第三章 通过搜索进行问题求解

算法性能评估：

1. 完备性：存在解时，算法一定能找到解；不存在解时，一定能保证报告失败

2. 代价最优性：找到了所有解中路径代价最小的解

3. 时间复杂度：找到解需要多长时间

4. 空间复杂度：执行搜索需要多少内存

搜索算法汇总

（一）无信息搜索 p66

1. 广度优先搜索 breadth-first search

旧结点出队时，对其进行扩展：生成其孩子结点并入队

目标测试在结点**生成的时候**，而非被扩展的时候

如果路径代价是**基于结点深度**的**非递减函数**，宽度优先搜索是**最优的**。

时间复杂度 ***O(b^d)*** 空间复杂度 ***O(b^d)*** ，b:分支因子，d:解的深度

2. 一致代价搜索 uniform-cost search

一致代价搜索（uniform-cost search）扩展的是路径消耗最小的结点，而非深度最浅的结点，可以通过将边缘结点集组织成按 **g** 值**排序的队列**来实现，结点的 g 值等于初始结点到该节点的**最小代价。**

时间和空间复杂度：p67

是完备的、代价最优的

与宽度优先搜索的不同：

1. 按路径代价对队列进行排序
2. **目标检测**应用于结点被选择**扩展时**，而不是在结点生成的时候进行(**为了避免选择次优路径**)
3. 对解路径的步数并不关心，只关心路径总代价。

3. 深度优先搜索 DFS

深度优先搜索总是扩展搜索树的**当前边缘结点集**中**最深**的结点

搜索很快推进到搜索树的最深层，那里的结点**没有后继。**当那些结点扩展完之后，就**从边缘结点集中去掉**，然后搜索算法**回溯到下一个还有未扩展后继的深度稍浅的结点。**

避免重复状态和冗余路径的**图搜索**，在有限状态空间是**完备的**，因为它至多扩展所有结点。而**树搜索**，则**不完备（可能生成循环的状态后继）**

不是最优的（找到目标结点即返回）

变体：回溯搜索：在回溯搜索中，每次只产生一个后继而不是生成所有后继；每个被部分扩展的结点要记住下一个要生成的结点。这样，内存只需要 O(m)而不是 O(bm)。

4. 深度受限搜索 DLS (depth-limited search)

设置界限 l，深度为 l 的结点被当作没有后继对待。

深度受限搜索可能因为两种失败而终止标准：***failure*** 返回值指示无解；***cutoff*** 值指示在深度界限内无解

空间复杂度：***O(bl)***　时间复杂度为 ***O(b^l)***

完备性：取决于 l 的选择

可以混合使用两种搜索算法，避免一部分重复状态的生成：先用宽度优先搜索直到有效内存耗尽，然后对边缘集中的所有结点应用迭代加深的深度优先搜索。

5. 迭代加深搜索 IDS (iterative deepening search)

不断地增大深度限制，首先为 0,接着为 1，然后为 2，依此类推直到找到目标。当深度界限达到 d，即最浅的目标结点所在深度时，就能找到目标结点。

空间复杂度：***O(bd)*** 时间复杂度为 ***O(b^d)***（与宽度优先搜索相近）

当**分支因子 b 有限**时是**完备**的

当**路径代价**是**结点深度**的**非递减函数**时该算法是**最优**的

6. 双向搜索 bidirectional search

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 $\geq \varepsilon > 0$，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

（二）有信息搜索

1. 最佳优先搜索 best-first search

选择评估函数 f(n)最小的结点扩展

可显著减少搜索空间，提高效率

不一定完备（取决于评估函数的选择）、**非最优性（贪心）**

2. 贪心最佳优先搜索

试图扩展离目标最近的结点，理由是这样可能可以很快找到解。因此，它只用启发式信息，

即 f(n)=h(n)

图搜索在**有限**状态空间中是**完备的**，但在**无限**状态空间中是**不完备**的

时间和空间复杂度为 **O(|V|)**，使用好的启发式函数可以大大降低

3. A*搜索

结点评估：f(n)=g(n)+h(n),

g(n): 从开始结点到结点 n 的路径代价

h(n): 从结点 n 到目标结点的最小代价路径的估计值

因此 f(n)= 经过结点 n 到一个目标状态的最优路径的估计代价值

A*搜索是**完备的**，算法与一致代价搜索类似

启发式是可容许的时，A*是最优的

使用一致的启发式时，A*不会搜索重复状态

4. 内存受限搜索

束搜索：只保留具有最优 f 值的 k 个结点，放弃其他已扩展结点

只保留最优 f 值δ范围内的所有结点

迭代加深 A*搜索 IDA*

递归优先最佳搜索 RBFS

1. **动态回溯**：通过 f_limit 变量记录当前路径之外的最优备选路径的启发式估值（f 值），当当前路径的 f 值超过 f_limit 时触发回溯。
2. **倒推值（Backed-up Value）**：记录每个节点子树的**最优 f 值**，用于后续重新扩展时判断是否需要回溯或继续搜索。
3. **递归分层**：以递归调用替代显式队列管理，通过函数调用栈隐式维护搜索路径。

IDA*和 RBFS 的问题在于它们**使用的内存过于小**了。两个算法都**忘记了它们做过什么**，所以终止时**有些状态可能重复扩多次**。图中的冗余路径会带来复杂度的潜在的指数级的增长

启发式函数

一个约束：当 n 为目标状态时，h(n)=0

（1）h(n)是一个可容许启发式：它从**不会过高估计**到达目标的代价

故 **f(n)=g(n)+h(n)　<=　经过结点 n 的解的实际代价**

（2）启发式 h(n)是一致的：**h(n) <= c(n, a, n') + h(n')**

c(n, a, n') ：结点 n 通过行动 a 生成 n 的后继结点 n’的**实际代价**

松弛问题中最优解的代价值可以作为原问题的一个一致(可容许)的启发式函数

总结：p91

第四章

1.爬山法

最陡上升爬山法(steepest ascent hill climbing)

随机爬山法（stochastic hill climbing）

在**上坡行动中**随机选择一个；被选中的**概率**随着上坡陡度的变化而变化。

收敛得更慢，但**可能找到更好的解。**

首选爬山法（first-choice hill climbing）

通过不断**随机地生成后继**直到生成一个**比当前状态更好**的后继为止来实现随机爬山。

当一个状态**存在众多（如数千个）后继**时，这是一个很好的策略。

随机重启爬山法（random-restart hill climbing）

它从**随机生成的初始状态**开始，执行一系列爬山搜索，直到找到目标。

算法**完备**的**概率为 1**

2.模拟退火法 stimulated annealing

随机移动。如果该移动使情况改善，该移动则被接受。否则，算法以某个小于 1 的概率($e^{-\Delta E/T}$，

ΔE 评估值变坏的量）接受该移动。

开始 T 高的时候可能允许坏的移动，T 越低则越不可能发生。

如果调度**让 T 下降得足够慢，算法找到全局最优解的概率逼近于 1。**

3. 局部束搜索

它从 k 个随机生成的状态开始。每一步全部 **k** 个状态的所有后继状态全部被生成。如果其中有一个是目标状态，则算法停止。否则，它从整个后继列表中选择 **k 个最佳**的后继，重复这个过程。　　**有用的信息在并行的搜索线程之间传递**

4. 进化算法 evolutionary algorithm

（1）遗传算法从 **k** 个随机生成的状态开始，称之为种群。每个状态(个体)用一个有限长字符串表示

（2）产生下一代状态：每个状态都由它的目标函数(适应度函数)给出评估值

（3）按照**适应度给出的概率**随机地选择两对进行繁殖。

（4）父串在杂交点上进行杂交而创造出后代

（5）后代**串每个位置**都会按照**某个小的独立概率随机变异**

第五章

α-β剪枝：

α = 到目前为止路径上发现的 **MAX** 的最佳(即极大值）选择 **至少**

β = 到目前为止路径上发现的 **MIN** 的最佳(即极小值）选择 **至多**

递归搜索时，子节点继承父节点的 α 和 β 值，

MAX 层：将所有可能的行动 a 依次丢给 MIN 层来决策，得到效用值 v2(和 MIN 决策 move2)

比较 v2 与当前最佳效用值 v，若 v2>v,，则更新 v 为 v2，更新最佳动作 move 为 a.

最后将最大的 v 更新给α

剪枝条件：若α≥β，直接返回 v, move

MIN 层: 将所有可能的行动 a 依次丢给 MAX 层来决策，得到 v2(和 MAX 决策 move2)

比较 v2 与当前最佳效用值 v，若 v2<v,，则更新 v 为 v2，更新最佳动作 move 为 a

最后将最小的 v 更新给β

剪枝条件：若β≤α，直接返回 v, move

第六章

增强弧一致性的算法：AC-3 算法

维护一个弧队列：

初始时，队列中包含 CSP 中的所有弧（每个二元约束都有两条弧，每个方向各一条）

#从队列中任意弹出一条弧（Xi，Xj）并使 Xi 相对于 Xj 弧一致（修正 Xi 的域 Di）

　　如果 Di 保持不变，继续处理下一条弧

　　但是如果 Di 得以修正（域变小），那么将所有的弧(Xk, Xi)（**Xi 的所有邻居**）添加到队列中

如果 **Di 变为空集**，那么表示整个 CSP 不存在一致解，**返回失败**

否则回到#，**直到队列中没有弧**

第七章

命题逻辑语句转 CNF：

（1）消去$\Leftrightarrow$，将$\alpha \Leftrightarrow \beta$替换为$(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$:

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

（2）消去$\Rightarrow$，将$\alpha \Rightarrow \beta$替换为$\neg\alpha \vee \beta$

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

（3）将所有¬移到文字前，保证只有文字前出现¬

一阶逻辑常见错误

1. 使用 ∧ 而非 ⇒ 与全称量词搭配，导致过强陈述，但修改不一定是改掉∧

2. 使用 ⇒ 而非 ∧ 与存在量词搭配，导致过弱陈述 （一般总能找到对象使前提为假）

$\exists x\ Crown(x) \Rightarrow OnHead(x, John)$ 一定有 x 不是小丑，导致恒真

3. 量词嵌套中，量词的顺序很重要；

两个量词与相同的变量名合用： $\forall x\ (Crown(x) \lor (\exists x\ Brother(Richard, x)))$

规则是，变量属于提及它的最内层量词，随后便不再受任何其他量词约束

4. 量化语句的德摩根律

∀实际上是对全体对象的合取而∃则是析取

$$\neg\exists x\ P \equiv \forall x\ \neg P \qquad \neg(P \lor Q) \equiv \neg P \land \neg Q$$

$$\neg\forall x\ P \equiv \exists x\ \neg P \qquad \neg(P \land Q) \equiv \neg P \lor \neg Q$$

$$\forall x\ P \equiv \neg\exists x\ \neg P \qquad P \land Q \equiv \neg(\neg P \lor \neg Q)$$

$$\exists x\ P \equiv \neg\forall x\ \neg P \qquad P \lor Q \equiv \neg(\neg P \land \neg Q)$$

否命题是改变量词（但不在量词前加┐），┐命题内容

归结证明的步骤

将证明 KB⇒Q 有效 转化为 KB∧┐Q 不可满足

1. KB∧┐Q 一阶逻辑化为 CNF

1. 蕴含消去：A⇒B 化为 ┐A∨B

2. ┐内移：**量词上的否定内移**，**括号上的否定内移**

3. 变量标准化：每个量词限定的变量要使用**不同的变量名**

4. 斯科伦化：消去**存在**量词

5. 全称量词消除：直接去掉∀x 符号，保留文字内的变量即可

6. 对∧分配∨

2. (对变量进行斯科伦化)，归结互补文字，直到推导出空语句或失败

每次选择┐Q 的 CNF 中的一项子句，使用合一子（若需要）对 KB 进行归结