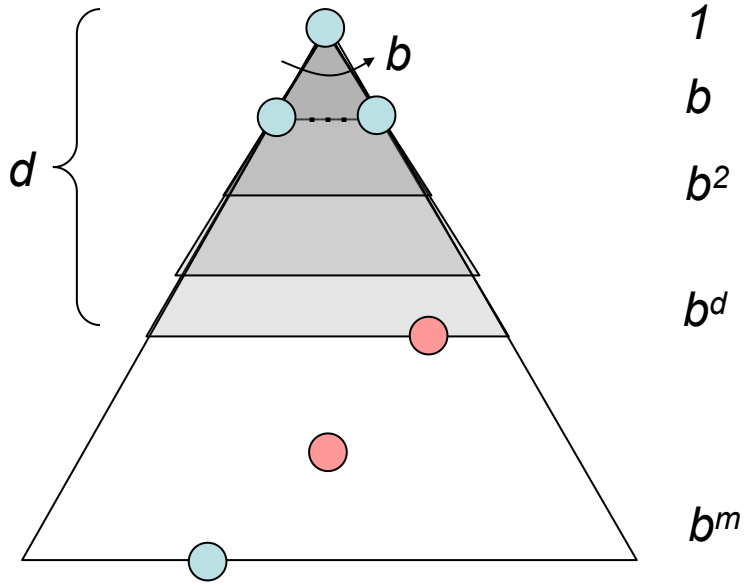
广度优先搜索

完备性：是（如果b是有限的）

时间：$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$（假设解的深度为$d$）

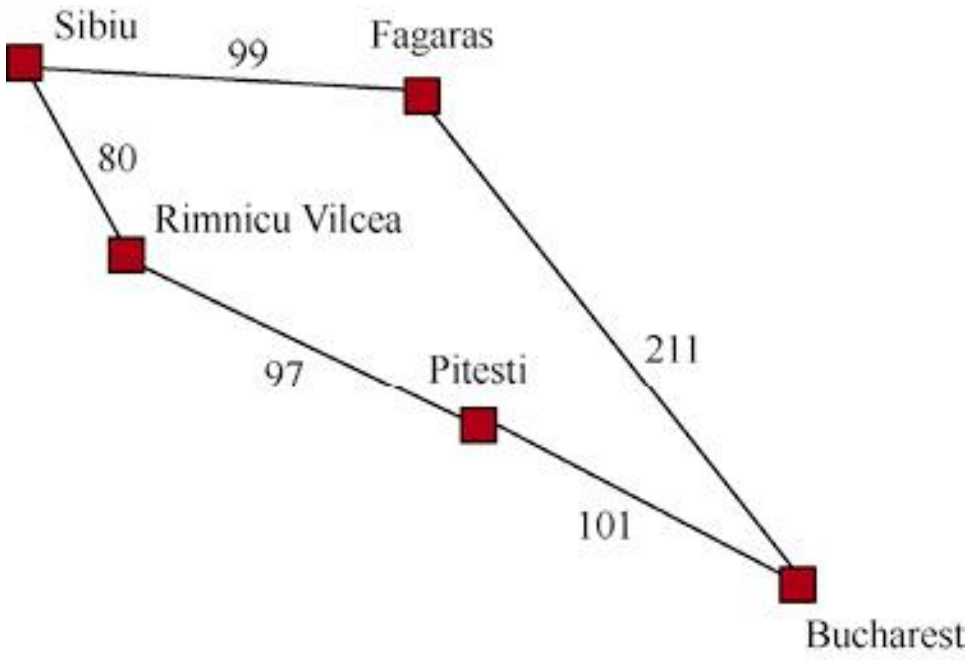
空间：$O(b^d)$（所有节点都存储在内存中）

代价最优：是（如果所有动作都具有相同代价，否则不一定）



一致代价搜索（Dijkstra算法）

- 优先扩展代价最小的未被扩展的节点
- 边界（frontier）可以实现为一个按路径代价排序的队列，最浅层的优先



罗马尼亚问题状态空间的一部分，选择这部分来说明一致代价搜索

无信息搜索策略

南京大学 NANJING UNIVERSITY

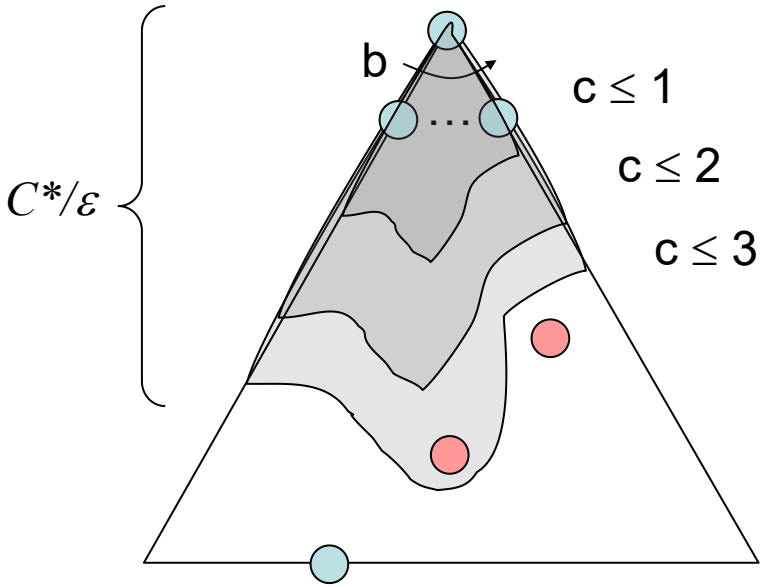
一致代价搜索（Dijkstra算法）

完备性：是（假设所有动作的代价> ε>0 ）

时间：最坏情况下$O(b^{C^*/\varepsilon})$（这里C^*是最优解的代价）

空间：最坏情况下$O(b^{C^*/\varepsilon})$

代价最优：是（节点以路径代价增加的顺序扩展）



无信息搜索策略

南京大学 NANJING UNIVERSITY

```
function BREADTH-FIRST-SEARCH(problem) returns 一个解节点或failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← 一个FIFO队列，其中一个元素为node
  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不在reached中 then
        将s添加到reached
        将child添加到frontier
    return failure

function UNIFORM-COST-SEARCH(problem) returns 一个解节点或failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

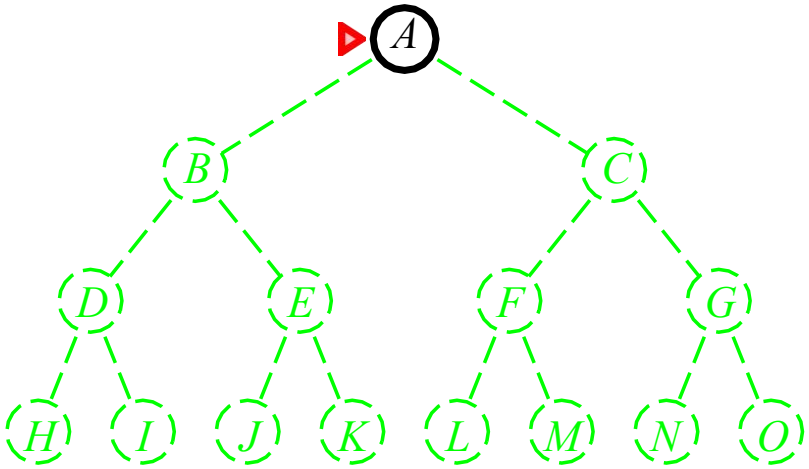
广度优先搜索和一致代价搜索算法

无信息搜索策略

南京大学 NANJING UNIVERSITY

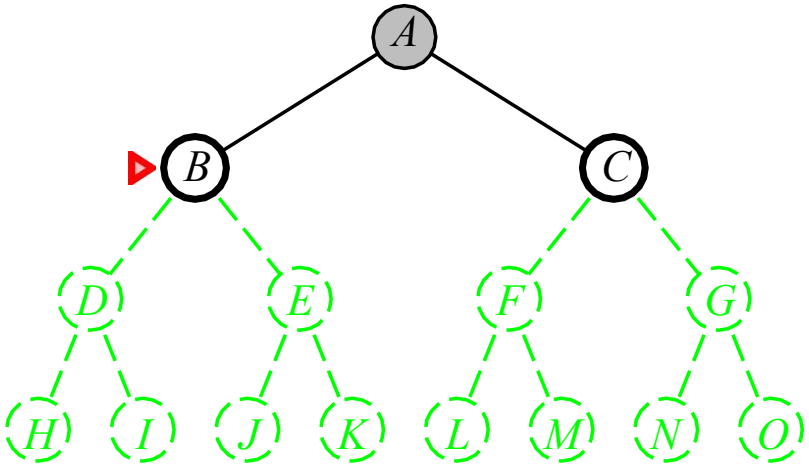
深度优先搜索

- 优先扩展最深的未被扩展的节点
- 边界（frontier）可以实现为一个LIFO队列，即后进先出。



二叉树的深度优先搜索过程中，从开始状态A到目标M

深度优先搜索

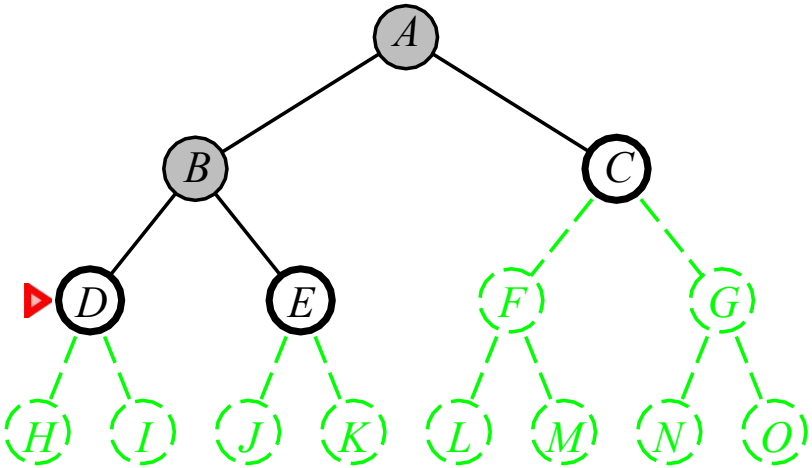


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

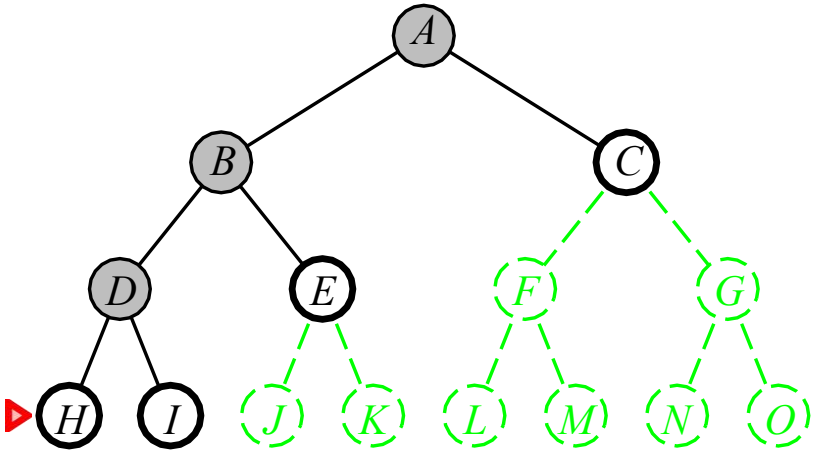


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

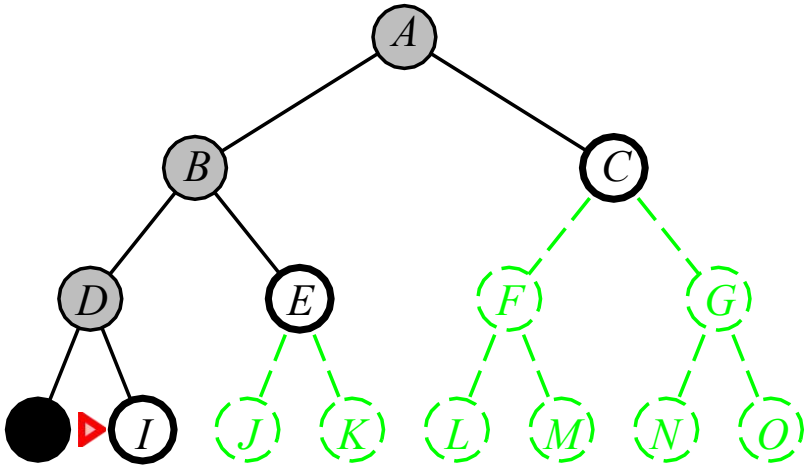


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

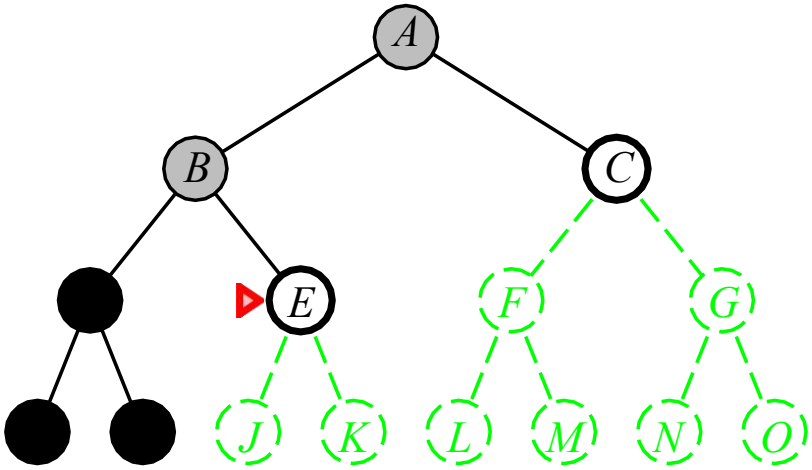


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

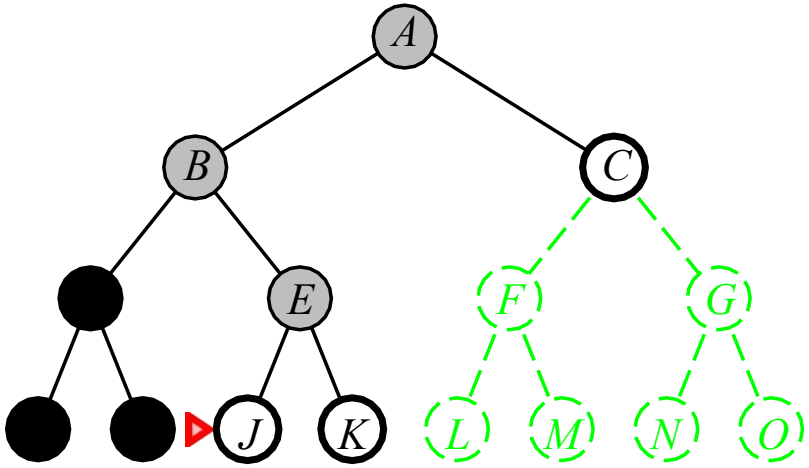


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

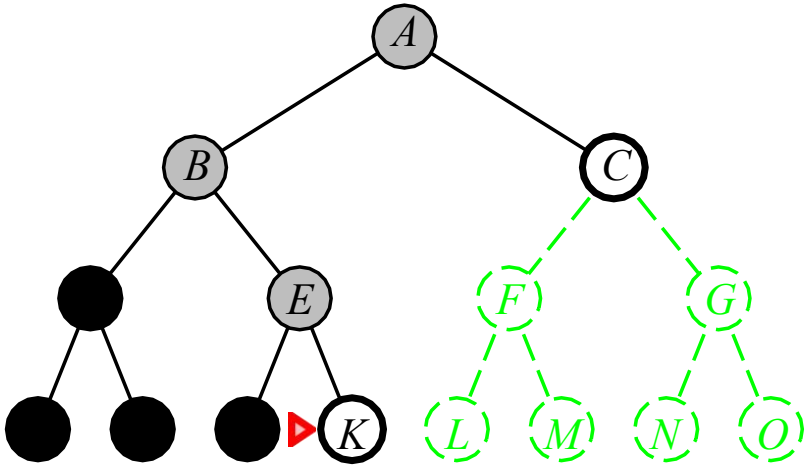


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

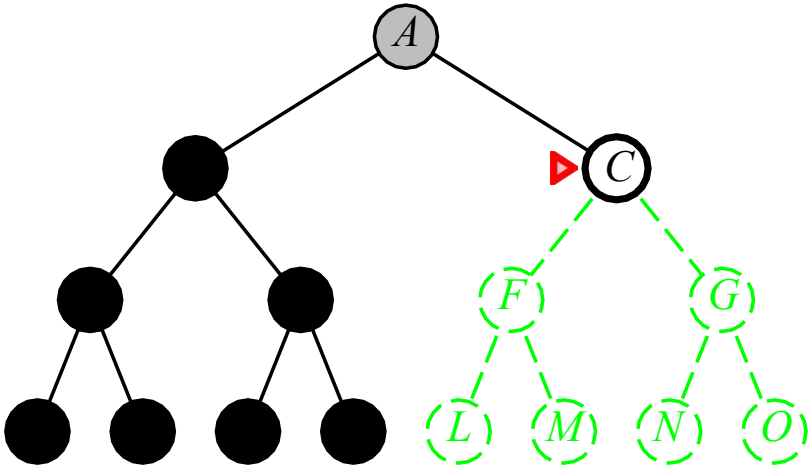


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

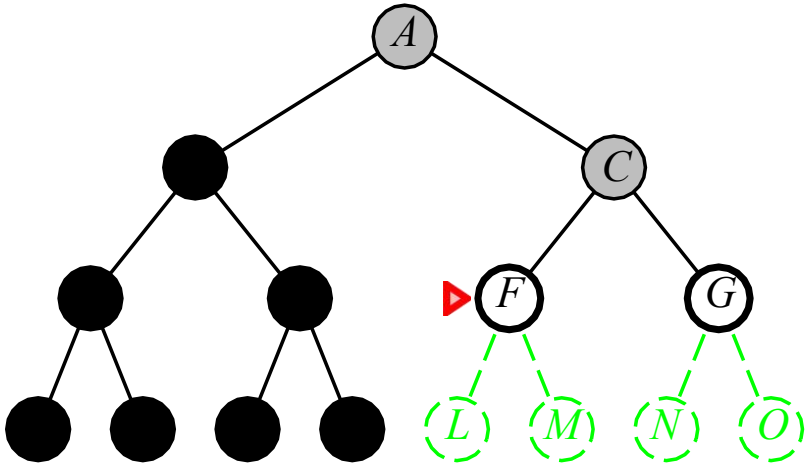


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

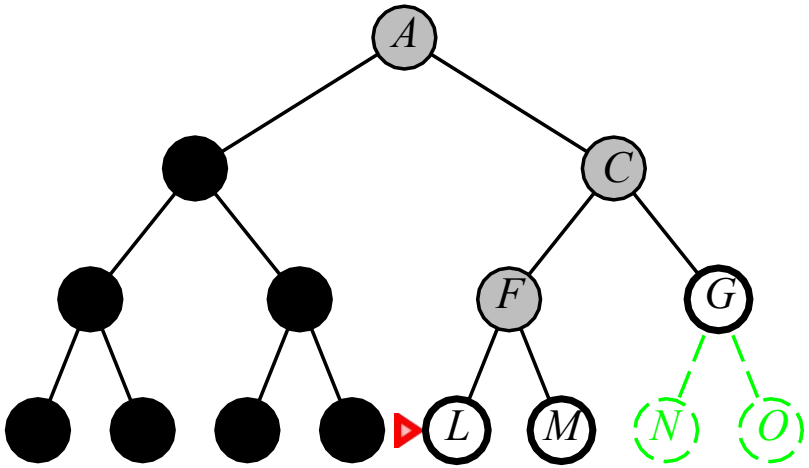


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索

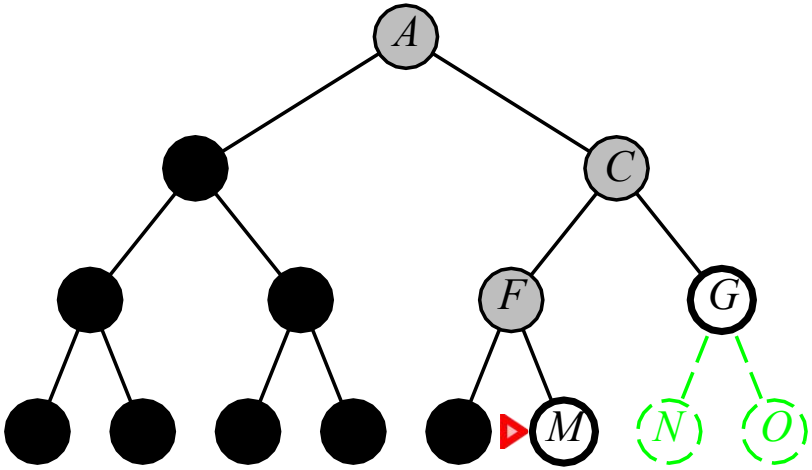


二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

南京大学 NANJING UNIVERSITY

深度优先搜索



二叉树的深度优先搜索过程中，从开始状态A到目标M

无信息搜索策略

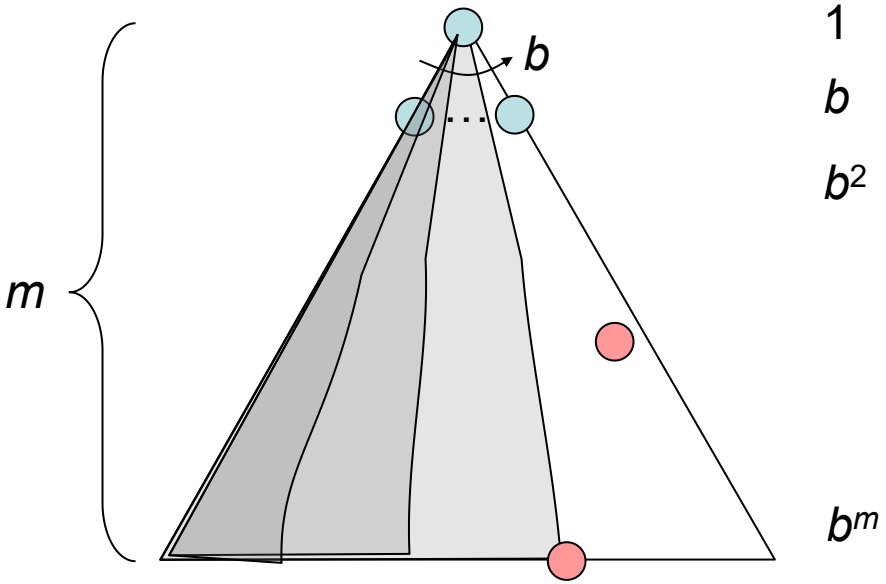
深度优先搜索

完备性：否（无限状态空间和有环状态空间，而对于树型的有限状态空间，算法是有效且完备的）

时间：$O(b^m)$（m是树的最大深度，如果解密集的，可能要比广度优先更快）

空间：$O(bm)$（线性空间）

代价最优：否，会找到最左边的解，而不管深度或者代价如何



无信息搜索策略

南京大学 NANJING UNIVERSITY

深度受限搜索

- 深度限制为 l 的深度优先搜索，即将深度 l 上的所有节点视为其不存在后继节点

```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns一个解节点或failure或cutoff
  frontier ← 一个LIFO队列（栈），其中一个元素为NODE(problem.INITIAL)
  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
        将child添加到frontier
  return result
```

无信息搜索策略

南京大学 NANJING UNIVERSITY

迭代加深搜索

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns 一个解节点或failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result
```

无信息搜索策略

南京大学 NANJING UNIVERSITY

迭代加深搜索

Limit = 0

二叉搜索树上的迭代加深搜索的4次迭代（目标为M），深度界限从0到3

无信息搜索策略

迭代加深搜索

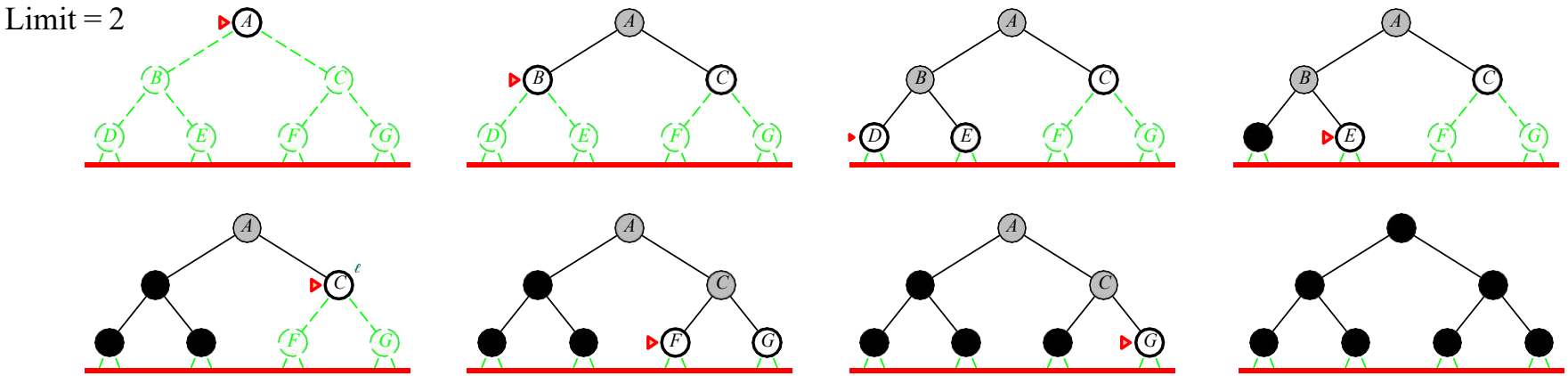
Limit = 1

二叉搜索树上的迭代加深搜索的4次迭代（目标为M），深度界限从0到3

无信息搜索策略

南京大学 NANJING UNIVERSITY

迭代加深搜索

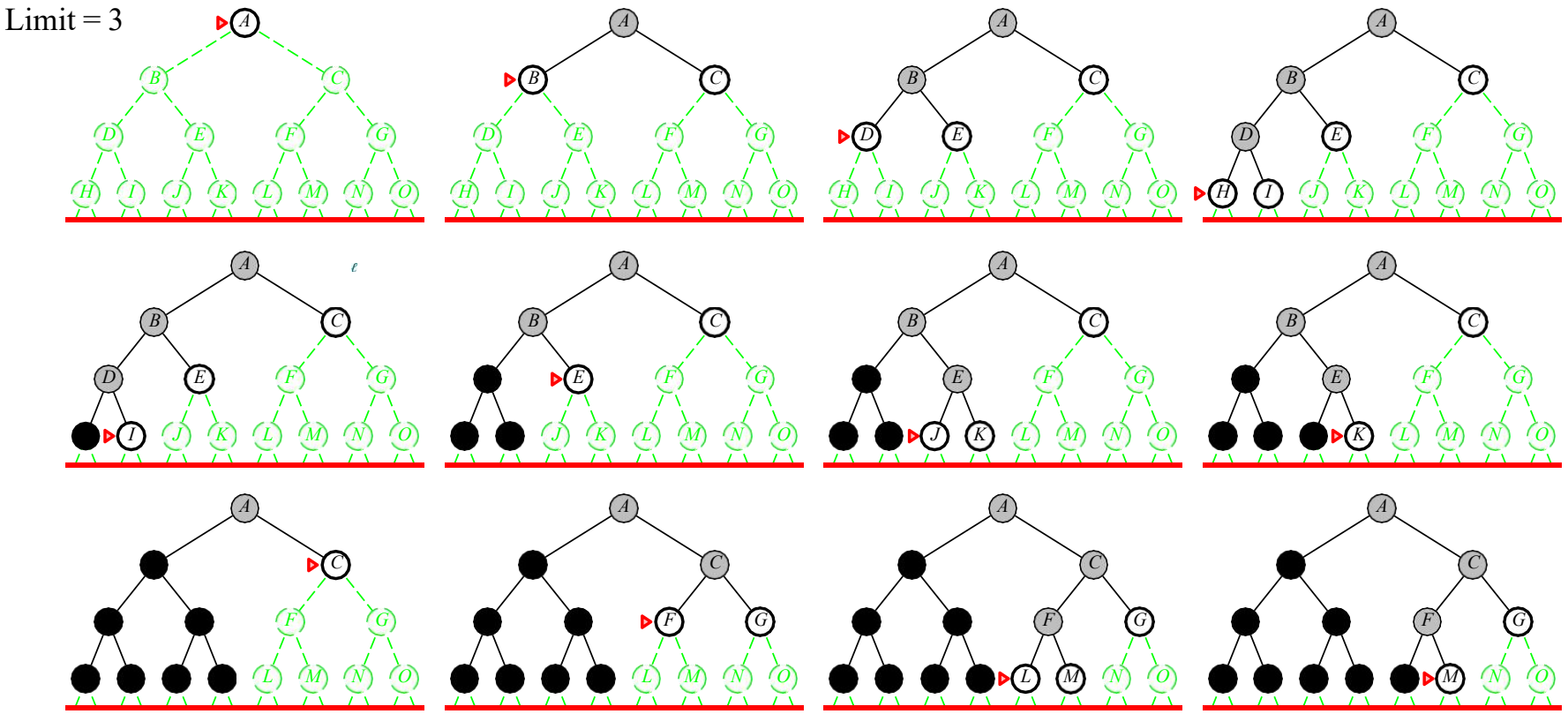


二叉搜索树上的迭代加深搜索的4次迭代（目标为M），深度界限从0到3

无信息搜索策略

南京大学 NANJING UNIVERSITY

迭代加深搜索



二叉搜索树上的迭代加深搜索的4次迭代（目标为M），深度界限从0到3

无信息搜索策略

迭代加深搜索

完备性：是

时间：$(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

空间：$O(b^d)$

代价最优：是（如果动作代价为1，能够被修改来探索代价一致树）

生成节点数的比较，当$b = 10$，$d = 5$:

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,100$$

无信息搜索策略

南京大学 NANJING UNIVERSITY

指标	广度优先	一致代价	深度优先	深度受限	迭代加深	双向（如适用）
完备性	是[1]	是[1,2]	否	否	是[1]	是[1,4]
代价最优	是[3]	是	否	否	是[3]	是[3,4]
时间复杂性	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
空间复杂性	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

注：[1] 如果 b 是有限的且状态空间要么有解要么有限，则算法是完备的。[2] 如果所有动作的代价都$\geqslant \varepsilon > 0$，算法是完备的。[3] 如果动作代价都相同，算法是代价最优的。[4] 如果两个方向均使用广度优先搜索或一致代价搜索

搜索算法比较。b是分支因子；m是搜索树的最大深度；d是最浅层解的深度，当不存在解时为m；l是深度界限

有信息（启发式）搜索策略

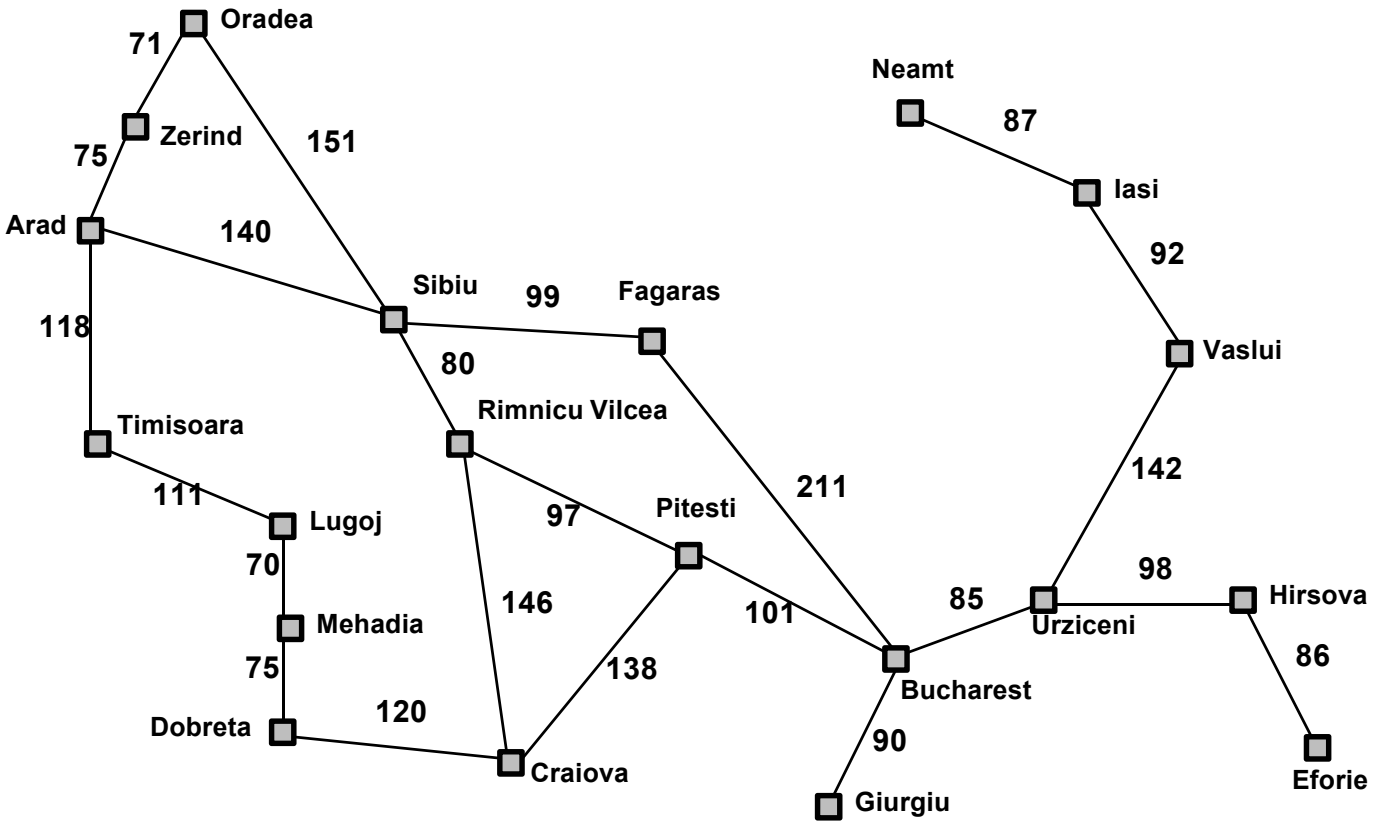
- **有信息搜索（informed search）策略**使用关于目标位置的特定领域线索来比无信息搜索策略更有效地找到解。

- 线索以**启发式函数（heuristic function）**的形式出现，记为h(n)：

  h(n) = 从节点n的状态到目标状态的最小代价路径的代价估计值

例如，在寻径问题中，我们可以通过计算地图上两点之间的直线距离来估计从当前状态到目标的距离。

有信息（启发式）搜索策略



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h_{SLD}（到布加勒斯特的直线距离）的值

有信息（启发式）搜索策略

南京大学 NANJING UNIVERSITY

贪心最佳优先搜索（greedy best-first search）

- 首先扩展h(n)值最小的节点，即看起来最接近目标的节点，因为这样可能可以更快找到解。
- 评价函数f(n) = h(n)

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

h_{SLD}（到布加勒斯特的直线距离）的值

有信息（启发式）搜索策略

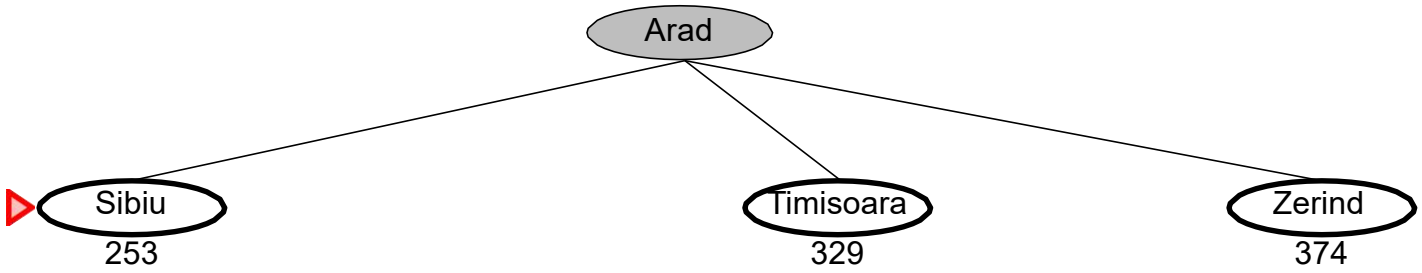
南京大学 NANJING UNIVERSITY

贪心最佳优先搜索（greedy best-first search）

基于直线距离启发式函数h_{SLD}的贪心最佳优先树状搜索的各个阶段（目标为布加勒斯特）。节点上标有h值。

有信息（启发式）搜索策略

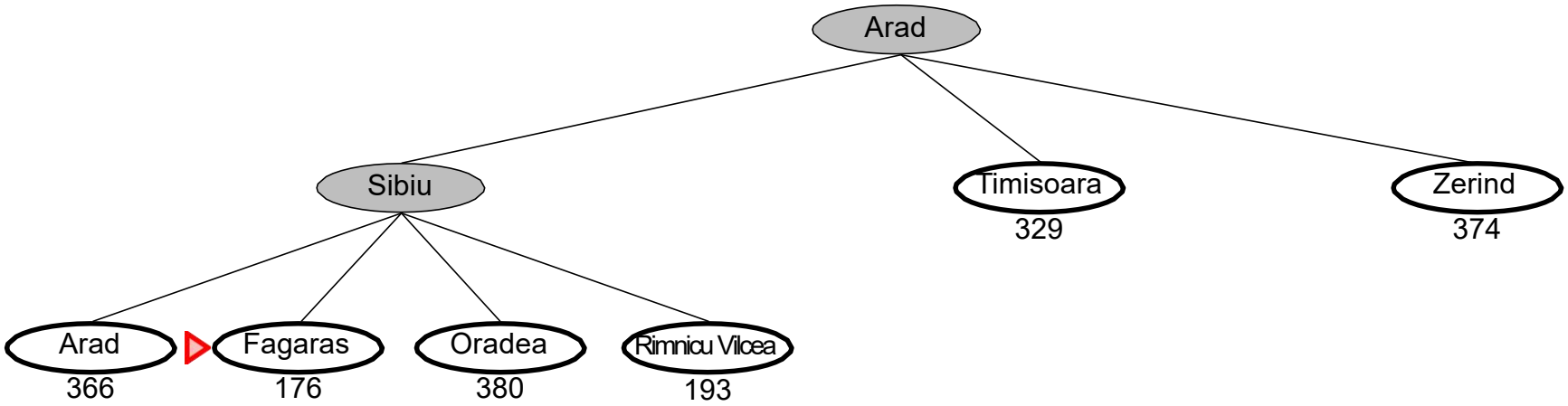
贪心最佳优先搜索



基于直线距离启发式函数h_{SLD}的贪心最佳优先树状搜索的各个阶段（目标为布加勒斯特）。节点上标有h值。

有信息（启发式）搜索策略

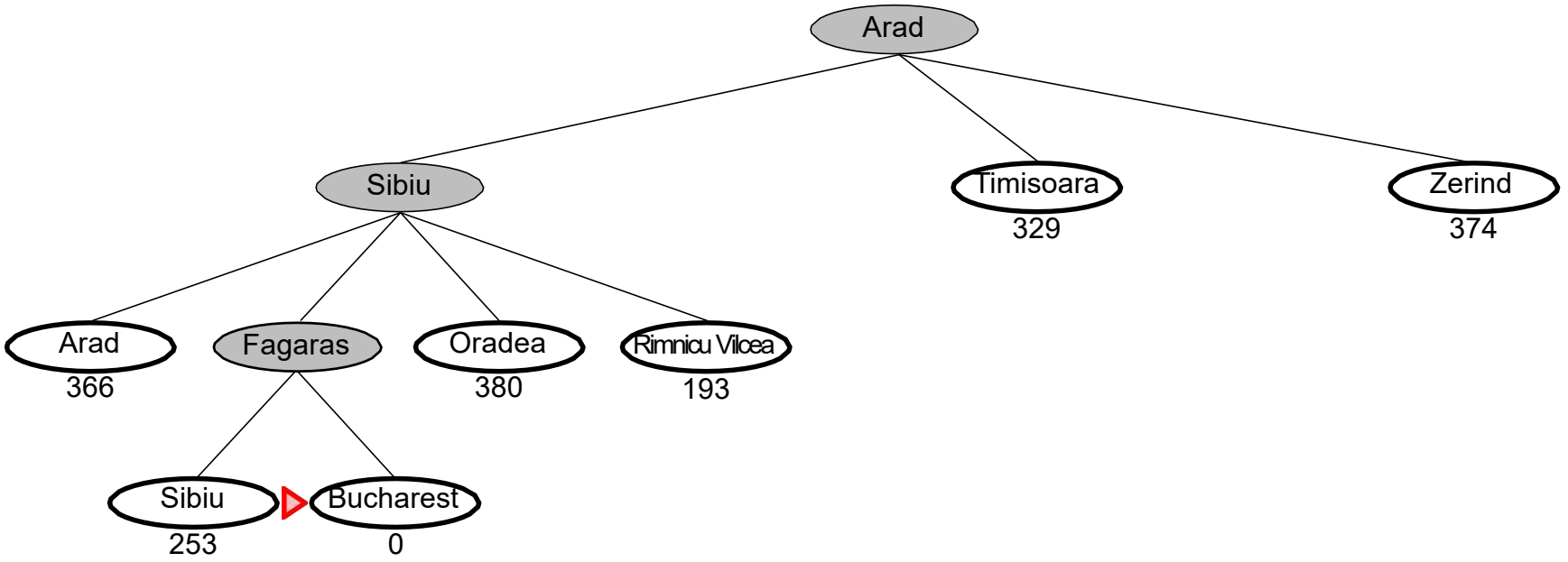
贪心最佳优先搜索



基于直线距离启发式函数h_{SLD}的贪心最佳优先树状搜索的各个阶段（目标为布加勒斯特）。节点上标有h值。

有信息（启发式）搜索策略

贪心最佳优先搜索



基于直线距离启发式函数h_{SLD}的贪心最佳优先树状搜索的各个阶段（目标为布加勒斯特）。节点上标有h值。

有信息（启发式）搜索策略

南京大学 NANJING UNIVERSITY

贪心最佳优先搜索（greedy best-first search）

完备性：否（可能会卡在循环当中，对于有着重复状态检测的有限状态空间是完备的）

时间： $O(b^m)$（一个好的启发式函数可以使复杂性大大降低）

空间： $O(b^m)$（所有节点都存储在内存中）

代价最优：否（如果所有动作都具有相同代价，否则不一定）

有信息（启发式）搜索策略

南京大学 NANJING UNIVERSITY

A*搜索

- 主要思想：避免扩展代价已经很高的路径
- 评价函数f(n) = g(n) + h(n)

  g(n) = 从初始节点到节点n的路径代价
  h(n) = 从节点n的状态到目标状态的最小代价路径的代价估计值
  f (n) = 经过n到一个目标状态的最优路径的代价估计值

- 对于**可容许的启发式（admissible heuristic）**函数，A*搜索是代价最优的，即

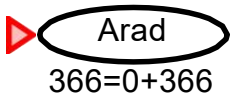
$$h(n) \le h^*(n)$$

这里 h*(n) 经过节点n到目标状态的真实代价．(h(n) ≥ 0, 对于任意目标G，h(G) = 0.)

有信息（启发式）搜索策略

南京大学 NANJING UNIVERSITY

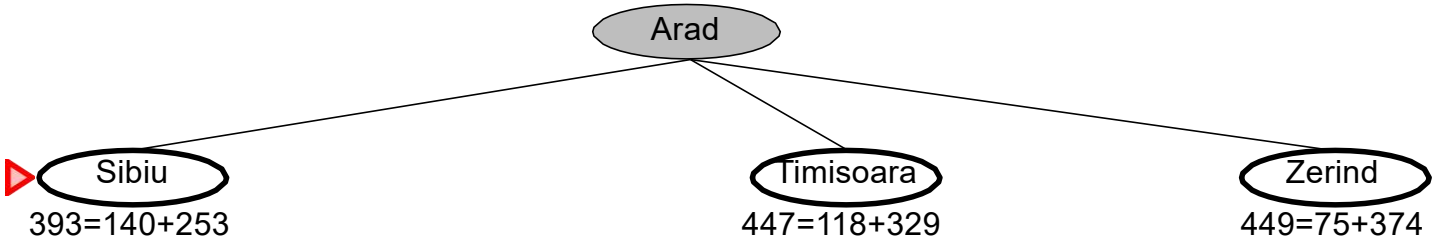
A*搜索



A*搜索的各个阶段（目标为Bucharest）。节点上标有$f = g + h$，h值为到Bucharest的直线距离

有信息（启发式）搜索策略

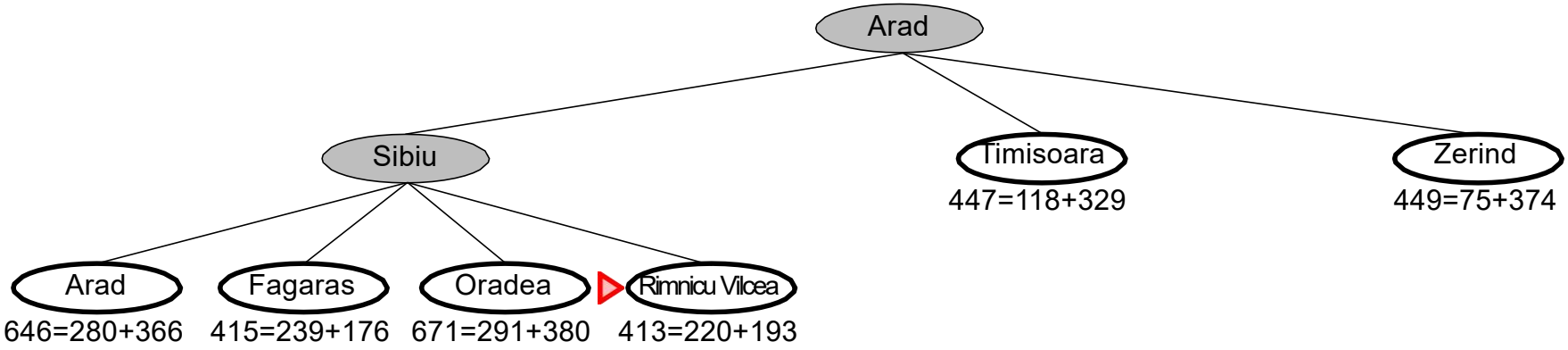
A*搜索



A*搜索的各个阶段（目标为Bucharest）。节点上标有$f = g + h$，h值为到Bucharest的直线距离

有信息（启发式）搜索策略

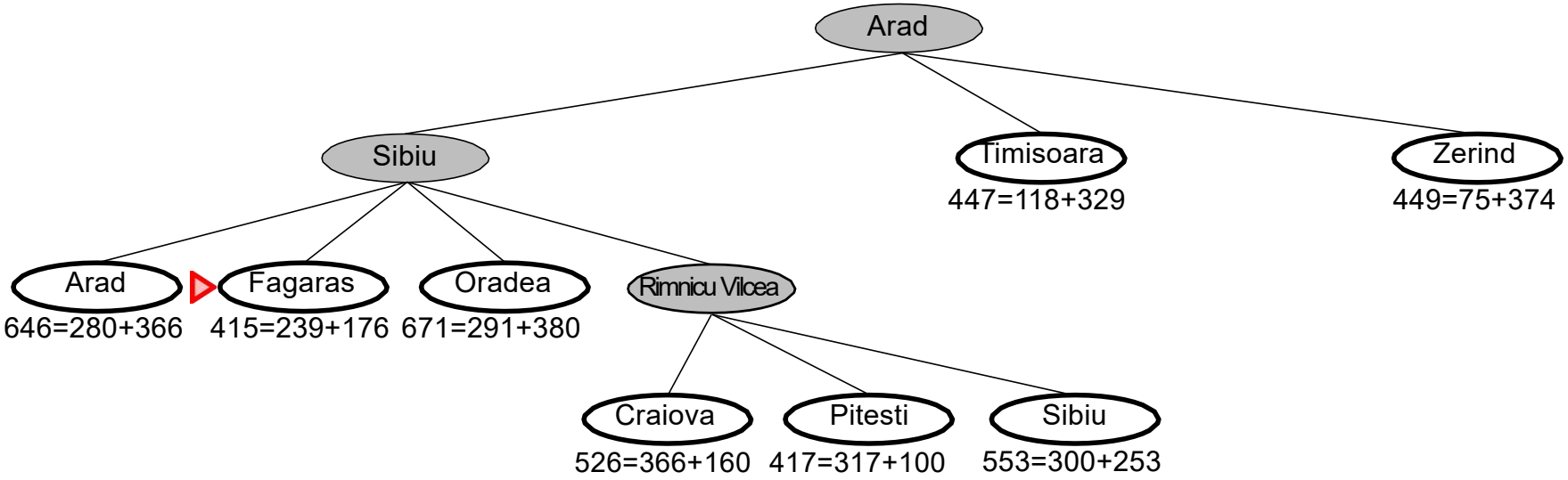
A*搜索



A*搜索的各个阶段（目标为Bucharest）。节点上标有$f = g + h$，h值为到Bucharest的直线距离

有信息（启发式）搜索策略

A*搜索

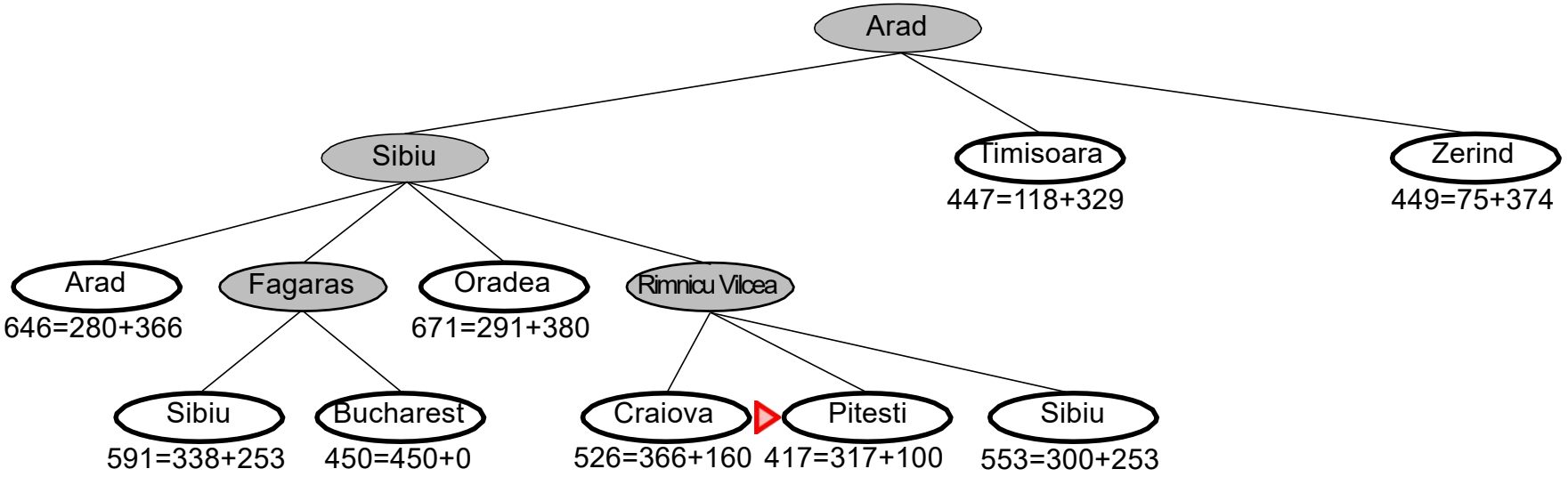


A*搜索的各个阶段（目标为Bucharest）。节点上标有$f = g + h$，h值为到Bucharest的直线距离

有信息（启发式）搜索策略

南京大学 NANJING UNIVERSITY

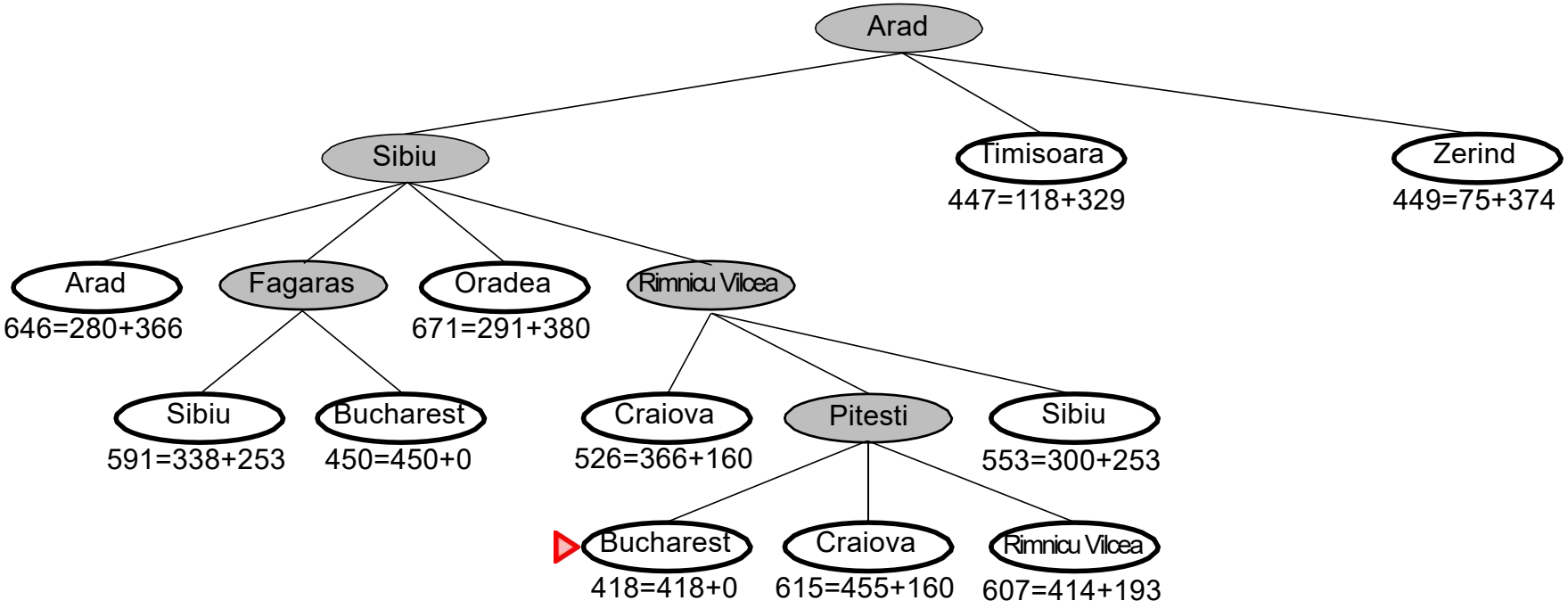
A*搜索



A*搜索的各个阶段（目标为Bucharest）。节点上标有$f = g + h$，h值为到Bucharest的直线距离

有信息（启发式）搜索策略

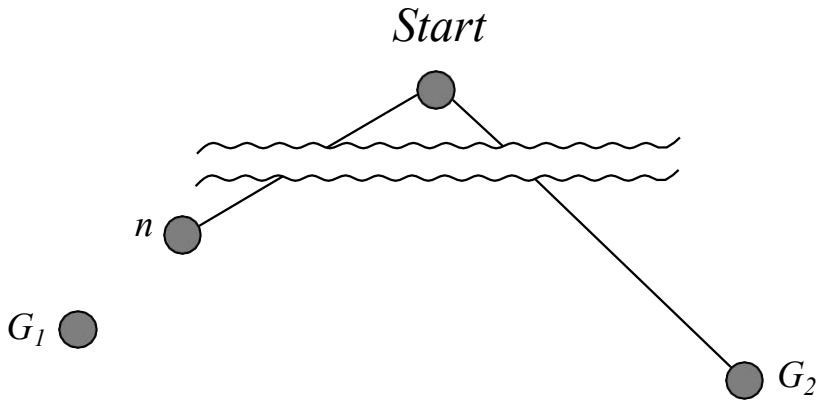
A*搜索



A*搜索的各个阶段（目标为Bucharest）。节点上标有$f = g + h$，h值为到Bucharest的直线距离

A*搜索

假设某个次优的目标节点G_2已经被生成并且在边界队列当中．令 n 是一个在到一个最优目标节点G_1的最短路径上未被扩展的节点：



$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{因为} h(G_2) = 0 \\
&> g(G_1) && \text{因为 } G_2 \text{ 是次优的} \\
&\geq f(n) && \text{因为 } h \text{ 是可容许的}
\end{aligned}
$$

因为 $f(G_2) > f(n)$，A* 一定不会选择 G_2 来扩展

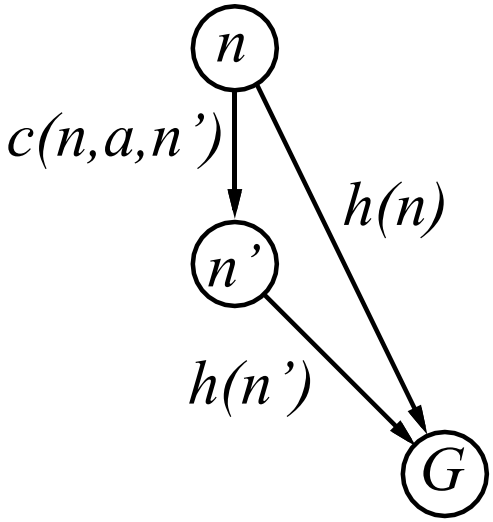
A*搜索

启发式函数是一致的，如果

$$h(n) \le c(n, a, n') + h(n')$$

如果 h 是一致的，我们有

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\ge g(n) + h(n) \\ &= f(n) \end{aligned}$$

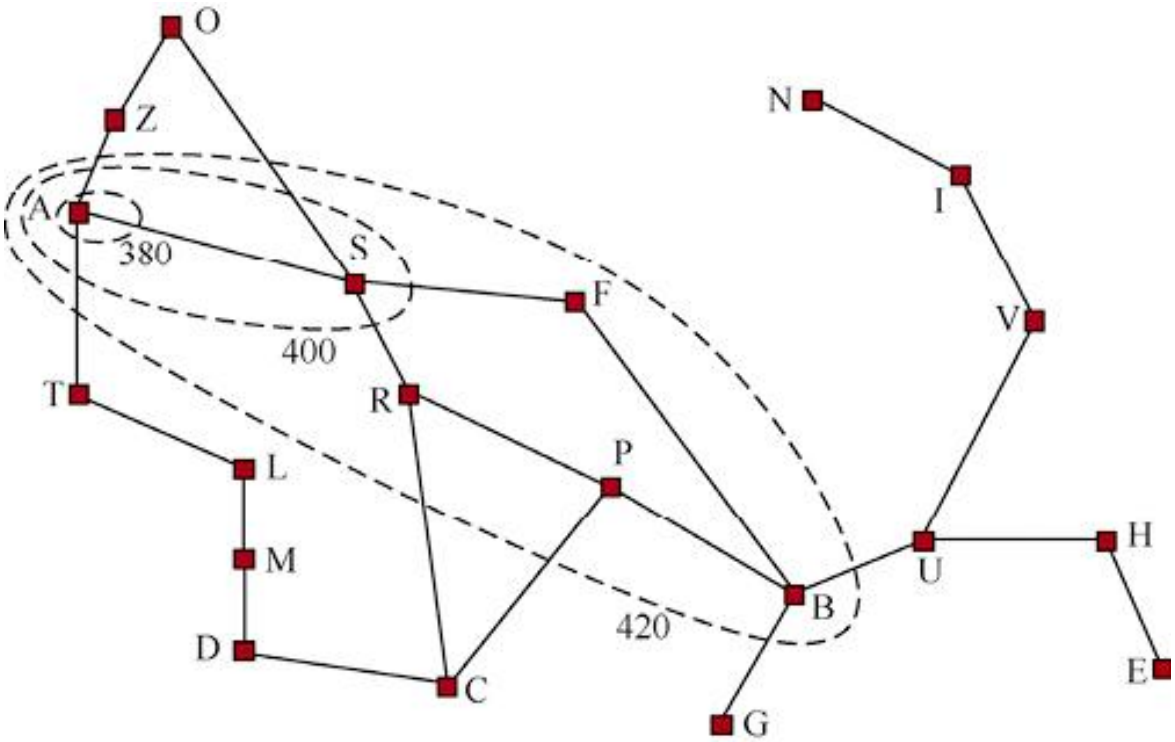


即，评价函数 $f(n)$ 沿着任意路径是单调递增的

有信息（启发式）搜索策略

南京大学 NANJING UNIVERSITY

搜索等值线



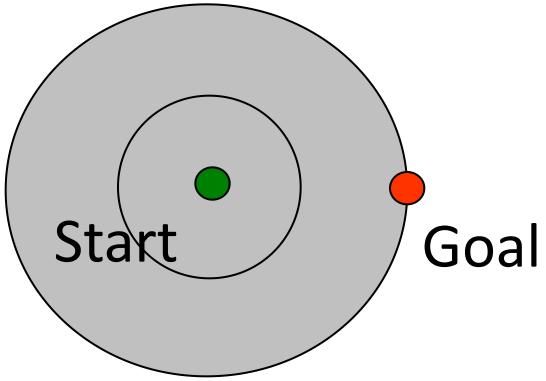
罗马尼亚地图，其中等值线为f = 380、f = 400和f = 420，初始状态为阿拉德。给定等值线内的节点的代价f = g + h小于或等于等值线值。

有信息（启发式）搜索策略

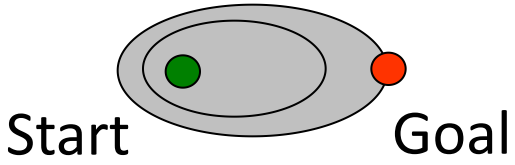
南京大学 NANJING UNIVERSITY

搜索等值线

一致代价搜索中，等值线将以初始状态为圆心呈“圆形”向各个方向均匀扩展。



对于具有好的启发式函数的A*搜索，等值线将朝目标状态延伸，并在最优路径周围收敛变窄。



有信息（启发式）搜索策略

加权A*搜索

f(n) = g(n) + W*h(n)，这里W > 1

同一网格上的两种搜索：（a）A*搜索，（b）加权A*搜索，权重 $W = 2$。灰色线条表示障碍，紫色线是一条从绿色起始点到红色目标点的路径，较小的点是每次搜索到达的状态。

有信息（启发式）搜索策略

A*搜索：$g(n)+h(n)$ （$W=1$）

一致代价搜索：$g(n)$ （$W=0$）

贪心最佳优先搜索：$h(n)$ （$W=+\infty$）

加权A*搜索：$g(n)+W\times h(n)$ （$1<W<+\infty$）

有信息（启发式）搜索策略

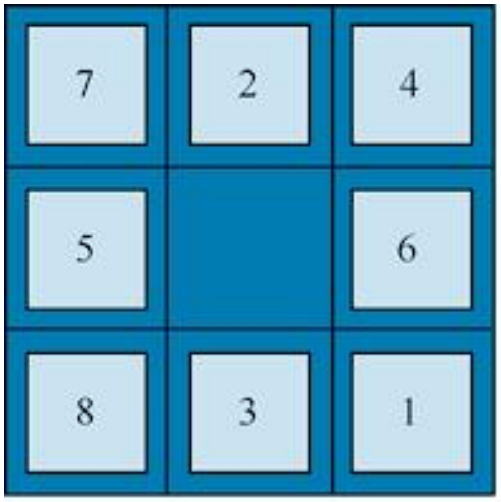
南京大学 NANJING UNIVERSITY

启发式函数

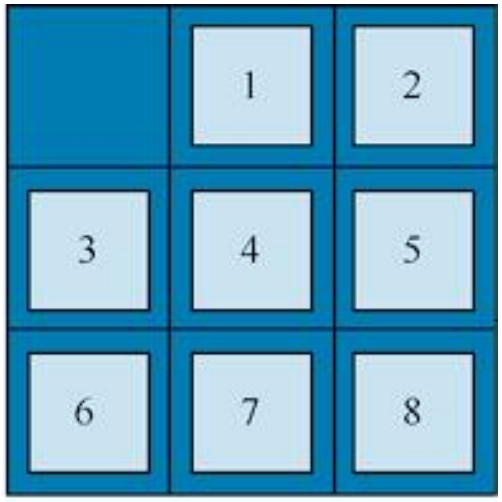
对于8数码问题，两个常用的启发式函数为:

$h_1(n)$ = 错位滑块的数量
$h_2(n)$ = 总的曼哈顿距离

$h_1(S)$ = 8
$h_2(S)$ = 3+1+2+2+2+3+3+2 = 18



开始状态



目标状态

8数码问题的典型实例。最短的解需要26步动作。

有信息（启发式）搜索策略

南京大学 NANJING UNIVERSITY

启发式函数的占优（Dominance）

	搜索代价：生成的节点数			有效分支因子		
d	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17 270	1683	364	1.74	1.48	1.32
18	41 558	4102	751	1.72	1.49	1.34
20	91 493	9905	1318	1.69	1.50	1.34
22	175 921	22 955	2548	1.66	1.50	1.34
24	290 082	53 039	5733	1.62	1.50	1.36
26	395 355	110 372	10 080	1.58	1.50	1.35
28	463 234	202 565	22 055	1.53	1.49	1.36

如果对于任意节点n， $h_2(n) \geq h_1(n)$，那么 h_2 占优于 h_1 并且更有利于搜索。

给定任意可容许的启发函数 h_a, h_b,

$h(n) = \max(h_a(n), h_b(n))$ 也是可容许的并且占有于 h_a, h_b

有信息（启发式）搜索策略

从松弛问题出发生成启发式函数

- 松弛问题中最优解的代价可以作为原问题的一个可容许的启发式函数
- 关键点：松弛问题最优解的代价不大于原问题最优解的代价

小结

- 问题由5部分组成：初始状态、动作集合、描述这些动作结果的转移模型、目标状态集合和动作代价函数。

- 问题的环境用**状态空间图**表示。通过状态空间（一系列动作）从初始状态到达一个目标状态的路径是一个解。

- 搜索算法通常将状态和动作看作原子的，即没有任何内部结构（尽管我们在学习时引入了状态特征）。

- 根据**完备性**、**代价最优性**、**时间复杂性**和**空间复杂性**来评估搜索算法。

- **无信息搜索方法**只能访问问题定义。算法构建一棵搜索树，试图找到一个解。算法会根据其首先扩展的节点而有所不同。

- **有信息搜索方法**可以访问启发式函数h(n)来估计从n到目标的解代价。

- **启发式搜索算法**的性能取决于启发式函数的质量。我们有时可以通过松弛问题定义、在模式数据库中存储预计算的子问题的解代价、定义地标点，或者从问题类的经验中学习来构建良好的启发式函数。

提纲

南京大学 NANJING UNIVERSITY

- 通过搜索进行问题求解
  - 问题求解智能体
  - 搜索算法
  - 无信息搜索策略
  - 有信息（启发式）搜索策略
- **复杂环境中的搜索**
  - 局部搜索和最优化问题
  - 连续空间中的局部搜索

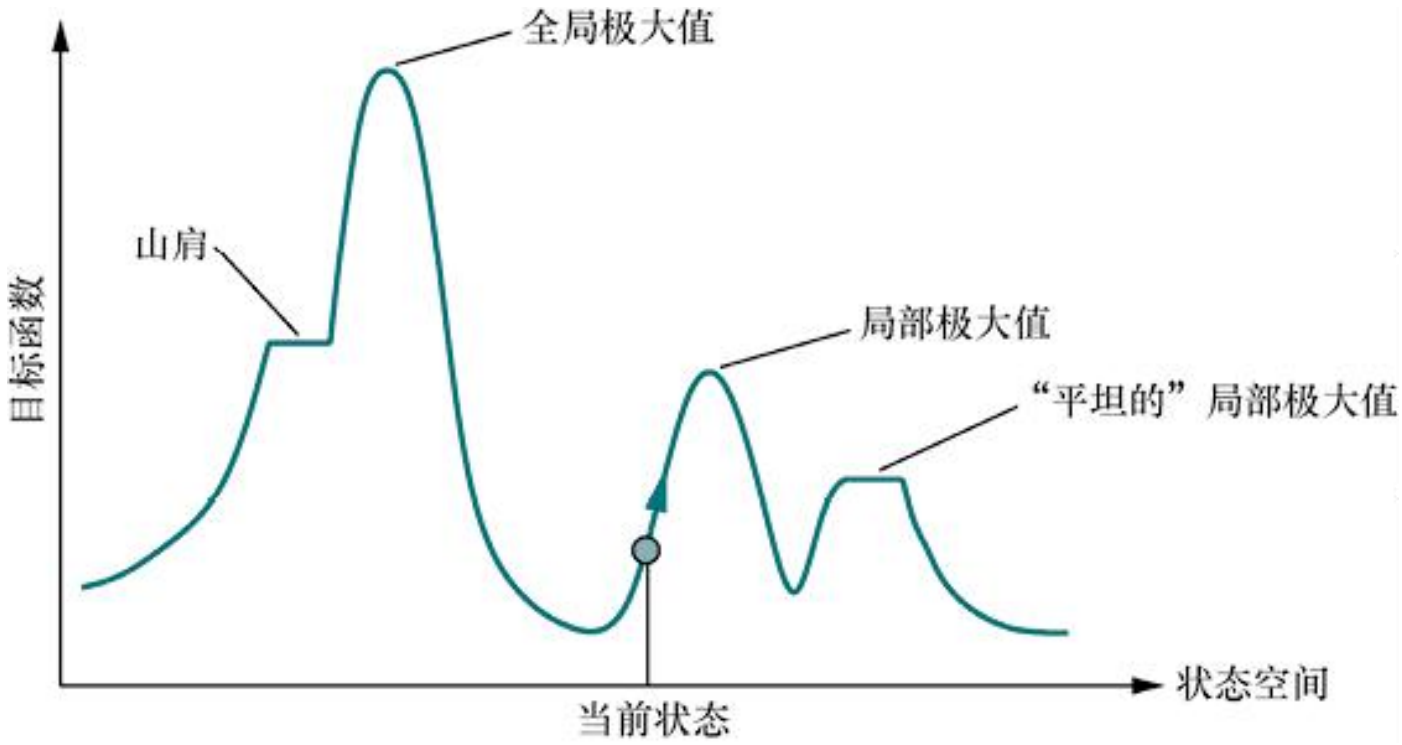
局部搜索和最优化问题

- 在许多优化问题中，我们只关心最终状态，而不是到达状态的路径。在这种情况下我们可以采用迭代优化算法，只记录当前状态，并尝试去优化它。

- **局部搜索算法**是从一个起始状态搜索到其相邻状态，它不记录路径，也不记录已达状态集。这意味着它们不是系统性的——可能永远不会探索问题的解实际所在的那部分搜索空间。

- 两个主要优点：
（1）使用很少的内存；
（2）通常可以在系统性算法不适用的大型或无限状态空间中找到合理的解。

局部搜索和最优化问题



一维状态空间地形图，其标高对应于目标函数。目的是找到全局极大值

局部搜索和最优化问题

南京大学 NANJING UNIVERSITY

爬山搜索

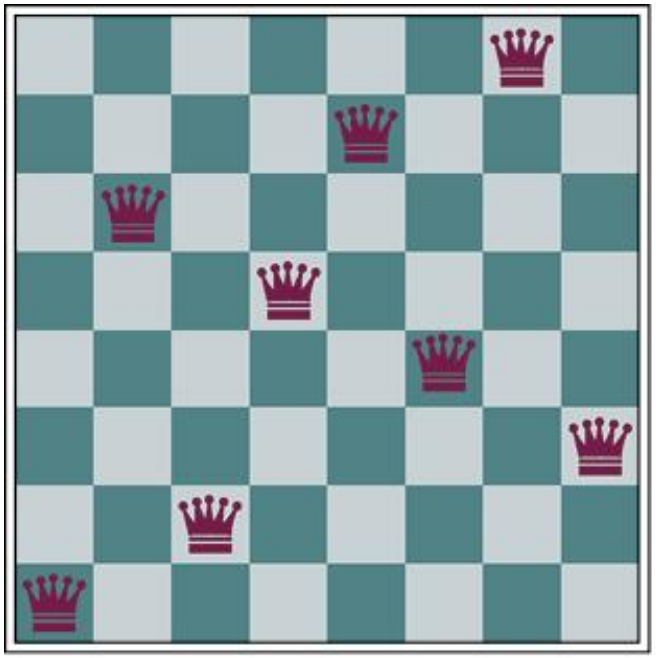
```
function HILL-CLIMBING(problem) returns 一个位于局部极大值的状态
  current ← problem.INITIAL
  while true do
    neighbor ← current的值最大的后继状态
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

爬山搜索算法是最基本的局部搜索技术。在每一步中，当前节点被其最优邻居节点替换。

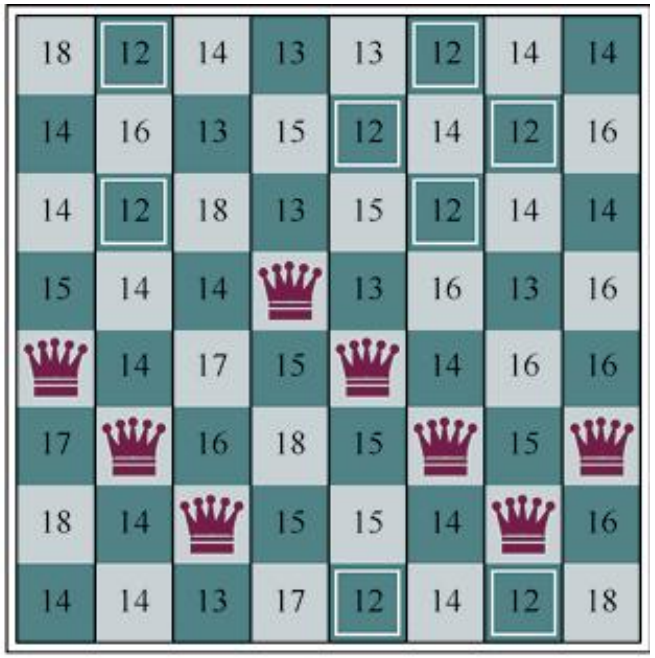
局部搜索和最优化问题

南京大学 NANJING UNIVERSITY

爬山搜索



(a)



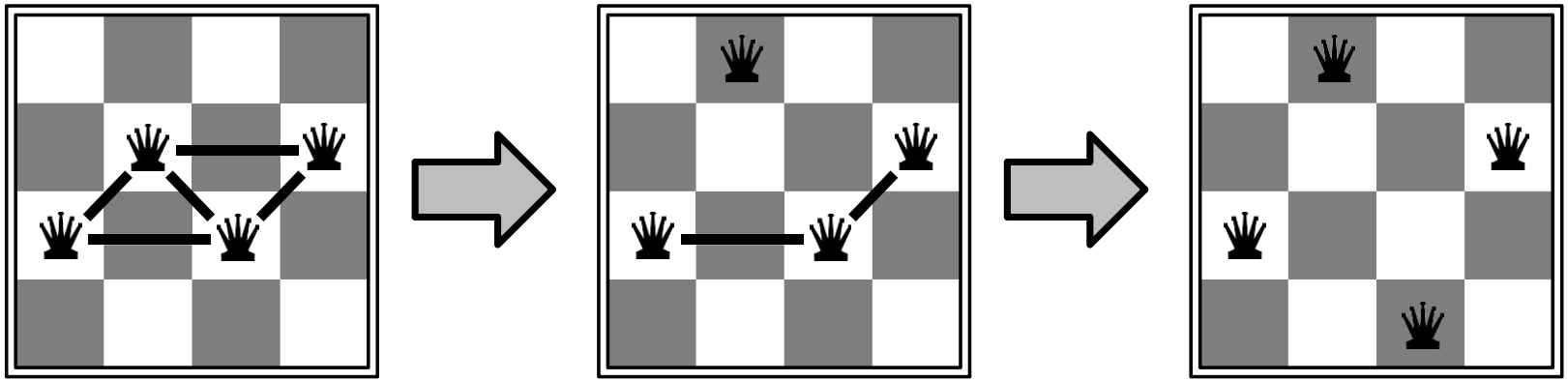
(b)

（a）8皇后问题：在棋盘上放置8个皇后，使得它们不能互相攻击。（皇后会攻击同一行、同一列或对角线上的任何棋子。）当前状态非常接近于一个解，除了第4列和第7列的两个皇后会沿对角线互相攻击。（b）一个8皇后状态，其启发式代价估计值h=17。棋盘显示了通过在同一列移动皇后而获得的每一个可能后继的h值。有8个移动并列最优，其h=12。爬山法将选择它们中的一个。

局部搜索和最优化问题

南京大学 NANJING UNIVERSITY

爬山搜索



h = 5　　h = 2　　h = 0

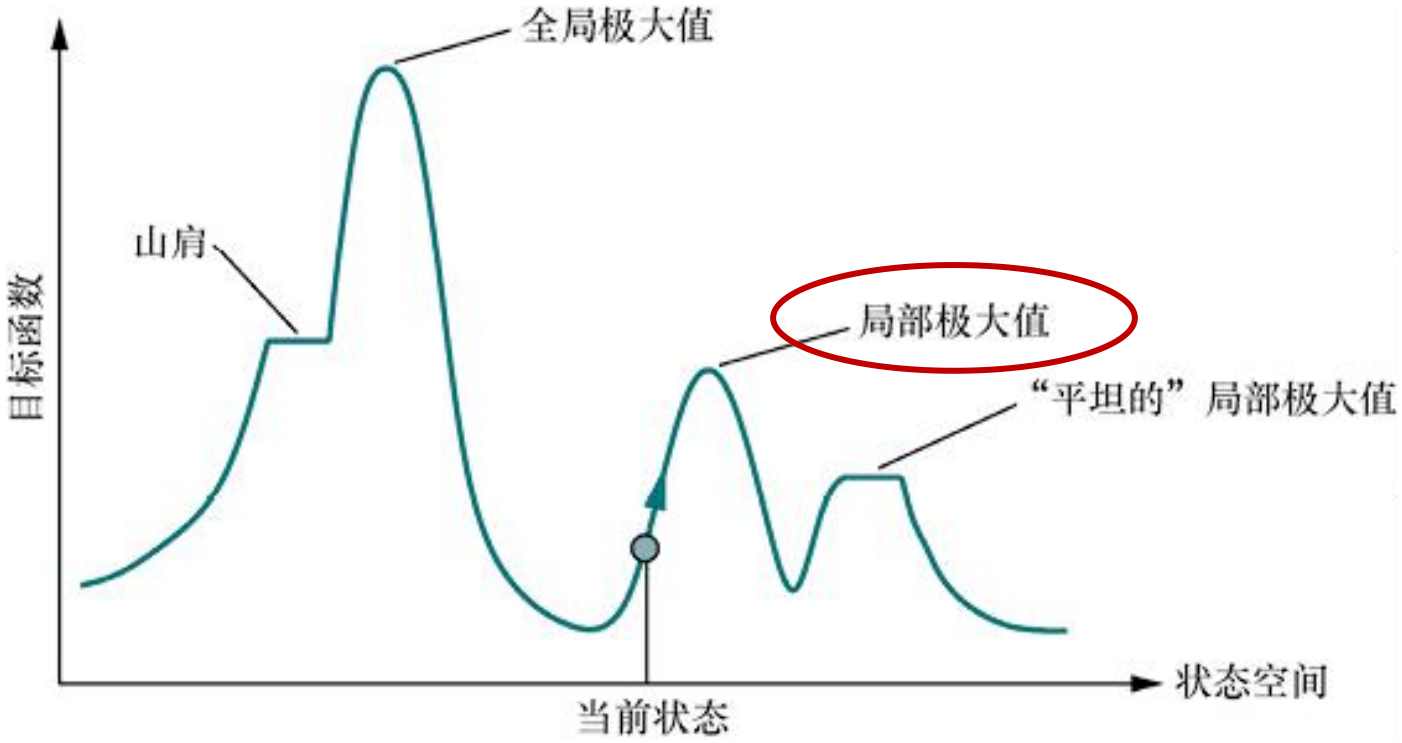
移动一个皇后来减少可相互攻击的皇后对的数量。

局部搜索和最优化问题

南京大学 NANJING UNIVERSITY

爬山搜索

爬山法可能会由于以下原因而陷入困境：

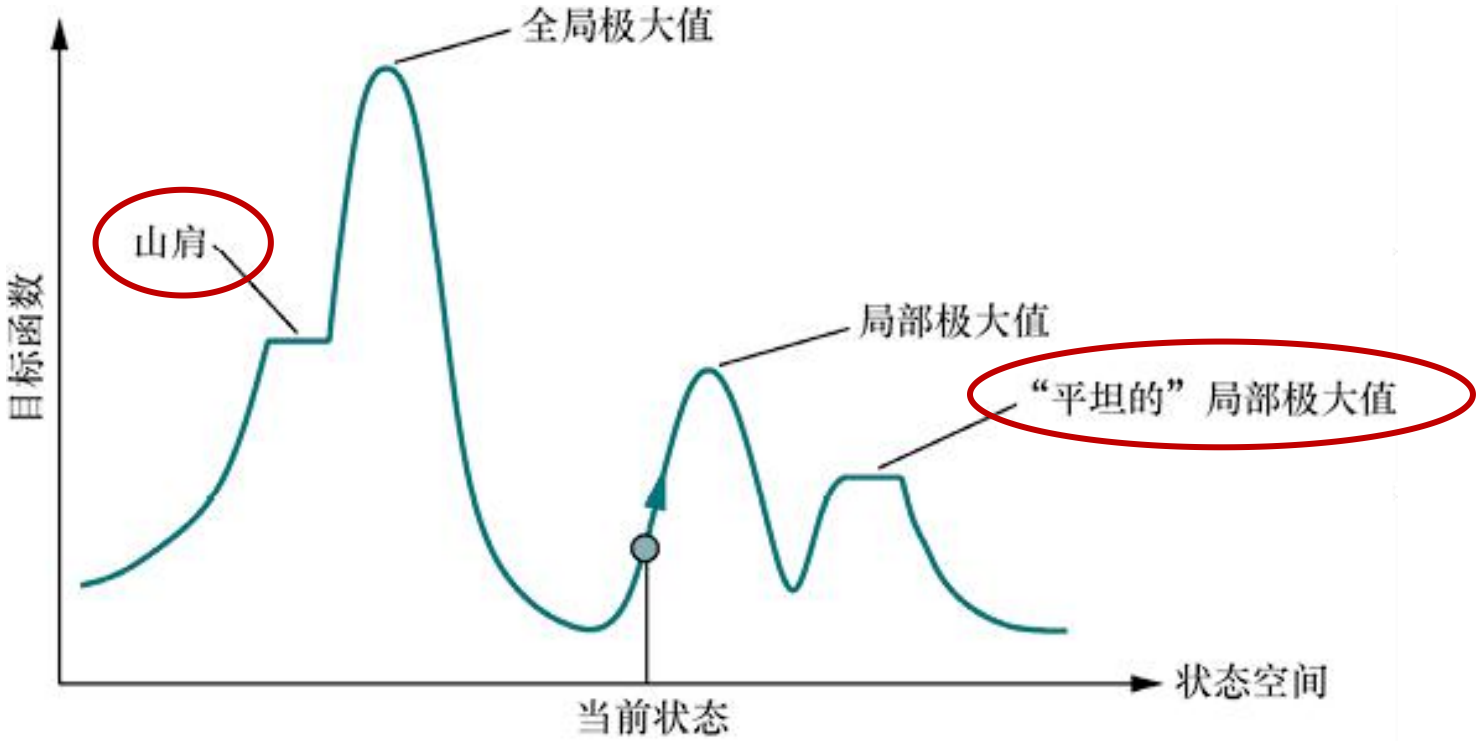


局部搜索和最优化问题

南京大学 NANJING UNIVERSITY

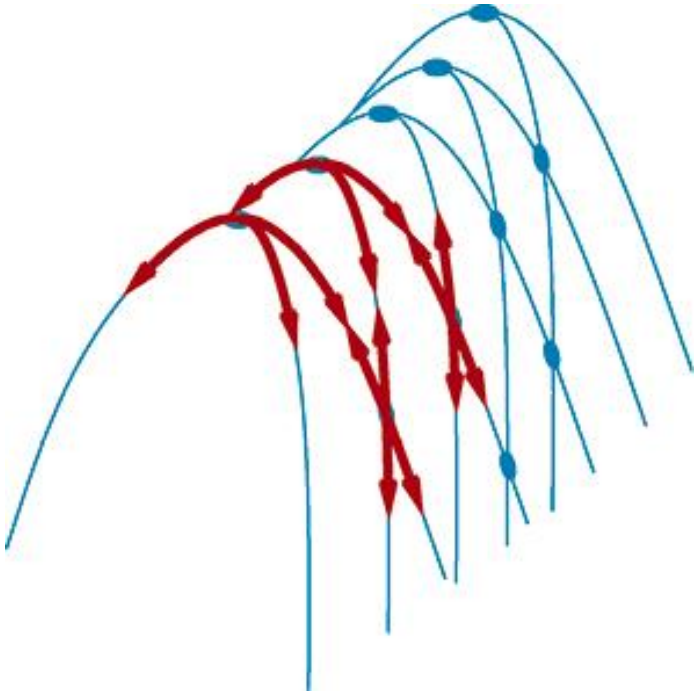
爬山搜索

爬山法可能会由于以下原因而陷入困境：



爬山搜索

爬山法可能会由于以下原因而陷入困境：



岭（ridge）为爬山法带来困难的示意图。

局部搜索和最优化问题

南京大学 NANJING UNIVERSITY

模拟退火（simulated annealing）

```
function SIMULATED-ANNEALING(problem, schedule) returns 一个解状态
  current ← problem.INITIAL
  for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current
    next ← current的一个随机选择的后继状态
    ΔE ← VALUE(current) – VALUE(next)
    if ΔE > 0 then current ← next
    else current ← next仅以e^(ΔE/T)的概率
```

模拟退火算法，一种允许某些下坡移动的随机爬山法。输入的schedule是关于时间的函数，它决定了“温度”T的值

连续空间中的局部搜索

假设我们希望在罗马尼亚新建3个机场：

- 6-D状态空间定义为 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$
- 目标函数 $f(x)$ = 每一个城市到其最近机场的均方距离之和

$$f(\boldsymbol{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

- **梯度(gradient)**法计算

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

通过 $\mathrm{x} \leftarrow \mathrm{x} + \alpha \nabla f(\mathrm{x})$来增加或减少$f$。

- **牛顿-拉弗森法**迭代更新 $\mathrm{x} \leftarrow \mathrm{x} - \mathrm{H_f^{-1}}(\mathrm{x}) \nabla f(\mathrm{x})$ 来去求解

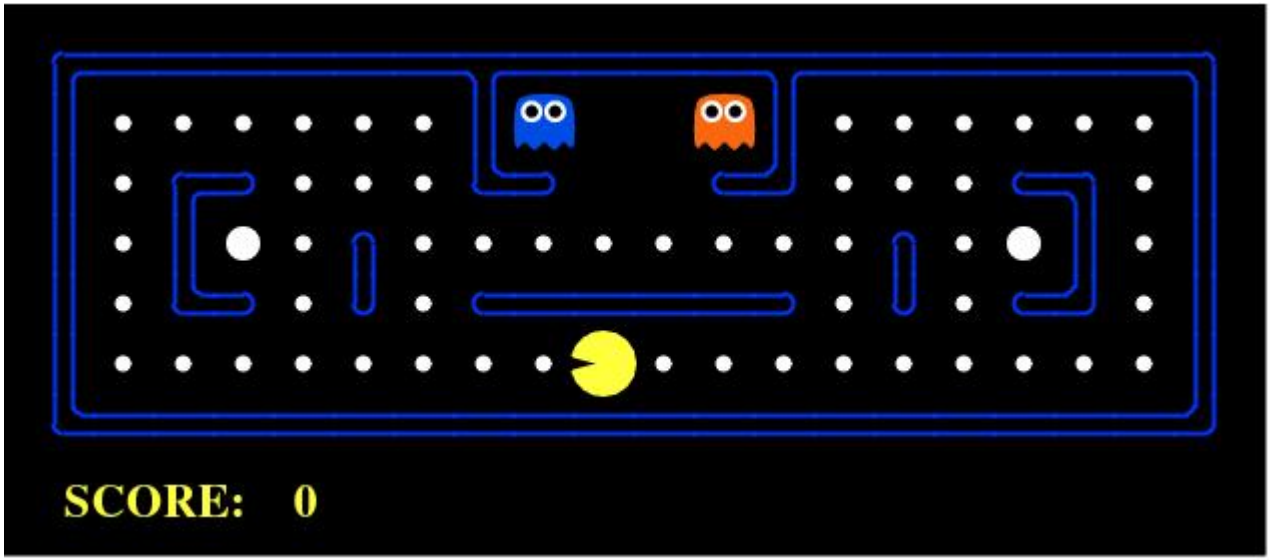
$\nabla f(\mathrm{x}) = 0$, 这里 $\mathrm{H_{ij}} = \partial^2 f / \partial x_\mathrm{i} \partial x_\mathrm{j}$

其中$\mathrm{H_f(x)}$为二阶导数的**黑塞矩阵**（Hessian matrix）

课程作业概况

基于UC Berkeley CS188 Intro to AI课程的**Pac-Man开源项目**：

https://inst.eecs.berkeley.edu/~cs188/sp23/projects/



该项目以《人工智能：现代方法》这一教材为蓝本设计，对于理解书中的知识点非常有帮助。