人工智能笔记

陈鸿峥

2020.01*

目录

*Build 20200106

本课程主要选用Stuart Russell和Peter Norvig的《人工智能—一种现代化方法》(Artificial Intelligence: A Modern Approach)一书，理论及实验作业请见https://github.com/chhzh123/Assignments/tree/master/ArtificalIntelligence，内容非常详实，基本涵盖了人工智能的几大板块。

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 expections

  - 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}) ∪ 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

[1]而不是探索的结点数目

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。

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！或者把根节点看作第0层也可以）

- 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：$1+b+b^2+\cdots+b^d+(b^d-1)b=O(b^{d+1})$，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$m >> d$，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

[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。

- 时间复杂性：$O(b^{\lfloor C^\star/\varepsilon \rfloor+1})$，对应着BFS中$d = C^\star/\varepsilon$，其中$C^\star$为最优解的开销，最坏情况就是每一层开销都很小为$\varepsilon$，那么需要$\lfloor C^\star/\varepsilon \rfloor$层
- 空间复杂性：$O(b^{\lfloor C^\star/\varepsilon \rfloor+1})$

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

注意算法中几个关键点：从队列Pop出一个结点后

1. 先做目标检测，如果是目标则立即返回（也就是说要扩展完结点进入下一层才会触发解返回）
2. 将结点状态标记为**已探索(explored)**，然后扩展(expand)该结点
3. 做环检测，如果孩子不在已探索或边界集中，将孩子节点加入边界集
4. 如果孩子在边界集中且有更高的路径开销，则将边界集中的结点用孩子结点进行替换

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

[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。

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(explored)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（注意不是展开）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。

2.1.8 总结

	BFS	UCS	DFS	Depth-limited	IDS	Bidirectional
完备性	✓	✓	✗	✗	✓	✓
时间复杂度	$O(b^{d+1})$	$O(b^{\lfloor C^*/\varepsilon \rfloor+1})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
空间复杂度	$O(b^{d+1})$	$O(b^{\lfloor C^*/\varepsilon \rfloor+1})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
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：$(M, C, 1) \overset{(m,c)}{\Longrightarrow} (M - m, C - c, 0)$
  $(M, C, 0) \overset{(m,c)}{\Longrightarrow} (M + m, C + c, 1)$

[4]注意这在启发式搜索中不一定成立

2.2 有信息搜索

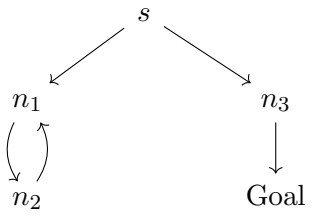
在无信息搜索中，我们从不估计边界集中最有期望(promising)获得最优解的结点，而是无区别地选择当前边界集中第一个结点。然而事实上，针对不同问题我们是有对结点的先验知识(apriori knowledge)的，即从当前结点到目标结点的开销有多大。而这就是有信息搜索(informed)，或者称为启发式搜索(heuristics)。

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。**当$f(n)$相同时，再用$h(n)$进行升序排序。**

$f(n)$需要满足下列两个性质。

定义 1 (可采纳的(admissibility))**.** *假设所有代价$c(n1 \to n2) \geq \varepsilon > 0$，令$h^\star(n)$为从节点$n$到目标节点$\infty$的最优解代价[5]，若*

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

*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）*

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

则称$h(n)$是单调的。

[5]如果没有路径则$h^\star(n) = \infty$

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$h(n) \leq h^\star(n) = \infty$恒成立
- 令$n = n_1 \to n_2 \to \cdots \to n_k$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$\forall i$: $h(n_i) \leq h^\star(n_i)$，如下从后往前推

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

定理 2. 可采纳性蕴含最优性

分析. 假设最优解有开销$C^\star$，则任何最优解一定会在开销大于$C^\star$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$c(p^\star)$且p在$p^\star$之前被扩展，则一定存在一个节点n在$p^\star$上且仍在边界集中，因此

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

矛盾。故在最优解展开之前的路径一定有开销$\leq C^\star$，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

分析.

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

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. $A^\star$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

$A^\star$算法有和BFS或UCS同样的空间复杂性问题，而迭代加深$A^\star$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. `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. `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）

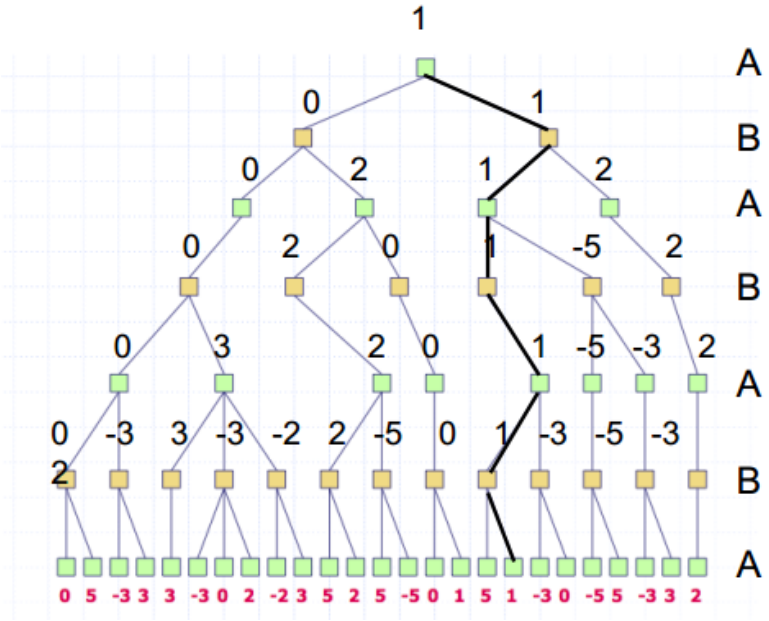
- 状态集合$\mathcal{S}$
- 初始状态$I \in \mathcal{S}$
- 终止位置$T \subset \mathcal{S}$
- 后继：下一可能状态的集合
- 效益(utility)/收益(payoff)函数$V : T \mapsto \mathbb{R}$，表明终止状态对A玩家有多好，对B玩家有多坏（都站在A角度给出）

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

$$U(n) = \begin{cases} \min\{U(c): \ c \text{ is a child of } n\} & n \text{ is a Min node} \\ \max\{U(c): \ c \text{ is a child of } n\} & n \text{ is a Max node} \end{cases}$$



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，即要看**所有祖先做决定**。

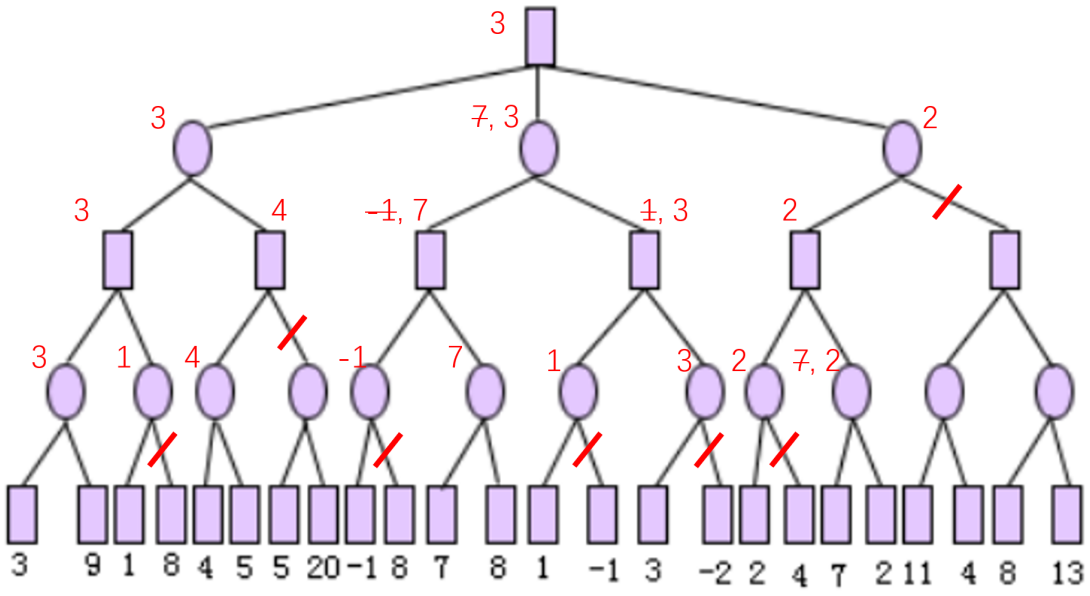
- 只要当前Max结点的值≥祖先某一Min结点的值，就可以在该Max结点上做α剪枝
- 只要当前Min结点的值≤祖先某一Max结点的值，就可以在该Min结点上做β剪枝

即**当前Min小等祖先Max，当前Max大等祖先Min可剪枝**。

Algorithm 2 Alpha-Beta Pruning

```
1:  procedure AlphaBeta(n,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, AlphaBeta(c,MIN,alpha,beta))
8:              if beta <= alpha then
9:                  break
10:         return alpha
11:     else                                             ▷ Player == MIN
12:         for c in ChildList do
13:             beta = min(beta, AlphaBeta(c,MAX,alpha,beta))
14:             if alpha >= beta then
15:                 break
16:         return beta
                                  ▷ Initial call: AlphaBeta(START-NODE,Player,−∞,+∞)
```

例 3. 如下采用$\alpha-\beta$剪枝进行搜索，方形为Max结点，圆形为Min结点。



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$V_1, \ldots, V_n$
- 每一个变量都有一个包含有限值的论域$\mathrm{dom}[V_i]$，如

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

- 一组限制条件$C_1, \ldots, C_m$
  - 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

$$C(V_1 = a, V_2 = b, V_4 = c) = True$$

  - 布尔函数可以以表形式给出，或以表达式形式给出，如$C(V_1, V_2, V_4) = (V_1 = V_2 + V_4)$
  - **注意限制是否写了变量不同allDifferent(Vars)**，这一条在很多问题中都会出现
- 一个状态可以通过给每一个变量赋值得到

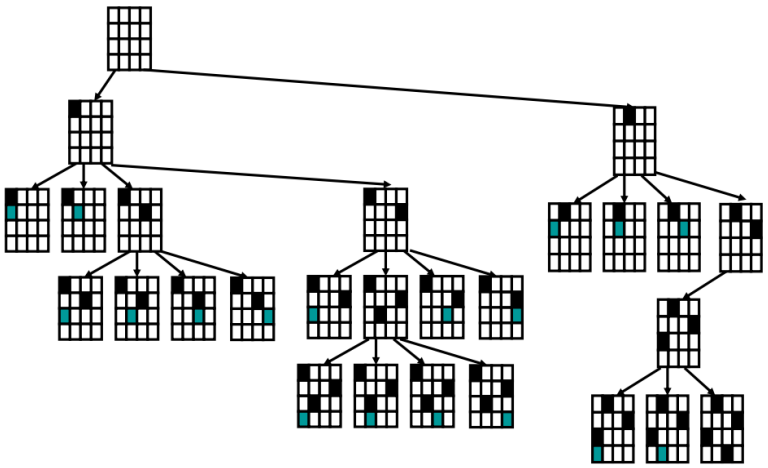
CSP不关心到目标状态的移动步骤，而只关心是否存在这样一组变量满足目标。

3.1 回溯搜索(Backtracking)

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

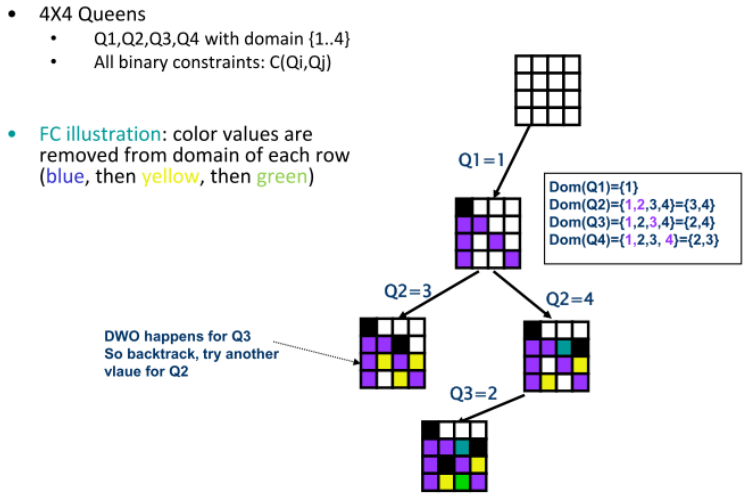
进而有下列两种传播算法：前向传播和GAC算法。

3.2 向前检测(Forward Checking)

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（注意这里移除的是原变量V中的值），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*/
                DWOoccurred:= 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;
```



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，可以进行试探然后将其他格中不能填的值都去除（约束传播）。

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$\forall X \exists Y : C(X,Y)$。

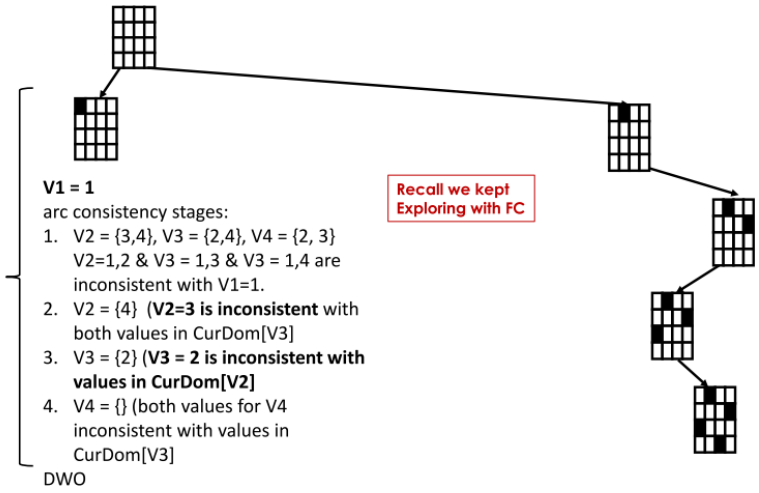
定义 5 (泛化边一致性(Generalized Arc Consistency, GAC)). 限制$C(V_1,V_2,\ldots,V_n)$是关于V_i边一致的，当且仅当$\forall V_i, \exists V_1,\ldots,V_{i-1},V_{i+1},\ldots,V_n$满足$C$。限制$C$是$GAC$的当且仅当对于每一变量都是$GAC$的。一个$CSP$是$GAC$的当且仅当它的限制都是$GAC$的。

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$V_i = d$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（即包含被移除值变量V的限制），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。

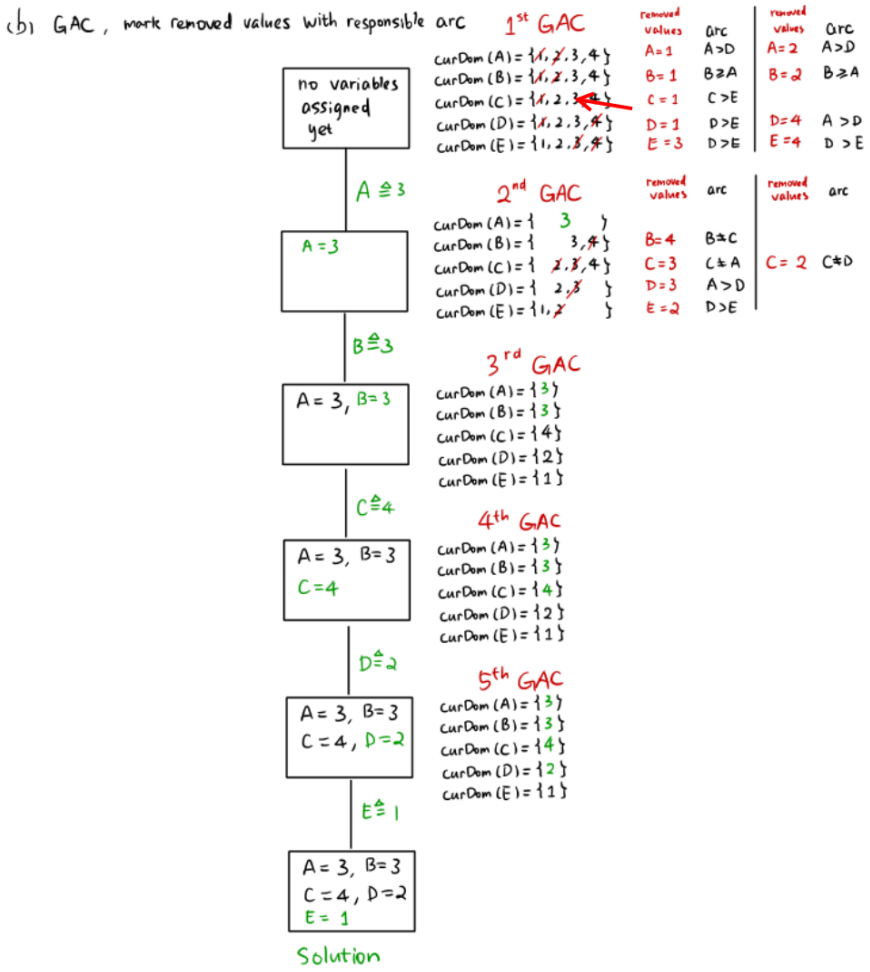
```
GAC(Level) /*Maintain GAC Algorithm */
   If all variables are assigned
      PRINT Value of each Variable
      RETURN or EXIT (RETURN for more solutions)
                     (EXIT for only one solution)
   V := PickAnUnassignedVariable()
   Assigned[V] := TRUE
   for d := each member of CurDom(V)
      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
```



其实如果直接手跑实验，则同样类比数独，先将每一个格不能填的数字都划掉（循环添加条件直至收敛），然后尝试赋值进入下一层。

例 4. 考虑5个变量A, B, C, D, E，论域都为$\{1, 2, 3, 4\}$，限制条件为$A > D$，$D > E$，$C \neq A$，$C > E$，$C \neq D$，$B \geq A$，$B \neq C$和$C \neq D + 1$。



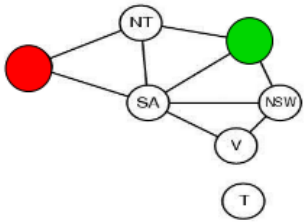
分析.

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，即FC只考虑当前限制下某一变量赋值带来的后果，而GAC则考虑所有限制下变量赋值带来的后果。

- Assign *{Q=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 ≠ 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)的主要语义元素为表示物体、关系和函数的符号(symbol)，分别对应着常量、谓词和函数。前者为抽象概念，后者为具体概念。

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

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \mid [\textit{Sentence}] \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
&\mid \textit{Quantifier}\ \textit{Variable}, \ldots\ \textit{Sentence} \\
\textit{Term} &\rightarrow \textit{Function}(\textit{Term}, \ldots) \\
&\mid \textit{Constant} \\
&\mid \textit{Variable} \\
\textit{Quantifier} &\rightarrow \forall \mid \exists \\
\textit{Constant} &\rightarrow A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} &\rightarrow a \mid x \mid s \mid \cdots \\
\textit{Predicate} &\rightarrow \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} &\rightarrow \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\text{OPERATOR PRECEDENCE} &: \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

FOL与命题逻辑的区别在于本体论的承诺(ontological commitment)，即是否确保实体(reality)的存在。而FOL确保这种承诺的方式是用关于这些句子事实的形式化的模型(model)来表达，其告诉了我们所有可能的世界(possible world)。比如，命题逻辑假设世界上存在要么为真要么为假的事实，而FOL则假设更多，即世界上存在某些存在特定联系(relation)的物体(object)使他们成立或不成立，实际上就是添加了**全称和特称**的概念。

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

定义 8 (公式(formular))**.** 公式包括以下几种情况：

- 若$t_1, \ldots, t_n$都是项，且P是n元谓词符号，则$P(t_1, \ldots, t_n)$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$\neg\alpha, (\alpha \wedge \beta), (\alpha \neg \beta), \exists v.\alpha, \forall v.\alpha$都是公式

通过全称特称符号进行约束的为约束*(bound)*变量，否则为自由*(free)*变量。没有自由变量的公式称为句子*(sentence)*。

注意区别项和公式，项的结果仍然是某一实体，而公式的结果是真值。

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

除了基本的物体、关系和函数，每一个模型还需提供一个假设说明到底什么物体、关系和函数是被常量、谓词和函数所指代的[6]。

[6]关于以下几个概念的解释可见https://max.book118.com/html/2017/1012/136804871.shtm

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、函数符号到具体实体的映射
- 常量符号映射为具体物体（全域U中的元素）
- 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$I(f) : D^n \to D$。如c是一个零元常量符号，则$I(c) \in D$。

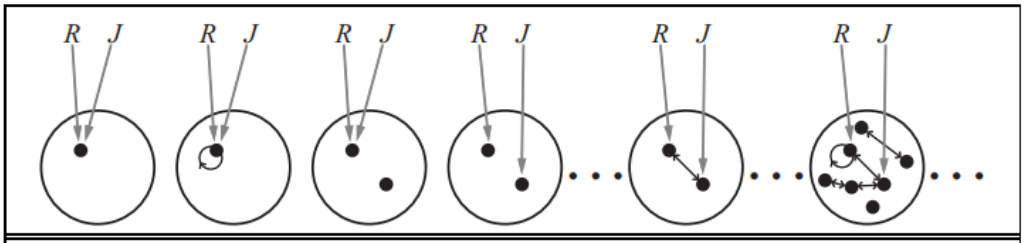


Figure 8.4 Some members of the set of all models for a language with two constant symbols, R and J, and one binary relation symbol. The interpretation of each constant symbol is shown by a gray arrow. Within each model, the related objects are connected by arrows.

考虑一个项$f(t_1, \ldots, t_n)$，函数符号f指向模型中的某一函数F，参数项指向论域中的物体$d_1, \ldots, d_n$，则完整的项指代函数F作用在$d_1, \ldots, d_n$上的值。比如Love是一个函数符号，Bob是一个常量符号，则Love(Bob)指明Bob Loves，也即解释确定了每一个项的引用(reference)。

注意若公式中含有自由变元，则其解释并不对变元进行指派，也得不到具体的真值。

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

$$\|v\|_{\mathcal{I},\mu} = \mu(v)$$
$$\|f(t_1, \ldots, t_n)\|_{\mathcal{I},\mu} = I(f)(\|t_1\|_{\mathcal{I},\mu}, \ldots, \|t_n\|_{\mathcal{I},\mu})$$

解释只是从抽象到具体

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

- $\mathcal{I}, \mu \models P(t_1, \ldots, t_n) \iff \langle \|t_1\|_{\mathcal{I},\mu}, \ldots, \|t_n\|_{\mathcal{I},\mu}\rangle \in I(P)$
  即对于解释$\mathcal{I}$下的μ赋值，t_i对应的具体实体都在P对应的实体关系的论域中
- $\mathcal{I}, \mu \models (t_1 = t_2) \iff \|t_1\|_{\mathcal{I},\mu} = \|t_2\|_{\mathcal{I},\mu}$

同样有一些逻辑基本性质

- $\mathcal{I}, \mu \models \neg\alpha \iff \mathcal{I}, \mu \not\models \alpha$
- $\mathcal{I}, \mu \models (\alpha \wedge \beta) \iff (\mathcal{I}, \mu \models \alpha \wedge \mathcal{I}, \mu \models \beta)$
- $\mathcal{I}, \mu \models (\alpha \vee \beta) \iff (\mathcal{I}, \mu \models \alpha \vee \mathcal{I}, \mu \models \beta)$

- $\mathcal{I}, \mu \models \exists v.\alpha \iff (\exists d \in D : \mathcal{I}, \mu\{d; v\} \models \alpha)$
- $\mathcal{I}, \mu \models \forall v.\alpha \iff (\forall d \in D : \mathcal{I}, \mu\{d; v\} \models \alpha)$

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 5. 通常的问题是下列句子集合是否是可满足的

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

即是否存在这样的解释使全部句子都成立。

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$\alpha$或$\alpha$是$S$一个逻辑结果*(consequence)*。注意$\{\alpha_1, \ldots, \alpha_n\} \models \alpha$当且仅当$\alpha_1 \wedge \cdots \wedge \alpha_n \to \alpha$合法，当且仅当$\alpha_1 \wedge \cdots \wedge \neg\alpha$不可满足。

例 6. 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 (推出(derivation)). 句子序列$c_1, \ldots, c_n = c$，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，这个过程称为推出，记为$S \vdash c$。

注意区别推出/逻辑后承$\vdash$与蕴含算子/实质蕴涵$\to$，见知乎讨论[7]。

定理 4. 若$S \vdash c$，则$S \models c$

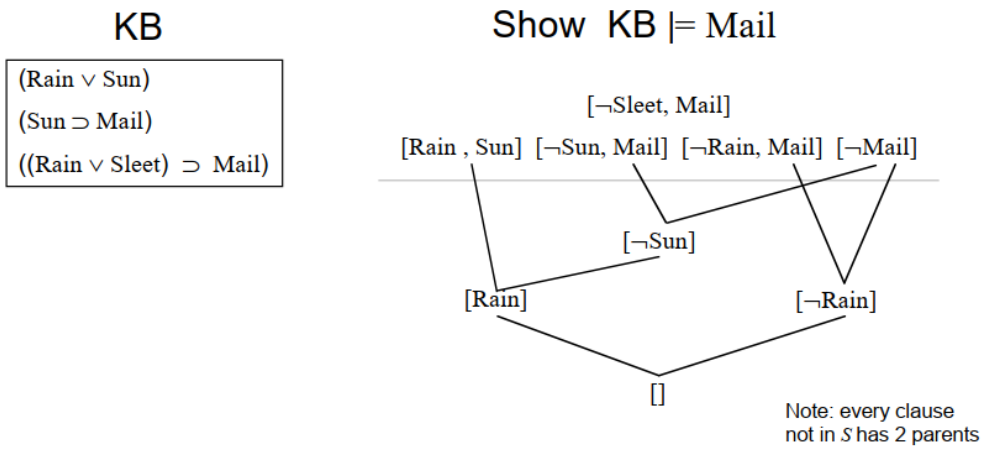
但注意其逆命题并不成立，考虑$(p) \models (p, q)$，但是$(p) \not\vdash (p, q)$。

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$KB \models \alpha \iff KB \wedge \neg\alpha$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$S \vdash ()$

[7]逻辑学中，前提为假而命题为真的推论如何解释？ - 罗心澄的回答 - 知乎 https://www.zhihu.com/question/21020308/answer/16917222



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：$A \to B \iff \neg A \vee B$
- 否定内移：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(skloemize)[8]：引入新的常量符号和函数符号，如$\forall x P(x)$改为$P(g(y))$
- 改为前缀形式：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

例 7. 用归结判断下列句子是否可满足

$$(\exists x \forall y P(x,y) \vee \exists x \forall y Q(x,y)) \to \exists x \forall y (P(x,y) \vee Q(x,y))$$

分析. 这里其实可以把KB看成空，进而S为上述句子的取反。按照子句形式的转换顺序，有

- 消除蕴含

$$(\exists x \forall y P(x,y) \vee \exists x \forall y Q(x,y)) \wedge \neg(\exists x \forall y (P(x,y) \vee Q(x,y)))$$

- 否定内移

$$(\exists x \forall y P(x,y) \vee \exists x \forall y Q(x,y)) \wedge (\forall x \exists y (\neg P(x,y) \wedge \neg Q(x,y)))$$

[8]数学上的例子是$\forall x: \mathrm{d}(x^y)/\mathrm{d}y = x^y$，则我们可以给其一个名字比如说e；但是却不能给一个已经存在的物体，比如说π，再命名。

- 标准化变量

$$(\exists x \forall y P(x,y) \vee \exists z \forall w Q(z,w)) \wedge (\forall a \exists b (\neg P(a,b) \wedge \neg Q(a,b)))$$

- *Skolemize*

$$(\forall y P(x,y) \vee \forall w Q(z,w)) \wedge (\forall a (\neg P(a,f(a)) \wedge \neg Q(a,f(a))))$$

- 改为前缀形式

$$\forall y \forall w \forall a ((P(x,y) \vee Q(z,w)) \wedge (\neg P(a,f(a)) \wedge \neg Q(a,f(a))))$$

- 变成子句形式

1. $P(x,y), Q(z,w)$
2. $\neg P(a,f(a))$
3. $\neg Q(a,f(a))$

归结过程如下

4. $R[1a,2]\{a=x, y=f(a)\}$: $(Q(z,w))$
5. $R[3,4]\{a=z, w=f(a)\}$: $()$

总结即当$a=x, y=f(a), a=z, w=f(a)$时得到归结结果，题目中的式子成立。 **（注意变量写左边，常量写右边）**

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 8. $P(f(x),a)$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 9. 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$\lambda = \{x=a\}$

算法 1 (MGU). 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 $\sigma_{k+1} = \sigma_k\{V = t\}$; $S_{k+1} = S_k\{V = t\}$; $k = k + 1$; Goto 2
5. Else stop, f and g cannot be unified.

4.2.4 完整例子

4.2.4.1 基于反驳的归结推理

例 10. 下面是一个例子

		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] ()

4.2.4.2 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$\forall x P(x) \rightarrow \text{answer}(x)$）
- 直到获得任意子句只包含答案的谓词

例 11. 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)：

- 世界被表示成封建世界知识库(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 12. *$pickup(X)$:*

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

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$a_0, \ldots, a_{n-1}$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$a_0, \ldots, a_{n-1}$也是$P'$的一个解。 □

因此最优放松问题的规划可以作为$A^\star$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：$s_0, a_1, s_1, \ldots, a_n, s_n$，直到$s_n \subset Goal$，或者状态层$s_n$不再改变（到达不动点）。 注意动作层的计算一定要小心是否漏动作。

命题 5. *令$a_0, \ldots, a_{n-1}$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$a_0, \ldots, a_{n-1}$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)
- NewG := G_P U preconditions of A.
- return CountAction(NewG,S_{K-1}) + 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。
CountActions从最后一层往前迭代，将目标G划分为G_P和G_N。

$$\begin{aligned} G_P &= G \cap S_{K-1} \\ G_N &= G - G_P \\ A &= \text{影响成为} G_N \text{的最小覆盖} \\ NewG &= G_P \cup A\text{的前提} \end{aligned}$$

定理 8. 假设$Goal \subset S_k$，对于$i < k$，令A'_{i-1}为调用$CountActions(G_i, S_i)$的结果，则$A'_0, \ldots, A'_{k-1}$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$V_1, \ldots, V_n$以及其对应的有限域$\mathrm{dom}[V_i]$，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

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

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$x \in \mathrm{dom}[X], y \in \mathrm{dom}[Y], z \in \mathrm{dom}[Z]$，

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

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

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

定理 11 (贝叶斯公式). 条件概率定义

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

注意与贝叶斯公式区分

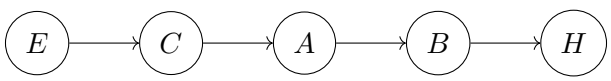
$$\mathbb{P}(Y \mid X) = \frac{\mathbb{P}(XY)}{\mathbb{P}(X)} = \frac{\mathbb{P}(X \mid Y)\,\mathbb{P}(Y)}{\mathbb{P}(X)}$$

定理 12 (链式法则).

$$\mathbb{P}(A_1 \cap A_2 \cap \cdots \cap A_n) = \mathbb{P}(A_1 \mid A_2 \cap \cdots \cap A_n)\,\mathbb{P}(A_2 \mid A_3 \cdots \cap A_n)\,\mathbb{P}(A_{n-1} \mid A_n)\,\mathbb{P}(A_n)$$

6.2 贝叶斯网络

6.2.1 基础知识



有以下概率公式成立

- $\mathbb{P}(H \mid B, A, E, C) = \mathbb{P}(H \mid B)$

- 由链式法则和独立性假设

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

- 通用公式

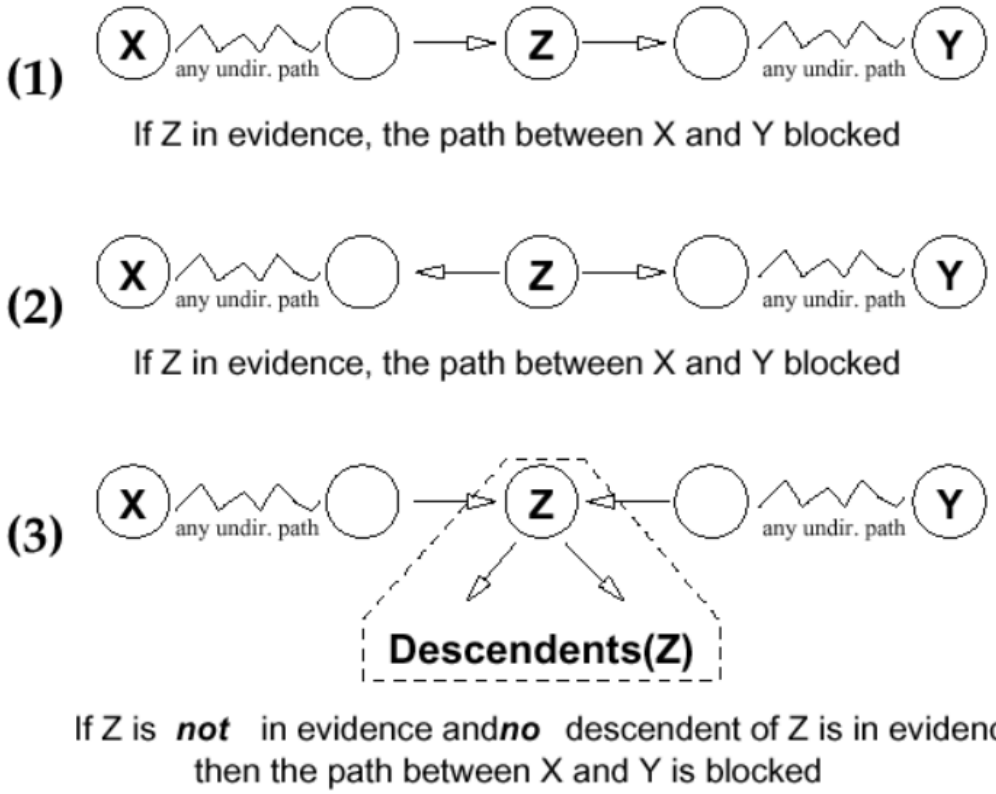
$$\mathbb{P}(X_1X_2\cdots X_n) = \mathbb{P}(X_n \mid Par(X_n))\cdots\mathbb{P}(X_1 \mid Par(X_1))$$

- 单点概率（由全概率公式和条件概率定义）

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

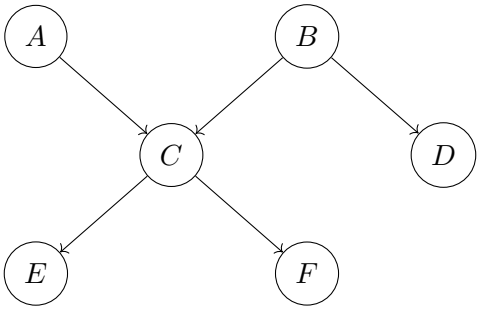
因此每个结点只需存一个条件概率表(conditional probability table, CPT)。

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：（一定要小心箭头方向）



注意第*3*种情况，**交叉结点及其子结点都没有在证据集中给出**，则父亲节点被阻隔。

例 13. 考虑如下贝叶斯网络



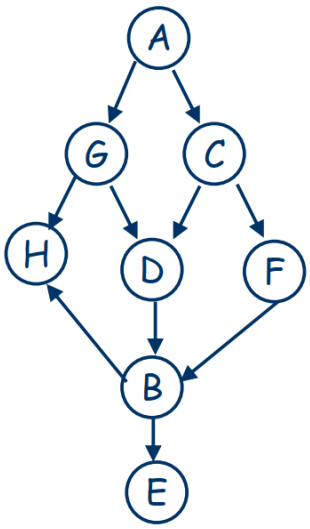
在此例中$\mathbb{P}(c \mid a, b, \neg d, \neg e, \neg f) \neq \mathbb{P}(c \mid a, b)$，要求前者还是拿条件概率展开为全变元非条件概率来求解。

例 14. 考虑下图的贝叶斯网络

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?



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：

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

$$\mathbb{P}(a, d, e) = \sum_{B,C} \mathbb{P}(a, B, C, d, e)$$

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。

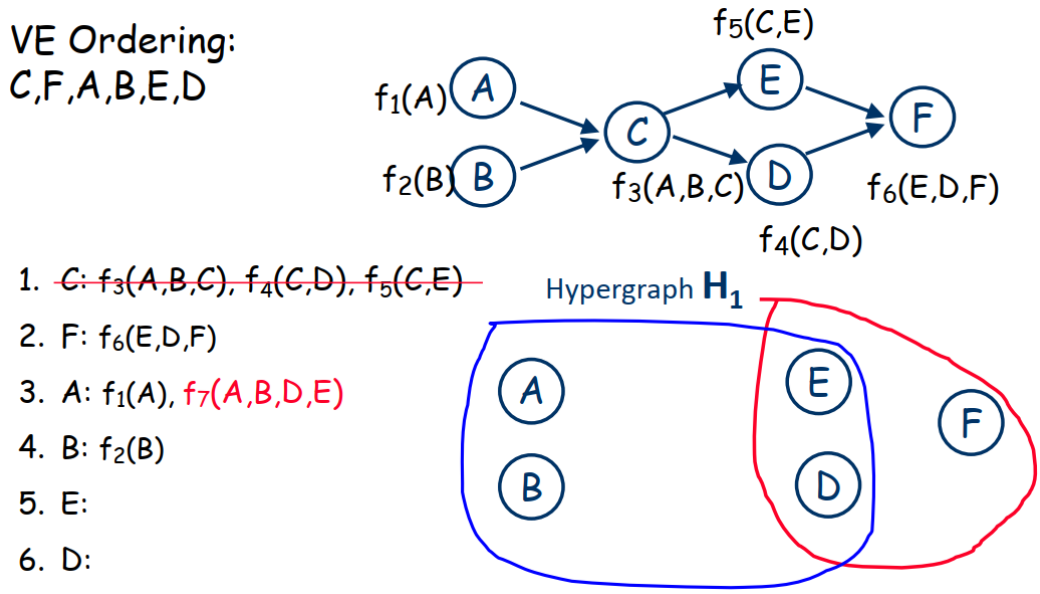
记因子为某些变量的函数，如$\mathbb{P}(C \mid A) = f(A, C)$

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

算法 2 (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：
   - $f_1, f_2, \ldots, f_k$为含有Z_j的因子
   - 计算新的因子$g_j = \sum_{Z_j} f_1 \times f_2 \times \cdots \times f_k$
   - 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。



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

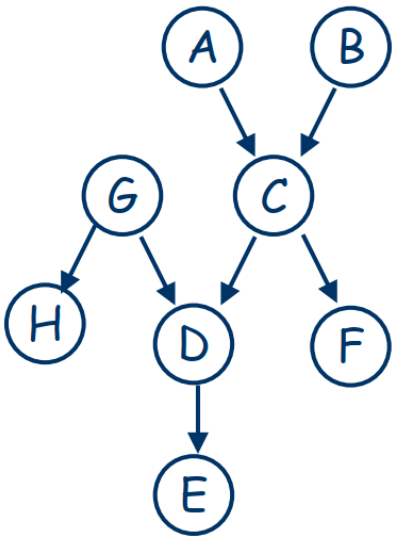
例 15. 若询问$P(F \mid H)$，则D, E, G是不相关的，这种算法会过度估计相关的变量。

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)



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$p_k(k=1,2,\ldots,|\mathcal{Y}|)$，则$D$的信息熵定义为

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

又假定离散属性a有V个可能取值$\{a^1,a^2,\ldots,a^V\}$，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

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

每次选择最大增益的属性进行划分，即

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

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

假设集H内有假设h_i，先验$\mathbb{P}(h_i)$、似然$\mathbb{P}(d\mid h_i)$、证据$d=\langle d_1,d_2,\ldots,d_n\rangle$，有贝叶斯公式（先验推后验）

$$\mathbb{P}(h_i\mid d)=\alpha\mathbb{P}(d\mid h_i)\,\mathbb{P}(h_i)$$

假设独立同分布$\mathbb{P}(d\mid h)=\prod_j\mathbb{P}(d_j\mid h)$，$X$为要预测的值

- 贝叶斯学习：由全概率公式

$$\mathbb{P}(X \mid d) = \sum_i \mathbb{P}(X \mid d, h_i)\,\mathbb{P}(h_i \mid d) = \sum_i \mathbb{P}(X \mid h_i)\,\mathbb{P}(h_i \mid d)$$

- 极大后验(MAP)学习：$\mathbb{P}(X \mid d) \approx \mathbb{P}(X \mid h_{MAP})$

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

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

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

注意贝叶斯在计算时应保证概率和为1，即要进行归一化。 上述三种方法都会在数据增加时收敛。

7.3 朴素贝叶斯

基于属性条件独立性假设

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

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

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

其中$\mathcal{Y}$为所有类别。令D_c表示训练集D中第c类样本组成的集合，有

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

7.4 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

7.4.1 EM算法

最大期望(Expectation-Maximization, EM)算法：用作软聚类

- E步：计算期望，利用对隐藏变量的现有估计值（实际上就是扩充了当前数据，初始化先随机赋值，然后进入迭代），计算其最大似然估计值
- M步：最大化在E步上求得的最大似然值来计算参数的值。

M步上找到的参数估计值被用于下一个E步计算中，这个过程不断交替进行。

7.4.2 K-means算法

- E步：对于每一个类别i和特征X_j，计算每个类别的均值向量

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

- M步：对于每一个样本e，指派e给类别i使得

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

7.5 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：

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

- 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

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

7.6 强化学习

目标：

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

给定策略π：

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

Q学习：随机选择动作a，观察回报r和下一状态s'。注意Q学习的公式有很多表达形式，如下面算法是考虑了带学习率α的更新公式，但不管怎样核心都是**当前回报+折扣下一动作的最大Q值**。

Algorithm 3 Q-Learning

随机初始化$Q[S, A]$
观测当前状态s
repeat
    选择动作a并执行
    观察回报r和状态s'
    依据下述公式进行更新

$$\begin{aligned} Q[s,a] &\leftarrow Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'] - Q[s,a]) \\ &= (1-\alpha)Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a']) \end{aligned}$$

    $s \leftarrow s'$
until 收敛