

人工智能笔记

陈鸿峥

2020.01*

目录

| | | |
|----------|------------------------------|-----------|
| 1 | 简介 | 2 |
| 1.1 | 概述 | 2 |
| 1.2 | 历史 | 2 |
| 2 | 搜索 | 3 |
| 2.1 | 无信息搜索 | 3 |
| 2.2 | 有信息搜索 | 6 |
| 2.3 | 博弈树搜索 | 8 |
| 3 | 限制可满足性问题 | 11 |
| 3.1 | 回溯(backtracking)搜索 | 11 |
| 3.2 | 向前检测 | 12 |
| 3.3 | 泛化边一致性(GAC) | 13 |
| 4 | 知识表示与推理 | 14 |
| 4.1 | 基本概念 | 14 |
| 4.2 | 推理 | 16 |
| 5 | 确定性规划 | 20 |
| 5.1 | 情景演算 | 20 |
| 5.2 | STRIPS | 21 |
| 5.3 | 松弛问题 | 21 |
| 6 | 不确定性规划 | 23 |
| 6.1 | 基础知识 | 23 |
| 6.2 | 贝叶斯网络 | 24 |

*Build 20200103

| | |
|---------------------|-----------|
| 7 机器学习 | 28 |
| 7.1 决策树 | 28 |
| 7.2 贝叶斯学习 | 28 |
| 7.3 聚类算法 | 29 |
| 7.4 神经网络 | 29 |
| 7.5 强化学习 | 30 |

1 简介

1.1 概述

- 1997 Deep Blue
- 2011 IBM Watson
- 2016 Google DeepMind

什么是AI?

- 像人类一样思考(thinking humanly): 中文屋子
- 理智思考(thinking rationally)
- 像人类一样行为(acting humanly): 图灵测试(1950)
- 理智行为(acting rationally)

常见术语

- 强AI: 机器像人类一样思考
- 弱AI: 机器有智能的行为
- 通用AI(AGI): 能够解决任何问题
- 窄AI: 专注于某一特定任务

不以模拟人类作为实现人工智能的最好方法

- 计算机和人类的体系结构不同: 数值计算、视觉、并行处理
- 对人类大脑的了解太少了!

1.2 历史

- 1950-70: Early excitement, great expectations
 - Samuel(1952)跳棋程序
 - Newell(1955)逻辑理论家
 - Dartmouth会议(1956): AI诞生

- 1970-90: Knowledge is power
- 1990-: rise of machine learning "AI Spring"
- 2010-: Deep learning

2 搜索

搜索主要包括无信息(uninformed)搜索和有信息搜索。

- 状态空间(state space)
 - 传统搜索：状态空间可见、动作确定性
 - 非传统搜索：局部搜索、模拟退火、爬坡
 - 动作(action)：不同状态之间的转换
 - 初始状态(initial state)
 - 目标/期望(goal)
- 树搜索，边界集(frontier)是未探索的状态集合

Algorithm 1 Tree Search

```

1: procedure TREESearch((Frontier, Successors, Goal?))
2:   if Frontier is empty then
3:     return failure
4:   Curr = select state from Frontier
5:   if Goal?(Curr) then
6:     return Curr
7:   Frontier' = (Frontier - {Curr})  $\cup$  Successors(Curr)
8:   return TreeSearch(Frontier', Successors, Goal?)

```

搜索需要关注的几个特性：

- 完备性：若解存在，搜索是否总能找到解
- 最优性：是否总能找到最小代价的解
- 时间复杂性：最大需要被生成或展开¹的结点数
- 空间复杂性：最大需要被存储在内存中的结点数

2.1 无信息搜索

无信息搜索并不考虑关于特定搜索问题的领域特定的信息，主要包括宽度优先、一致代价、深度优先、深度受限、迭代加深五种算法。

¹而不是探索的结点数

2.1.1 宽度优先搜索(BFS)

将后继加入边界集的**后面**， b 为最大状态后继数目/分支因子(branching factor)， d 为最短距离解的行动数（注意是**边数**，而不是**层数**！）

- 完备性与最优性：所有短路总在长路前被探索，某一长度只有有限多条路径，最终可以检测所有长度为 d 的路径，从而找到最优解
- 时间复杂度： $1 + b + b^2 + \dots + b^d + (b^d - 1)b = O(b^{d+1})$ ，最差情况在最后一层的最后一个节点才探索到最优解，从而前面 b 个节点都要展开第 $d + 1$ 层
- 空间复杂度： $b(b^d - 1) = O(b^{d+1})$ ，需要将边界集都存储下来，同上最后一层

2.1.2 深度优先搜索(DFS)

将后继加入边界集的**前面**，即总是展开边界集中最深的节点

- 完备性
 - 无限状态空间：不能保证
 - 有限状态空间无限路径：不能保证（可能有环）
 - 有限状态空间+路径/重复状态剪枝：可以保证
- 最优性：因完备性不能保证，故最优性也不能保证
- 时间复杂性： $O(b^m)$ ，其中 m 为状态空间的最长路的长度（若 $m \gg d$ ，则非常糟糕；但如果有大量解路径，则会快于BFS）
- 空间复杂性： $O(bm)$ ，**线性空间复杂性**是DFS最大的优点。边界集只包含当前路径的最深节点以及回溯节点（backtrack points为当前路径上节点的未探索的兄弟sibling）。注意这里需要记录路径上每个结点的孩子，因已被扩展。

2.1.3 一致代价(Uniform-cost)

一致代价搜索(Uniform cost search, UCS)²的边界集以路径开销升序排序，总是先展开最低开销的路径。如果每一个动作都是一样的代价，则一致代价等价于BFS。

- 完备性与最优性：假设所有转移都有代价 $\geq \varepsilon > 0$ ，所有更低代价的路径都在高代价路径之前被展开，只有有限多的路径开销小于最优解的开销，故最终一定会到达最优解
- 时间复杂性： $O(b^{\lfloor C^*/\varepsilon \rfloor + 1})$ ，对应着BFS中 $d = C^*/\varepsilon$ ，其中 C^* 为最优解的开销，最坏情况就是每一层开销都很小为 ε ，那么需要 $\lfloor C^*/\varepsilon \rfloor$ 层
- 空间复杂性： $O(b^{\lfloor C^*/\varepsilon \rfloor + 1})$

²至于为什么叫Uniform， 可以看<https://math.stackexchange.com/questions/112734/in-what-sense-is-uniform-cost-search-uniform>和<https://cs.stackexchange.com/questions/6072/why-is-uniform-cost-search-called-uniform-cost-search>，比较合理的解释是到达同一结点的cost都被认为是相同的（寻找最优解时）。一致的算法总是选择边界集中第一个元素。

2.1.4 深度受限搜索(Depth-limited)

只在最大深度执行DFS，因此无穷路径长不会存在问题

- 完备性与最优性：不能保证，若解的深度大于 L
- 时间复杂度： $O(b^L)$
- 空间复杂度： $O(bL)$

2.1.5 迭代加深搜索(Iterative Deepening)

IDS逐渐增加最大深度 L ，对每一个 L 做深度受限搜索

- 完备性：可以保证
- 最优性：如果开销一致³，则可以保证
- 时间复杂性： $(d+1)b^0 + db + (d-1)b^2 + \dots + b^d = O(b^d)$ ，第0层搜了 $(d+1)$ 次，以此类推。可以看到时间复杂度是比BFS优的，因为最后一层的结点并未进行展开。
- 空间复杂性： $O(bd)$ ，同DFS

2.1.6 双向搜索(Bidirectional)

从源结点和汇结点同时采用BFS，直到两个方向的搜索汇聚到中间。

- 完备性：由BFS保证
- 最优性：若一致代价则可保证
- 时间复杂性： $O(b^{d/2})$
- 空间复杂性： $O(b^{d/2})$

2.1.7 环路/路径检测

所有检测都是在扩展时进行：

- 环路(cycle)检测：检测当前状态是否与所有已探索的状态重复(BFS)
- 路径(path)检测：只检测当前状态是否与该路径上的状态重复(DFS)

注意不能将环路检测运用在BFS上，因为这会破坏其空间复杂度的优势。

环路检测运用到UCS上依然可以保证最优性⁴。因为UCS第一次探索到某一状态的时候已经发现最小代价路径，因而再次探索该状态不会发现路径比原有的更小。

³若开销不一致，则可以采用代价界(cost bound)来代替：仅仅展开那些路径开销小于代价界的路径，同时要记录每一层深搜的最小代价。这种方式的搜索开销会非常大，有多少种不同路径开销就需要多少次迭代循环。

⁴注意这在启发式搜索中不一定成立

2.1.8 总结

| | BFS | UCS | DFS | Depth-limited | IDS | Bidirectional |
|-------|----------|--|----------|---------------|----------|---------------|
| 完备性 | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |
| 时间复杂度 | $O(b^d)$ | $O(b^{\lfloor C^*/\varepsilon \rfloor + 1})$ | $O(b^m)$ | $O(b^l)$ | $O(b^d)$ | $O(b^{d/2})$ |
| 空间复杂度 | $O(b^d)$ | $O(b^{\lfloor C^*/\varepsilon \rfloor + 1})$ | $O(bm)$ | $O(bl)$ | $O(bd)$ | $O(b^{d/2})$ |
| 最优性 | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |

例 1. N 个传教士和 N 个食人族要过河，他们都在河的左岸。现在只有一条船能够运载 K 个人，要把他们都运往右岸。要满足无论何时何地，传教士的数目都得大于等于食人族的数目，或者传教士数目为0。

分析. 考虑对问题形式化为搜索问题

- 状态 (M, C, B) ，其中 M 为左岸传教士数目， C 为左岸食人族数目， $B = 1$ 指船在左岸
- 动作 (m, c) 指运 m 个传教士和 c 个食人族到对岸
- 先决条件：传教士数目和食人族数目满足限制
- 效果： $(M, C, 1) \xrightarrow{(m, c)} (M - m, C - c, 0)$
 $(M, C, 0) \xrightarrow{(m, c)} (M + m, C + c, 1)$

2.2 有信息搜索

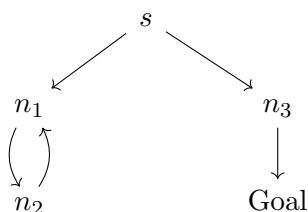
在无信息搜索中，我们从不估计边界集中最有期望(promising)获得最优解的结点，而是无区别地选择当前边界集中第一个结点。然而事实上，针对不同问题我们是有对结点的先验知识(apriori knowledge)的，即从当前结点到目标结点的开销有多大。而这就是有信息搜索(informed)，或者称为启发式搜索(heuristics)。

关键在于领域特定启发式函数 $h(n)$ 的设计，它估计了从结点 n 到目标结点的开销(cost)。注意满足目标状态的结点 $h(n) = 0$ 。

2.2.1 贪心最优搜索(Greedy Best-First Search)

直接使用 $h(n)$ 对边界集进行排序，但这会导致贪心地选择看上去离目标结点开销最小的路径。

如果存在环路，贪心最优搜索是不完备的，会陷入死循环。



2.2.2 A*搜索

综合考虑当前已走的开销和未来估计的开销。定义一个估值函数

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

其中 $g(n)$ 为路径到节点 n 的代价, $h(n)$ 为从 n 到目标节点的代价, 采用 $f(n)$ 对边界集内的节点进行排序。
 $f(n)$ 需要满足下列两个性质。

定义 1 (可采纳的(admissibility)). 假设所有代价 $c(n_1 \rightarrow n_2) \geq \varepsilon > 0$, 令 $h^*(n)$ 为从节点 n 到目标节点 ∞ 的最优解代价⁵, 若

$$\forall n: h(n) \leq h^*(n)$$

则称 $h(n)$ 是可采纳的。即一个可采纳的启发式函数总是**低估**了当前结点到目标结点的真实开销（这样才能保证最优解不被排除）。因此对于任何目标结点 g 都有 $h(g) = 0$ 。

定义 2 (一致性(consistency)/单调性(monotonicity)). 若对于所有的结点 n_1 和 n_2 , $h(n)$ 满足（三角不等式）

$$h(n_1) \leq c(n_1 \rightarrow n_2) + h(n_2)$$

则称 $h(n)$ 是单调的。

定理 1. 一致性蕴含可采纳性

分析. 分类讨论

- 当结点 n 没有到目标结点的路径, 则 $h(n) \leq h^*(n) = \infty$ 恒成立
- 令 $n = n_1 \rightarrow n_2 \rightarrow \dots \rightarrow n_k$ 为从结点 n 到目标结点的最优路径, 则可以用数学归纳法证明 $\forall i: h(n_i) \leq h^*(n_i)$, 如下从后往前推

$$h(n_i) \leq c(n_i \rightarrow n_{i+1}) + h(n_{i+1}) \leq c(n_i \rightarrow n_{i+1}) + h^*(n_{i+1}) = h^*(n_i)$$

定理 2. 可采纳性蕴含最优性

分析. 假设最优解有开销 C^* , 则任何最优解一定会在开销大于 C^* 的路径之前被展开（有限条路径）。

反证若路径 p 的代价 $c(p)$ 大于最优解路径代价 $c(p^*)$ 且 p 在 p^* 之前被扩展, 则一定存在一个节点 n 在 p^* 上且仍在边界集中, 因此

$$c(p) \leq f(p) \leq f(n) = g(n) + h(n) \leq g(n) + h^*(n) = c(p^*)$$

矛盾。故在最优解展开之前的路径一定有开销 $\leq C^*$, 最终我们一定会检测到最优解, 而且次优解不会在最优解之前被检测。

做环检测可能导致找不到最优解。可采纳性不能保证最优解, 但**单调性环检测保证最优解**。

单调性有以下几个性质

命题 1. 路径上的 f 一定是非递减的

⁵如果没有路径则 $h^*(n) = \infty$

分析.

$$\begin{aligned}f(n) &= g(n) + h(n) \\&\leq g(n) + c(n \rightarrow n') + h(n') \\&= g(n') + h(n') \\&= f(n')\end{aligned}$$

命题 2. 如果 n_2 在 n_1 之后被扩展, 则 $f(n_1) \leq f(n_2)$

命题 3. 当 n 在任何小于 f 值得路径之前被展开

命题 4. A^* 算法第一次展开某个状态时, 它已经找到了到达那个状态的最小开销路径。

2.2.3 迭代加深 A^* (IDA)算法

A^* 算法有和BFS或UCS同样的空间复杂性问题, 而迭代加深 A^* 算法同样解决空间复杂度的问题。与迭代加深类似, 但IDA*则用 $f = g + h$ 作为截断阈值。每一轮迭代中选择上一轮中超过截断阈值最小的 f 。

2.2.4 构造启发式函数

常常需要考虑一个更加简单的问题 (松弛问题), 然后让 $h(n)$ 为到达一个简单问题解的开销。比如考虑 A 和 B 之间没有屏障, A 和 B 相邻等。

例 2. 现有积木若干, 积木可以放在桌子上, 也可以放在另一块积木上面。有两种操作:

1. $\text{move}(x, y)$: 把积木 x 放到积木 y 上面, 前提是积木 x 和积木 y 上面都没有其他积木
2. $\text{moveToTable}(x)$: 把积木 x 放到桌子上, 前提是积木 x 上面无其他积木, 且积木 x 不在桌子上

设计一个可采纳的启发式函数 $h(n)$

分析. $h(n) = h_1(n) + h_2(n)$, 其中 $h_1(n)$ 为在目标状态积木的数目, $h_2(n)$ 为符合上下关系的积木数目 (A 在 B 下, 且 A 在目标状态)

定理 3. 在松弛问题中的最优解也是原问题中的一个可满足的启发式函数。

定义 3 (支配). 如果 h_2 支配(*dominate*) h_1 且除了目标结点都可满足, 则 $h_1(n) \leq h_2(n)$

2.3 博弈树搜索

博弈的一些前提

- 两个博弈玩家
- 离散值: 游戏和决策都可以映射到离散空间
- 有限的: 只有有限的状态和可能的决策
- 零和博弈: 完全竞争, 即如果一个玩家胜利, 则另外一个失去同样数量的收益
- 确定性的: 没有牵涉到概率性事件, 如色子、抛硬币等
- 完美信息博弈: 状态的所有方面都可以被完全观察, 即没有隐藏的卡牌

剪刀石头布是简单的一次性(one-shot)博弈

- 一次移动
- 在博弈论中称为策略或范式博弈(strategic/normal form)

但很多游戏是牵涉到多步操作的

- 轮回(turn-taking)游戏，如棋类
- 在博弈论中称为扩展形式博弈(extensive form)

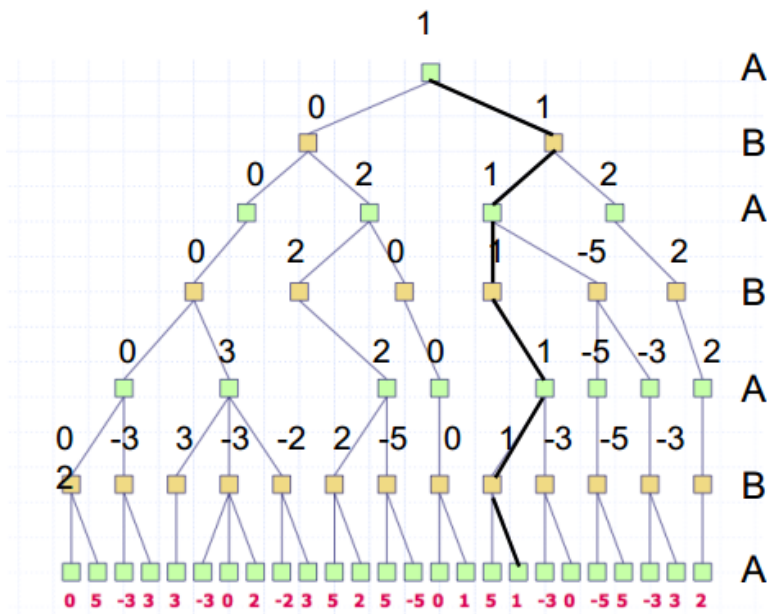
两个玩家A（最大化己方收益）和B（最小化对方收益）

- 状态集合 \mathcal{S}
- 初始状态 $I \in \mathcal{S}$
- 终止位置 $T \subset \mathcal{S}$
- 后继：下一可能状态的集合
- 效益(utility)/收益(payload)函数 $V : T \mapsto \mathbb{R}$ ，表明终止状态对A玩家有多好，对B玩家有多坏（都站在A角度给出）

2.3.1 Minimax策略

自己选max，对方选min

- 构建整棵博弈树，然后将终止/叶子结点标上收益
- 回溯整棵树，然后将每个结点都标记上收益



用DFS可以遍历整棵树，同时保持线性的空间复杂度，每次回溯时更新结点为min/max即可

2.3.2 Alpha-Beta剪枝

注意 α - β 剪枝只要有一祖先大于/小于后代节点的值就可以进行剪枝。

- 只要当前Max结点的值 \geq 祖先某一Min结点的值，就可以在该Max结点上做 α 剪枝
- 只要当前Min结点的值 \leq 祖先某一Max结点的值，就可以在该Min结点上做 β 剪枝

Algorithm 2 Alpha-Beta Pruning

```
1: procedure ALPHABETA( $n, \text{Player}, \alpha, \beta$ )
2:   if  $n$  is TERMINAL then
3:     return  $V(n)$  ▷ Return terminal states utility
4:   ChildList =  $n$ .Successors(Player)
5:   if Player == MAX then
6:     for  $c$  in ChildList do
7:        $\alpha = \max(\alpha, \text{AlphaBeta}(c, \text{MIN}, \alpha, \beta))$ 
8:       if  $\beta \leq \alpha$  then
9:         break
10:    return  $\alpha$ 
11:  else ▷ Player == MIN
12:    for  $c$  in ChildList do
13:       $\beta = \min(\beta, \text{AlphaBeta}(c, \text{MAX}, \alpha, \beta))$ 
14:      if  $\beta \leq \alpha$  then
15:        break
16:    return  $\beta$  ▷ Initial call: AlphaBeta(START-NODE, Player,  $-\infty, +\infty$ )
```

可以证明，如果原始情况需要访问 $O(b^D)$ 个结点，则经过 α - β 剪枝后只需访问 $O(b^{D/2})$ 个结点。

2.3.3 其他补充

但在现实生活的游戏中，即使采用了 α - β 剪枝，博弈树也太过庞大。如棋类的分支因子大致是35，深度为10的树已经到 2.7×10^{14} 个结点。因此不能将整棵博弈树展开，需要采用一些启发式方式进行估计。

评价函数(evaluation)的一些需求：

- 对于终止状态，评价函数的序应与真实的收益函数相同
- 对于非终止状态，评价函数则应该与真实的胜率相关联
- 计算时间不能花太长
- 通常取多个特征，然后进行加权求和（先验知识）

在线(online)/实时(real-time)搜索

- 没有办法展开全部的边界集，因此限制展开的大小（在没找到去目标的真实路径就做出决定/直接选一条路就开始走）
- 在这种情况下，评价函数不仅仅引导搜索，更是提交真实的动作
- 虽然找不到最优解，但是求解时间大大缩减

3 限制可满足性问题

在搜索问题中，状态表示是个黑箱，可以有多种多样的方法来表达。但实际上我们可以有特定的状态表示方法来解决大量不同的问题，在这种情况下的搜索算法可以变得很高效。

限制可满足性问题(Constraint Satisfaction Problem, CSP)指每一个状态都可以用一组特征值向量表示的问题。

- k 个特征/变量的集合 V_1, \dots, V_n
- 每一个变量都有一个包含有限值的论域 $\text{dom}[V_i]$ ，如

$$\text{height} = \{\text{short}, \text{average}, \text{tall}\}$$

- 一组限制条件 C_1, \dots, C_m
 - 每个限制条件都有一个作用域(scope)，表示作用在什么变量上，如 $C(V_1, V_2, V_4)$
 - 相当于一个布尔函数，从变量赋值到布尔值的映射，如

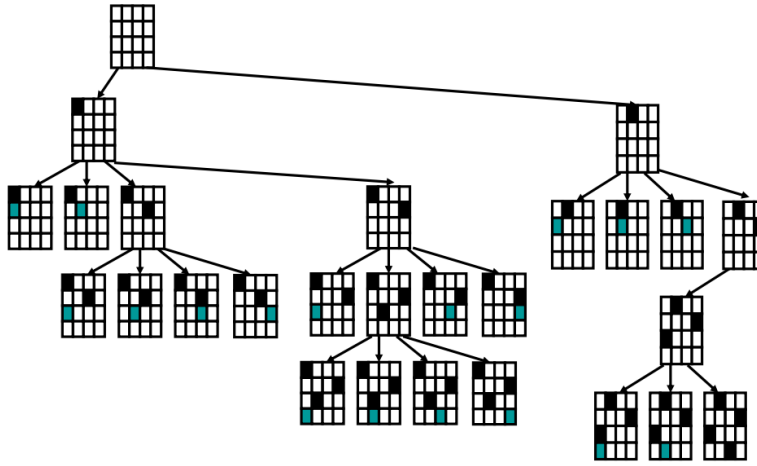
$$C(V_1 = a, V_2 = b, V_4 = c) = \text{True}$$

- 布尔函数可以以表形式给出，或以表达式形式给出，如 $C(V_1, V_2, V_4) = (V_1 = V_2 + V_4)$
 - 一个状态可以通过给每一个变量赋值得到
- CSP不关心到目标状态的移动步骤，而只关心是否存在这样一组变量满足目标。

3.1 回溯(backtracking)搜索

对每一个变量分别赋值，深搜方式，同时结合启发式函数，用于在每一步选择不同的赋值变量。

```
BT(Level)
  If all variables assigned
    PRINT Value of each Variable
    RETURN or EXIT (RETURN for more solutions)
                    (EXIT for only one solution)
  V := PickUnassignedVariable()
  Assigned[V] := TRUE
  for d := each member of Domain(V) (the domain values of V)
    Value[V] := d
    ConstraintsOK = TRUE
    for each constraint C such that
      a) V is a variable of C and
      b) all other variables of C are assigned:
          ;(rarely the case initially high in the search tree)
      IF C is not satisfied by the set of current
        assignments:
          ConstraintsOK = FALSE
    If ConstraintsOk == TRUE:
      BT(Level+1)
  Assigned[V] := FALSE //UNDO as we have tried all of V's values
  return
```



回溯的问题在于不能提前探测到某一变量已经没有可以赋值了，导致依然要进入一层进行搜索。因此考虑前瞻式算法，即限制传播(propagation)。

- 甚至可以在还未进行搜索之前就采用
- 传播本身需要耗费资源，因此这里存在一个权衡

进而有下列两种传播算法：前向传播和GAC算法。

3.2 向前检测

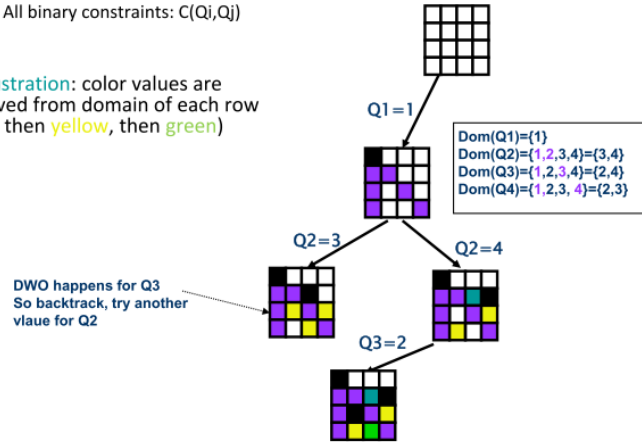
1. 选择一个未被赋值的变量 V 。这里可以采用最小剩余值(Minimum Remaining Values, MRV)作为启发式函数，即先选论域小的变量。
2. 选择论域 $\text{dom}[V]$ 中的值对 V 进行赋值 d
3. 将 d 向前传递给含有 V 的限制 C ，主要考察那些只剩一个未赋值变量 X 的限制
4. 检测 $\text{FCCheck}(C, X)$ 是否出现论域清空(Domain Wipe Out, DWO)，即 X 没得选值了。这里 FCCheck 做的则是核心的限制传播部分，当 $V = d$ 后把 X 不能取的值删去
5. 如果不出现DWO，则进入下一层（选择新的未赋值变量赋值）
6. 否则需要恢复当层 FCCheck 剪枝的部分，即 d 不可取， V 要重新取值

```

FC(Level) /*Forward Checking Algorithm */
    If all variables are assigned
        PRINT Value of each Variable
        RETURN or EXIT (RETURN for more solutions)
                        (EXIT for only one solution)
    V := PickAnUnassignedVariable()
    Assigned[V] := TRUE
    for d := each member of CurDom(V)
        Value[V] := d
        DWOoccured:= False
        for each constraint C over V such that
            a) C has only one unassigned variable X in its scope
            if (FCCheck(C,X) == DWO) /* X domain becomes empty*/
                DWOoccured:= True
                break /* stop checking constraints */
        if(not DWOoccured) /*all constraints were ok*/
            FC(Level+1)
        RestoreAllValuesPrunedByFCCheck()
    Assigned[V] := FALSE //undo since we have tried all of V's values
    return;
    
```

- 4X4 Queens
 - Q1,Q2,Q3,Q4 with domain {1..4}
 - All binary constraints: $C(Q_i, Q_j)$

- FC illustration: color values are removed from domain of each row (blue, then yellow, then green)



其实这是人工解数独一种很常见的方法，即当前格填了某一个值后会不会导致其他格填不了。

3.3 泛化边一致性(GAC)

定义 4 (一致). 限制 $C(X, Y)$ 是一致的，当且仅当对于所有 X 的值都存在某些 Y 满足 C ，即 $\forall X \exists Y : C(X, Y)$ 。

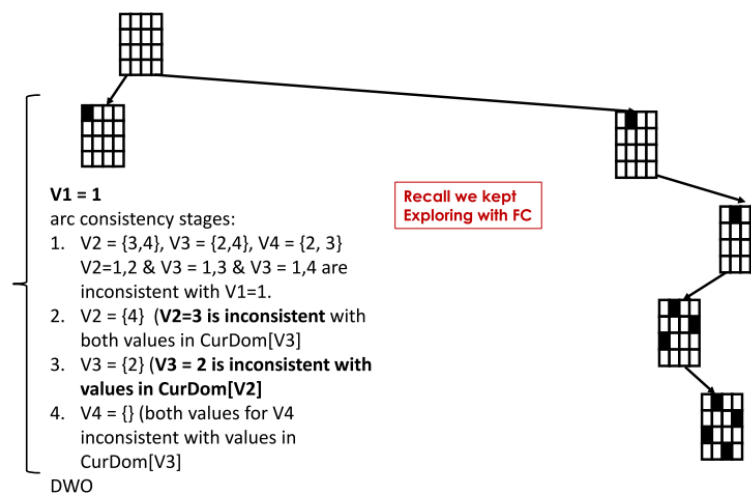
定义 5 (泛化边一致性(Generalized Arc Consistency, GAC)). 限制 $C(V_1, V_2, \dots, V_n)$ 是关于 V_i 边一致的，当且仅当 $\forall V_i, \exists V_1, \dots, V_{i-1}, V_{i+1}, \dots, V_n$ 满足 C 。限制 C 是 GAC 的当且仅当对于每一变量都是 GAC 的。一个 CSP 是 GAC 的当且仅当它的限制都是 GAC 的。

有 GAC 算法：

- 在 $V_i = d$ 下，没有其他变量赋值能够满足该限制，则 d 是边不一致的，进而可以被剪枝剪掉。
- 注意当从论域中移除一个值时可能导致新的不一致，故需要采用队列的方式，不断将需要检测边一致性限制添加，直到队列为空，即限制条件变为 GAC。
- 近似可理解为将后续搜索步骤都前移到剪枝部分。
- 注意在未进入循环之前就可以先进行检测了。（同样类比数独，先将每一个格不能填的数字都划掉）

```
GAC(Level) /*Maintain GAC Algorithm */
If all variables are assigned
  PRINT Value of each Variable
  RETURN or EXIT (RETURN for more solutions)
  (EXIT for only one solution)
V := PickAnUnassignedVariable()
Assigned[V] := TRUE
for d := each member of CurDom(V)
  Value[V] := d
  Prune all values of V ≠ d from CurDom[V]
  for each constraint C whose scope contains V
    Put C on GACQueue
  if (GAC_Enforce() != DWO)
    GAC(Level+1) /*all constraints were ok*/
  RestoreAllValuesPrunedFromCurDoms()
Assigned[V] := FALSE
return;
```

```
GAC_Enforce()
// GAC-Queue contains all constraints one of whose variables has
// had its domain reduced. At the root of the search tree
// first we run GAC_Enforce with all constraints on GAC-Queue
while GACQueue not empty
  C = GACQueue.extract()
  for V := each member of scope(C)
    for d := CurDom[V]
      Find an assignment A for all other
      variables in scope(C) such that
      C(A ∪ V=d) = True
      if A not found
        CurDom[V] = CurDom[V] - d
        if CurDom[V] = ∅
          empty GACQueue
          return DWO //return immediately
        else
          push all constraints C' such that
          V ∈ scope(C') and C' ∉ GACQueue
          on to GACQueue
return TRUE //while loop exited without DWO
```

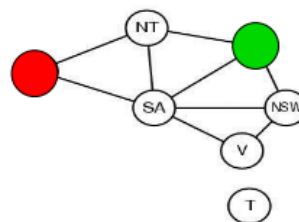


定义 6 (支撑). 在限制 C 下变量赋值 $V = d$ 的支撑是对 C 中其他变量的赋值 A , 使得 $A \cup \{V = d\}$ 满足 C 。
 即 **其他变量的指派构成了当前变量的支撑**。

向前检测和边一致性检测的区别如下

- Assign $\{Q=\text{green}\}$
- Effects on other variables connected by constraints with Q
 - NT can no longer be green = $\{B\}$
 - NSW can no longer be green = $\{R, B\}$
 - SA can no longer be green = $\{B\}$
- DWO there is no value for SA that will be consistent with $NT \neq SA$ and $NT = B$

Note Forward Checking would not have detected this DWO.



4 知识表示与推理

4.1 基本概念

一阶逻辑(First-Order Logic,FOL)

- 个体/常量(0-arity)
- 一元(unary)谓词: $A(x), B(x)$
- 关系/二元谓词: $L(x, y)$

定义 7 (项(term)). 每一个变量都是一个项。若 t_1, \dots, t_n 都为项, 且 f 为 n 参数的函数符号, 则 $f(t_1, \dots, t_n)$ 是一个项。

定义 8 (公式(formular)). 公式包括以下几种情况:

- 若 t_1, \dots, t_n 都是项, 且 P 是 n 元的谓词符号, 则 $P(t_1, \dots, t_n)$ 是一个公式
- 若 t_1, t_2 都是项, 那么 $(t_1 = t_2)$ 是一个原子公式
- 若 α, β 都是公式, v 是一个变量, 则 $\neg\alpha, (\alpha \wedge \beta), (\alpha \rightarrow \beta), \exists v.\alpha, \forall v.\alpha$ 都是公式

注意公式中的变量是自由的 (*free*) 还是约束的 (*bound*), 通过全称特称符号可进行约束。没有自由变量的公式称为句子 (*sentence*)。

定义 9 (替换). $\alpha[v/t]$ 表示 α 中所有自由出现的 v 都用项 t 替代

定义 10 (解释(interpretation)). 一个解释是一个对 (pair) $\mathcal{I} = \langle D, I \rangle$, 其中

- D 是论域, 可以是任何非空集
- I 是从谓词到函数符号的映射
- 若 P 是一个 n 元谓词符号, 则 $I(P)$ 是一个在 D 上的 n 元关系, 即 $I(P) \subset D^n$ 。如 p 是一个常量/命题符号, 则 $I(p) \in \{True, False\}$ 。
- 若 f 是一个 n 元函数符号, 则 $I(f)$ 是一个在 D 上的 n 元函数, 即 $I(f) : D^n \rightarrow D$ 。如 c 是一个零元常量符号, 则 $I(c) \in D$ 。

定义 11 (赋值(denotation)). 变量指派 (*assignment*) μ 是一个从变量集合到论域 D 的映射

$$\begin{aligned} \|v\|_{\mathcal{I}, \mu} &= \mu(v) \\ \|f(t_1, \dots, t_n)\|_{\mathcal{I}, \mu} &= I(f)(\|t_1\|_{\mathcal{I}, \mu}, \dots, \|t_n\|_{\mathcal{I}, \mu}) \end{aligned}$$

定义 12 (满足). $\mathcal{I}, \mu \models \alpha$ 读作 \mathcal{I}, μ 满足 α

- $\mathcal{I}, \mu \models P(t_1, \dots, t_n) \iff \langle \|t_1\|_{\mathcal{I}, \mu}, \dots, \|t_n\|_{\mathcal{I}, \mu} \rangle \in I(P)$
- $\mathcal{I}, \mu \models (t_1 = t_2) \iff \|t_1\|_{\mathcal{I}, \mu} = \|t_2\|_{\mathcal{I}, \mu}$

同样也有命题逻辑的一些性质

- $\mathcal{I}, \mu \models \neg\alpha \iff \mathcal{I}, \mu \not\models \alpha$
- $\mathcal{I}, \mu \models (\alpha \wedge \beta) \iff (\mathcal{I}, \mu \models \alpha \wedge \mathcal{I}, \mu \models \beta)$
- $\mathcal{I}, \mu \models (\alpha \vee \beta) \iff (\mathcal{I}, \mu \models \alpha \vee \mathcal{I}, \mu \models \beta)$
- $\mathcal{I}, \mu \models \exists v.\alpha \iff (\exists d \in D : \mathcal{I}, \mu\{d; v\} \models \alpha)$
- $\mathcal{I}, \mu \models \forall v.\alpha \iff (\forall d \in D : \mathcal{I}, \mu\{d; v\} \models \alpha)$

令 S 为句子的集合, 若对于所有 $\alpha \in S, \mathcal{I} \models \alpha$, 则称 \mathcal{I} 满足 S , 记作 $\mathcal{I} \models S$, \mathcal{I} 又被称为 S 的一个模型。若 S 是可满足的, 则存在 \mathcal{I} 使得 $\mathcal{I} \models S$ 。

例 3. 通常的问题是下列句子集合是否是可满足的

$$\{\forall x(P(x) \rightarrow Q(x)), P(a), \neg Q(a)\}$$

即是否存在这样的解释/变量赋值使全部句子都成立。

定义 13 (蕴含(entail)). $S \models \alpha$ 是指对于任意满足 $\mathcal{I} \models S$ 的 \mathcal{I} 有 $\mathcal{I} \models \alpha$, 读作 S 蕴含 α 或 α 是 S 一个逻辑结果 (consequence)。注意 $\{\alpha_1, \dots, \alpha_n\} \models \alpha$ 当且仅当 $\alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \alpha$ 合法, 当且仅当 $\alpha_1 \wedge \dots \wedge \neg \alpha$ 不可满足。

例 4. 令知识库 KB 为句子集合, α 为一个询问, 想要知道知识库是否能推出询问, 即求蕴含 $KB \models \alpha$ 。若要证明 $KB \not\models \alpha$, 则只需给出一个解释满足 KB 但不满足 α 。

4.2 推理

4.2.1 归结

定义 14 (子句 (clause)). 文字 (literal) 是原子公式或它的取反, 一个子句是文字的析取 (disjunction), 如 $p \vee \neg r \vee s$, 写作 $(p, \neg r, s)$ 。特殊地, 空子句 $()$ 代表为假。公式 (formula) 则是子句的合取 (conjunction)。

定义 15 (归结 (resolution)). 从两条子句 $\{p\} \cup c_1$ 和 $\{\neg p\} \cup c_2$ 中推出子句 $c_1 \cup c_2$ 的过程称为归结。

定义 16 (获取 (derivation)). 句子序列 $c_1, \dots, c_n = c$, 对于每一个 c_i , 或者 $c_i \in S$, 或者 c_i 是先前两个子句的归结, 这个过程称为获取, 记为 $S \vdash c$ 。

定理 4. 若 $S \vdash c$, 则 $S \models c$

但注意其逆命题并不成立, 考虑 $(p) \models (p, q)$, 但是 $(p) \not\models (p, q)$ 。

定理 5. $S \vdash () \iff S \models () \iff S$ 是不可满足的

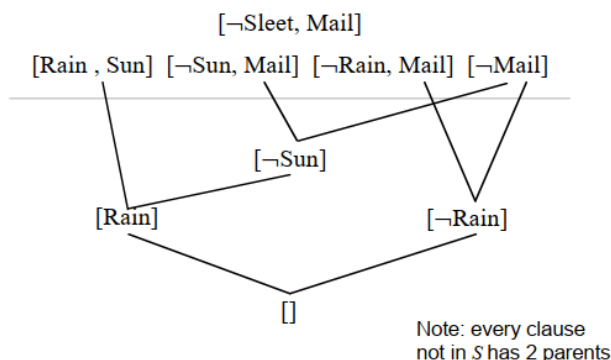
进而得到基于归结的推理过程: 反驳 (refutation), 即 $KB \models \alpha \iff KB \wedge \alpha$ 是不可满足的

- 将 KB 和 $\neg \alpha$ 改写为子句形式构成 S
- 检测是否 $S \vdash ()$

KB

| |
|--|
| $(\text{Rain} \vee \text{Sun})$ $(\text{Sun} \supset \text{Mail})$ $((\text{Rain} \vee \text{Sleet}) \supset \text{Mail})$ |
|--|

Show $\text{KB} \models \text{Mail}$



Similarly $\text{KB} \not\models \text{Rain}$

Can enumerate all resolvents given $\neg \text{Rain}$, and $[]$ will not be generated

利用归结（两条文字合一变真删除）看是否能得到空子句。

4.2.2 子句形式转换

- 消除蕴含: $A \rightarrow B \iff \neg A \vee B$
- 将非向内推: 德摩根定律
- 标准化变量: 重命名变量使得每一个量词都是唯一的
- 消除存在量词(skolemize): 引入新的函数符号, 如 $\forall x P(x)$ 改为 $P(g(y))$
- 将所有量词带到最前面: 只有全局量词, 且名字均不同
- 析取分配到合取
- 压平嵌套合取和析取
- 转化为子句: 将量词全部移除, 然后分离成几条子句

4.2.3 合一

归结过程中可能需要合一(unification), 用来消除两个子句中不同的变量。

定义 17 (合一子(unifier)). 令公式 f 和 g 语义等价的替换 σ 称为 f 和 g 的合一子

例 5. $P(f(x), a)$ 和 $P(y, f(w))$ 不能被合一, 因没有令 $a = f(w)$ 的替换使得两者相同

定义 18 (最一般合一子(Most General Unifier, MGU)). 两个公式 f 和 g 的替换 σ 是 MGU , 若

- σ 是一个合一子
- 对于其他合一子 θ 必然存在第三个替换 λ 使得 $\theta = \sigma\lambda$

也就是说其他的合一子都比 σ 更特殊(*specialized*)。

例 6. 考虑 $P(f(x), z)$ 和 $P(y, a)$

- $\sigma = \{y = f(a), x = a, z = a\}$ 是一个合一子, 但不是 MGU
- $\theta = \{y = f(x), z = a\}$ 是一个 MGU
- $\sigma = \theta\lambda$, 其中 $\lambda = \{x = a\}$

计算MGU的算法: 不断代入新的元, 使其一致

Given two atomic formulas f and g

- ① $k = 0; \sigma_0 = \{\}; S_0 = \{f, g\}$
- ② If S_k contains an identical pair of formulas, stop and return σ_k as the MGU of f and g .
- ③ Else find the disagreement set $D_k = \{e_1, e_2\}$ of S_k
- ④ If $e_1 = V$ a variable, and $e_2 = t$ a term not containing V (or vice-versa) then let $\sigma_{k+1} = \sigma_k\{V = t\}; S_{k+1} = S_k\{V = t\}; k = k + 1; \text{Goto } 2$
- ⑤ Else stop, f and g cannot be unified.

4.2.4 完整例子

基于反驳的归结推理

| | |
|---|---|
| | 1. $A(tony)$ |
| | 2. $A(mike)$ |
| | 3. $A(john)$ |
| | 4. $L(tony, rain)$ |
| | 5. $L(tony, snow)$ |
| $\forall x(A(x) \wedge \neg S(x)) \rightarrow C(x)$ | \Rightarrow 6. $(\neg A(x), S(x), C(x))$ |
| $\forall x(C(x) \rightarrow \neg L(x, rain))$ | \Rightarrow 7. $(\neg C(y), \neg L(y, rain))$ |
| $\forall x(\neg L(x, snow) \rightarrow \neg S(x))$ | \Rightarrow 8. $(L(z, snow), \neg S(z))$ |
| $\forall x(L(tony, x) \rightarrow \neg L(mike, x))$ | \Rightarrow 9. $(\neg L(tony, u), \neg L(mike, u))$ |
| $\forall x(\neg L(tony, x) \rightarrow L(mike, x))$ | \Rightarrow 10. $(L(tony, v), L(mike, v))$ |
| $\neg \exists x(A(x) \wedge C(x) \wedge \neg S(x))$ | \Rightarrow 11. $(\neg A(w), \neg C(w), S(w))$ |

Note that we must standardize variables.

12. $R[5, 9a]u = snow \neg L(mike, snow)$
13. $R[8, 12]z = mike \neg S(mike)$
14. $R[6b, 13]x = mike (\neg A(mike), C(mike))$
15. $R[2, 14a] C(mike)$
16. $R[8a, 12]z = mike \neg S(mike)$
17. $R[2, 11]w = mike (\neg C(mike), S(mike))$
18. $R[15, 17] S(mike)$
19. $R[16, 18] ()$

答案抽取(answer extraction)

- 将询问 $\exists x P(x)$ 用 $\exists x [P(x) \wedge \neg \text{answer}(x)]$ 替换 (因为取非后变成 $\forall x P(x) \implies \text{answer}(x)$)
- 直到获得任意子句只包含答案的谓词

例 7. 对下列查询进行归结及答案查询

- *Whoever can read $R(x)$ is literate $L(x)$*
- *Dolphins $D(x)$ are not literate*
- *Flipper is an intelligent dolphin $I(x)$*

Who is intelligent but cannot read?

分析. 对语句进行形式化

| | | |
|---|---|--------------------------------|
| $\forall x(R(x) \rightarrow L(x))$ | 1 | $(\neg R(u), L(u))$ |
| $\forall x(D(x) \rightarrow \neg L(x))$ | 2 | $(\neg D(v), \neg L(v))$ |
| $D(Flip) \wedge I(Flip)$ | 3 | $D(Flip)$ |
| | 4 | $I(Flip)$ |
| $Q:\exists x(I(x) \wedge \neg R(x))$ | 5 | $(\neg I(y), R(y), answer(y))$ |
| $R[4, 5]/y = Flip$ | 6 | $(R(Flip), answer(Flip))$ |
| $R[1, 6]/u = Flip$ | 7 | $(L(Flip), answer(Flip))$ |
| $R[2, 7]/v = Flip$ | 8 | $(\neg D(Flip), answer(Flip))$ |
| $R[3, 8]$ | 9 | $(answer(Flip))$ |

因此得到 *Flipper* 是聪明的但是不能阅读

5 确定性规划

智能体应该能够对世界做出动作(action)，而不仅仅是通过搜索解决问题或推理及知识表示。核心是对动作的效果进行推理，并且计算什么动作能够达成特定的效果。

本节中主要关注确定性规划，即有完全的初始状态描述及确定性的动作效果。

5.1 情景演算

情景演算(Situation Calculation, SitCalc)三个基本部分

- 动作(action): 一组谓词
- 情景(situations): 动作序列, $do(a, s)$ 为动作、情景到新情景的函数映射, S_0 为初始情景

$$do(put(a, b), do(put(b, c), S_0))$$

要区别情景与状态(state)，如将硬币转两次，情景/动作历史不同，但状态都是一样的

- 流(fluent): 从情景到情景的谓词或函数（动态变化过程），用谓词或函数符号描述，其中最后一个参数为情景，如 $Holding(r, x, s)$ 代表机器人 r 在状态 s 下拿着物体 x ，有

$$\neg Holding(r, x, s) \wedge Holding(r, x, do(pickup(r, x), s))$$

经过后面的流操作会得到一个新情景下的谓词

- 条件(precondition): 动作执行的前提条件
- 影响(effect): 执行动作后改变的流。如下在情景 s 下执行修复动作后， x 就不是破碎的

$$\neg Broken(x, do(repair(r, x), s))$$

情景演算的形式化仅陈述了执行动作的影响，而没有阐述没影响的部分。但是给定一个流只有很少的动作会被影响，而大多数都保持不变。

形式化后的情景即可以利用归结进行规划，但是开销会非常大。而框架(frame)问题则是找到一种高效的方法来确定动作的非效果(non-effects)。而不是显式地将它们全部写下来，可以考虑用一阶逻辑。

5.2 STRIPS

传统的规划没有不完全或不确定的信息，因此可以做以下假设。

- 封闭世界假设(Closed-World Assumption, CWA)：初始状态的信息是完备的，用于表示世界状态的知识库是一系列真实的原子事实（与数据库类似）。如 $emp(A, C)$ 不在数据库中，则 $\neg emp(A, C)$ 为真。
- 动作的前提只能是原子命题的合取
- 动作的影响只会使原子命题变真或变假，不会出现条件影响或析取影响

STRIPS(Stanford Research Institute Problem Solver)：

- 世界被表示成封闭世界知识库(CW-KB)，一个STRIPS的动作表示成更新CW-KB的方式
- 一个动作生成新的KB，用以描述新的世界

在SitCalc中我们可能有不完全的信息（用一阶逻辑公式表示），而在STRIPS中，我们有完整的信息（用CW-KB表示）。

STRIPS中需要用3个列表来表示一个动作

- 动作的前提 pre
- 动作增加的影响 add
- 动作减少的影响 $delete$

例 8. $pickup(X)$ ：

- $Pre : handempty, clear(X), ontable(X)$
- $Adds : holding(X)$
- $Dels : handempty, clear(X), ontable(X)$

注意STRIPS的表达力还是有限制的，如它没有条件(conditional)影响。因此需要有表达力更强的语言Action Description Language(ADL)，允许在前提中有任意公式，而且可以有条件和全称影响。

可以将规划看成是一个搜索问题，每个动作都是由一个CW-KB到CW-KB的映射，但搜索空间会非常巨大。

5.3 松弛问题

传统的规划给出

- 一个封闭世界知识库表示原始状态
- 一组STRIPS动作（从状态到新状态的映射）
- 一个目标状态

放松(relaxed)问题即考虑忽略删除动作的列表, 这样可能得到有用的启发式函数估计。

定理 6. 放松问题的一个最优规划的长度是原始问题最优规划长度的下界

证明. 由于我们将所有影响都添加了, 并且不删除负面影响, 因此很直觉地会很快得到目标。令 P 为原始问题, P' 为放松问题, 我们希望得到 $Sols(P) \subset Sols(P')$, 则有 $Minlen(Sols(P')) \leq Minlen(Sols(P))$ 。

- 令 a_0, \dots, a_{n-1} 为 P 的解, 证明它也是 P' 的一个解。令 s_0 为初始状态, 令 $s_{i+1} = s_i \cup add(a_i) - del(a_i), \forall i < n$, 且 $Goal \subset s_n, pre(a_i) \subset s_i, \forall i < n$ 。
- 又令 $s'_0 = s_0$, 且 $s'_{i+1} = s'_i \cup add(a_i)$, 通过数学归纳法证明 $s_i \subset s'_i, \forall i \leq n$ 。
 - 归纳奠基: $i = 0$ 时 $s_0 \subset s'_0 = s_0$
 - 归纳假设: 假设 $s_i \subset s'_i, \forall i < n$, 证明 $s_{i+1} \subset s'_{i+1}$
 - 推理: 因为 a_i 在 s_i 中是可采纳的, 即 $pre(a_i) \subset s_i$, 因此 $pre(a_i) \subset s_i \subset s'_i$, 即 a_i 在 s'_i 中是可采纳的。所以得到

$$s_{i+1} = s_i \cup add(a_i) - del(a_i) \subset s'_i \cup add(a_i) = s'_{i+1}$$

进而 $Goal \subset s_n \subset s'_n$, 且 $pre(a_i) \subset s_i \subset s'_i, \forall i < n$, 因此 a_0, \dots, a_{n-1} 也是 P' 的一个解。□

因此最优放松问题的规划可以作为 A^* 算法的可采纳启发式函数。然而在放松问题中计算一个最优的问题是NP难的, 可以从 S 开始建立层次(layered)结构使其到达目标。

可达性分析: $s_0, a_1, s_1, \dots, a_n, s_n$, 直到 $s_n \subset Goal$, 或者状态层 s_n 不再改变(到达不动点)。

命题 5. 令 a_0, \dots, a_{n-1} 为 S_0 中可采纳的动作序列, 令 $s_0 = S_0$, 且 $s_{i+1} = s_i \cup add(a_i) - del(a_i)$ 。那么 $\forall i < n, \exists j, k \leq i$ 使得 $a_i \in A_k$ 且 $s_i \subset S_j$

分析. 用数学归纳法

- 归纳奠基: $i = 0$ 显然成立
- 归纳假设: 假设 $\forall i < n \exists j \leq i: s_i \subset S_j$
- 因为 $pre(a_i) \subset s_i, pre(a_i) \subset S_j$, 令 k 为最小的 $u \leq j$ 使得 $pre(a_i) \subset S_u$, 那么 $a_i \in A_k$, 故 $add(a_i) \subset S_{k+1} \subset S_{j+1}$ 因此

$$s_{i+1} \subset s_i \cup add(a_i) \subset S_j \cup S_{j+1} = S_{j+1}$$

定理 7. 假设状态层不再改变且目标不被满足, 则原始规划问题是不可解的。

证明. 用反证法, 假设 a_0, \dots, a_{n-1} 为原始问题的一个解, 则 $Goal \subset s_n$ 。由前面的命题, 存在 $m \leq n$ 使得 $Goal \subset s_n \subset S_m$, 导致矛盾。□

假设目标 G 包含在状态层中, 现在想要计算一个好的放松规划, 想法即对每个 i 选择最小的 A_i 子集。

CountActions(G, S_K):

/* Here G is contained in S_K , and we compute the number of actions contained in a relaxed plan achieving G . */

- If $K = 0$ return 0
- Split G into $G_P = G \cap S_{K-1}$ and $G_N = G - G_P$
 - G_P contains the previously achieved (in S_{K-1}), and
 - G_N contains the just achieved parts of G (only in S_K).
- Find a **minimal** set of actions A whose add effects cover G_N .
 - cannot contain redundant actions,
 - **but may not be the minimum sized set**
(computing the minimum sized set of actions is the set cover problem and is NP-Hard)
- $\text{NewG} := G_P \cup \text{preconditions of } A$.
- return $\text{CountAction}(\text{NewG}, S_{K-1}) + \text{size}(A)$

进而变成一个集合覆盖问题：从集合的集合 S 中选择 m 个集合使得其并集可以覆盖全集。

定理 8. 假设 $\text{Goal} \subset S_k$, 对于 $i < k$, 令 A'_{i-1} 为调用 $\text{CountActions}(G_i, S_i)$ 的结果, 则 A'_0, \dots, A'_{k-1} 为松弛问题的一个解。

证明. 用数学归纳法

□

6 不确定性规划

6.1 基础知识

一组变量 V_1, \dots, V_n 以及其对应的有限域 $\text{dom}[V_i]$, 很容易导致指数的计算复杂度。

定理 9 (全概率公式). $\{B\}_{i=1}^k$ 为全集 U 的一个划分, 则

$$\begin{aligned}\mathbb{P}(A) &= \mathbb{P}(A \cap B_1) + \dots + \mathbb{P}(A \cap B_k) \\ &= \mathbb{P}(A \mid B_1) \mathbb{P}(B_1) + \dots + \mathbb{P}(A \mid B_k) \mathbb{P}(B_k)\end{aligned}$$

定理 10 (条件独立). 若

$$\mathbb{P}(B \mid A \cap C) = \mathbb{P}(B \mid A)$$

, 则在给定 A 的情况下 B 条件独立于 C (C 没有给 A 增加知识)。若对于所有 $x \in \text{dom}[X], y \in \text{dom}[Y], z \in \text{dom}[Z]$,

$$\mathbb{P}(X = x \wedge Y = y \mid Z = z) = \mathbb{P}(X = x \mid Z = z) \mathbb{P}(Y = y \mid Z = z)$$

则在给定 $Z = z$ 下, $X = x$ 和 $Y = y$ 条件独立。

命题 6 (独立性性质). • 若 A 和 B 独立, 则 $\mathbb{P}(A \cap B) = \mathbb{P}(A) \cdot \mathbb{P}(B)$

- 给定 A , B 和 C 条件独立, 则

$$\mathbb{P}(B \cap C | A) = \mathbb{P}(B | A) \mathbb{P}(C | A)$$

定理 11 (贝叶斯公式). 条件概率定义

$$\mathbb{P}(X | Y) = \mathbb{P}(XY) / \mathbb{P}(Y) \implies \mathbb{P}(XY) = \mathbb{P}(X | Y) \mathbb{P}(Y)$$

注意与贝叶斯公式区分

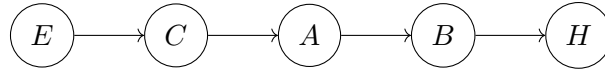
$$\mathbb{P}(Y | X) = \frac{\mathbb{P}(XY)}{\mathbb{P}(X)} = \frac{\mathbb{P}(X | Y) \mathbb{P}(Y)}{\mathbb{P}(X)}$$

定理 12 (链式法则).

$$\mathbb{P}(A_1 \cap A_2 \cap \dots \cap A_n) = \mathbb{P}(A_1 | A_2 \cap \dots \cap A_n) \mathbb{P}(A_2 | A_3 \cap \dots \cap A_n) \mathbb{P}(A_{n-1} | A_n) \mathbb{P}(A_n)$$

6.2 贝叶斯网络

6.2.1 基础知识



有以下概率公式成立

- $\mathbb{P}(H | B, A, E, C) = \mathbb{P}(H | B)$
- 由链式法则和独立性假设

$$\mathbb{P}(HBACE) = \mathbb{P}(H | BACE) \mathbb{P}(B | ACE) \mathbb{P}(A | CE) \mathbb{P}(C | E) \mathbb{P}(E)$$

$$\mathbb{P}(HBACE) = \mathbb{P}(H | B) \mathbb{P}(B | A) \mathbb{P}(A | C) \mathbb{P}(C | E) \mathbb{P}(E)$$

- 通用公式

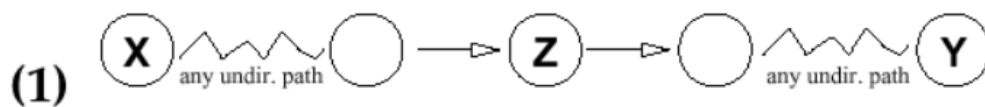
$$\mathbb{P}(X_1 X_2 \dots X_n) = \mathbb{P}(X_n | \text{Par}(X_n)) \dots \mathbb{P}(X_1 | \text{Par}(X_1))$$

- 单点概率 (由全概率公式和条件概率定义)

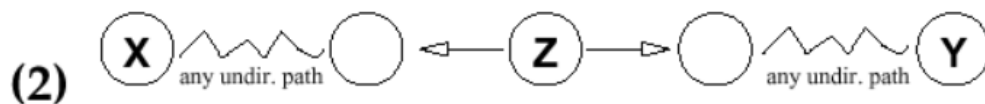
$$\begin{aligned} \mathbb{P}(a) &= \sum_{c_i \in \text{dom}[C]} \mathbb{P}(a | c_i) \mathbb{P}(c_i) \\ &= \sum_{c_i \in \text{dom}[C]} \mathbb{P}(a | c_i) \sum_{e_i \in \text{dom}[E]} \mathbb{P}(c_i | e_i) \mathbb{P}(e_i) \end{aligned}$$

因此每个结点只需存一个条件概率表(conditional probability table, CPT)。

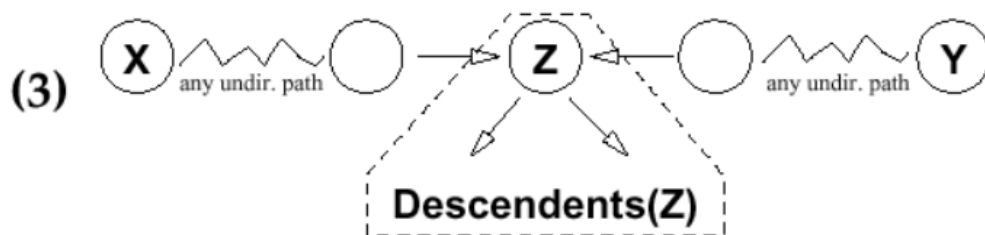
定义 19 (D分隔(separation)). 若一组变量 E 阻隔(block)了 X 到 Y 的所有无向路径 P , 则称 E D-分隔了 X 和 Y , 且有给定证据 E 下, X 和 Y 条件独立。 E 阻隔(block)了路径 P 当且仅当在路径上存在某点 Z 使得下面任一成立。



If Z in evidence, the path between X and Y blocked



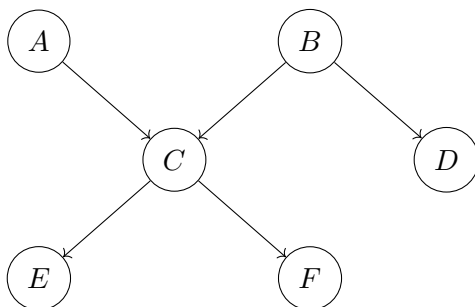
If Z in evidence, the path between X and Y blocked



If Z is **not** in evidence and **no** descendent of Z is in evidence, then the path between X and Y is blocked

注意第3种情况，证据集中没有给交叉结点及其子结点，则父亲节点都独立。

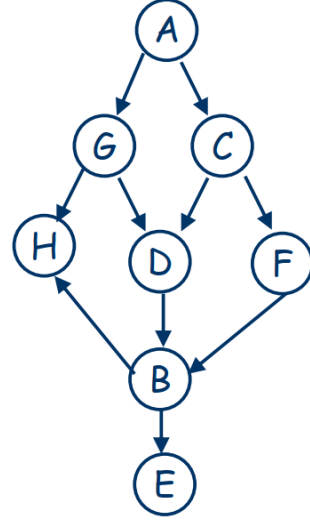
例 9. 考虑如下贝叶斯网络



在此例中 $\mathbb{P}(c \mid a, b, \neg d, \neg e, \neg f) \neq \mathbb{P}(c \mid a, b)$ 。

determine if **A and E are independent** given the evidence:

1. A and E given no evidence? **No**
2. A and E given {C}? **No**
3. A and E given {G,C}? Y
4. A and E given {G,C,H}? Y
5. A and E given {G,F}? **No**
6. A and E given {F,D}? Y
7. A and E given {F,D,H}? **No**
8. A and E given {B}? Y
9. A and E given {H,B}? Y
10. A and E given {G,C,D,H,D,F,B}? Y



Why the answer to 7 is No?

例 10. 注意第3种情况的适用条件, 由于 H 在证据集中, 所以并不满足第3种情况。在第7个例子中, $AGHBE$ 没有被阻隔, 故给定 FDH , A, E 也不独立。

6.2.2 贝叶斯推断

推断过程如下:

$$\begin{aligned}
 \mathbb{P}(a \mid d, e) &= \mathbb{P}(a, d, e) / \mathbb{P}(d, e) \\
 &= \mathbb{P}(a, d, e) / \sum_A \mathbb{P}(a, d, e) \\
 \mathbb{P}(a, d, e) &= \sum_{B, C} \mathbb{P}(a, B, C, d, e)
 \end{aligned}$$

故只需计算 $\mathbb{P}(a, d, e)$ 。

采用动态规划的思想, 存储子项, 减少计算量。

记因子为某些变量的函数, 如 $\mathbb{P}(C \mid A) = f(A, C)$

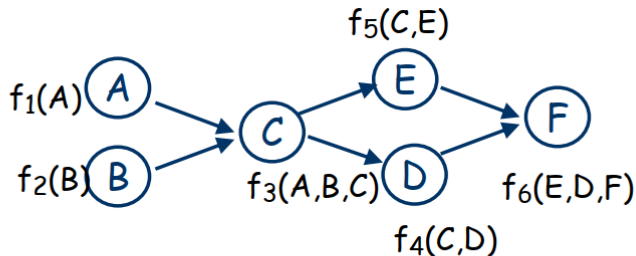
- 乘积: $h(X, Y, Z) = f(X, Y) \times g(Y, Z)$
- 求和: $h(Y) = \sum_{x \in \text{dom}[X]} f(x, Y)$
- 因子限定: $h(Y) = f(a, Y)$

算法 1 (变量消除(Variable Elimination, VE)). 给定贝叶斯网络, 条件概率表 F , 询问 Q , 证据 E , 其余变量为 Z , 计算 $\mathbb{P}(Q \mid E)$ 。

1. 对于 $f \in F$ 中每一变量, 将其替换为 $f_{E=e}$ (因子限定)
2. 对于每一 $Z_j \in Z$, 按给定 Z_j 顺序, 并按照以下步骤消除:
 - f_1, f_2, \dots, f_k 为含有 Z_j 的因子
 - 计算新的因子 $g_j = \sum_{Z_j} f_1 \times f_2 \times \dots \times f_k$
 - 将 f_i 从 F 中移除, 并将新的因子 g_j 添加到 F 中
3. 剩下的因子只包含询问 Q 中的变量, 则计算它们的乘积归一化得到 $\mathbb{P}(Q \mid E)$

可以采用桶消除(bucket elimination)算法，每次将新生成的因子放在第一个可被应用的桶中。下图展示的是超图(hypergraph)，最大超边(hyperedge)的大小即为最大的CPT表项/VE算法的复杂度。

VE Ordering:
C, F, A, B, E, D



1. ~~C: $f_3(A, B, C)$, $f_4(C, D)$, $f_5(C, E)$~~

2. F: $f_6(E, D, F)$

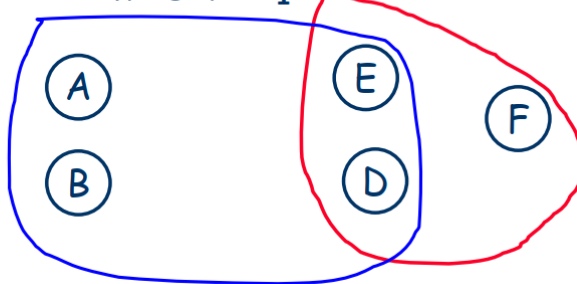
3. A: $f_1(A)$, $f_7(A, B, D, E)$

4. B: $f_2(B)$

5. E:

6. D:

Hypergraph H_1



多树(polytree): 单连通(singly connected)的贝叶斯网络，即在任意两个结点间只有一条路径。

最小填充(min-fill)启发式: 总是先消除产生最小因子大小的变量，这种方法可以使得在线性时间内求解多树。

定义 20 (相关性(relevance)). 给定证据 E 询问 Q ，则有以下几种情况:

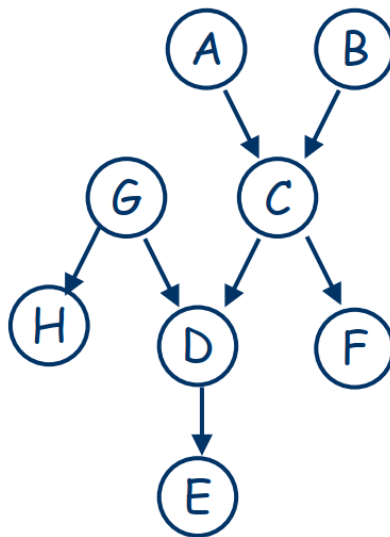
- Q 自身当然是相关的
- 若结点 Z 相关，则它的父母也相关
- 若 $e \in E$ 是一个相关结点的后代，则 E 也是相关的

Query: $P(F)$

- relevant: F, C, B, A

Query: $P(F|E)$

- relevant: F, C, B, A
- **also: E, hence D, G**
- intuitively, we need to compute $P(C|E)$ to compute $P(F|E)$



例 11. 但询问 $P(F|H)$ ，则 D, E, G 是不相关的。而且这种算法会过度估计相关的变量。

7 机器学习

7.1 决策树

假定样本集合 D 中第 k 类样本所占比例为 $p_k(k = 1, 2, \dots, |\mathcal{Y}|)$ ，则 D 的信息熵定义为

$$\text{Ent}(D) = - \sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k$$

又假定离散属性 a 有 V 个可能取值 $\{a^1, a^2, \dots, a^V\}$ ，第 v 个分支结点包含了 D 中所在属性 a 上取值为 a^v 的样本，记为 D^v ，进而可定义信息增益

$$\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Ent}(D^v)$$

每次选择最大增益的属性进行划分，即

$$a_* = \arg \max_{a \in A} \text{Gain}(D, a)$$

以信息增益为准则来选择划分属性的算法即为ID3决策树。

7.2 贝叶斯学习

先验 $\mathbb{P}(H)$ 、似然 $\mathbb{P}(d | H)$ 、证据 $d = \langle d_1, d_2, \dots, d_n \rangle$ ，有贝叶斯公式（先验推后验）

$$\mathbb{P}(H | d) = \alpha \mathbb{P}(d | H) \mathbb{P}(H)$$

假设独立同分布 $\mathbb{P}(d | h) = \prod_j \mathbb{P}(d_j | h)$

- 贝叶斯学习： $\mathbb{P}(X | d) = \sum_i \mathbb{P}(X | d, h_i) \mathbb{P}(h_i | d) = \sum_i \mathbb{P}(X | h_i) \mathbb{P}(h_i | d)$
- 极大后验(MAP)学习： $\mathbb{P}(X | d) \approx \mathbb{P}(X | h_{MAP})$

$$h_{MAP} = \arg \max_{h_i} \mathbb{P}(h_i | d) = \arg \max_{h_i} \mathbb{P}(h_i) \mathbb{P}(d | h_i)$$

- 最大似然(ML)学习： $\mathbb{P}(X | d) \approx \mathbb{P}(X | h_{ML})$

$$h_{ML} = \arg \max_{h_i} \mathbb{P}(d | h_i)$$

基于属性条件独立性假设

$$\mathbb{P}(c | \mathbf{x}) = \frac{\mathbb{P}(c) \mathbb{P}(\mathbf{x} | c)}{\mathbb{P}(x)} = \frac{\mathbb{P}(c)}{\mathbb{P}(\mathbf{x})} \prod_{i=1}^d \mathbb{P}(x_i | c)$$

其中 d 为属性数目, x_i 为 \mathbf{x} 在第 i 个属性上的取值。因对所有类别来说 $\mathbb{P}(\mathbf{x})$ 相同, 因此贝叶斯判定准则为

$$h_{NB}(\mathbf{x}) = \arg \max_{c \in \mathcal{Y}} \mathbb{P}(c) \prod_{i=1}^d \mathbb{P}(x_i | c)$$

令 D_c 表示训练集 D 中第 c 类样本组成的集合, 有

$$\mathbb{P}(c) = \frac{|D_c|}{|D|} \quad \mathbb{P}(x_i | c) = \frac{|D_{c,x_i}|}{|D_c|}$$

7.3 聚类算法

- 硬聚类: 每个样本都决定放在哪一个类别中
- 软聚类: 每个样本都被指派每个类别的概率分布

K-means算法

- E步: 对于每一个类别 i 和特征 X_j , 有

$$pval(i, X_j) \leftarrow \frac{\sum_{e: class(e)=i} val(e, X_j)}{|\{e : class(e) = i\}|}$$

- M步: 对于每一个样本 e , 指派 e 给类别 i 使得

$$\min_i \sum_{j=1}^n (pval(i, X_j) - val(e, X_j))^2$$

7.4 神经网络

定理 13 (一致近似理论(Universal Approximator Theorem)). 具有至少一个隐层的深度神经网络可以无限逼近任意连续函数

前向后向传播过程:

- 前向过程:

$$in_j = \sum_i w_{ij} a_i \quad a_j = g(in_j)$$

- 后向过程:

$$\text{output: } \Delta_j = g'(in_j)(y_j - a_j)$$

$$\text{hidden: } \Delta_i = g'(in_i) \sum_j w_{ij} \Delta_j$$

注意以下两条求导公式

$$\begin{aligned} \sigma(x) &= \frac{1}{1 + e^{-x}} & \frac{\partial \sigma(x)}{\partial x} &= (1 - \sigma(x))\sigma(x) \\ \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} & \frac{\partial \tanh(x)}{\partial x} &= 1 - \tanh^2(x) \end{aligned}$$

7.5 强化学习

目标:

$$\max_{\pi_{\theta}} \left[\sum_{t=1}^{\infty} \gamma^t r_t \right]$$

给定策略 π :

$$Q^{\pi}(s, a) = \sum_{s'} \mathbb{P}(s' | a, s) (R(s, a, s') + \gamma V^{\pi}(s'))$$

$$V^{\pi}(s) = Q(s, \pi(s))$$

Q学习: 随机选择动作 a , 观察回报 r 和下一状态 s'

$$Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$$