人工智能原理

第三章　搜索技术

前文回顾

在前序课程中，我们通过对于智能行为的若干定义来尝试刻画人工智能的目标

像人一样思考 "有头脑的机器"（Haugeland，1985） "与人类思维相关的活动，如决策、求解、学习等的自动化"（Bellman，1978）	合理地思考 "通过计算机模型研究智力"（Charniak & McDermott，1985） "使感知、推理和行动成为可能的计算的研究"（Winston，1992）
像人一样行动 "创造能执行某些功能的机器，这些功能如果是人来执行的话需要智能"（Kurzweil，1990） "研究如何使计算机能够像林日前人比计算机更擅长的事情"（Rich & Knight，1991）	合理地行动 "智能Agent的设计"（Poole，1998） "自关心人类造物的智能行为"（Nilsson，1998）

3.1.1 Agent与环境

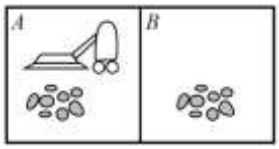
理性Agent的概念是人工智能方法的核心

行为	传感器	执行器
人类Agent	眼睛、耳朵、皮肤	手脚、声音
机器人Agent	摄像头、红外测距仪	马达
软件Agent	键盘敲击、文件、web	屏幕、打印机、文件

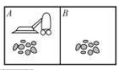
3.1.1 Agent与环境

吸尘器的世界

- 传感器（Percept）：位置、内容
- 执行器（Action）：左、右、吸尘、等待



吸尘器Agent函数



Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
...	...

```
function REFLEX-VACUUM-AGENT([location, status]) returns an action
  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left
```

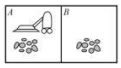
什么是**正确的**行为?

做正确的事

- 答案：**考虑Agent的行动后果。**
- 如果Agent在一个环境中，针对收到的感知信息生成一个行动序列，这个行动序列能够导致环境状态朝着机器人希望的方向改变，那么这个Agent性能良好。
- 注意环境状态不是Agent的状态，如果从Agent的角度定义，Agent可以达到完美理性，它只需要欺骗自己说环境是完美的。

（人类如果得不到有时候也会欺骗自己这个不重要）

做正确的事：理性的概念

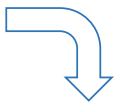


- 所谓的符合理性的度量，是一个很难有通用定义的概念。
- 对于吸尘器Agent：
- 如果定义吸尘的量多少，那么它可能一边吐一边吸，很符合理性的概念。
- 如果定义清洁方格的数量，它就可能来回吸尘永远停不下来。
- 如果定义最快把两个格子清洁干净，就一定真的“理性”吗？
- 如果两个Agent一个一直在工作，一个在短时间内工作然后休息很长时间，哪个更好？
- 这只是个简单的问题吗？

理性的概念

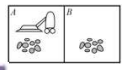
理性的判断依赖下面四个方面：

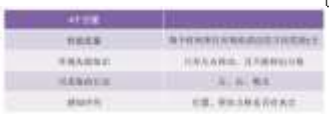
- 定义成功标准的性能度量。
- Agent对环境的先验知识。
- Agent可以完成的行动。
- Agent截止到此时的感知序列



对每一个可能的**感知序列**，根据已知的感知序列提供的**证据**和Agent具有的**先验知识**，理性Agent应该选择能**使其性能度量最大化**的行动。

吸尘器的理性



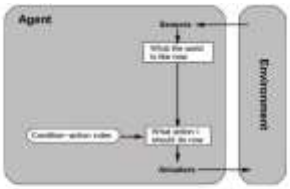


- 同样的Agent在不同的环境下会变成非理性。
- 例如，一旦所有灰尘都吸干净了，它就跑来跑去：如果性能度量含有对左右移动罚1分，这个Agent的性能评价就会很糟糕。
- 这意味着Agent应该在所有地方干净了以后什么都不干——但这样可能吗？
- 换成定时检查，确认干净以后什么都不干——足够了吗？

Agent与环境

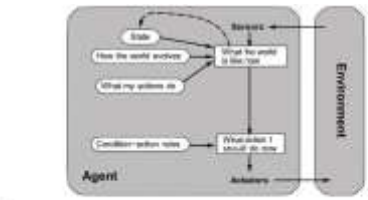
任务环境的性质：可观察（Observable）
任务环境的性质：单或多Agent（Agents）
任务环境的性质：确定或不确定（Deterministic）
任务环境的性质：片段式与延续式（Episodic）
任务环境的性质：静态或动态（Episodic）
任务环境的性质：离散的与连续的（Discrete）

Agent程序：简单反射Agent



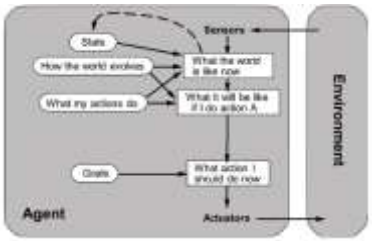

```
function Reflex-Vacuum-Agent([location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

Agent程序：基于模型的反射Agent

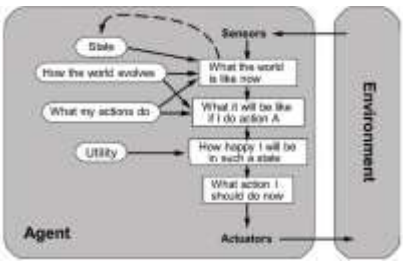



```
function Reflex-Vacuum-Agent( [location,status]) returns an action
    static: last A, last B, numbers, initially ∞
    if status = Dirty then . . .
```

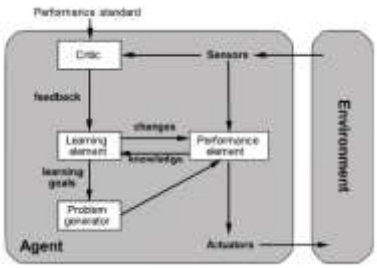

Agent程序：基于目标的Agent



Agent程序：基于效用的Agent



Agent程序：学习Agent

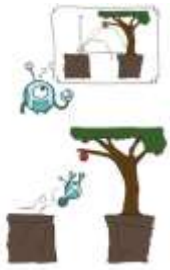


第三章　搜索技术

问题求解Agent

问题求解Agent

- 理性的行为：
- 内部性：能够针对当前状况做决策
- 什么是当前？如何刻画当前？
- 外部性：对于世界的变化有认知
- 世界变化是什么？如何刻画？需要考虑世界因为他的行为所产生的影响吗？
- 如果机器人只会条件反射，能算作理性Agent吗？



问题求解Agent

```
function Simple-Problem-Solving-Agent(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation
  state ← Update-State(state, percept)
  if seq is empty then
      goal ← Formulate-Goal(state)
      problem ← Formulate-Problem(state, goal)
      seq ← Search(problem)
  action ← Recommendation(seq, state)
  seq ← Remainder(seq, state)
  return action
```

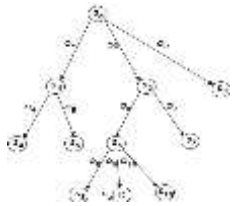

3.2.1 什么是搜索

搜索的概念：搜索(Search)——根据问题的实际情况，按照一定的策略或者规划。从知识库中寻找到可以利用的知识，从而构造出一条使得问题获得解决的推理路线的过程。

搜索的含义：搜索包含两层含义，一是根据问题的实际情况， 按照一定的策略，从知识库中寻找可利用的知识，从而构造出一条使问题获得解决的推理路线；另一是找到的这条路线是时空复杂度最小的求解路线。

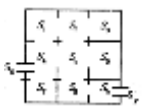
3.2.2 状态空间表示法

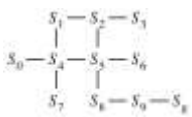
状态空间法的基本思想是用 “状态” 和 “操作” 来表示和求解问题。



3.2.3 状态空间的搜索策略

状态空间搜索的基本思想：①先把问题的初始状态作为当前扩展节点对其进行扩展，生成一组子节点。②然后检查问题的目标状态是否出现在这些子节点中。③ 若出现，则搜索成功，找到了问题的解；若没出现，则再按照某种搜索策略从已生成的子节点中选择一个节点作为当前扩展节点。④重复上述过程，直到目标状态出现在子节点中或者没有可供操作的节点为止。





3.2.4 举个例子

- 描述目标:
  到达布加勒斯特（Bucharest）
- 描述问题:
  状态：许多城市
  操作：从一个城市到另一个城市
- 查找解答:
  一系列到达的城市列表，如：
  Arad, Sibiu, Fagaras, Bucharest

3.2.4 举个例子

1、初始状态（Initial state）

2、动作（Actions）

3、转换模型（Transition model）

4、目标测试（Goal test）

5、路径代价（Path cost）

3.2.4 举个例子：初始状态（Initial state）

- Agent出发时的状态
- 初始状态位于Arad可记作：

In(Arad)

3.2.4 举个例子：动作（Actions）

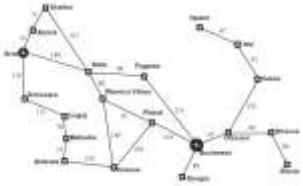
- 描述Agent可执行的动作
- Action(s) 返回 s 状态下可执行的动作序列：

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

3.2.4 举个例子：转换模型（Transition model）

- 描述Agent每个动作做什么
- RESULT(s, a)返回在 s 下动作 a 之后的状态：

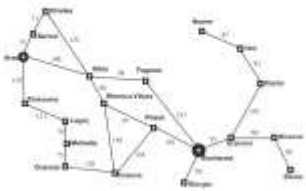
RESULT(In(Arad), Go(Zerind)) = In(Zerind)



3.2.4 举个例子：目标测试（Goal test）

- 确定一个Agent给定的状态是否是目标状态
- Agent在Bucharest的目标是单元素集合：

{In(Bucharest)}



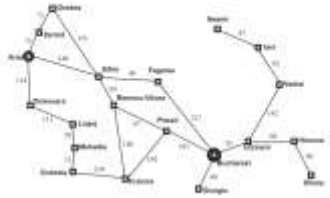
3.2.4 举个例子：路径代价（Path cost）

- 每条路径所分配的一个数值代价
- 状态 s 下执行动作 a 到达状态 s' 的步骤代价为：

c(s, a, s')

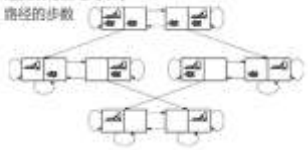
3.2.4 举个例子

- 通过图搜索在该罗马尼亚地图上生成一系列搜索路径



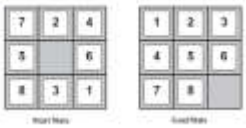
3.2.4 举个例子：真空吸尘器问题

- 状态：2个地点：$2x2^2 = 8$
- 1、初始状态：不限
- 2、动作：左移L、右移R、吸尘S
- 3、转换模型：任何转换均可，除了以下几种：
  - 在最左边左移、在最右边右移、在干净区域吸尘
- 4、目标测试：是否所有区域均干净
- 5、路径代价：路径的步数



3.2.4 举个例子：8滑块问题

- 状态：8个数字滑块每个占据一个方格，空格在最后
- 1、初始状态：不限
- 2、动作：上、下、左、右
- 3、转换模型：给定状态和动作，返回结果状态
- 4、目标测试：检查状态是否达到目标布局
- 5、路径代价：路径步数

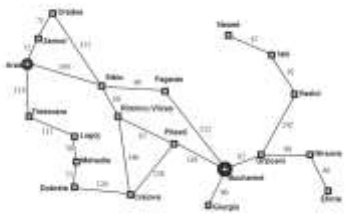


3.2.5 状态空间的搜索策略

状态空间搜索的基本分类。

搜索回顾

- 通过图搜索在该罗马尼亚地图上生成一系列搜索路径

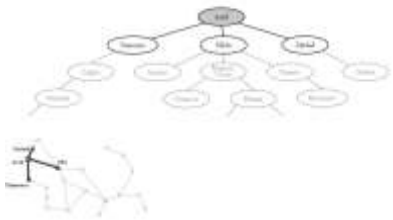


搜索回顾

- 初始状态

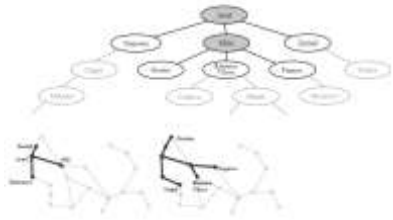
搜索回顾

- 扩展 Arad



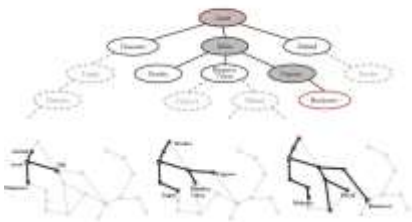
搜索回顾

- 扩展 Sibiu



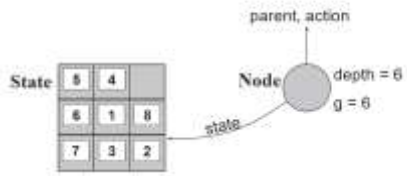
搜索回顾

- 扩展 Fagaras



nodes vs. states

- node：数据结构，包括：parent, children, depth, path cost g(x)
- state：状态表示，不包括node的元素



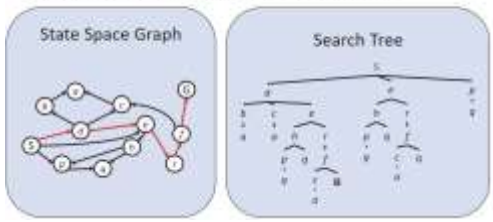
通用树搜索算法

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

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
```


State Space Graphs vs. Search Trees

- 搜索树中的每个节点都是状态空间图中的完整路径



搜索策略的常用评估指标

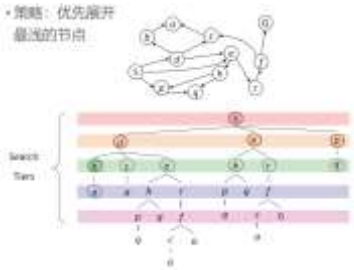
- **完备性（Completeness）**
  如果问题有解，算法就能找到，称此搜索方法是完备的。
- **最优性（Optimality）**
  如果解存在，总能找到最优解。
- **时间复杂度（Time Complexity）**
- **空间复杂度（Space Complexity）**

第三章 搜索技术

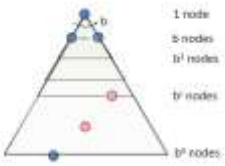
3.3.1 宽度优先搜索的基本思想

在宽度优先搜索算法中，解答树上结点的扩展是按它们在树中的层次进行的。首先生成第一层结点，同时检查目标结点是否在所生成的结点中，如果不在，则将所有的第一层结点逐一扩展，得到第二层结点，并检查第二层结点是否包含目标结点，……，对层次为n+1的任一结点进行扩展之前，必须先考虑层次完层次为n的结点的每种可能的状态。因此，对于同一层结点来说，求解问题的价值是相同的，可以按任意顺序来扩展它们。通常采用的原则是先生成的结点先扩展。

3.3.2 宽度优先搜索的过程



3.3.3 搜索性能分析



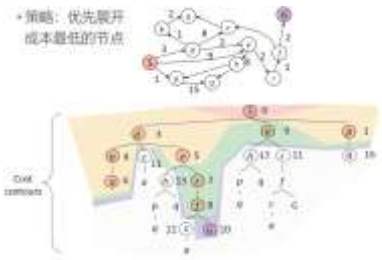
- 完备的
- 最优的
- 时间复杂度：$O(b^d)$
- 空间复杂度：$O(b^d)$

$$b + b^2 + b^3 + \cdots + b^d = O(b^d)$$

3.3.4 一致代价搜索

宽度优先搜索可以找到最短路径，但不是**成本最低**的路径。

3.3.4 一致代价搜索



第三章 搜索技术

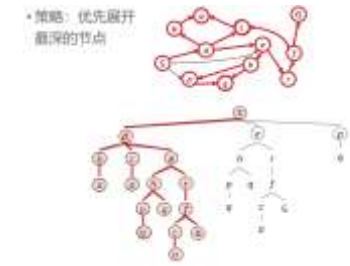
3.4.1 深度优先搜索的基本思想

深度优先搜索属于图算法的一种,它的基本思想简单的说就是对每一个分支的路径搜索到最深处，并且每一个节点只能被访问一次。深度优先搜索遍历和树的先根遍历比较类似，是沿着树的深度遍历树的节点，尽可能深的搜索树的分支。思想是从一个顶点开始，沿着一条路径一直走到最后一个节点，如果发现不能达到目标节点，就需要返回上一个节点，换一条路径重复以上过程。

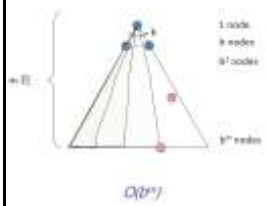
3.4.2 深度优先搜索过程

深度优先搜索DFS（Depth-First-Search）是一种沿着树的深的节点遍历的方式，尽可能的搜索更深的节点的策略。如下图是一个无向图，如果从A点发起深度优先搜索（以下的访问次序并不是唯一的，第二个点既可以是B也可以是C,D），可以得到如下的一个访问过程：A→B→E分支结束，重新回到A，A→C→F→H→G→D没有路，最终回到A，A也没有未访问的相邻节点，本次搜索结束。深度优先搜索的特点：每次深度优先搜索的结果必然是图的一个连通分量.深度优先搜索可以从多个节点发起。

3.4.2 深度优先搜索过程



3.4.3 时间复杂度



- 不完备的
- 不是最优的
- 时间复杂度：O（b^m）
- 空间复杂度：O（$b*m$）

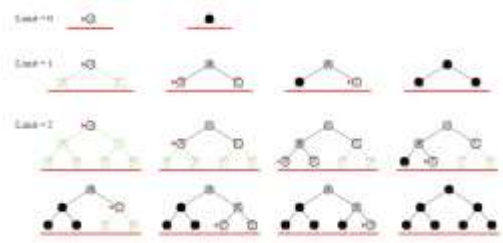
3.4.3 深度受限搜索

- 在无限状态空间深度优先搜索会有可能失败，这个问题可以通过对深度优先搜索设置界限 *l* 来避免，这种方法称为**深度受限搜索**。
- 深度受限搜索虽然使得搜索不至于无止尽地搜索下去，但是这种搜索得到的解是不完备的

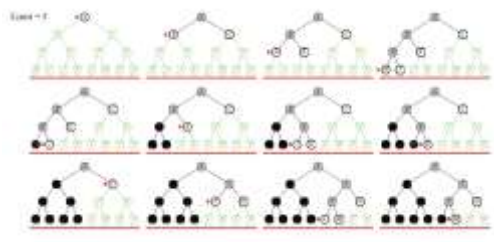
3.4.4 迭代加深的深度优先搜索

迭代加深的深度优先搜索用于确定最好的深度界限。
- 通过不断增加最大深度限制，0->1->2->...->d，直到找到目标。
- 这一算法结合了深度优先搜索和宽度优先搜索的优点，**当路径代价是搜索深度的非递减函数时该算法是最优的**

3.4.4 迭代加深的深度优先搜索搜索过程

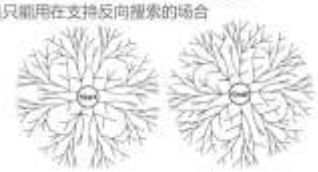


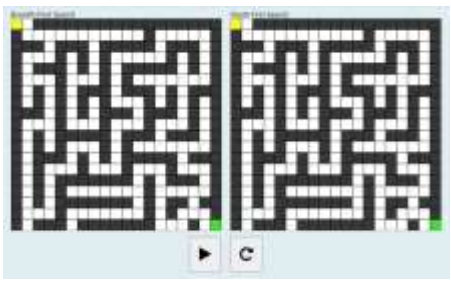
3.4.4 迭代加深的深度优先搜索搜索过程



双向搜索

- 双向搜索：同时运行两个搜索
  - 一个从初始状态向前搜索
  - 另一个从目标向后搜索
  - 如果在中间某点相遇则搜索终止
- 这种搜索的优点是搜索区域比深度搜索要小
- 限制是只能用在支持反向搜索的场合





https://seanperfecto.github.io/BFS-DFS-Pathfinder/

	广度优先	一致代价	深度优先	深度受限	迭代加深	双向
完备性	是	是	否	否	是	是
时间复杂度	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
空间复杂度	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
是否最优	是	是	否	否	是	是

一个简单的视频总结

习题：

1.请阐述状态空间的一般搜索过程。OPEN表与CLOSED表的作用是什么？2.宽度优先搜索与深度优先搜索各有什么特点？3.何谓估价函数？ 在估价函数中,g(x)和h(x)各起什么作用？