课程《人工智能原理与技术》

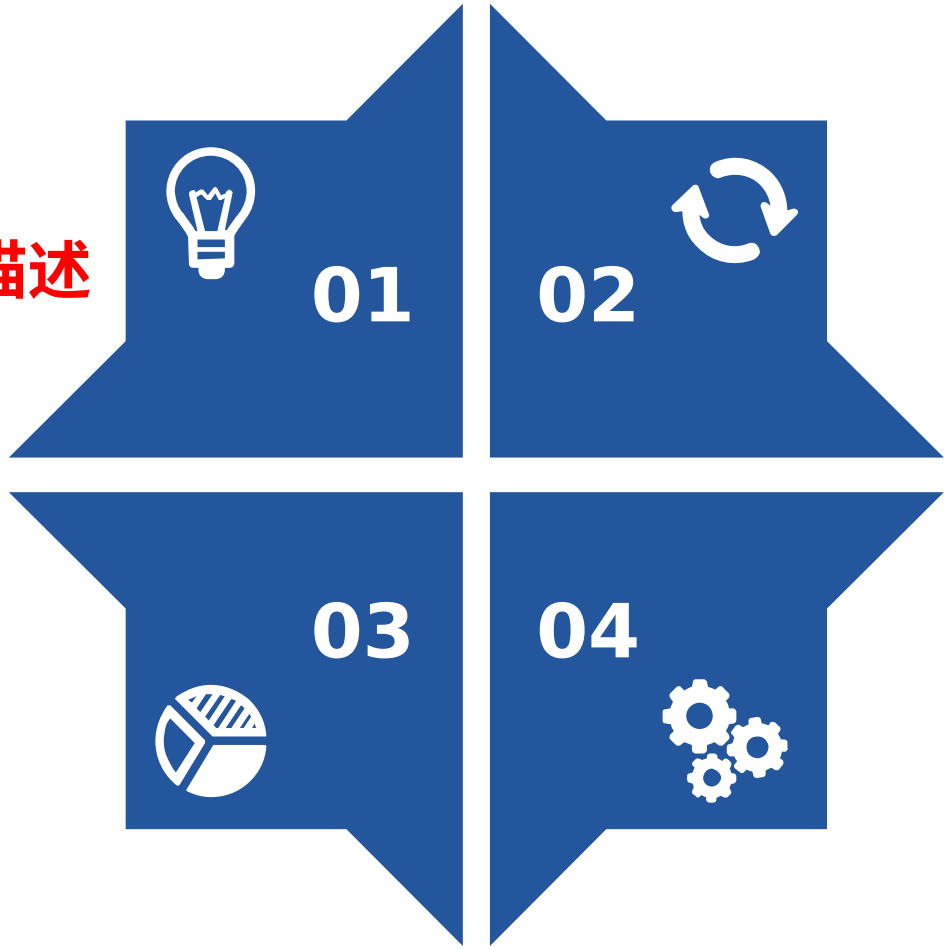
同济大学 TONGJI UNIVERSITY

第三章搜索探寻与问题求解

搜索树构建与启发式搜索

同济大学 TONGJI UNIVERSITY

目录



搜索的形式化描述

搜索树构建

启发式搜索

性能分析和案例

故记诵者，学问之舟车也

—— 清 · 章学诚《文史通义》

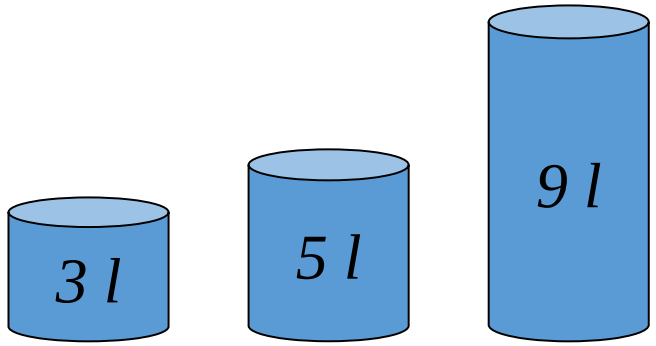
“学问之始，未能记诵；博涉既深，将趋记诵；故记诵者，学问之舟车也”。背诵是学问的基本功，犹如搜索算法从庞大解空间中搜索得到与问题相对应、满足一定约束条件的最佳答案，寻找一条从初始状态（问题所处状态）到目标状态（答案所处状态）的最佳路径。

搜索的形式化描述

问题导入

测量问题：

问题： 用 3 个杯子，测量出 7 升水



搜索的形式化描述

同济大学 TONGJI UNIVERSITY

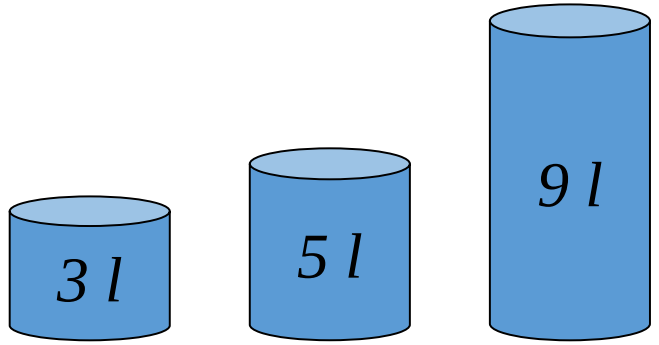
- **Solution 1:**

a	b	c
0	0	0
start		
3	0	0
0	0	3
3	0	3
0	0	6
3	0	6
0	3	6
3	3	6
1	5	6
0	5	**7**
goal		

- **Solution 2:**

a	b	c
0	0	0
start		
0	5	0
3	2	0
3	0	2
3	5	2
3	**0**	**7**
goal		

哪一个更优?



搜索的形式化描述

同济大学 TONGJI UNIVERSITY

＜状态、动作、状态转移、路径 / 代价、目标测试〉

状态（state）

广义来说状态是对搜索算法和搜索环境当前所处情形的描述信息。在左图中，搜索算法要从城市 A 出发找到一条到城市 K 最短路线，**影响其下一步行动的信息只有算法当前所位于的城市**，因此每个城市即为一个状态。搜索算法刚开始所在状态（即城市 A ）被称为初始状态，完成任务时所在状态（即城市 K ）被称为终止状态。

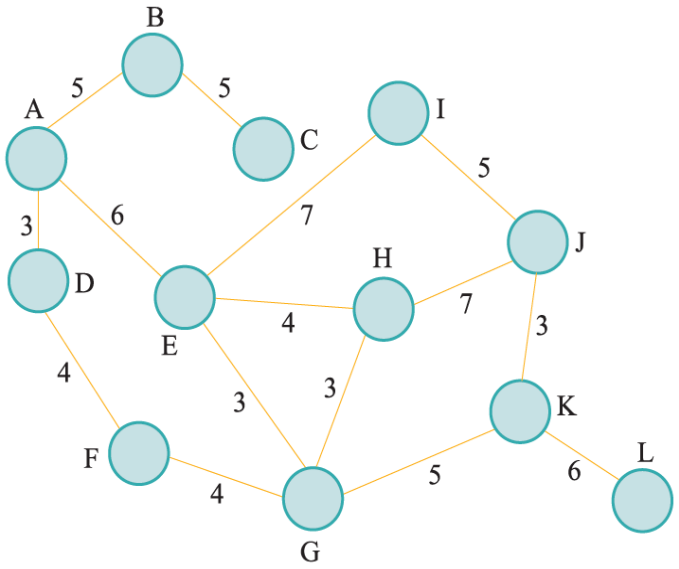


图 3.1　某地区的交通网络图

以求取从城市 A 到城市 K 之间一条行驶时间最短路线为例

搜索的形式化描述

同济大学 TONGJI UNIVERSITY

< 状态、动作、状态转移、路径 / 代价、目标测试〉

动作（ action ）

为了完成最短路径的探索，算法需要不断从一个状态转移到下一个状态。**算法从一个状态转移到另外一个状态所采取的行为被称为动作。**在这个问题中，动作即为乘坐一趟列车从当前城市到下一城市。显然在这个问题中，任何状态下可选择的动作都是离散的、而非连续。

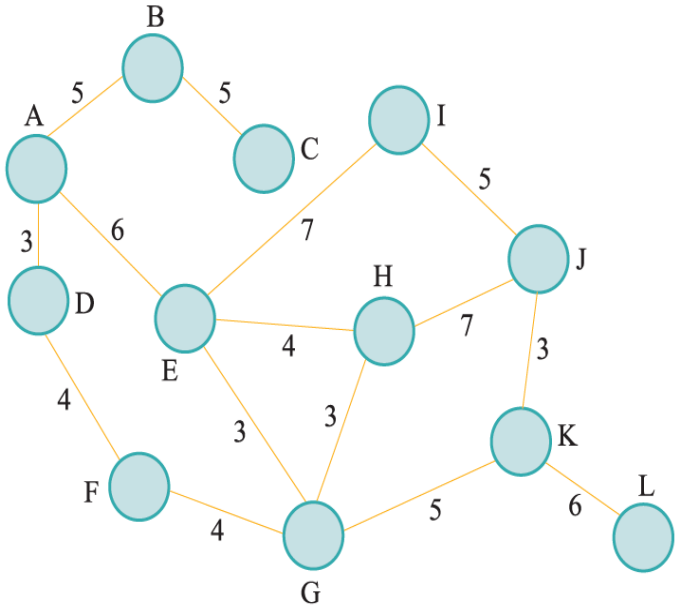


图 3.1　某地区的交通网络图

以求取从城市 A 到城市 K 之间一条行驶时间最短路线为例

搜索的形式化描述

同济大学 TONGJI UNIVERSITY

＜状态、动作、状态转移、路径 / 代价、目标测试〉

状态转移（state transition）

算法选择了一个动作之后，其所处状态也会发生相应变化，这个过程被称为状态转移。如算法在当前城市 A 选择乘上开往邻接城市 D 的列车，那么将发生从状态 A 到状态 D 的状态转移。

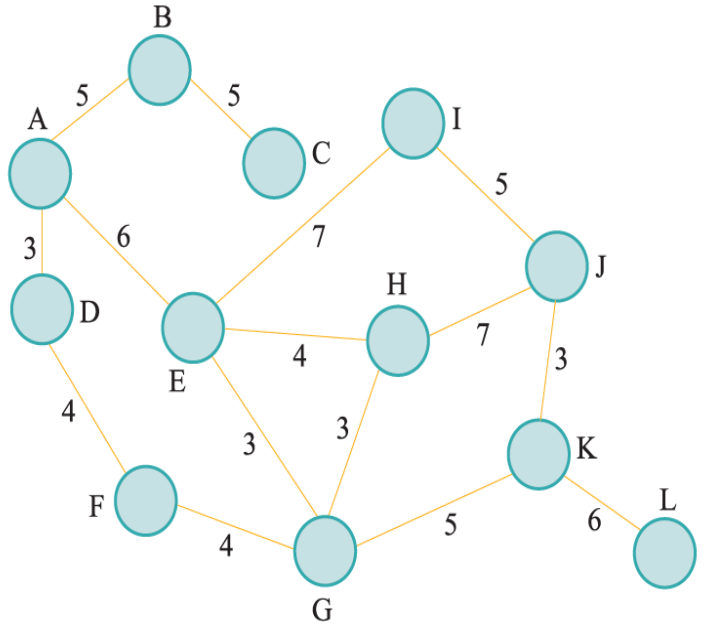


图 3.1　某地区的交通网络图

以求取从城市 A 到城市 K 之间一条行驶时间最短路线为例

搜索的形式化描述

＜状态、动作、状态转移、路径 / 代价、目标测试〉

路径（path）和代价（cost）

搜索过程中可以得到一个状态序列，这个状态序列被称为一条路径。如从状态A出发，经过状态E、G，最终到达状态K，就形成了A → E → G → K这一路径。**每条路径对应一个代价，图中路径的代价就是通过该路径的时间开销**，图中每条边上的数值即为单步代价取值。

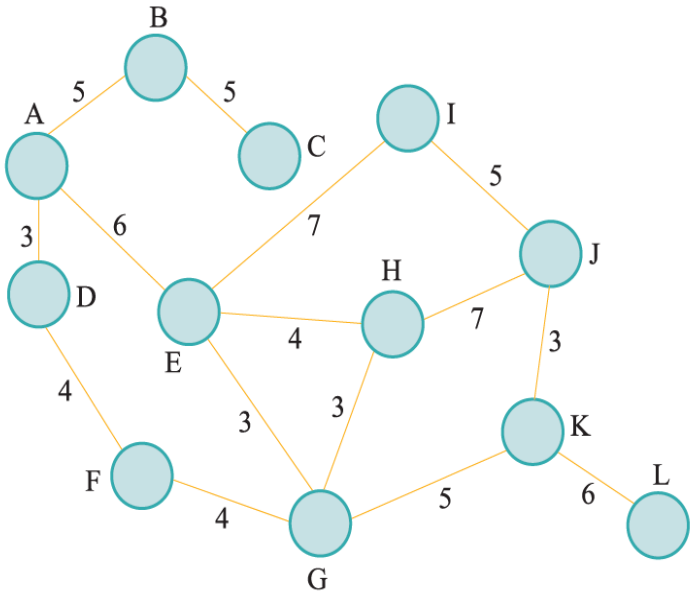


图 3.1　某地区的交通网络图

以求取从城市 A 到城市 K 之间一条行驶时间最短路线为例

搜索的形式化描述

同济大学 TONGJI UNIVERSITY

< 状态、动作、状态转移、路径 / 代价、目标测试〉

目标测试（goal test）

目标测试函数goal_test(s)用于判断状态s是否为目标状态。

在图中，如果当前状态为K，那么目标测试通过，否则不通过。

当然，寻找所得的路径（从初始状态到终止状态）是否满足“路径最短”这一约束条件，还需要另行判断。

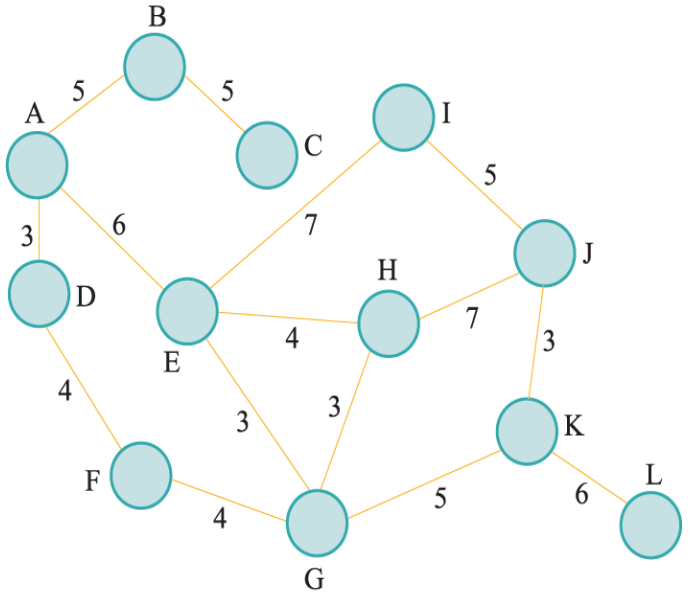


图 3.1　某地区的交通网络图

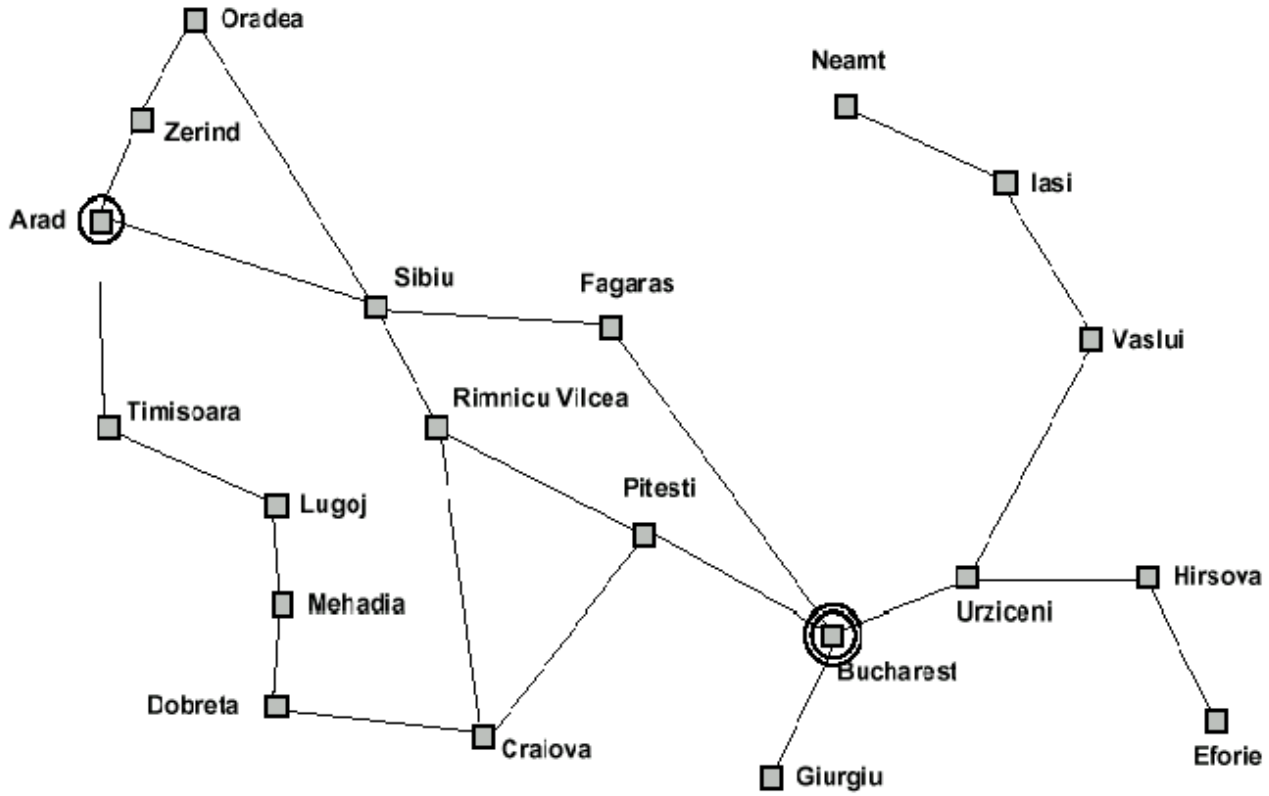
以求取从城市 A 到城市 K 之间一条行驶时间最短路线为例

搜索的形式化描述实例

同济大学 TONGJI UNIVERSITY

- 罗马尼亚问题：

  一个 Agent 如何从罗马尼亚的 Arad 走到 Bucharest？



搜索的形式化描述例子

- **一个问题用 5 个部分进行形式化描述（以罗马尼亚问题为例）**

1）初始状态： In Arad

2）行动： ACTIONS(s) ，给定一个状态 s， ACTIONS(s) 返回状态 s 下可以执行的动作的集合，例如状态 s 为“ In Arad”， ACTIONS(s) 返回的行动为

{ Go (Sibiu), Go (Timisoara), Go (Zerind) }

3）转移模型： RESULT(s， a)，在状态 s 下，执行 a 动作后，达到的状态。

RESULT(In (Arad)， Go (Sibiu)) = In (Sibiu)

搜索的形式化描述实例

- **一个问题用 5 个部分进行形式化描述（以罗马尼亚问题为例）**

4）目标测试： 确定给定的状态是不是目标状态。目标状态集为 { In (Bucharest) }

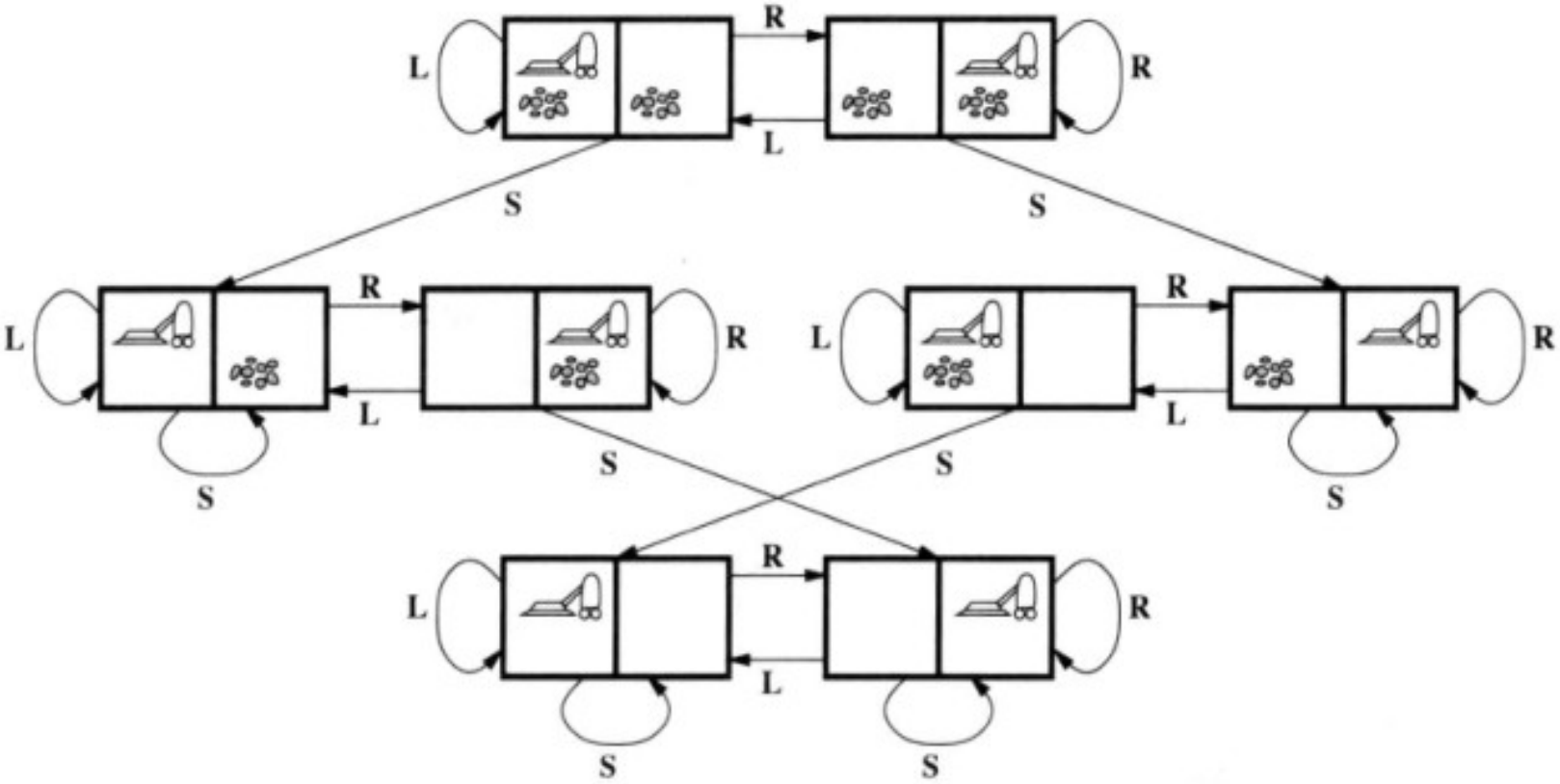
5）路径耗散： 路径耗散函数为每条路径赋一个耗散值，即边加权。罗马尼亚案例中，路径耗散可用公里数表示的路径长度。采用行动 a 从状态 s 走到状态 s’ 所需要的单步耗散用 c(s, a, s’) 。

问题的**解**是从**初始状态**到**目标状态**的一组行动序列。解的质量由路径耗散函数衡量，路径耗散值最小的即为**最优解。**

搜索的形式化描述实例

同济大学 TONGJI UNIVERSITY

- 真空吸尘器世界问题



搜索的形式化描述实例

真空吸尘器世界问题可形式化如下：

状态：状态由 Agent 位置和灰尘位置确定。 Agent 的位置有两个，每个位置都可能有灰尘，因此可能的世界状态有 2x2x2=8 个。对于具有 n 个位置的大型环境，可能的世界状态数为 nx2^n 个。

初始状态：任何一个状态都可能是初始状态。

行动：每个状态下可执行的行动只有三个 Left （左移）， Right （右移），Suck （吸尘）

转移模型：行动会产生对应的预期效果。除了在最左边位置不能再左移，最右边位置不能再右移，且干净的位置吸尘没有效果以外，其他所有可能的行动后的状态空间如图所示。

目标测试：检查所有位置是否干净

路径消耗：每一步耗散值为 1 ，整个路劲耗散值为路径的总步数。

搜索的形式化描述实例

八数码问题游戏：

一个 3x3 的棋盘中有 8 个数字棋子和一个空格，与空格相邻棋子可滑动到　空格中。游戏目标要达到右侧给出的指定状态。

2	3	1
5		8
4	6	7

初始状态

1	2	3
8		4
7	6	5

目标状态

搜索的形式化描述实例

八数码的问题可形式化如下：

可能状态： 所有 8 个棋子及空格在棋盘上的可能分布

初始状态： 任何可能状态里的一种。

行动： 与空格相邻的数字棋子可能发生的合法移动（上移，下移，左移，右移）

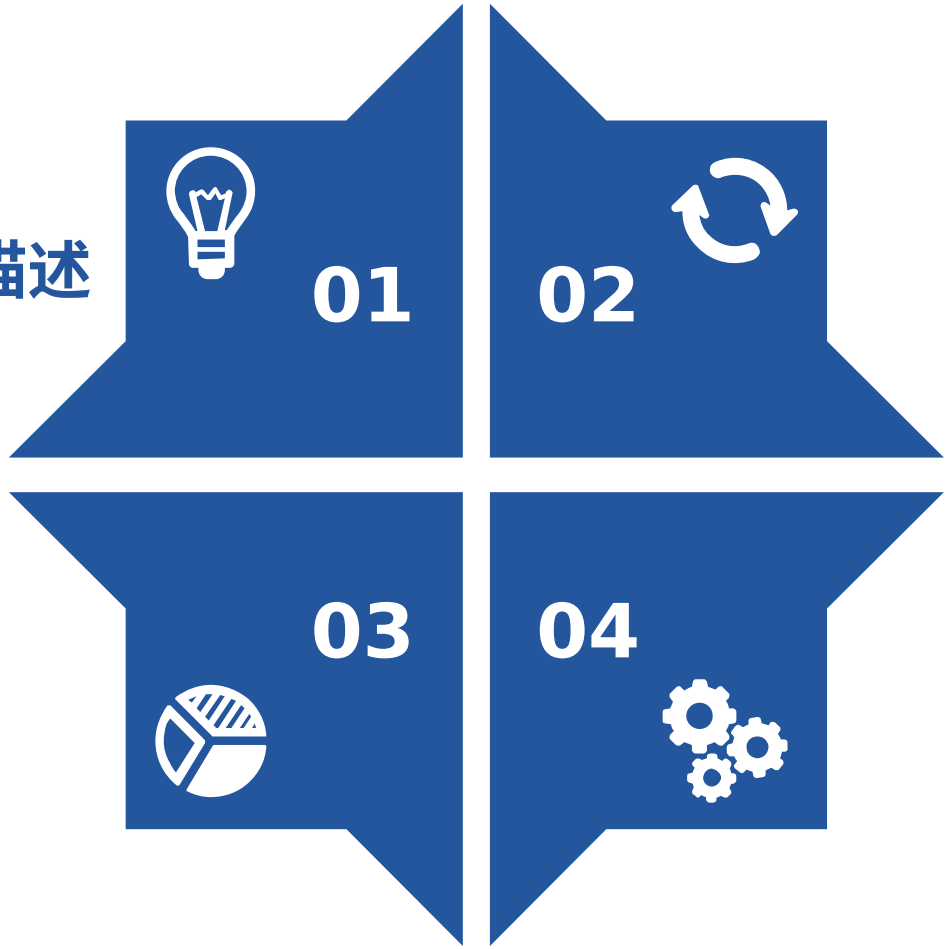
转移模型： 对应棋子移动后的合法状态

目标测试： 检测最终状态布局是否与图中右侧给定状态一致。

路径耗散： 每一步耗散值为 1，总的路径耗散值为路径中的步数。

同济大学 TONGJI UNIVERSITY

目录



搜索的形式化描述

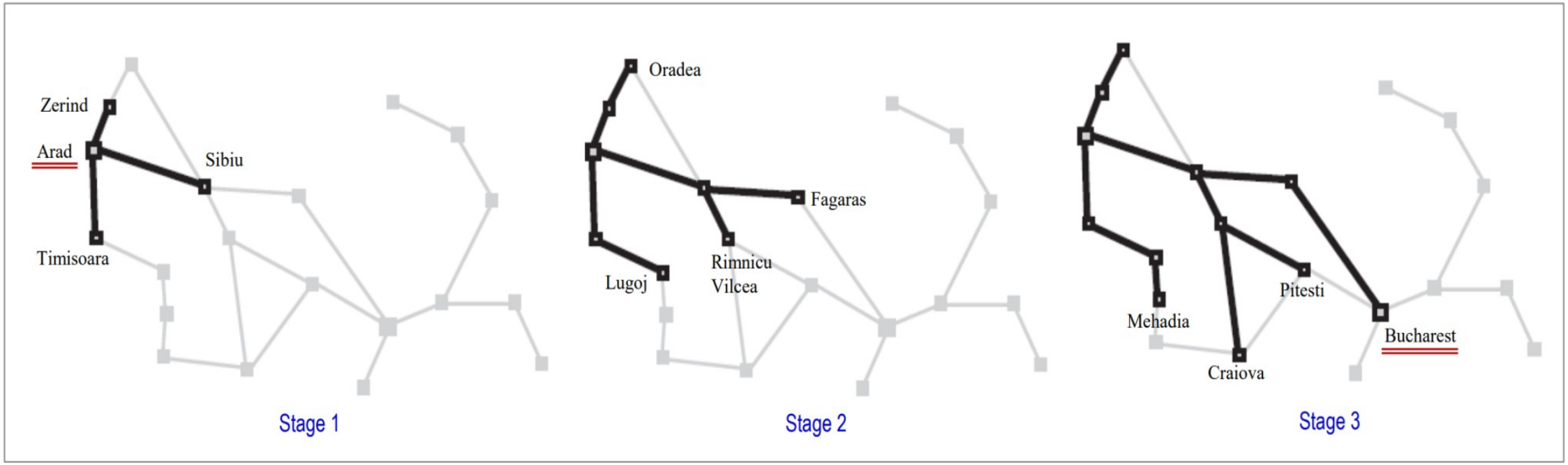
搜索树构建

启发式搜索

性能分析和案例

搜索过程可视为搜索树的构建

- 通过图搜索在该罗马尼亚地图上生成一系列搜索路径



搜索树的构建

- 树的根节点为初始状态
- 从根节点出发，通过各种行为扩张当前状态，生成树的中间节点
- 状态之间用边 S 表示
- 树的叶子表示最终状态，边用行动和代价表示

搜索树的构建

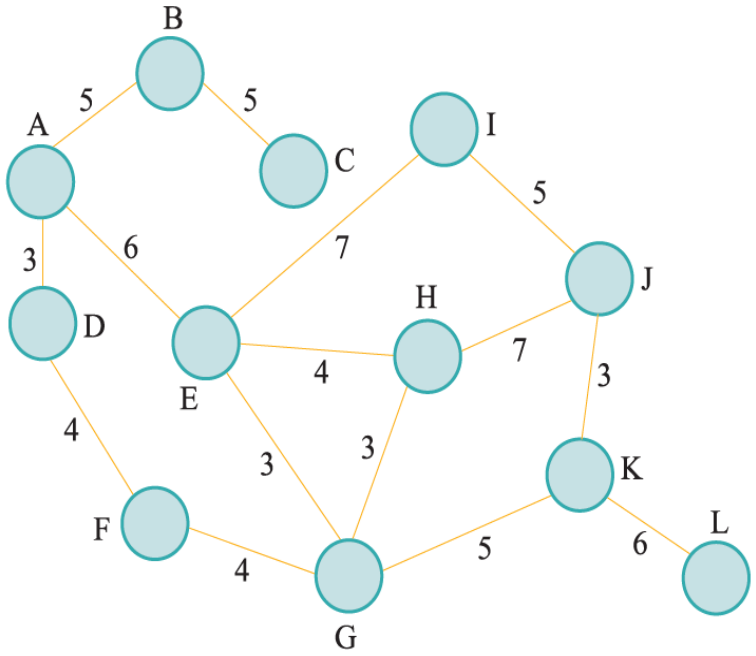


图 3.1　某地区的交通网络图

以求取从城市 A 到城市 K 之间

一条行驶时间最短路线为例

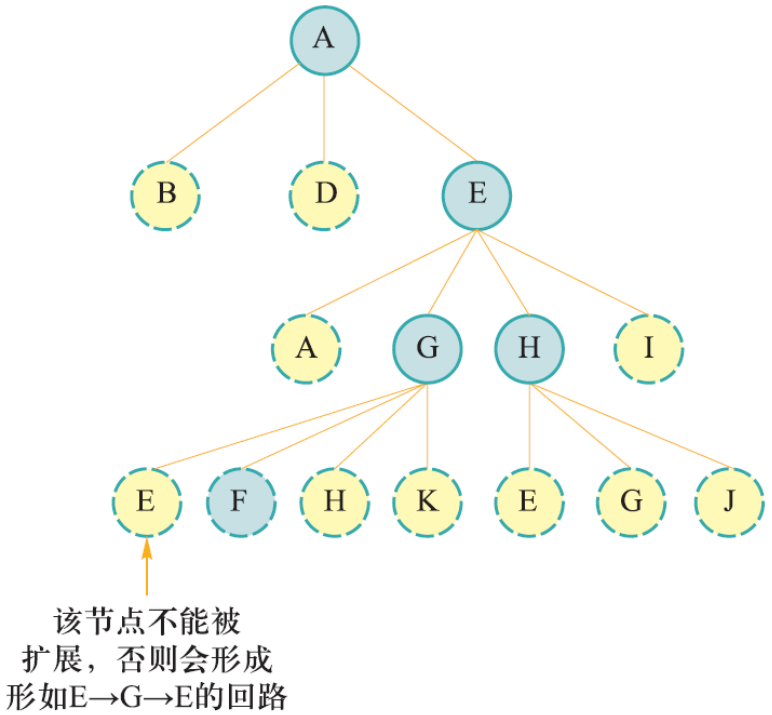


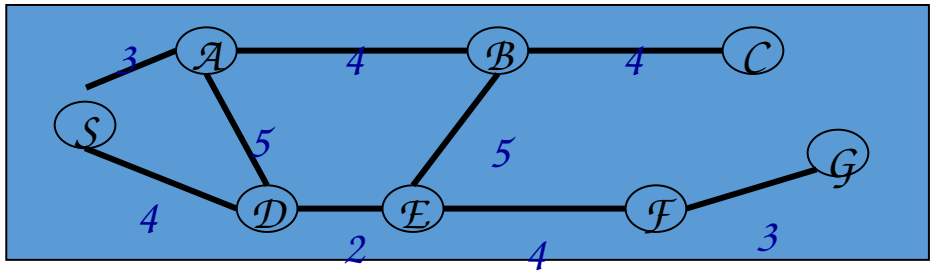
图 3.3 正在构建中的搜索树

（注：图中实线结点表示已被扩展，虚线结点在边缘集合中）

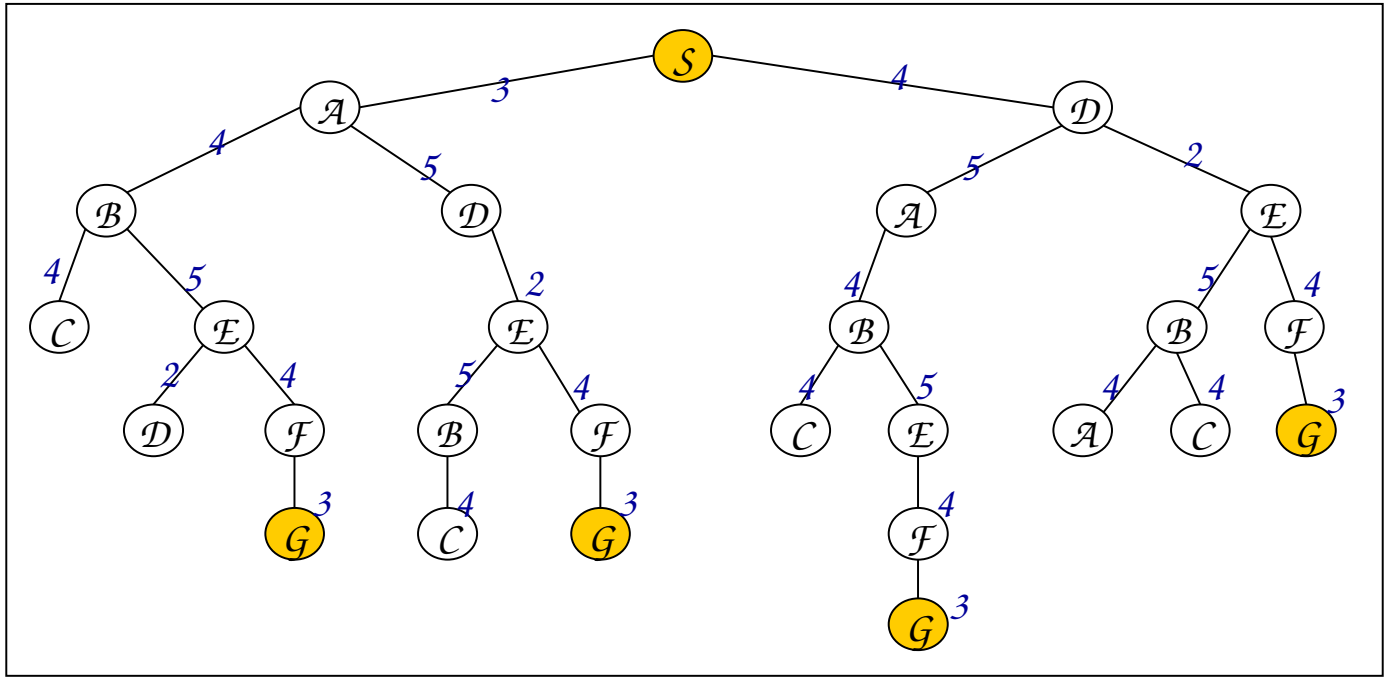
搜索树的构建

同济大学 TONGJI UNIVERSITY

问题空间

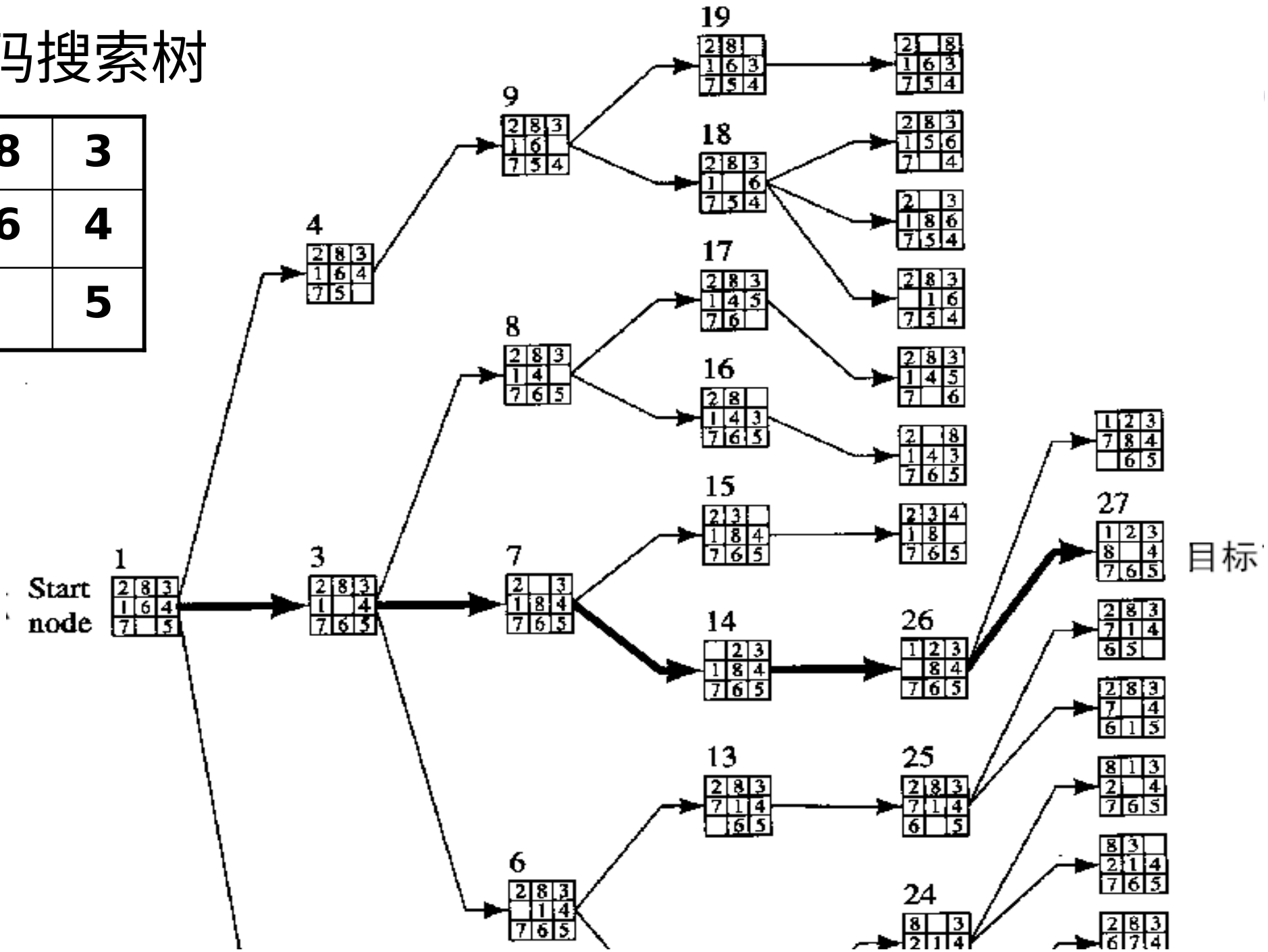


对应的搜索树

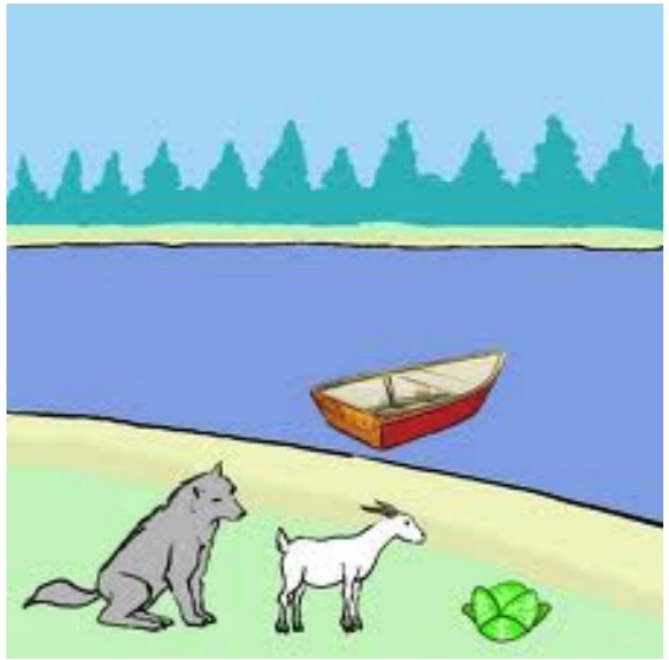


8 数码搜索树

2	8	3
1	6	4
7		5



搜索树的构建思考题



- 农夫、白菜、羊和狼过河问题。现在农夫、白菜、羊和狼都在左岸，农夫有一条船，过河时，除农夫外，每次只能带一样东西，农夫不在，狼要吃羊，羊要吃白菜。现在要把他们全部安全送到右岸去。

1. 写出问题的形式化描述．画出状态空间图（树）。

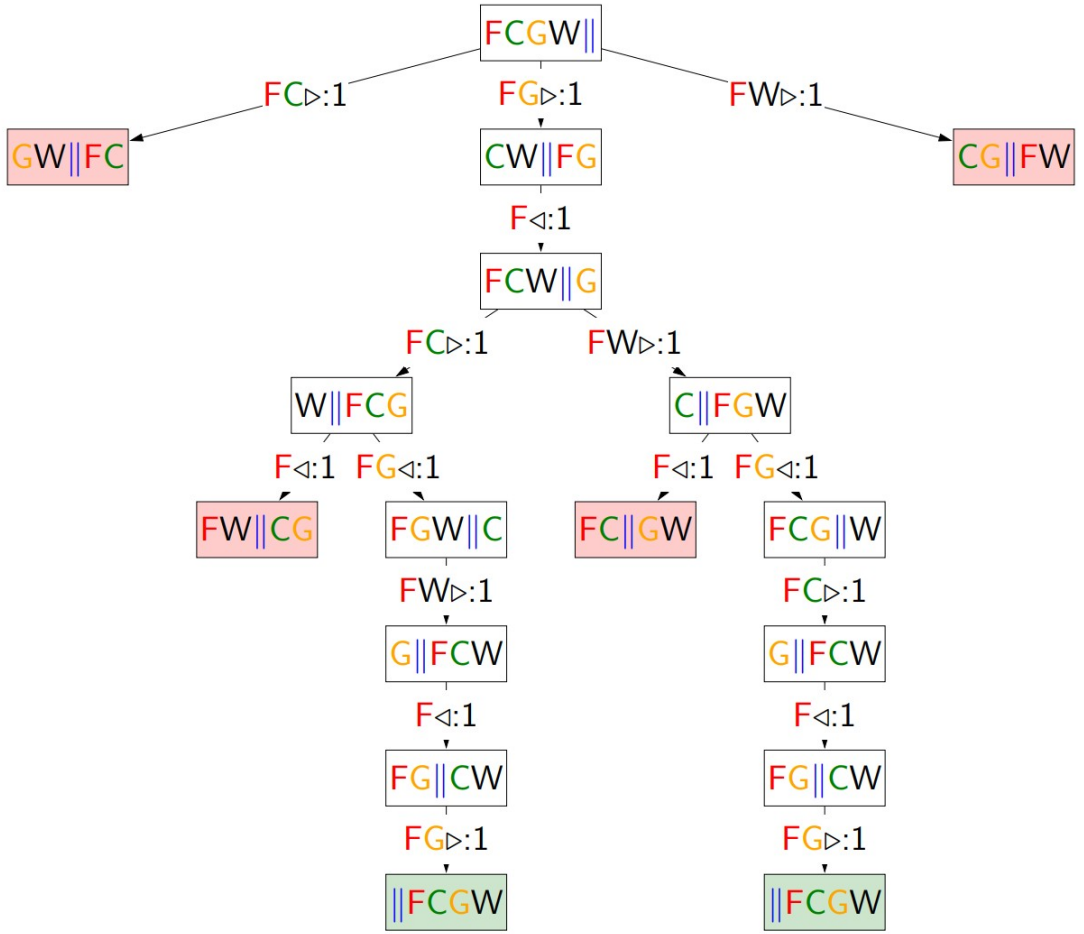
搜索树的构建思考题

- 用四元组表示（f,c,g,w）农夫、白菜、羊和狼是否在左岸，1 表示在左岸，0 表示不在左岸。
- 初始状态（1，1，1，1），目标状态（0，0，0，0）
- **行动**：L(f，j) 表示从左岸把 j 样东西送到右岸(j=1 表示白菜，j=2 表示羊，j=3 表示狼，j=0 表示农夫什么都不载）；R(f，j) 表示从右岸把 j 样东西送到左岸
- L(f，1) 则（1，1，1，1）→（0，0，1，1）

搜索树的构建思考题

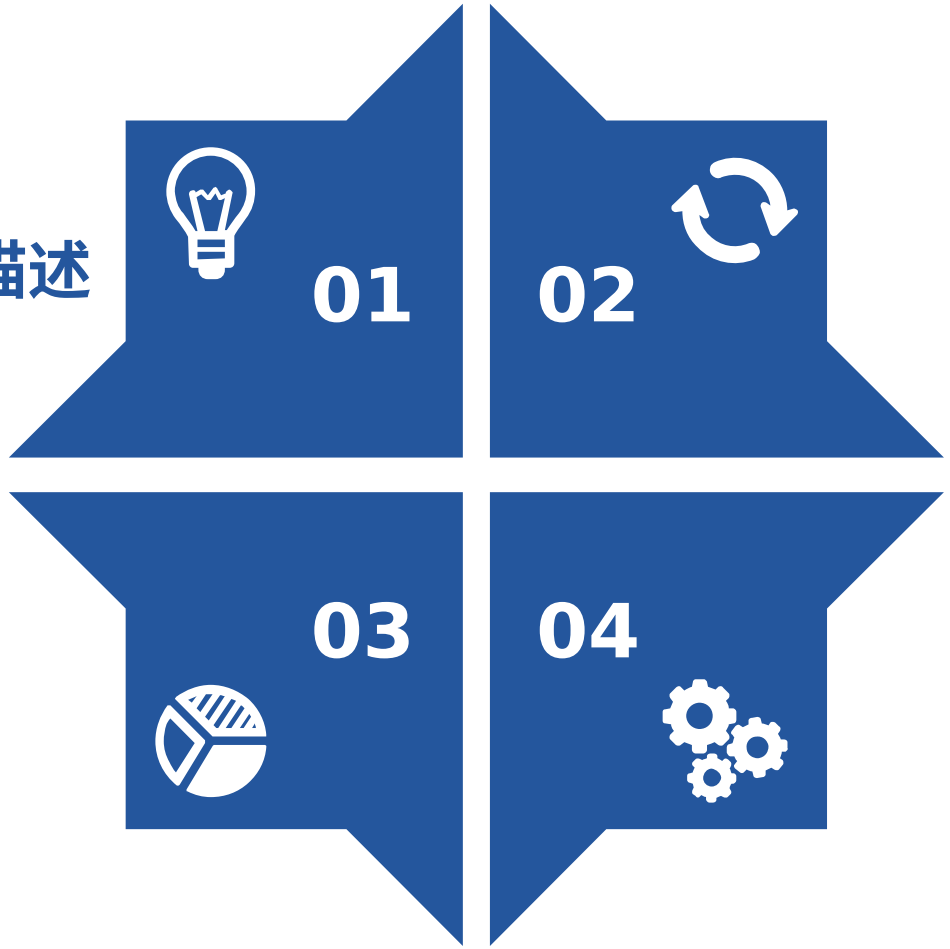
同济大学 TONGJI UNIVERSITY

搜索树



同济大学 TONGJI UNIVERSITY

目录



搜索的形式化描述

搜索树构建

启发式搜索

性能分析和案例

搜索中需要解决的基本问题：

（1）是否一定能找到一个解**（完备性）**。

（2）找到的解是否是最佳解。**（最优性）**

（3）找到解需要花费多长时间？**（时间复杂度）**

（4）在执行搜索的过程中需要多少内存？**（空间复杂度）**

（5）是否终止运行或是否会陷入一个死循环。

1. 搜索方向：

(1) **数据驱动**：从初始状态出发的正向搜索。

正向搜索——从问题给出的条件出发。

(2) **目的驱动**：从目的状态出发的逆向搜索。

逆向搜索：从想达到的目的入手，看哪些操作算子能产生该目的以及应用这些操作算子产生目的时需要哪些条件。

(3) **双向搜索**

双向搜索：从开始状态出发作正向搜索，同时又从目的状态出发作逆向搜索，直到两条路径在中间的某处汇合为止。

2. 无信息搜索与启发式搜索：

什么是无信息搜索?

- 无信息搜索也被称为盲目搜索。
- 该术语（无信息、盲目的）意味着该搜索策略没有超出问题定义提供的状态之外的附加信息。
- 所有能做的就是生成后继结点，并且从区分一个目标状态或一个非目标状态。
- 所有的搜索策略是由节点扩展的顺序加以区分。
- 这些搜索策略是：宽度优先、深度优先、以及一致代价搜索（数据结构中已经学过）。

启发式策略和信息

- “**启发**”（heuristic）：关于发现和发明操作算子及搜索方法的研究。
- 在状态空间搜索中，**启发式**被定义成一系列操作算子，并能从状态空间中选择最有希望到达问题解的路径。
- **启发式策略**：利用与问题有关的启发信息进行搜索。
- 在具体求解中，能够利用与该问题有关的信息来简化搜索过程，称此类信息为**启发信息**。
- **启发式搜索**：利用启发信息的搜索过程。

评价函数

- 评价函数的任务就是估计待搜索结点的“有希望”程度，并依次给它们排定次序（在 open 表中）。

- 评价函数 $f(n)$：从初始结点经过 n 结点到达目的 结点的路径的最小代价估计值，其一般形式是

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

- 一般地，在 $f(n)$ 中，$g(n)$ 的比重越大，越倾向于宽度优先搜索方式，而 $h(n)$ 为启发式函数，比重越大，表示启发性能越强。

最佳优先搜索

- 想法：创建一个评价函数 f(n) 用于每个结点
- 评估 “可取性”
  - 扩展最可取的结点
- 实现：
  - 按可取性递减的顺序排列未扩展结点
- 特殊情况：
  - 贪婪最佳优先搜索
  - A^* 搜索

两个重要函数：评价函数与启发函数

辅助信息	所求解问题之外、与所求解问题相关的特定信息或知识。	
评价函数（evaluation function） f(n)	从当前节点n出发，根据评价函数来选择后续结点	**下一个结点是谁?**
启发函数（heuristic function） h(n)	计算从结点n到目标结点之间所形成路径的最小代价值，这里将两点之间的直线距离作为启发函数	**完成任务还需要多少代价?**

贪婪最佳优先搜索 (Greedy best-first search)：评价函数 f(n)= 启发函数 h(n)

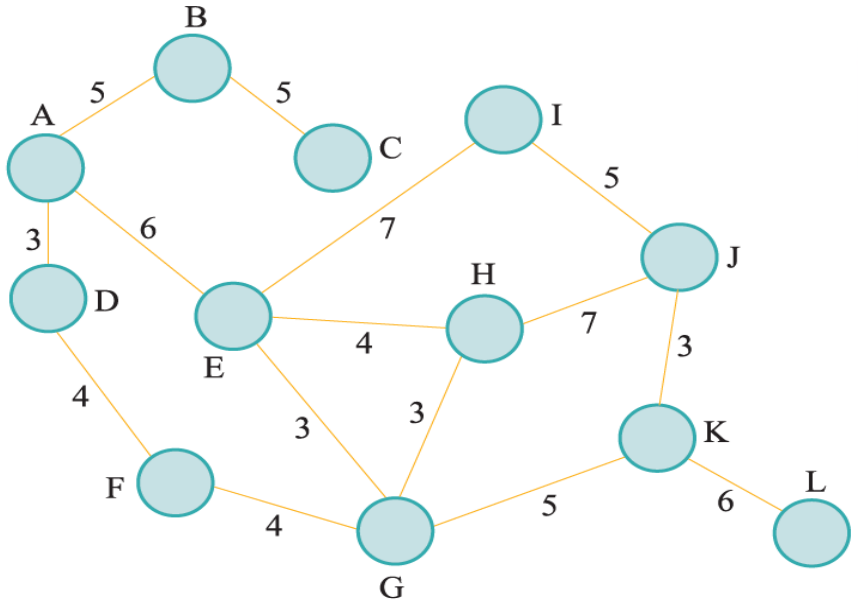


图 3.1　某地区的交通网络图

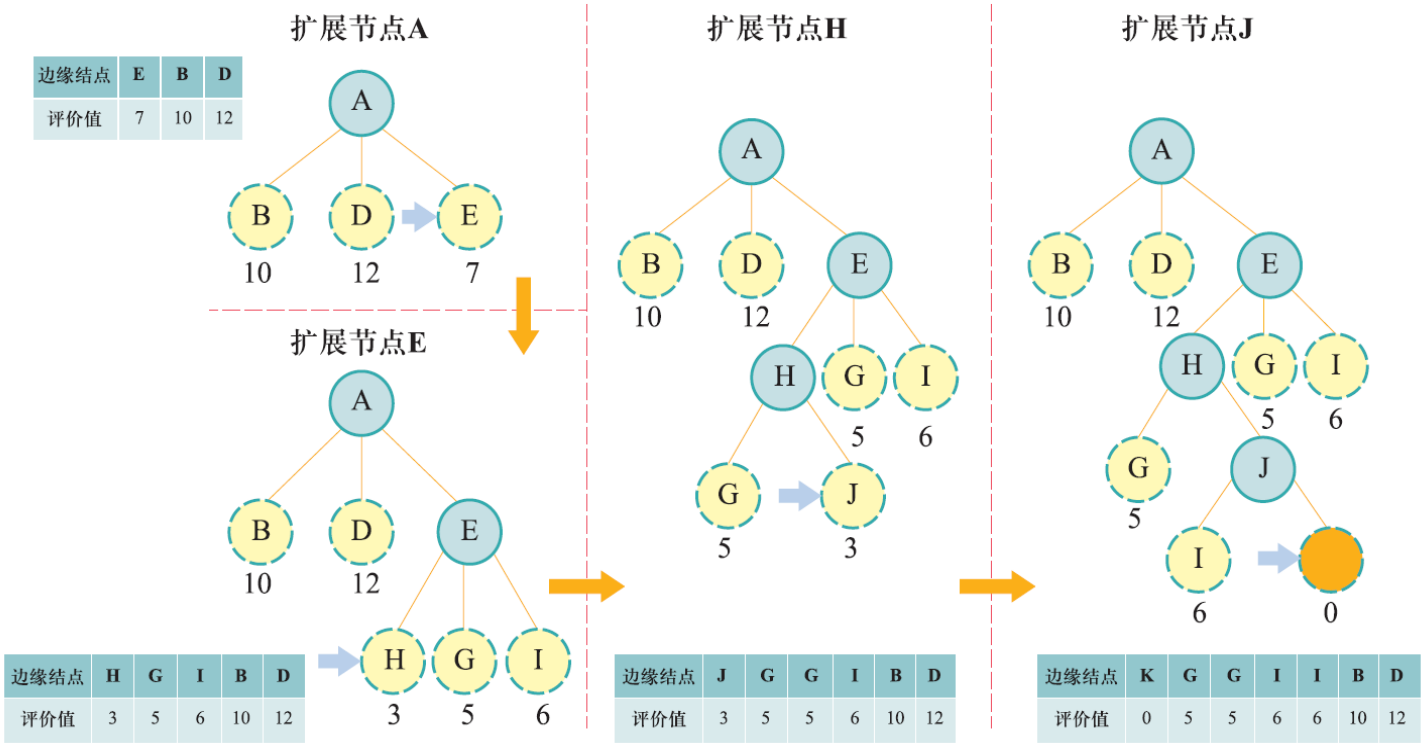
表 3.2　每个城市到目标城市的直线距离（启发函数取值）

状态	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
$h(n)$	13	10	6	12	7	8	5	3	6	3	0	6

每个城市到目标城市（即城市 K）的直线距离（启发函数取值）

求取从城市 A 到城市 K 之间一条行驶时间最短路线

贪婪最佳优先搜索：将启发函数作为评价函数的搜索过程



边缘结点	E	B	D
评价值	7	10	12

边缘结点	H	G	I	B	D
评价值	3	5	6	10	12

边缘结点	J	G	G	I	B	D
评价值	3	5	5	6	10	12

边缘结点	K	G	G	I	I	B	D
评价值	0	5	5	6	6	10	12

图 3.4　贪婪最佳优先搜索的过程

（注：图中边缘结点下的数字表示其评价函数取值）

算法找到了一条从起始结点到目标结点的路径A → E → H → J → K

但这条路径并不是最短路径，实际上最短路径为A → E → G → K

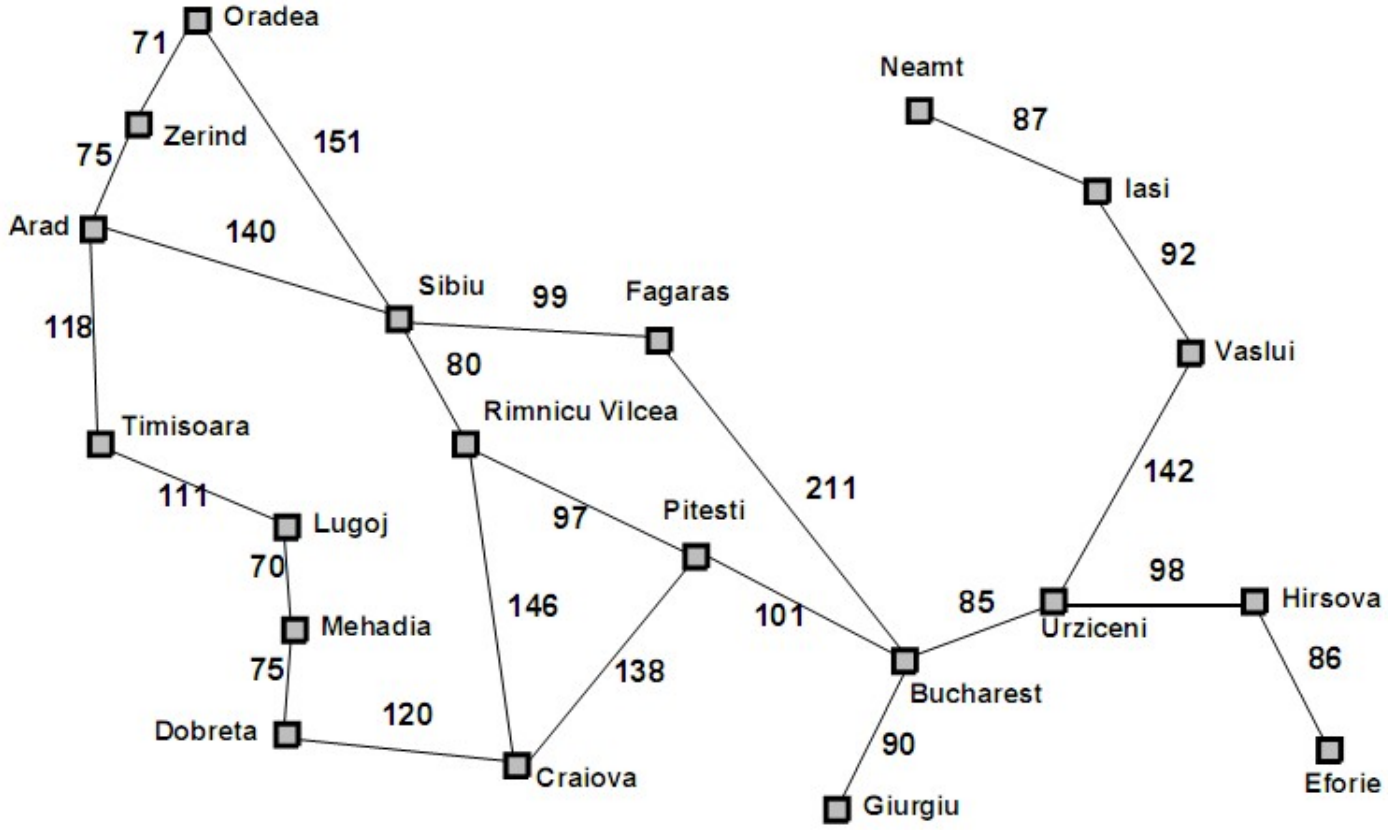
贪婪最佳优先搜索的过程。
图中边缘结点下数字表示其评价函数取值

贪婪最佳优先搜索：为什么会失败

欲速则不达

- 在上述搜索步骤中，虽然第四步扩展的J结点距离目标结点的估计代价很小（取值为3），但从起始结点到达该结点的路径A → E → H → J代价为17，已经超过了实际上最短路径的代价（值为14），因此路径A → E → H → J并不是一个好的探索方向。

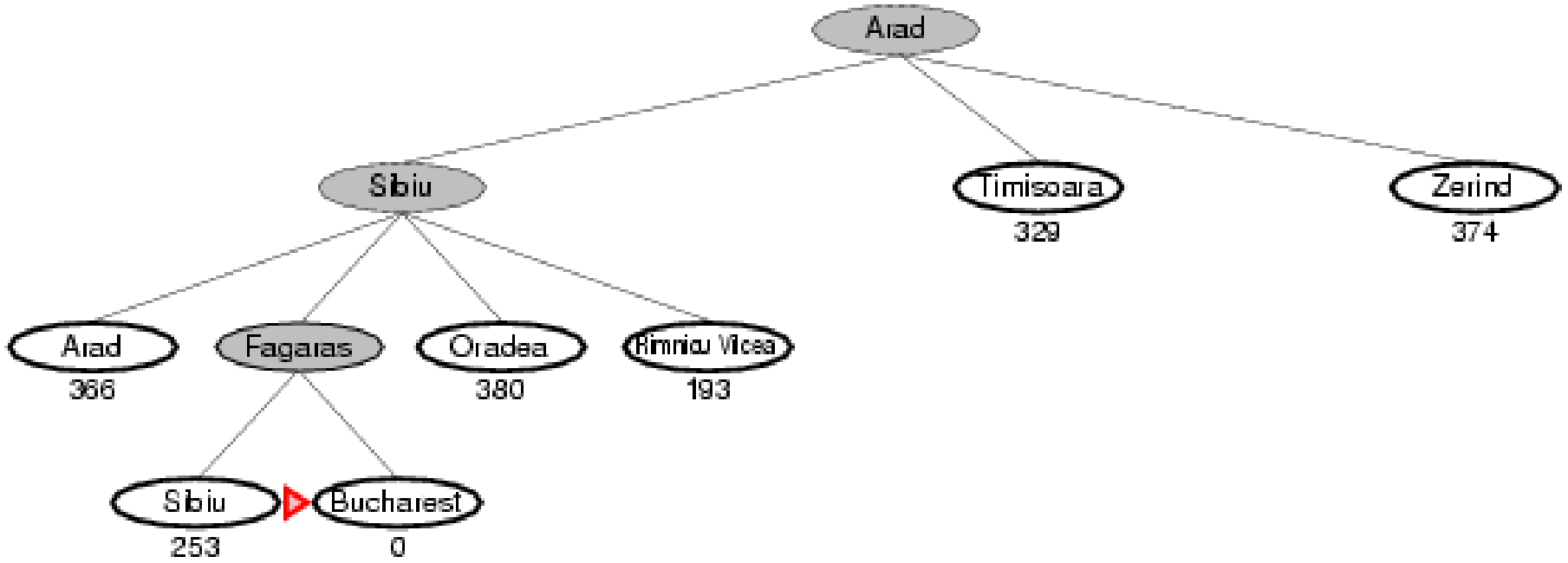
贪婪最佳优先搜索另一个例子



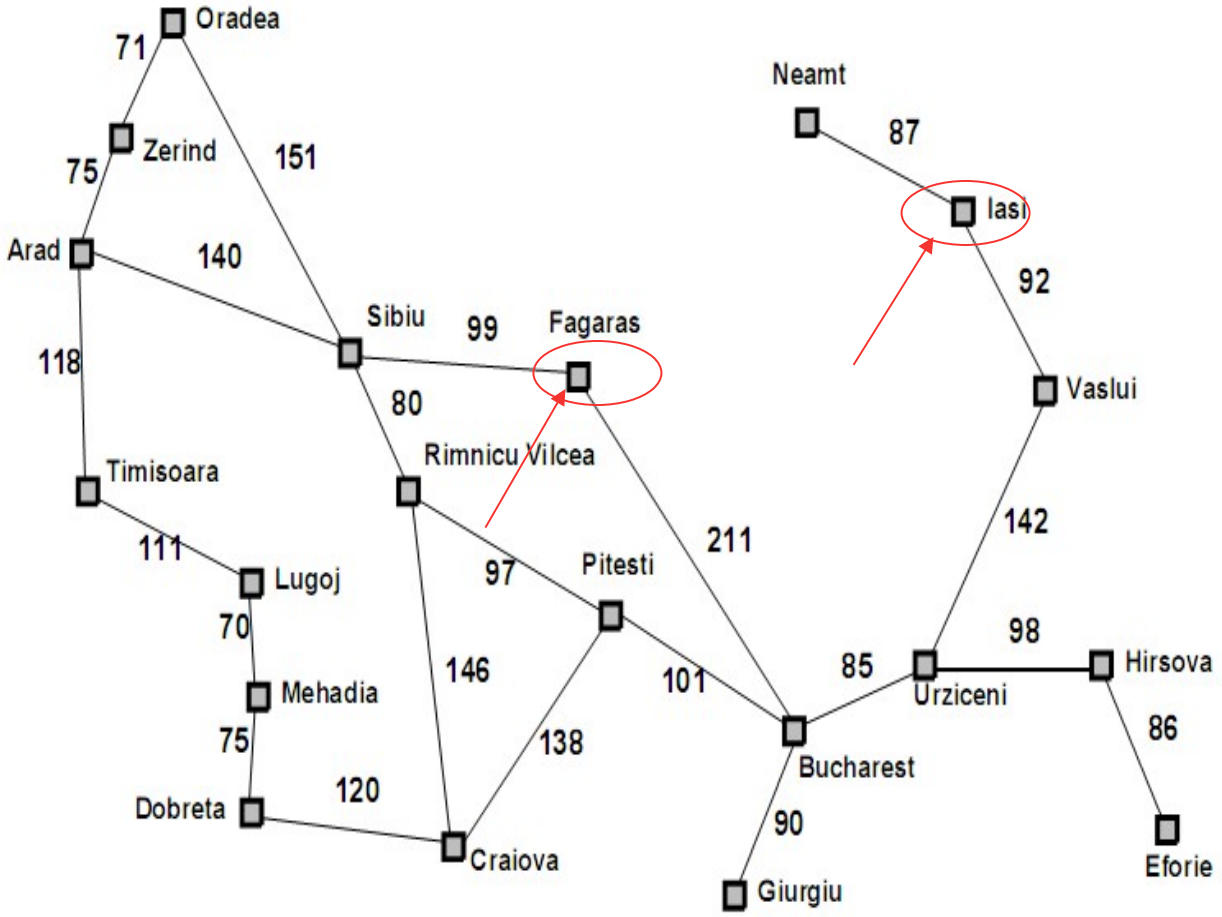
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

贪婪最佳优先搜索另一个例子



贪婪最佳优先搜索另一个例子



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

贪婪最佳优先搜索的性能

- 完备性？　否–会陷入死循环，除非采用了排除环路的剪枝方法
- 最优性？　否
- 时间复杂度？　$O(b^m)$，一个好的启发式函数可以有效降低复杂度。
- 空间复杂度？$O(b^m)$

b--- 分支因子　**m--** 搜索空间的最大深度

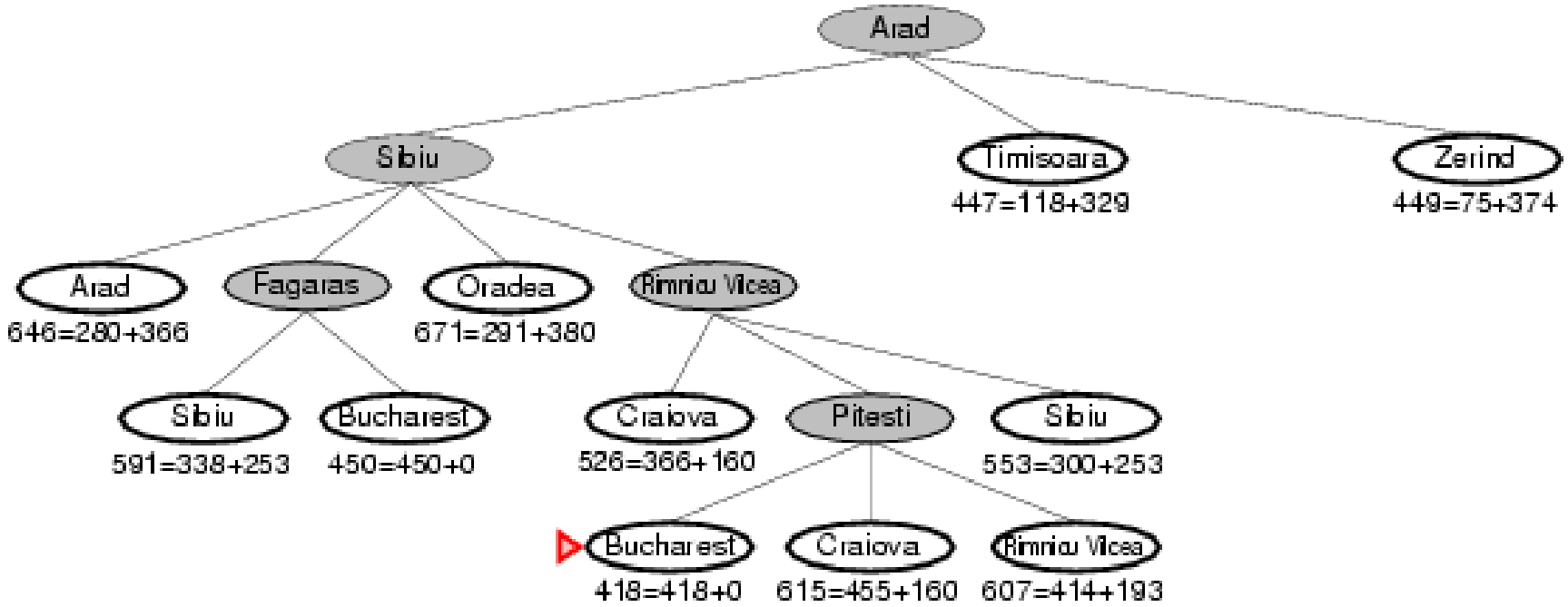
A* 搜索算法：耗散值和启发函数各司其职

在评价函数中考虑从起始结点到当前结点的路径代价

$$\underbrace{f(n)}_{\text{评价函数}\ (\text{选择后续结点})} = \underbrace{g(n)}_{\text{起始结点到结点}n\text{代价}\ (\text{当前最小代价})} + \underbrace{h(n)}_{\text{结点}n\text{到目标结点代价}\ (\text{后续估计最小代价})}$$

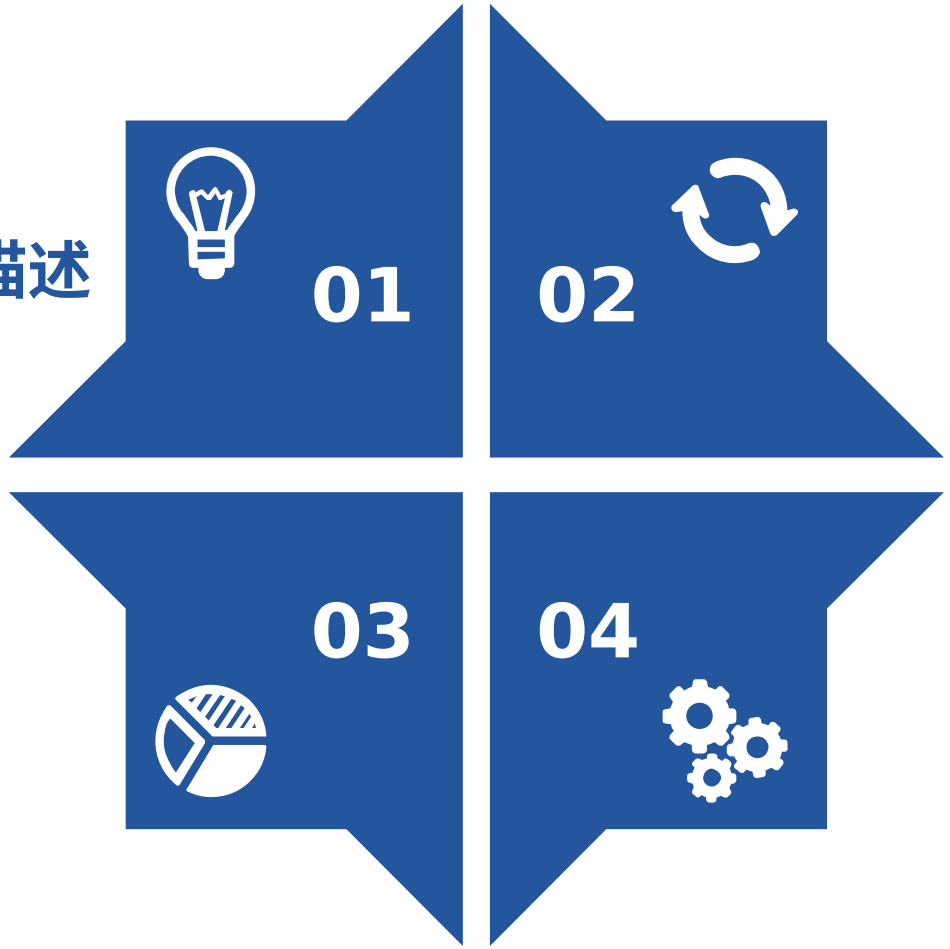
定义从起始结点到结点n的路径代价函数$g(n)$，在此基础上评价函数$f(n)$被修改为：$f(n) = g(n) + h(n)$，即评价函数等于从起始结点到达结点n的的实际代价取值与结点n到目标结点估计代价取值之和。根据每个结点的评价函数之值不断选择后续结点的搜索算法被称为A*算法

A* 搜索算法实例


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

同济大学 TONGJI UNIVERSITY

目录



搜索的形式化描述

搜索树构建

启发式搜索

性能分析和案例

A* 搜索性能分析：可容性

- 若对每个结点 n ，满足 $h(n) \le h^*(n)$, 则 $h(n)$ 是**可采纳的**。
  - 其中 $h^*(n)$ 是从结点 n 到达目标结点的真实代价。

- 可采纳启发式不会过高估计到达目标的代价。
  - 例如 : $h_{SLD}(n)$ (不会高估真实距离)

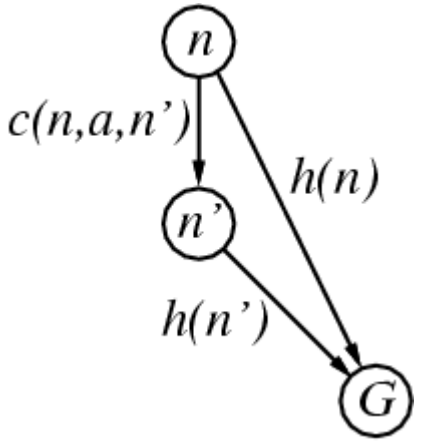
- **定理**： 如果 $h(n)$ 是可采纳的，A* 搜索树搜索版本是最优的。

A* 搜索性能分析：一致性

- 如果对于每个结点 n 和通过任一行动 a 生成的 n' 的每个后继结点满足 $h(n) \le c(n,a,n') + h(n')$，则称启发式 $h(n)$ 是**一致的。**

- 若 h 是一致的，则

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



i.e., $f(n)$ 在任何路径上都是非递减的。

- **定理：** 如果 $h(n)$ 是一致的，A* 搜索图搜索版本是最优的。

A* 搜索性能分析：一致性必然导致可容性

- 一致性是一个比可容性更严格的性质，即一个启发函数如果满足一致性条件，那么该启发函数必然满足可容性条件。

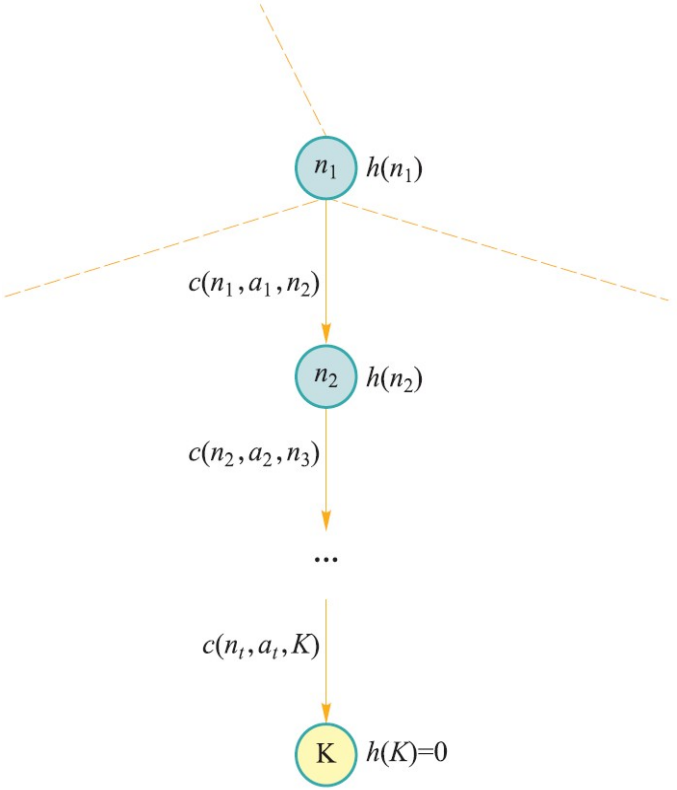


图 3.6　从 n_1 结点到终止结点的最短路径

如图中以n_1为根结点的子树，假设从该结点到达终止结点代价最小的路径为$n_1 \to n_2 \to \cdots \to$ K。由于$h(\mathrm{K}) = 0$且启发函数满足一致性条件，因此存在：

$$\begin{aligned} h(n_1) &\le h(n_2) + c(n_1, a_1, n_2) \\ &\le h(n_3) + c(n_2, a_2, n_3) + c(n_1, a_1, n_2) \\ &\le \cdots \\ &\le c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + \cdots \\ &= h^*(n_1) \end{aligned}$$

即满足一致性条件的启发函数一定满足可容性条件。

A* 搜索性能分析：

保证找到最短路径（最优解）的条件，关键在于估价函数 f(n) 的选取（或者说 h(n) 的选取）。

我们以 h*(n) 表达状态 n 到目标状态的距离，那么 h(n) 的选取大致有如下三种情况：

- 1. 如果 h(n)< h*(n) ，这种情况下，搜索的点数多，搜索范围大，效率低。但能得到最优解。
- 2. 如果 h(n)= h*(n) ，即距离估计 h(n) 等于最短距离，那么搜索将严格沿着最短路径进行， 此时的搜索效率是最高的。
- 3. 如果 h(n)> h*(n) ，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

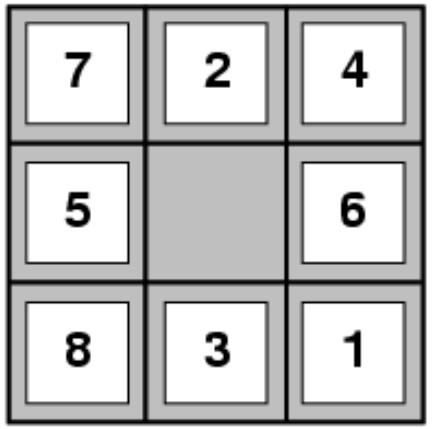
A* 搜索性能分析：

- 完备性？ 是 (unless there are infinitely many nodes with f ≤ *f(G)*)
- 最佳性？ 是
- 时间复杂度？ 指数级
- 空间复杂度？ Keeps all nodes in memory

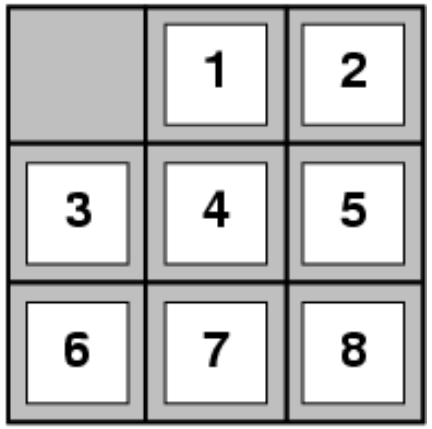
A* 搜索的应用

E.g.，针对八数码问题：

- $h_1(n)$ = 不在位的棋子数
- $h_2(n)$ = 所有棋子到其目标位置的距离和
-



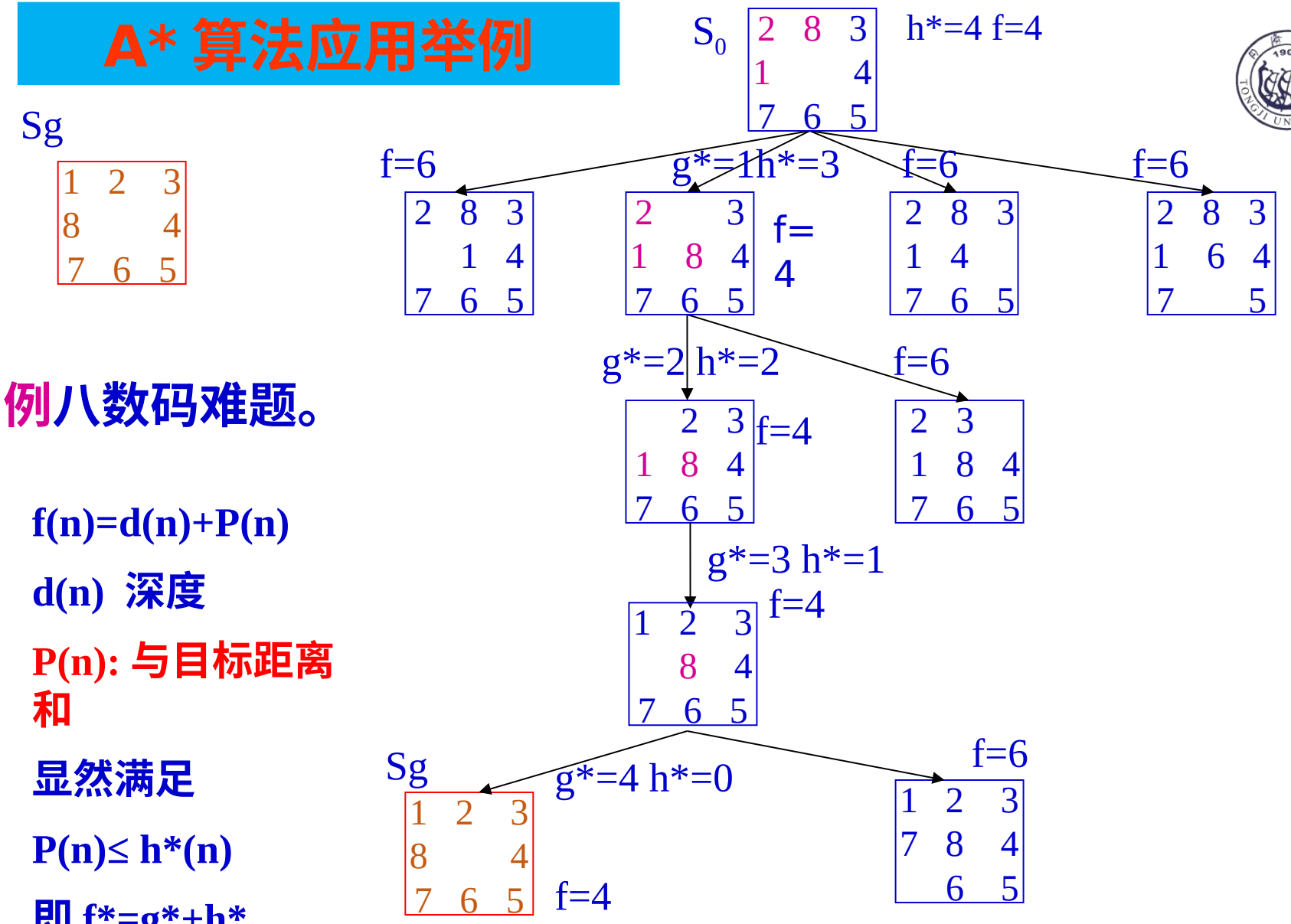
Start State



Goal State

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

A* 算法应用举例



八数码难题 h(n)=P(n) 的搜索树

例八数码难题。

f(n)=d(n)+P(n)

d(n) 深度

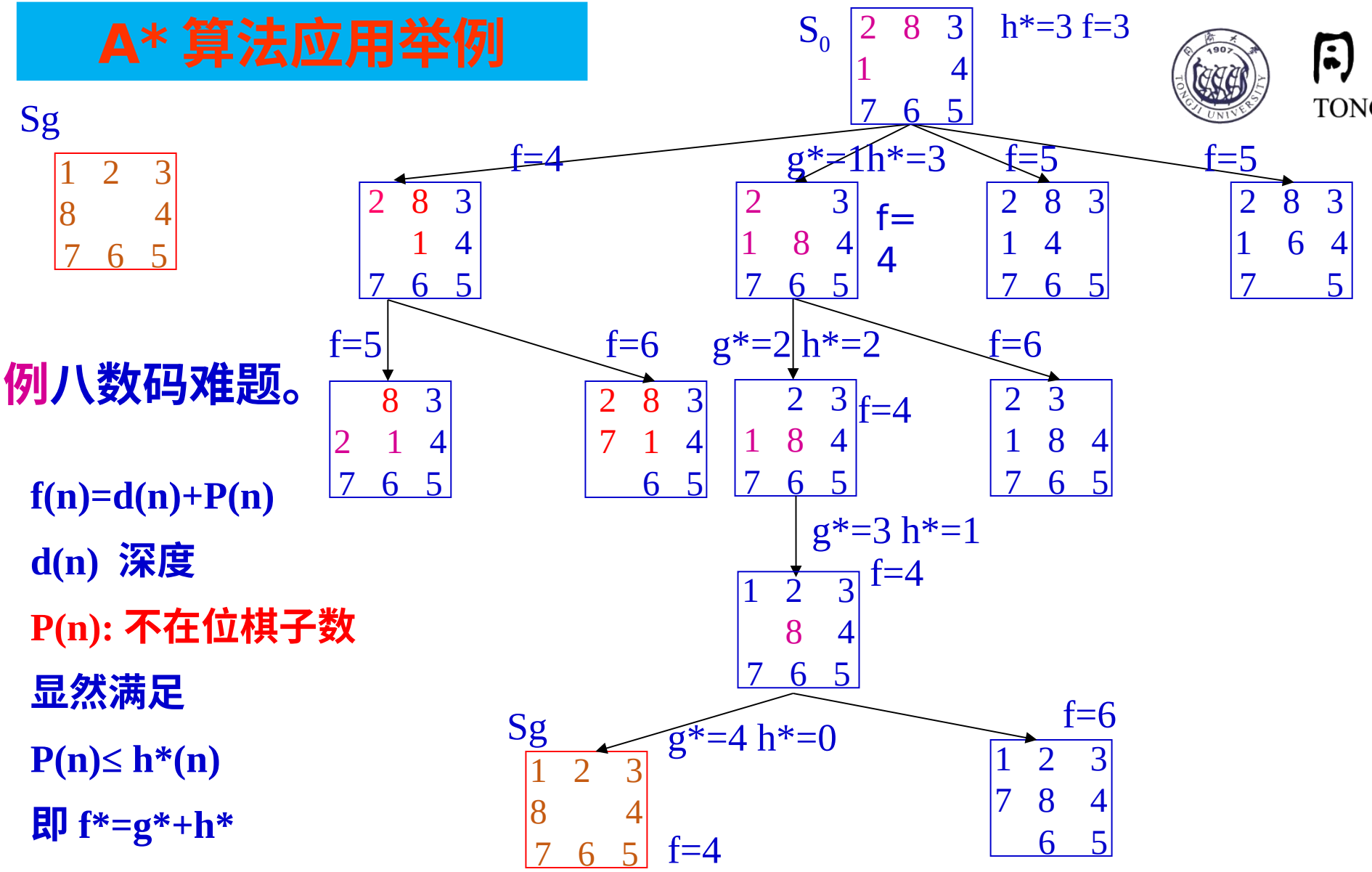
P(n): 与目标距离和

显然满足

P(n)≤ h*(n)

即 f*=g*+h*

A* 算法应用举例



八数码难题 h(n)=P(n) 的搜索树

启发函数：表示所有图片到其目标位置的距离和

启发函数：表示不在位的图片数

A* 搜索的应用

- 对于任意结点 *n* ， 若 $h_2(n) \geq h_1(n)$ ，称 h_2 比 h_1 占优势， h_2 更利于搜索。

- 搜索代价 (扩展的平均结点数):
  - *d=12* IDS = 3,644,035 nodes (迭代加深搜索)
    $A^*(h_1)$ = 227 nodes
    $A^*(h_2)$ = 73 nodes
  - *d=24* IDS = too many nodes
    $A^*(h_1)$ = 39,135 nodes
    $A^*(h_2)$ = 1,641 nodes

从松弛问题出发设计可采纳的启发式

- 减少了行动限制的问题称为**松弛问题。**
- 一个松弛问题的最优解是原有问题的**可采纳的启发式。**
- **八数码问题松弛问题**
  - 棋子可以移动到任一位置，则 $h_1(n)$ 能够给出最短路径解。
  - 棋子可以移动到相邻位置，则 $h_2(n)$ 能够给出最短路径解。